Computerarchitektur und Betriebssysteme

https://vfhcab.eduloop.de

Stand 2024-02-13 19:45:12

Inhalt

Computerarchitektur und Betriebssysteme ... 8
1 Motivation ... 10
1.1 Hinweise für Studierende ... 11
1.2 Hinweise für Experten ... 12
1.3 Copyright ... 13
2 Computerarchitektur ... 14
2.1 Vom Anwender zur digitalen Schaltung ... 15
2.1.1 Erstmal aufschrauben ... 16
2.1.2 Von-Neumann-Rechner ... 17
2.1.2.1 Von-Neumann-Architektur ... 18
2.1.2.2 Von-Neumann-Flaschenhals ... 19
2.1.3 Komponenten eines Von-Neumann-Rechners ... 20
2.1.3.1 Zentraleinheit / CPU ... 21
2.1.3.1.1 Steuerwerk / Leitwerk ... 22
2.1.3.1.2 Rechenwerk ... 28
2.1.3.2 Speicherwerk ... 31
2.1.3.3 Ein- / Ausgabewerk ... 36
2.1.3.4 Gesamtbild eines Von-Neumann-Rechners ... 38
2.1.3.5 Von-Neumann-Zyklus ... 38
2.1.3.6 Animation der Zusammenarbeit ... 41
2.1.4 Eigenschaften eines Von-Neumann-Rechners ... 43
2.1.5 Digitale Schaltungen ... 44
2.1.5.1 Aufbau und Arbeitsweise eines Registers ... 45
2.1.5.2 Aufbau und Arbeitsweise des Speicherwerks ... 49
2.1.5.3 Aufbau und Arbeitsweise der ALU ... 49
2.1.5.4 Aufbau und Arbeitsweise eines Busses ... 50
2.1.6 Gatter ... 50
2.2 Prozessoren und ihre Befehle ... 52
2.2.1 Vom Quellcode zum Prozessor ... 53
2.2.2 Befehlssatz ... 59
2.2.3 Befehlsformat ... 60
2.2.3.1 Einadressformat ... 61
2.2.3.2 Zweiadressformat ... 62
2.2.3.3 Dreiadressformat ... 62
2.2.3.4 Das Adressformat und der Von-Neumann-Zyklus ... 63
2.2.3.5 Aufgaben & Co. zum Befehlsformat ... 64
2.2.4 Adressierungsarten ... 66

2.2.4.1 Unmittelbare Adressierung ... 67
2.2.4.2 Registeradressierung ... 67
2.2.4.3 Direkte/absolute Adressierung ... 68
2.2.4.4 Registerindirekte Adressierung ... 68
2.2.4.5 Indizierte Adressierung mit Verschiebung ... 69
2.2.5 Vom Programm zum Prozess ... 70
2.2.6 Gesamtbild der Programmausführung ... 72
2.2.7 Aufgaben & Co. zu Prozessoren ... 72
2.3 Weitere Komponenten der Computerarchitektur ... 73
2.3.1 Stackregister ... 75
2.3.2 Basisregister ... 76
2.3.2.1 Mehrere Prozesse gleichzeitig im Speicher ... 78
2.3.2.2 Swapping: Aus- und Einlagern von kompletten Prozessen ... 81
2.3.3 Limitregister zum Speicherschutz ... 82
2.3.4 Interrupt-Controller ... 85
2.3.4.1 Gründe für eine Interrupt-Auslösung ... 89
2.3.4.2 Speicherschutzverletzung ... 90
2.3.4.3 Quasi-gleichzeitige Ausführung mehrerer Prozesse ... 91
2.3.4.4 Kommunikation mit E/A-Geräten ... 93
2.3.4.4.1 Allgemeiner Aufbau eines Controllers ... 94
2.3.4.4.2 Zeit und Kosten machen den Unterschied ... 96
2.3.4.4.3 Datentransfer und Interrupts ... 98
2.3.5 DMA-Controller ... 102
2.3.5.1 Aufbau und Arbeitsweise eines DMA-Controllers ... 106
2.3.5.2 Direkt kann wirklich direkt bedeuten ... 109
2.3.6 MMU - Memory Management Unit ... 112
2.3.7 Moderne Bussysteme ... 116
2.4 Fazit Computerarchitektur ... 119
3 Betriebssysteme ... 120
3.1 Einführung Betriebssysteme ... 121
3.1.1 Geschichtlicher Überblick zu Betriebssystemen ... 123
3.1.2 Nur ein Prozessor mit einem Kern ... 123
3.1.3 Zwischen Benutzer und Hardware ... 124
3.1.4 Betriebsmittel ... 126
3.1.5 Betriebsmittel sind Prozessen zugeordnet ... 127
3.1.6 Zentrale Aufgabe eines Betriebssystems ... 130
3.1.7 Betriebssystemarchitekturen ... 131
3.1.8 Betriebssystemarten ... 131

3.1.9 Vom Batch-Job zum Multitasking ... 133
3.1.10 Kernel-Mode, User-Mode und Systemaufrufe ... 134
3.2 Prozessverwaltung ... 138
3.2.1 Prozess ... 139
3.2.2 Prozesskontext ... 140
3.2.3 Kontextwechsel ... 140
3.2.3.1 Die Statistik der Kontextwechsel unter Windows ... 141
3.2.3.2 Die Statistik der Kontextwechsel unter Linux ... 142
3.2.4 Prozesse erzeugen ... 143
3.2.4.1 Fork ... 144
3.2.4.2 CreateProcess ... 146
3.2.5 Prozesskontrollblock ... 147
3.2.5.1 Prozesskontrollblock unter Windows ... 148
3.2.5.2 Prozesskontrollblock unter Linux ... 148
3.2.6 Prozesstabelle ... 158
3.2.7 Prozesszustände ... 158
3.2.8 Verwalten von Prozessen ... 160
3.2.8.1 Prozessverwaltung aus Admin-Sicht unter Windows ... 160
3.2.8.2 Prozessverwaltung aus Admin-Sicht unter Linux ... 161
3.2.9 Threads ... 162
3.2.9.1 Java-Beispiel mit Threads ... 166
3.2.9.2 Prozesse und Threads unter Windows ... 167
3.2.9.3 Prozesse und Threads unter Unix und Linux ... 168
3.2.10 Scheduling ... 168
3.2.10.1 Scheduling-Ziele ... 172
3.2.10.2 Scheduling-Verfahren ... 173
3.2.10.2.1 First Come First Serve ... 175
3.2.10.2.2 Shortest Job First ... 176
3.2.10.2.3 Shortest Remaining Time Next ... 178
3.2.10.2.4 Round Robin ... 179
3.2.10.2.5 Priority Scheduling ... 182
3.2.10.2.6 Weitere Verfahren ... 184
3.2.10.3 Scheduling in gängigen Betriebssystemen ... 184
3.2.10.4 Vergleichskriterien ... 185
3.2.11 Synchronisation ... 185
3.2.11.1 Grundlegende Begriffsdefinitionen zur Synchronisation ... 186
3.2.11.1.1 Nebenläufigkeit ... 187
3.2.11.1.2 Race Conditions ... 188

3.2.11.1.3 Kritischer Abschnitt ... 190
3.2.11.2 Aktives Warten ... 194
3.2.11.2.1 Aktives Warten mit while ... 195
3.2.11.2.2 Das Problem des ungünstigsten Moments ... 199
3.2.11.2.3 Aktives Warten mit TSL ... 201
3.2.11.3 Semaphore ... 203
3.2.11.3.1 Mutex ... 206
3.2.11.3.1.1 Wechselseitiger Ausschluss ... 207
3.2.11.3.1.2 Reihenfolgedurchsetzung ... 214
3.2.11.3.2 Zählsemaphor ... 215
3.2.11.3.2.1 Erzeuger- / Verbraucherproblem ... 216
3.2.11.3.2.2 Philosophenproblem ... 216
3.2.11.4 Monitore ... 217
3.2.11.5 Zusammenfassung Synchronisation ... 218
3.2.11.6 Synchronisationstechniken moderner Betriebssysteme ... 219
3.2.11.7 Synchronisationsmechanismen in Programmiersprachen ... 219
3.2.12 Deadlocks ... 219
3.2.12.1 Vier Bedingungen nach Coffman ... 221
3.2.12.2 Deadlocks erkennen ... 222
3.2.12.3 Deadlocks ignorieren ... 226
3.2.12.4 Deadlocks vermeiden ... 226
3.2.12.5 Deadlocks verhindern ... 228
3.2.12.6 Deadlock-Fazit ... 229
3.2.13 Interprozesskommunikation ... 230
3.2.13.1 Zwei Threads kommunizieren über gemeinsame Variablen ... 231
3.2.13.2 Zwei Prozesse kommunizieren über gemeinsame Speicherobjekte ... 231
3.2.13.3 Zwei Prozesse kommunizieren über Shared Memory ... 232
3.2.13.4 Zwei Prozesse kommunizieren über Pipes ... 233
3.2.13.5 Zwei Prozesse kommunizieren über Sockets ... 233
3.2.13.6 Interprozesskommunikation-Fazit ... 234
3.3 Speicherverwaltung ... 235
3.3.1 Virtuelle Speicherverwaltung ... 236
3.3.1.1 Arbeitsweise der MMU ... 240
3.3.1.2 Seitentabellen ... 240
3.3.1.2.1 Einstufige Seitentabellen ... 241
3.3.1.2.2 Mehrstufige Seitentabellen ... 245
3.3.2 Swapping und Paging ... 246
3.3.2.1 Page Fault ... 248

3.3.2.2 Seitenersetzung ... 251
3.3.2.2.1 Was bei der Seitenersetzung passiert ... 253
3.3.2.2.2 Das Modifiziert-Bit ... 254
3.3.2.2.3 Seitenersetzungsverfahren ... 256
3.3.2.2.3.1 Optimaler Seitenersetzungsalgorithmus ... 258
3.3.2.2.3.2 NRU - Not Recently Used Algorithmus ... 259
3.3.2.2.3.3 FIFO - First In First Out Algorithmus ... 262
3.3.2.2.3.4 Second Chance Algorithmus ... 263
3.3.2.2.3.5 Working Set Algorithmus ... 263
3.3.3 Shared Memory ... 265
3.3.4 Speicherverwaltung moderner Betriebssysteme ... 268
3.3.4.1 Speicherverwaltung unter Linux ... 268
3.3.4.2 Speicherverwaltung in Windows ... 269
3.3.4.3 Speicherverwaltung unter Android ... 269
3.4 Geräteverwaltung ... 269
3.4.1 Rolle der Geräteverwaltung ... 271
3.4.2 Abhängig und gleichzeitig unabhängig ... 273
3.4.3 Gerätetreiber ... 273
3.4.4 Aufgaben eines Treibers ... 275
3.4.4.1 Initialisierung des Geräte-Controllers ... 275
3.4.4.2 Gerät dem Betriebssystem bekannt machen ... 276
3.4.4.3 Bereitstellen einer Schnittstelle zum Geräte-Controller ... 278
3.4.4.4 Interruptbehandlung für ein Gerät ... 282
3.4.4.5 Bereitstellen einer Schnittstelle zur Geräteverwaltung ... 286
3.4.4.6 Pufferung von Daten ... 287
3.4.4.7 Koordination nebenläufiger Zugriffe auf ein Gerät ... 288
3.4.5 Geräteklassen ... 288
3.4.5.1 Blockorientierte Geräte ... 288
3.4.5.2 Zeichenorientierte Geräte ... 292
3.4.5.3 Sonstige Geräte ... 295
3.4.6 Memory-Mapped-I/O ... 295
3.4.7 DMA - Direct Memory Access ... 295
3.4.8 Windows-Treiber auf GitHub ... 295
3.5 Dateiverwaltung ... 296
3.5.1 Datei ... 297
3.5.2 Dateisystem ... 298
3.5.3 Aufgaben eines Dateisystems ... 299
3.5.4 Dateisystemkonzepte ... 300

3.5.5 Von Windows unterstützte Dateisysteme ... 300
3.5.5.1 FAT - File Allocation Table ... 303
3.5.5.2 NTFS - New Technology File System ... 307
4 Aufgaben zur Prüfungsvorbereitung ... 309
4.1 Aufgaben zum Kapitel Vom Anwender zur digitalen Schaltung ... 309
4.2 Aufgaben zum Kapitel Prozessoren und ihre Befehle ... 312
4.3 Aufgaben zum Kapitel Weitere Komponenten der Computerarchitektur ... 316
4.4 Aufgaben zum Kapitel Einführung Betriebssysteme ... 318
4.5 Aufgaben zum Kapitel Prozessverwaltung ... 318
4.5.1 Aufgaben zu Prozesse und Threads ... 319
4.5.2 Aufgaben zum Scheduling ... 321
4.5.3 Aufgaben zur Synchronisation ... 323
4.5.4 Aufgaben zu Deadlocks ... 326
4.6 Aufgaben zum Kapitel Speicherverwaltung ... 328
4.7 Aufgaben zum Kapitel Geräteverwaltung ... 332
4.8 Aufgaben zum Kapitel Dateiverwaltung ... 335
5 Logbuch ... 338
5.1 Jahr 2013 ... 338
5.2 Jahr 2014 ... 342
5.3 Jahr 2015 ... 343
5.4 Jahr 2016 ... 344
6 Anhang ... 345
6.1 Zweier-Potenzen ... 345
6.2 Bits und Bytes ... 347
6.3 GiB, MiB, KiB im Vergleich zu GB, MB, KB ... 347
6.4 Java-Applets ... 348
6.5 Inhalt dieses LOOPs ausdrucken ... 350
6.6 Impressum ... 361

Anhang

I Literaturverzeichnis ... 364
II Abbildungsverzeichnis ... 366
III Medienverzeichnis ... 372
IV Listingverzeichnis ... 373
V Aufgabenverzeichnis ... 374
VI Index ... 380

Computerarchitektur und Betriebssysteme

Über den Autor

Andreas Wilkens

www.dozaw.de

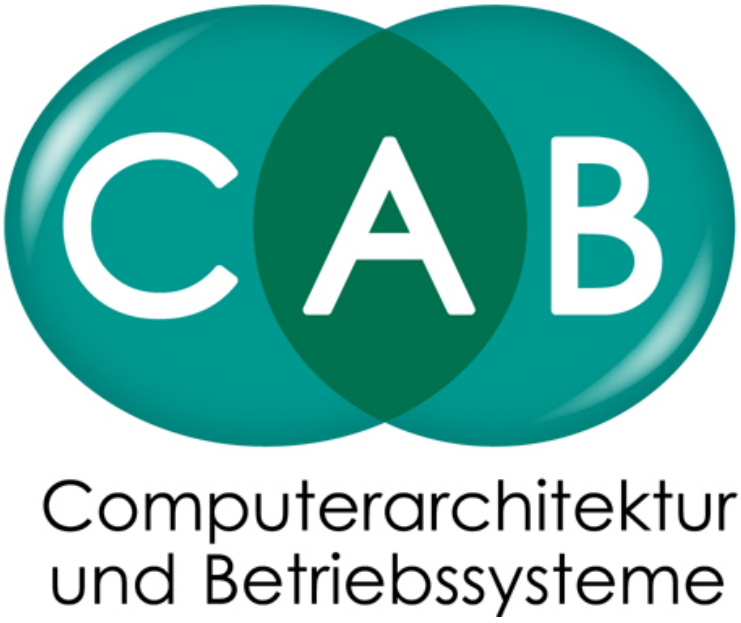
Kontakt:

E-Mail: aw(at)dozaw.de

Impressum:

vfhcab.eduloop.de/loop/Impressum

Hier geht es los!



Gliederung

Computerarchitektur und Betriebssysteme
1 Motivation
2 Computerarchitektur
3 Betriebssysteme
4 Aufgaben zur Prüfungsvorbereitung
5 Logbuch
6 Anhang

Seit dem Wintersemester 2013/14 werden die hier veröffentlichten Inhalte im gleichnamigen Modul "Computerarchitektur und Betriebssysteme" des Studiengangs "Online-Medieninformatik" an verschiedenen Hochschulen des Verbundes Virtuelle Fachhochschule als Lernmaterialien verwendet.

Ein Modul wie dieses wird niemals fertig. Viele Inhalte sind zwar vorhanden, jedoch fehlen immer noch einzelne Teile oder Unterkapitel. Von Zeit zu Zeit werden Erweiterungen vorgenommen.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

1 Motivation

Die Vorlesung "Betriebssysteme" gehört zu den Klassikern in der Informatik-Ausbildung. Aber jeder Dozent, der diese Veranstaltung ein oder mehrmals gehalten hat, stellt immer wieder fest, dass wichtige Konzepte von Betriebssystemen von den Studierenden gar nicht, oder nur lückenhaft, verstanden werden. Der Grund hierfür kann unter anderem in fehlenden Kenntnissen der zu Grunde liegenden Computerarchitektur vermutet werden.

Hieraus entstand letztlich die Motivation dazu, nicht nur ein reines Betriebssystem-Modul zu erstellen, sondern zuvor eine Einführung in die wichtigsten und bedeutsamsten Aspekte der Computerarchitektur zu geben.

Die Zielgruppe dieses Kurses sind Studierende im ersten Semester, sowie alle weiteren interessierten Personen, die gerne mehr über die Thematik erfahren möchten.

Der Aufbau und die Gliederung der einzelnen Kapitel folgt jeweils den Grundsätzen:

- Vom Bekannten zum Unbekannten.
- Vom Leichten zum Schweren.
- Vom Einfachen zum Komplexen.
- Abstraktion erleben, erlernen, anwenden.

Dieses Vorgehen soll ein schnelles Begreifen der Inhalte fördern und so den Lernerfolg erhöhen. Insbesondere der letztgenannte Punkt, die Abstraktion, mag an dieser Stelle für Diskussionsstoff sorgen. Aus eigener Erfahrung des Autors dieser Zeilen folgt, dass viele Studierende sich (gerade zu Beginn ihres Studiums) mit der Abstraktion sehr schwer tun. Auf der anderen Seite gibt es viele Fachbücher und Lehrveranstaltungen, die von Anfang an auf Abstraktion setzen. Es prallen also zwei Welten aufeinander.

Viele Studierende wünschen sich nun, dass die Hochschulen und Fachbücher sich ihnen anpassen, dass also die Abstraktion vermieden wird. Aber (und das ist jetzt wieder die persönliche Meinung des Autors) das ist nicht sinnvoll. Abstraktion ist, wenn man sich erst einmal daran gewöhnt hat, ein wunderbares Mittel in vielen Lebenslagen, insbesondere in Lehre und Forschung. Die Studierenden müssen also lernen mit Abstraktion umzugehen, sie zu verstehen, und letztlich auch sie selbst anzuwenden.

Die Inhalte sollen - so weit wie möglich - nicht nur als Text und Grafik aufbereitet sein (dann könnte man ja gleich ein Buch lesen), sondern die vielfältigen Möglichkeiten des Internets sollen ganz bewusst genutzt werden, um ausdrücklich jede Variante der Informationsvermittlung einzubinden. Das ist ein hochgestecktes Ziel und es wird sicher den Einen oder Anderen geben, der hierzu weitere Ideen entwickelt. Falls dem so ist: Macht alle mit und erweitert dieses Werk!

Die Umsetzung von Computerarchitektur und Betriebssysteme erfolgt mit Hilfe von LOOP - Learning Object Online Platform . Loop ist eine flexible Autorensoftware, die auf Mediawiki basiert und insbesondere eine Zusammenarbeit von verschiedenen Autoren ermöglicht.

Viel Spaß beim Durcharbeiten wünscht

Andreas Wilkens
im Juni 2013

http://www.dozaw.de

1.1 Hinweise für Studierende

Dieses Online-Modul dient dem selbständigen Erarbeiten der Inhalte. Links am Rand befindet sich der Kasten "Ausgabeformate". Über die dort enthaltenen Links kann eine Offline-Version des Moduls erzeugt und heruntergeladen werden.

Hinweis

Hinweis

Dieses Modul gehört zu den sogenannten "Cloud-Learning-Modulen", d.h. eine optimale Nutzung setzt eine dauerhafte Internet-Verbindung voraus. Viele Inhalte lassen sich nicht in die unterstützten (offline) Ausgabeformate überführen. Die Offline-Variante stellt damit nur eine Art von "Notlösung" dar. Das ist sicher nicht für alle befriedigend, aber es ermöglicht andere Wege in der Wissensvermittlung, und auch andere Wege der kollaborativen Weiterentwicklung dieses Moduls.

Wer zwingend auf Offline-Materialien angewiesen ist, dem seien die unten aufgelisteten downloadbaren Offline-Inhalte empfohlen. Weiterführend natürlich auch die im Literaturverzeichnis aufgeführten Bücher, die teilweise über Springerlink heruntergeladen werden können.

Downloadbare Offline-Inhalte

In der folgenden Liste sind verschiedene Materialien aufgeführt, die aus dem Internet heruntergeladen und offline genutzt werden können.

- PDF-Version dieses Lernmoduls
- Rechner- und Betriebssysteme, Kommunikationssysteme, Verteilte Systeme ; Online-Ausgabe eines Lehrtextes der Otto-Friedrich-Universität Bamberg; Autoren: Martin Eisenhardt, Andreas Henrich, Stefanie Sieber; (Creative Commons BY-NC-ND-Lizenz)

Downloadbare Bücher von Springerlink

Über Springerlink können Angehörige vieler Hochschulen (dazu zählen insbesondere auch Studierende!) Fachbücher, Journale, veröffentlichte Forschungsergebnisse, etc. ohne weitere entstehende Kosten herunterladen.

Grundlage dafür bietet jeweils ein Vertrag, den die betreffende Hochschule (bzw. deren Bibliothek) mit den Betreibern der Internet-Plattform Springerlink abgeschlossen hat.

Hinweis

Der Download ist dabei i.d.R. nur möglich, sofern

- entweder der Zugriff auf die Online-Ressource aus dem Rechnernetz der betreffenden Hochschule erfolgt, (Springerlink prüft offenbar beim ersten Verbindungsaufbau der Browser-Session die Absender-IP-Adresse)
- oder nachdem man sich auf der Springerlink-Seite mit gültigen Login-Daten der Hochschule angemeldet hat.

Studierende, deren Hochschulen sich der *eduroam-Initiative* angeschlossen haben, können sich mit ihren Zugangsdaten üblicherweise in den WLAN-Netzen vieler Hochschulen im In- und Ausland anmelden. In den meisten Fällen dürfte der Zugriff auf Ressourcen bei Springerlink aus diesen WLAN-Netzen heraus möglich sein.

Hinweis

Im Literaturverzeichnis findest du jeweils einen direkten Link zum Buch bei Springerlink , sofern dieses hierüber verfügbar ist. Beachte die gerade beschriebene Zugriffsbeschränkung über die Rechnernetze der Hochschulen! Erkundige dich ggf. bei deiner Hochschule.

1.2 Hinweise für Experten

Zitat

"Wenn alle Experten sich einig sind, ist Vorsicht geboten."
Bertrand Russell , *1872 bis †1970,
britischer Philosoph und Mathematiker, 1950 Nobelpreis für Literatur

Ein schönes Zitat, nicht wahr? Es drückt aus, was in unserer Welt natürlich ist. Andere Menschen, andere Meinungen. So soll es auch sein!

Und genau deshalb ist die Idee hinter diesem Modul (eigentlich: die Idee hinter LOOP , der Autorensoftware, mit der dieses Modul erstellt wurde), dass viele Menschen mit ihrem Wissen an der Weiterentwicklung dieses Moduls teilhaben können.

Das fängt an bei den Experten für Rechtschreibung und Interpunktion, die gerne entsprechende Fehler korrigieren können. (Bitte tun Sie es!) Es geht weiter bei den Experten für digitale Dinge aller Art, die ein zusätzliches Video, eine Animation oder irgendetwas anderes erstellen, um Sachverhalte aus dem Modul auf zusätzliche Weise zu erläutern. Und es endet bei den (Wissens-) Experten des Fachgebiets, die neue oder überholte Inhalte dieses Moduls identifizieren und auf den aktuellen Stand bringen können.

Und natürlich ist klar, dass ein Modul wie dieses nicht gänzlich fehlerfrei sein kann. Aber mit Hilfe von LOOP können alle erkannten Fehler schnell und einfach beseitigt werden. Ein unschlagbarer Vorteil von Cloud-Learning-Materialien.

So kann man mitarbeiten

Hinweis

Einfach ein kostenloses Benutzerkonto anlegen , und schon kann es losgehen. Angemeldete Benutzer können Seiten editieren und/oder neue Seiten hinzufügen. Änderungen oder Ergänzungen werden vor der Veröffentlichung noch einmal von einer verantworlichen Person geprüft (4-Augen-Prinzip), anschließend werden sie freigeschaltet.

1.3 Copyright

Bei der Erstellung von Computerarchitektur und Betriebssysteme war es ein Ziel, eine Copyright-Lizenz mit möglichst wenig Einschränkungen zu verwenden. Soweit nicht anders vermerkt, ist dieses die Creative Commons Namensnennung 3.0 Unported Lizenz .

http://i.creativecommons.org/l/by/3.0/88x31.png

Üblicherweise wird die für eine einzelne Seite gültige Lizenz jeweils am Ende der Seite erwähnt. Da immer wieder auch auf Inhalte von externen Webseiten (z.B. Texte, Grafiken oder Videos) verwiesen wird, deren Veröffentlichung unter einer abweichenden Lizenz erfolgte, ist zu beachten, dass prinzipiell jede einzelne Seite unter einer anderen Lizenz veröffentlicht sein kann, sogar unterschiedliche Teile einer einzelnen Seite können verschiedenen Lizenzen unterliegen.

2 Computerarchitektur

Dieses Lernmodul wurde entwickelt für Studierende im ersten Semester.

Das Ziel einer jeden Hochschulausbildung ist immer, möglichst alle Studierenden am Ende eines Semesters auf das gleiche Wissensniveau zu bringen. Ein übliches Mittel dazu ist die "Bottom-Up-Strategie", welche praktisch *"bei Null"* beginnt und mit den elementarsten Grundlagen anfängt.

Dieses Lernmodul geht den entgegengesetzten Weg: "Top-down". D.h. es wird ein gewisses Vorwissen vorausgesetzt und dieses vorhandene Vorwissen wird verfeinert und erweitert. Die Denkweise soll in bestimmte Bahnen gelenkt werden, damit die Studierenden eigene Schlüsse ziehen können, und früher oder später *"der Groschen fällt"*.

Doch welches Vorwissen kann bei Studierenden im ersten Semester im Bereich Computerarchitektur vorausgesetzt werden? Und zwar bei allen (!) Studierenden gleichermaßen? (Wenn sicherlich auch nicht überall gleich stark ausgeprägt.)

Kleinster gemeinsamer Nenner

Was ist *"der kleinste gemeinsame Nenner"* in Bezug auf die Computerarchitektur bei allen Studierenden?

Es ist die *"Anwendung des Computers"*. Alle Studierenden sind Anwender an ihrem PC oder Laptop:

- Sie wissen üblicherweise wie Tastatur, Maus, Monitor usw. angeschlossen werden.
- Sie können das Gerät einschalten.
- Sie können mit den Komponenten (--> Maus, Tastatur, Drucker, etc.) umgehen.
- Sie können mit Betriebssystem und Anwendungsprogrammen in normalem Umfang arbeiten.

Und bei genau dieser Gemeinsamkeit beginnt dieses Lernmodul auf der nächsten Seite. Vorher aber noch eine Definition:

Definition: Computerarchitektur

Definition

Die **Computerarchitektur** oder **Rechnerarchitektur** ist ein Teilgebiet der (technischen) Informatik, welche sich mit dem internen sowie externen Aufbau eines Computersystems beschäftigt.

So geht es weiter:

2 Computerarchitektur
2.1 Vom Anwender zur digitalen Schaltung

Gliederung

2.2 Prozessoren und ihre Befehle
2.3 Weitere Komponenten der Computerarchitektur
2.4 Fazit Computerarchitektur

Weblinks

Rechnerarchitektur bei Wikipedia

Literatur

Einen umfassenden Einblick in das Thema Computerarchitektur liefert *Tanenbaum 2005* . Weitere Zusammenfassungen sind zu finden bei *Eisenhardt et.al. 2007* sowie bei *Plate DVS* .

2.1 Vom Anwender zur digitalen Schaltung

Wichtig

Hallo Welt!

Liebe Anwenderin,

Lieber Anwender,

schön, dass du da bist!

Überblick

Das folgende Video zeigt in einem kurzen Überblick die Themen der Unterkapitel.

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/DcsVh7UtrO0

Med. 1: Überblick: Vom Anwender zur digitalen Schaltung (00:54)
http://youtu.be/DcsVh7UtrO0
CC-BY

So geht es weiter:

Gliederung

2.1 Vom Anwender zur digitalen Schaltung
2.1.1 Erstmal aufschrauben
2.1.2 Von-Neumann-Rechner
2.1.3 Komponenten eines Von-Neumann-Rechners
2.1.4 Eigenschaften eines Von-Neumann-Rechners
2.1.5 Digitale Schaltungen
2.1.6 Gatter

2.1.1 Erstmal aufschrauben

Dass alle Leserinnen und Leser schon einmal mit einem Computer gearbeitet haben, wird an dieser Stelle vorausgesetzt. Aber wie sieht so ein Gerät im Innern aus? Die Antwort folgt mit dem Schraubenzieher in der Hand.

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/tY6DkMy2e18

Med. 2: Erstmal aufschrauben! (08:31)
http://youtu.be/tY6DkMy2e18
CC-BY

In diesem Video wird ein handelsüblicher PC aufgeschraubt und auseinander gebaut. Einzelne Bauteile (Komponenten) werden dadurch sichtbar.

Hinweis

Hinweis!
Da der im Film verwendete PC bereits defekt war, wurden keine Vorkehrungen zur Vermeidung von Schäden durch elektrostatische Aufladung getroffen. Es wurde allerdings explizit darauf geachtet, den Stromfluss durch ziehen des Netzsteckers vor dem Aufschrauben des Gehäuses zu unterbrechen. Falls die im Film gezeigte Szene anhand eines eigenen Geräts nachgestellt werden soll, so ist auf die Einhaltung von Sicherheitsmaßnahmen zu achten. Bei Nichteinhaltung besteht Lebensgefahr oder die Gefahr von Schäden am Gerät. Weitere Hinweise dazu gibt es beispielsweise unter http://de.wikipedia.org/wiki/F%C3%BCnf_Sicherheitsregeln

Komponenten

Die im Video identifizierten Komponenten des PC sind:

- CPU (Central Processing Unit, Hauptprozessor)
- RAM (Random-Access Memory, Hauptspeicher oder Arbeitsspeicher)
- HDD (Hard Disk Drive, Festplattenlaufwerk oder Festplatte)
- CD-ROM (Compact Disc Read-Only Memory, CD-Laufwerk)
- DVD-Brenner (Digital Versatile Disc, Digitale Vielseitige Scheibe)
- FDD (Floppy Disk Drive, Diskettenlaufwerk)
- NIC (Network Interface Card, Netzwerkkarte)

- Mainboard (Hauptplatine)

Eine Grafikkarte wurde nicht als eigenständige Komponente ausgebaut, sondern befindet sich als "On-Board-Grafik" fest verlötet auf der Hauptplatine.
Darüber hinaus kann es noch eine Reihe weiterer Teile geben, die im Video nicht direkt zu sehen waren, oder die im gezeigten Beispielrechner ganz einfach nicht eingebaut waren.

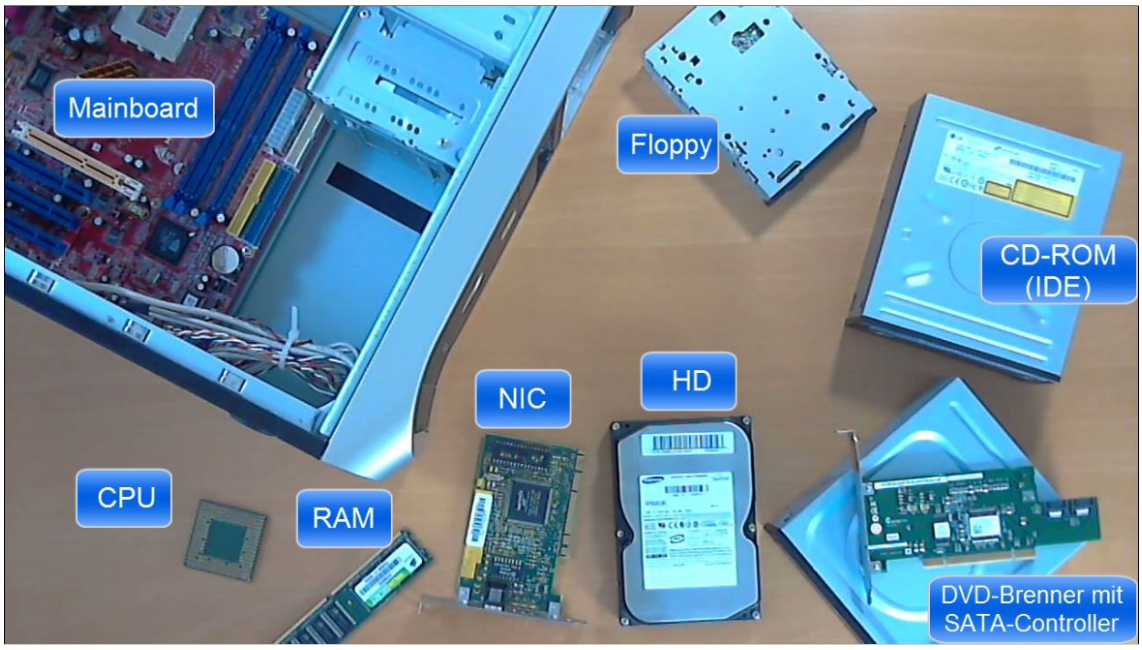


Abb. 1: Das Mainboard und die ausgebauten Komponenten
CC-BY

Über die Hauptplatine sind alle Komponenten miteinander verbunden. Das ist in modernen Systemen ein überaus kompliziertes Verbindungsgeflecht, weshalb man sich dieser Materie zunächst mit Hilfe einer sehr vereinfachten Darstellung nähert, der Von-Neumann-Architektur.

2.1.2 Von-Neumann-Rechner

John von Neumann lebte von *1903 bis †1957. Er gilt als einer der Väter der Informatik.

Abb. 2: John von Neumann (um 1940)
Bildquelle: 'http://upload.wikimedia.org/wikipedia/commons/thumb/5/5e/JohnvonNeumann-LosAlamos.gif/184px-JohnvonNeumann-LosAlamos.gif'
By LANL [Public domain or Public domain], via Wikimedia Commons

Nach ihm wurde die "Von-Neumann-Architektur" benannt, deren Prinzip er im Jahre 1945 in seinem Aufsatz *"First Draft of a Report on the EDVAC "* beschrieb. Der Grundidee dieser Architektur folgen noch heute nahezu alle modernen Rechner - wenn auch mit einer Reihe von Optimierungen.

Bevor auf der folgenden Seite die Von-Neumann-Architektur näher betrachtet wird, kann bereits die Definition eines Von-Neumann-Rechners erfolgen:

Definition: Von-Neumann-Rechner

Definition

Einen Computer (oder Rechner), der nach den Prinzipien der Von-Neumann-Architektur aufgebaut ist, nennt man einen **Von-Neumann-Rechner** oder kurz **VNR**.

2.1.2.1 Von-Neumann-Architektur

Das folgende Video erläutert den Aufbau der Von-Neumann-Architektur:

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/UomUEphRAwM

Med. 3: Von-Neumann-Architektur (01:35)
http://youtu.be/UomUEphRAwM
CC-BY

Skizze der Von-Neumann-Architektur

Dies ist die im Video erarbeitete Skizze der Von-Neumann-Architektur:

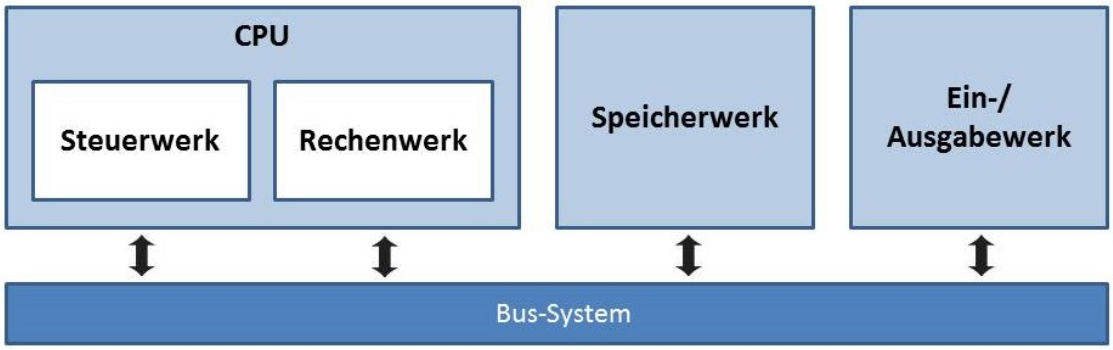


Abb. 3: Von-Neumann-Architektur
CC-BY

Definition: Von-Neumann-Architektur

Definition

Die **Von-Neumann-Architektur** (kurz: **VNA**) bildet ein Referenzsystem für Computer (oder Rechner) und besteht aus den Komponenten

- CPU mit Steuerwerk und Rechenwerk
- Ein-/Ausgabewerk
- Speicherwerk
- Bus-System

Bus-System

Das Bus-System bildet das Bindeglied aller Komponenten. Mit ihm wird eine sehr einfache Kommunikationsmöglichkeit von und zu jeder einzelnen Komponente geschaffen. ("Bus" ist die Abkürzung für "Binary Unit System", etwas später folgt eine genauere Definition dieses Begriffs.)

Hinweis

Das Bus-System ist zwar eine sehr einfache Möglichkeit, die Kommunikation unter allen beteiligten Komponenten zu ermöglichen, gleichzeitig ist es aber auch der größte Kritikpunkt an dieser Architektur. Es kommt zum sogenannten Von-Neumann-Flaschenhals.

2.1.2.2 Von-Neumann-Flaschenhals

Den "Flaschenhals" der Von-Neumann-Architektur bildet das Bus-System. Genauer wird unter dem **Von-Neumann-Flaschenhals** der Sachverhalt verstanden, dass bei einem Bus-System immer nur eine angeschlossene Komponente zur Zeit schreibend auf

den Bus zugreifen darf. Alle anderen Komponenten können in dieser Phase höchstens lesend den Bus nutzen.

Kollision

Sollten einmal zwei (oder mehr) angeschlossene Komponenten zur selben Zeit auf den Bus schreiben, so kommt es zu einer Überlagerung der Signale, wodurch diese unbrauchbar werden. Man nennt dies eine **Kollision** auf dem Bus.

Das folgende Video geht auf diese Kollisionen ein:

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/q1sdBp-8fYM

Med. 4: Von-Neumann-Flaschenhals (01:53)
http://youtu.be/q1sdBp-8fYM
CC-BY

Strategien zur Kollisionsvermeidung

Damit ein Rechner trotz Von-Neumann-Flaschenhals funktionieren kann, sind Strategien zur Vermeidung und Erkennung von Kollisionen auf dem Bus nötig. Letztlich ist diesen Strategien gemein, dass sie sich negativ auf die Geschwindigkeit auswirken, mit der eine Datenübertragung über den Bus möglich ist. Damit ist dann auch die Geschwindigkeit des gesamten Rechners auf einem niedrigen Niveau begrenzt.

Hinweis

Eine detailliertere Betrachtung der Strategien zur Kollisionsvermeidung im Umfeld eines Bus-Systems wird an dieser Stelle nicht vorgenommen. Bei Bedarf sei auf weiterführende Literatur zum Thema verwiesen.

2.1.3 Komponenten eines Von-Neumann-Rechners

In diesem Kapitel werden die im vorangegangenen Abschnitt identifizierten Komponenten eines Von-Neumann-Rechners detaillierter betrachtet und ihre Arbeitsweise wird erläutert.

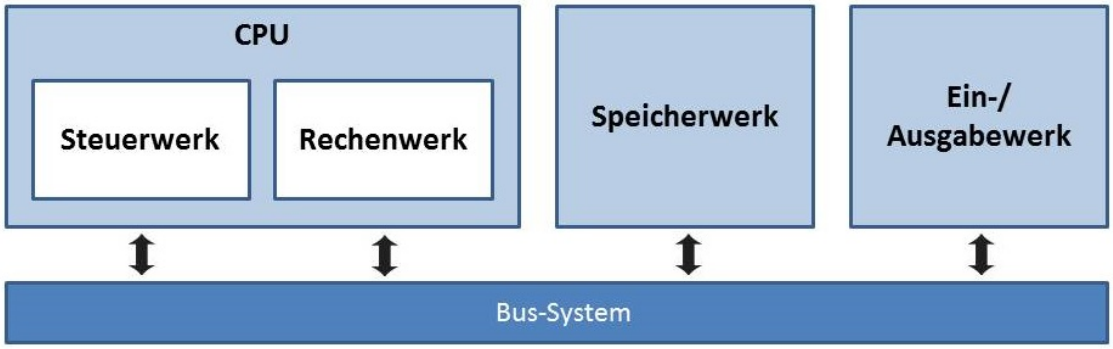


Abb. 4: Von-Neumann-Architektur
CC-BY

So geht es weiter:

Gliederung

2.1.3 Komponenten eines Von-Neumann-Rechners
2.1.3.1 Zentraleinheit / CPU
2.1.3.2 Speicherwerk
2.1.3.3 Ein- / Ausgabewerk
2.1.3.4 Gesamtbild eines Von-Neumann-Rechners
2.1.3.5 Von-Neumann-Zyklus
2.1.3.6 Animation der Zusammenarbeit

2.1.3.1 Zentraleinheit / CPU

Die **CPU** oder **Zentraleinheit** ist ein zentraler Bestandteil eines Computers. Sie besteht in erster Linie aus dem Steuerwerk, dem Rechenwerk sowie den Registern.

Während Steuerwerk und Rechenwerk auf den kommenden Seiten getrennt betrachtet werden, erfolgt hier bereits die Definition der Register:

Definition: Register

Definition

Ein **Register** ist ein digitaler Speicherbereich, der unmittelbar auf der CPU angesiedelt ist, und eine kleine Gruppe von binären Werten (Bits) speichern kann.

Definition: Registerbreite

Definition

Die Anzahl der gleichzeitig in einem Register speicherbaren Bits nennt man die **Breite des Registers** oder **Registerbreite**. Bei einer Breite von *n* Bit spricht man auch von einem **n-Bit-Register**.

Übliche Registerbreiten in Vergangenheit und Gegenwart waren bzw. sind 4, 8, 16, 32 oder 64 Bit.

Schematischer Aufbau einer CPU

Die folgende Abbildung zeigt den schematischen Aufbau einer CPU. Die oben angesprochenen Register werden hinzugefügt, sobald ihre Funktion angesprochen wird.

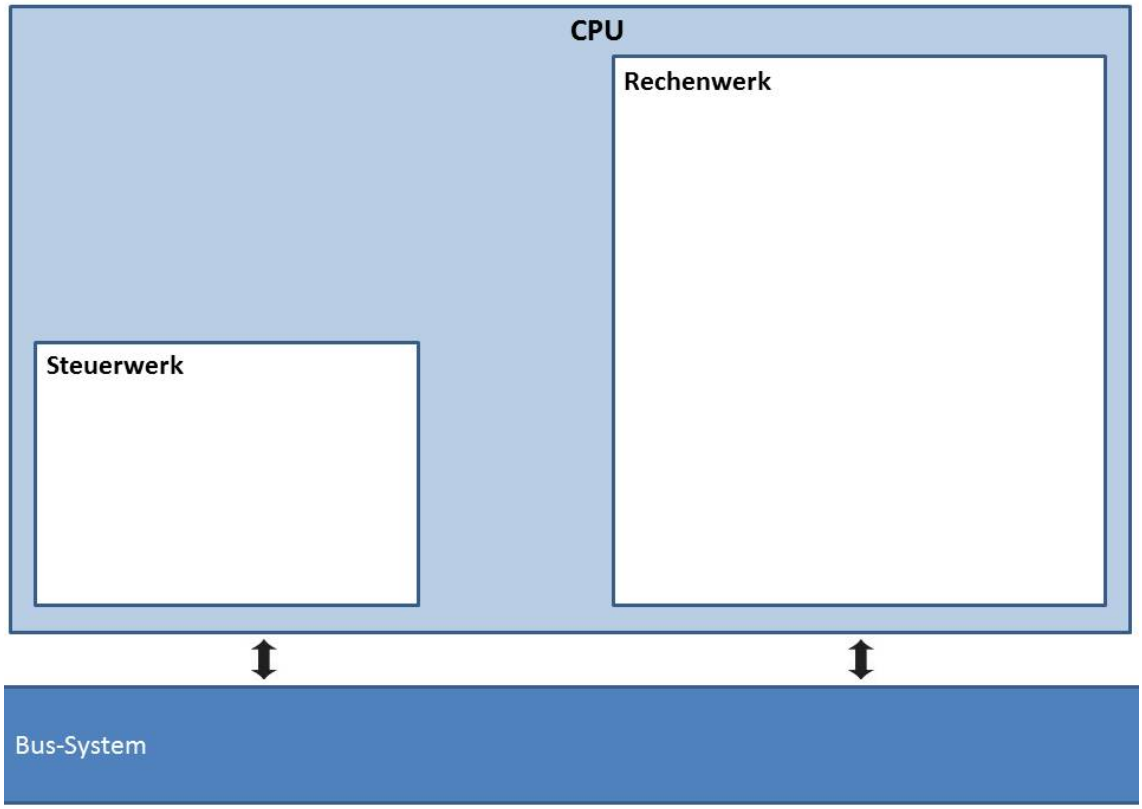


Abb. 5: CPU mit Steuerwerk und Rechenwerk
CC-BY

2.1.3.1.1 Steuerwerk / Leitwerk

Eines der Bestandteile der CPU ist das Steuerwerk:

Definition: Steuerwerk

Definition

Das **Steuerwerk** (oder **Leitwerk**) ist ein Bestandteil der CPU und für die sequentielle Abarbeitung des im Speicherwerk befindlichen Programms zuständig.

Damit ist sofort ersichtlich, dass eine Verbindung zwischen Steuerwerk und Speicherwerk existieren muss, damit das abzuarbeitende Programm, bzw. dessen einzelnen Anweisungen, vom Speicherwerk in das Steuerwerk übertragen werden können.

Definition: Arbeitsweise des Steuerwerks

Definition

Das Steuerwerk sendet die Adresse des aktuellen Befehls an das Speicherwerk und erhält von diesem als Antwort genau jenen Befehl, der an der übermittelten Adresse innerhalb des Speicherwerks steht. Sobald das Steuerwerk den Befehl erhalten hat, wird dieser ausgeführt.

Einfache Sicht auf die Arbeitsweise des Steuerwerks

Es ergibt sich damit für die Arbeitsweise des Steuerwerks ein Kreislauf, der immer wieder durchlaufen wird:

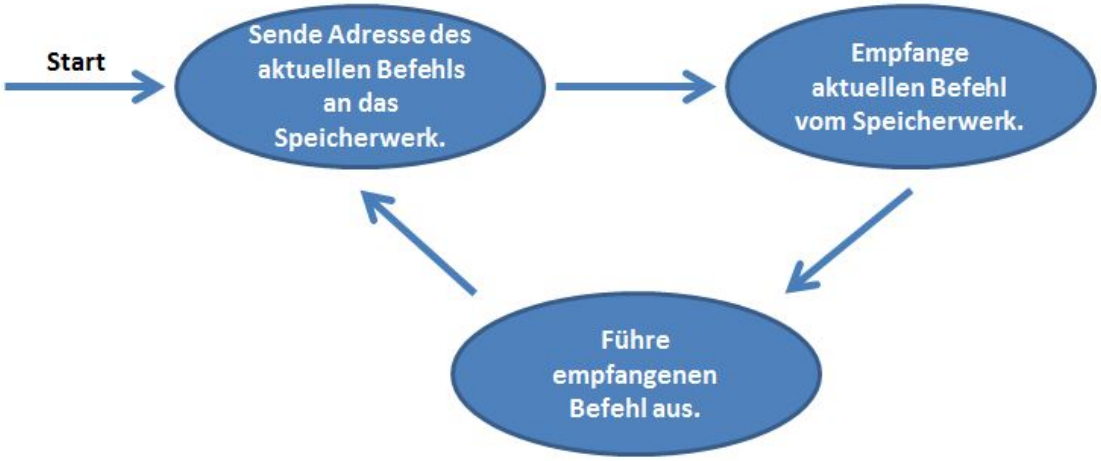


Abb. 6: Eine einfache Sicht auf die Arbeitsweise des Steuerwerks
CC-BY

Diese Darstellung ist leicht verständlich, stellt aber auch nur eine erste vereinfachte Sicht auf die Arbeitsweise dar.

Bestandteile der CPU

Aus der einfachen Sicht der Arbeitsweise des Steuerwerks lassen sich einige Bestandteile der CPU ableiten:

- Damit die Adresse des aktuellen Befehls vom Steuerwerk verwaltet werden kann, muss es einen Speicherbereich dafür geben. Verwendet wird dazu ein Register, der sogenannte **Befehlszähler**, engl. **Program Counter**, kurz: **PC**.
- Damit der vom Speicherwerk empfangene Befehl verwaltet werden kann, muss es einen Speicherbereich dafür geben. Verwendet wird dazu ein Register, das sogenannte **Befehlsregister**, engl. **Instruction Register**, kurz: **IR**.

- Die Adresse des aktuellen Befehls kann über das vorhandene Bus-System an das Speicherwerk gesendet werden. Eine Verbindung der Register PC sowie IR mit dem Bus-System muss daher gegeben sein.

Erweitertes Bild der CPU

Man erhält ein leicht erweitertes Bild der CPU:

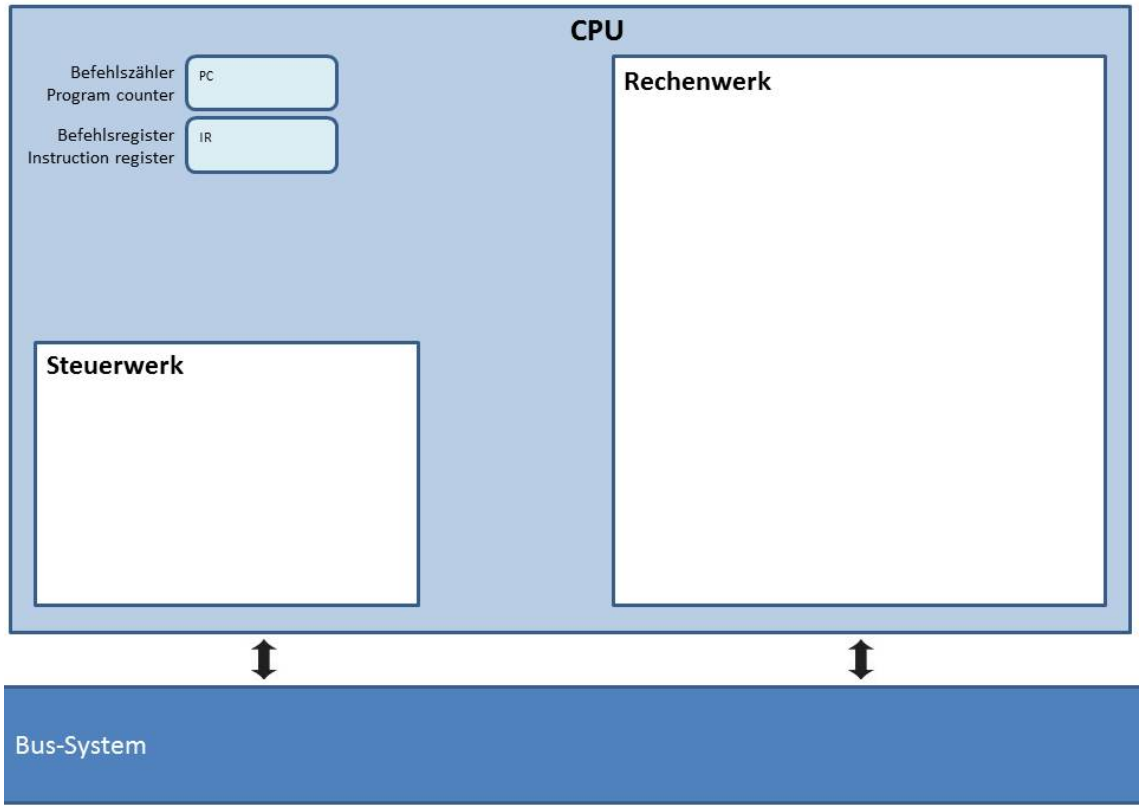


Abb. 7: Steuerwerk mit Befehlszähler und Befehlsregister
CC-BY

Zusammenspiel PC, IR und Bus

Das folgende Video geht näher auf das Zusammenspiel von Befehlszähler (PC) und Befehlsregister (IR) mit dem Bus-System ein. Die Fachbegriffe **Adressbus** und **Datenbus** werden eingeführt und erläutert.

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/2IxOCLy5jms

Med. 5: Befehlszähler und Befehlsregister im Zusammenspiel mit dem Bus-System (03:07)
http://youtu.be/2IxOCLy5jms
CC-BY

Zu den Erläuterungen im Video folgen hier noch einige Definitionen.

Definition: (Paralleler) Bus

Definition

Ein **Bus** (**Binary Unit System**) dient zur parallelen Übertragung einer Gruppe von Bits.

Hinweis: Es wird hier ausdrücklich ein "paralleler Bus" definiert. Gleichzeitig sei darauf hingewiesen, dass auch "serielle Busse" existieren, die zum jetzigen Zeitpunkt in diesem Modul aber (noch) keine Rolle spielen.

Es können mehrere Komponenten an einem Bus angeschlossen sein, so dass sich **Quelle** und **Ziel** der übertragenen Bits benennen lassen.

Definition: Busbreite

Definition

Die Anzahl der parallel übertragbaren Bits auf einem Bus nennt man die **Breite des Busses** oder **Busbreite**.

Übliche Busbreiten in Vergangenheit und Gegenwart waren bzw. sind 4, 8, 16, 32 oder 64 Bit.

Definition: Schreibender Zugriff auf einen Bus (Senden)

Definition

Ein **schreibender Zugriff** auf einen Bus liegt vor, wenn eine an diesem Bus angeschlossene Komponente eine Gruppe von Bits zur Übertragung auf den Bus gibt. Man spricht dann auch vom **Senden** von Informationen über den Bus.

Nur eine einzige am Bus angeschlossene Komponente darf zur gleichen Zeit schreibend auf den Bus zugreifen.

Definition: Lesender Zugriff auf einen Bus (Empfangen)

Definition

Ein **lesender Zugriff** auf einen Bus liegt vor, wenn eine an diesem Bus angeschlossene Komponente eine Gruppe von Bits vom Bus entgegennimmt. Man spricht dann auch vom **Empfangen** von Informationen über den Bus.

Es können beliebig viele Komponenten zur gleichen Zeit Informationen über den Bus empfangen.

Definition: Kollision auf einem Bus

Definition

Eine **Kollision** auf einem Bus liegt vor, falls zur gleichen Zeit mehrere Komponenten schreibend auf den Bus zugreifen.

Beim Betreiben eines Busses ist also sicher zu stellen, dass immer nur eine Komponente zur Zeit schreibend auf den Bus zugreift. Falls einmal mehr als nur eine Komponente zur gleichen Zeit auf den Bus schreibt, so werden aufgrund der physikalischen Gesetze die übertragenen Bits unbrauchbar, d.h. sie können von den lesenden Komponenten am Bus nicht mehr verlässlich empfangen werden. Die übertragenen Informationen gehen verloren. (Siehe hierzu auch das Video im Abschnitt Von-Neumann-Flaschenhals).

Definition: Adressbus

Definition

Ein **Adressbus** ist ein Bus, bei dem die parallel übertragene Gruppe von Bits als Adresse zu interpretieren ist.

Definition: Datenbus

Definition

Ein **Datenbus** ist ein Bus, bei dem die parallel übertragene Gruppe von Bits als Daten zu interpretieren ist.

Detailliertere Sicht auf die Arbeitsweise des Steuerwerks

Im folgenden Video wird die Arbeitsweise des Steuerwerks etwas detaillierter aufgeschlüsselt und erläutert.

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/eEZIlmMHl7c

Med. 6: Detailliertere Arbeitsweise des Steuerwerks (06:01)
http://youtu.be/eEZIlmMHl7c
CC-BY

Die im Video erarbeitete detailliertere Sicht auf die Arbeitsweise des Steuerwerks sieht damit so aus:

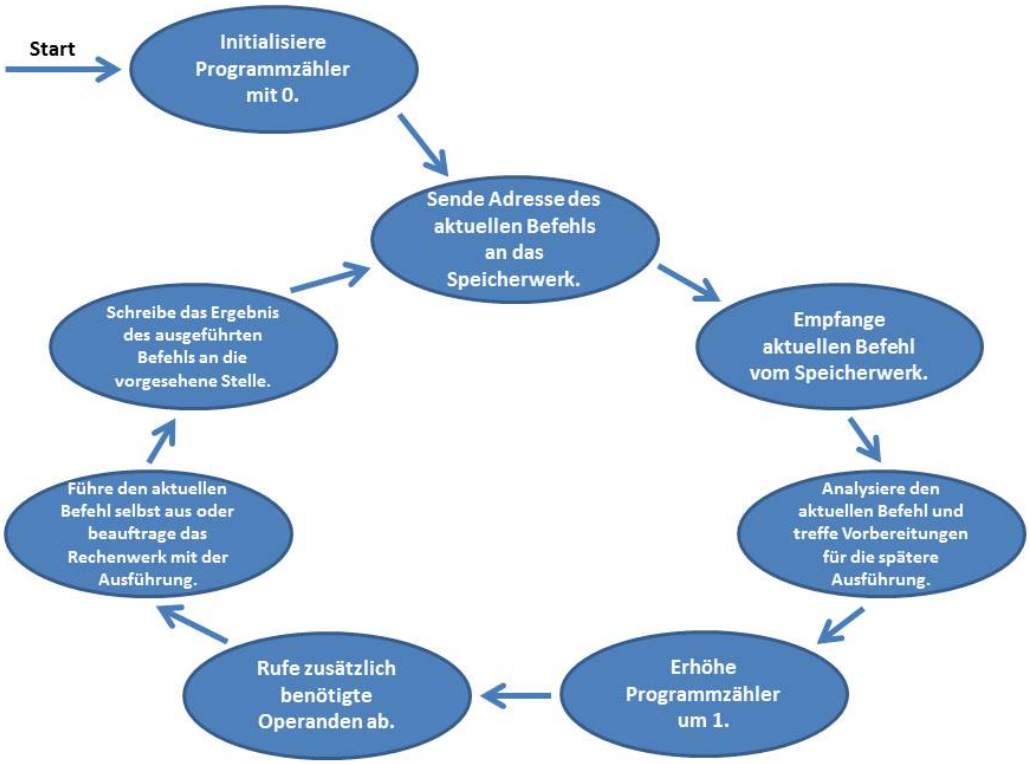


Abb. 8: Eine detailliertere Sicht auf die Arbeitsweise des Steuerwerks
CC-BY

Und der aktuelle Stand bei der Erläuterung von CPU und insbesondere dem Steuerwerk ist:

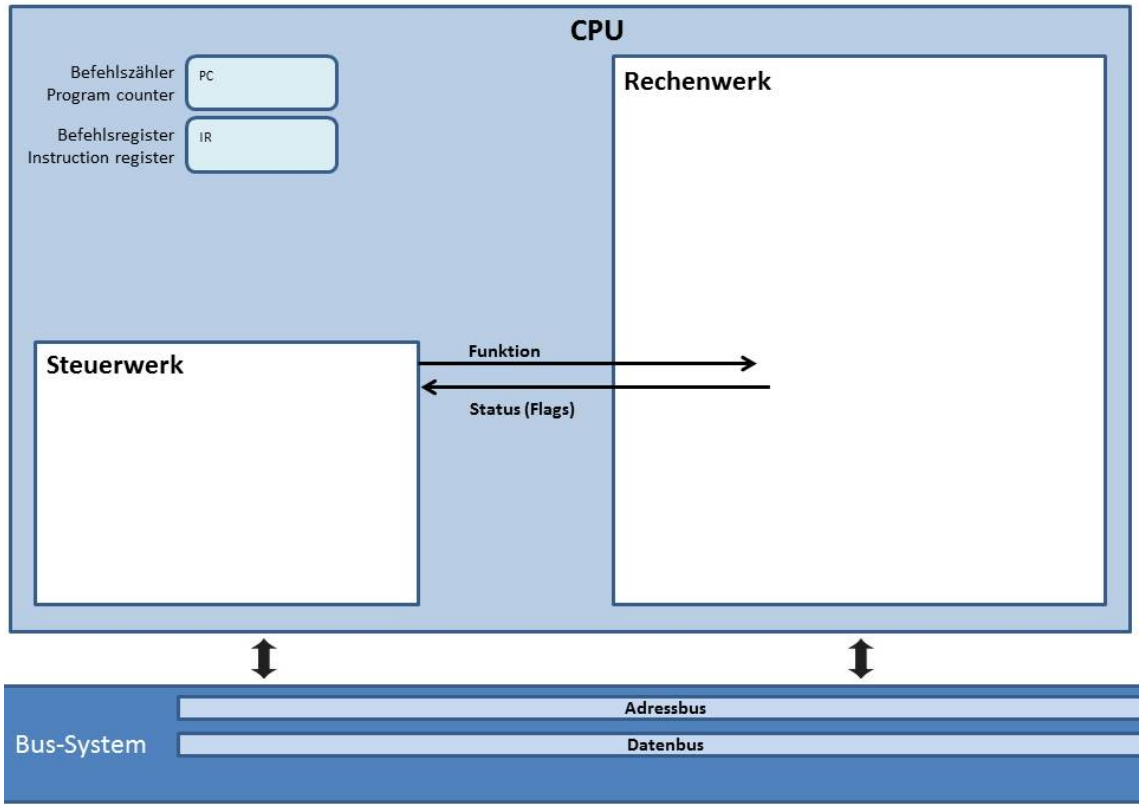


Abb. 9: Steuerwerk mit Adress- und Datenbus
CC-BY

2.1.3.1.2 Rechenwerk

Ein weiterer Bestandteil der CPU ist das Rechenwerk:

Definition: Rechenwerk

Definition

Das **Rechenwerk** ist ein Bestandteil der CPU und führt vom Steuerwerk in Auftrag gegebene Berechnungen durch. Der Hauptbestandteil des Rechenwerks ist die **ALU** (arithmetic-logical Unit, arithmetisch-logische Einheit).

Damit ist sofort ersichtlich, dass eine Verbindung zwischen Steuerwerk und Rechenwerk existieren muss. Diese Verbindung ist bidirektional, d.h. das Steuerwerk gibt über entsprechende Steuerleitungen eine zu berechnende Funktion in Auftrag und das Rechenwerk liefert den Status der Berechnung, die sogenannten Flags, an das Steuerwerk zurück.

Arbeitsweise des Rechenwerks

Das folgende Video erläutert die Arbeitsweise des Rechenwerks, insbesondere auch die Zusammenarbeit mit dem Steuerwerk.

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/Zsreix2gLkc

Med. 7: Arbeitsweise des Rechenwerks (02:26)
http://youtu.be/Zsreix2gLkc
CC-BY

Das Bild der CPU vervollständigt sich langsam:

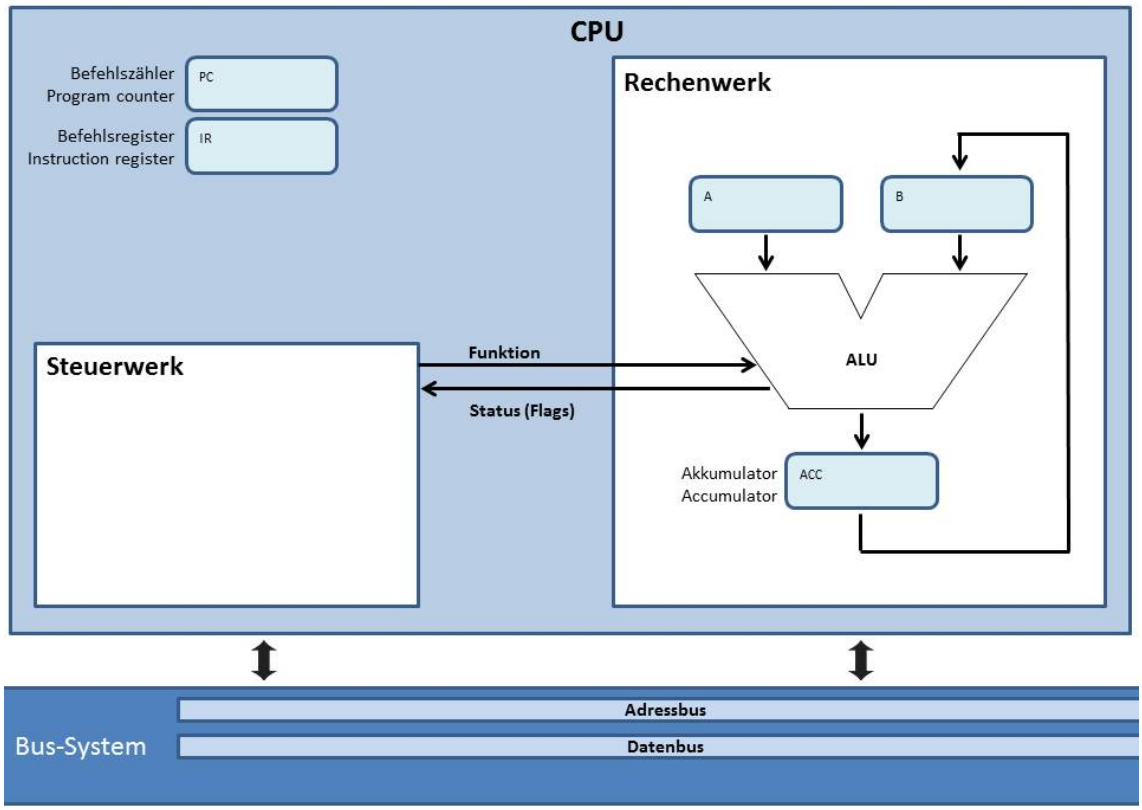


Abb. 10: Rechenwerk mit ALU

Vom Rechenwerk bereitgestellte Funktionen

Wie im Video erläutert, kann das Steuerwerk eine Reihe von Funktionen beim Rechenwerk in Auftrag geben. Diese Funktionen sind u.a.:

- Addition der Operanden (A + B)
- Subtraktion (A - B)
- Multiplikation (A * B)
- Konjunktion (logisches UND, bitweise)
- Disjunktion (logisches ODER, bitweise)

- Vergleich von A und B (bitweise)
- etc.

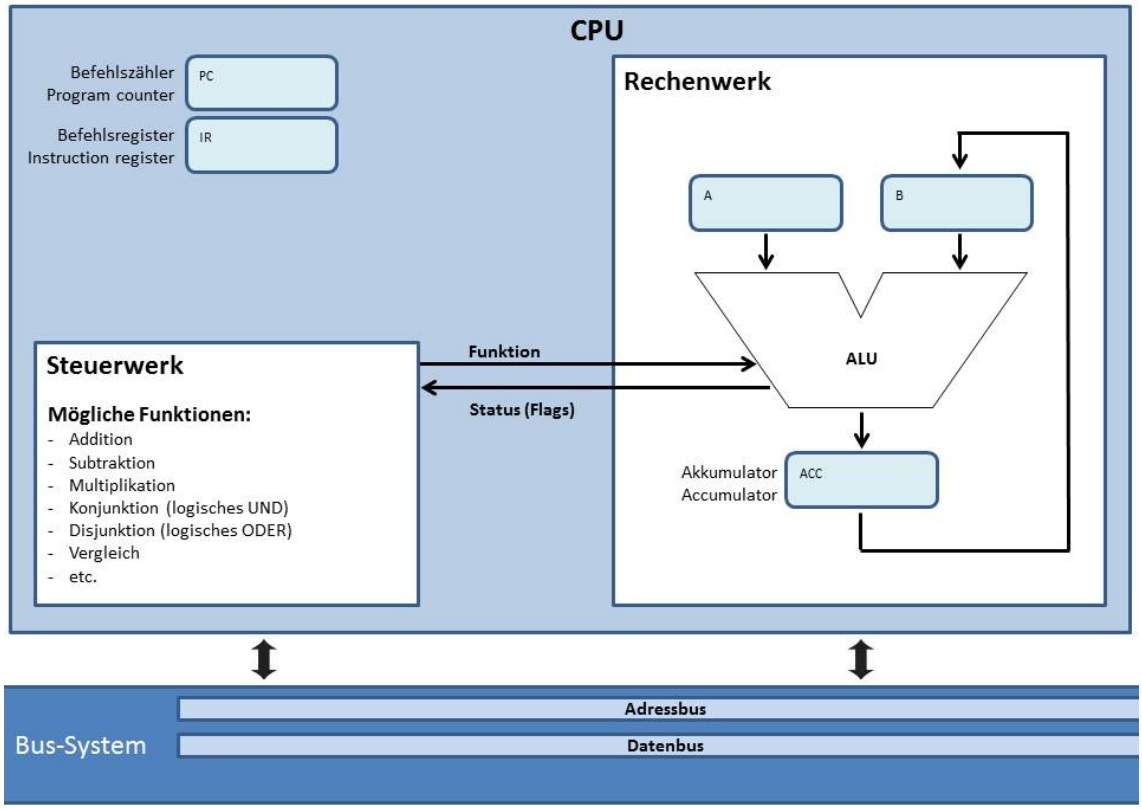


Abb. 11: Funktionen des Rechenwerks, die vom Steuerwerk in Auftrag gegeben werden können

Flags

Ebenfalls im Video erläutert wurden die **Statusbits** (**Flags**), die vom Rechenwerk als Ergebnis jeder Operation an das Steuerwerk übermittelt werden. Diese sind u.a.

- War letztes Ergebnis gleich Null? (ACC = 0?)
- Waren beide Operanden gleich? (A = B?)
- War A kleiner als B? (A < B?)
- Gab es einen Überlauf?
- etc.

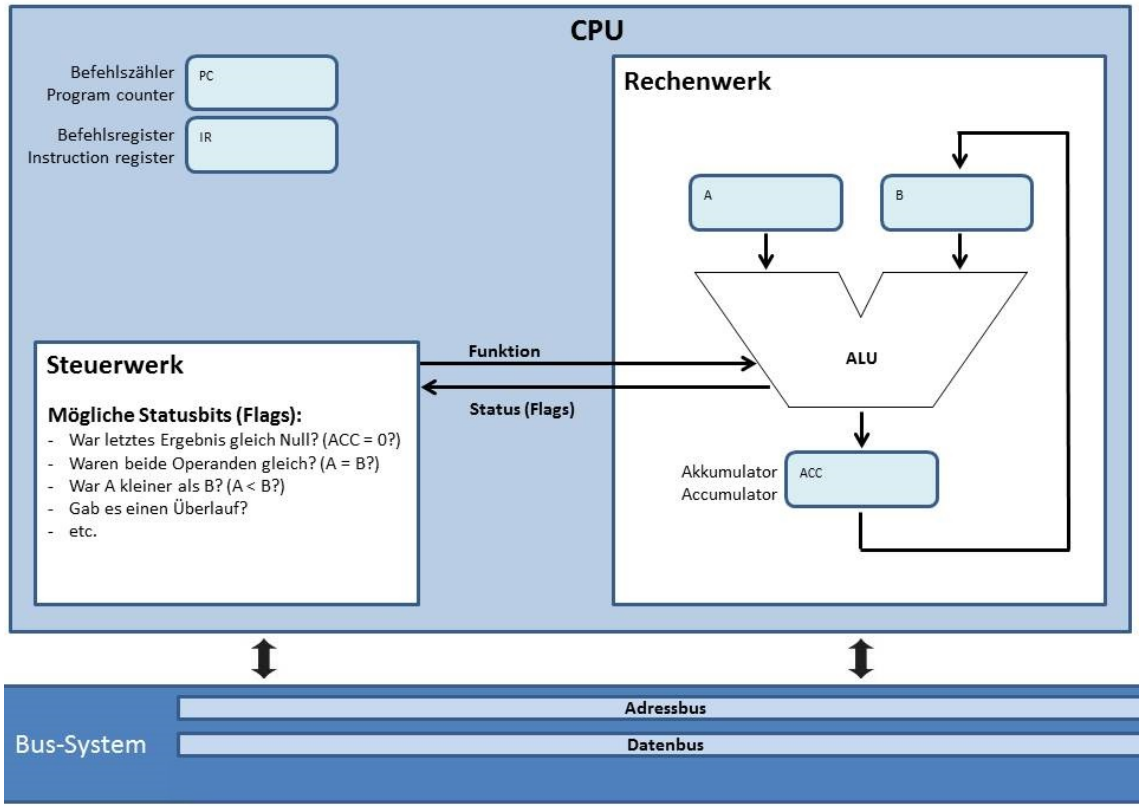


Abb. 12: Statusbits des Rechenwerks, die vom Steuerwerk ausgewertet werden können

Aufgabe 1

Aufgabe

Aufg. 1: Größer als

In den Beispielen für mögliche Statusbits (Flags) sieht man u.a.
"Waren beide Operanden gleich?" sowie
"War A kleiner als B?".

Die dritte Variante *"War A größer als B?"* muss jedoch nicht vorhanden sein. Warum kann das Steuerwerk auch ohne dieses Statusbit auskommen?

2.1.3.2 Speicherwerk

Das Speicherwerk ist ein Bestandteil der Von-Neumann-Architektur:

Definition: Speicherwerk

Definition

Das Speicherwerk ist in eine endliche Anzahl gleichgroßer (aber verhältnismäßig kleiner) Speicherzellen unterteilt. Jede Zelle verfügt dabei über eine eindeutige Adresse. Alle Adressen sind fortlaufend, beginnend bei Null (0, 1, 2, 3, ...).

Das Speicherwerk gehört zu den sogenannten "flüchtigen Speichern". Es kann nur Informationen speichern, solange es mit Strom versorgt wird, also bei eingeschaltetem Rechner. Sobald der Rechner ausgeschaltet wird, gehen alle im Speicherwerk abgelegten Informationen unwiederbringlich verloren.

Hinweis

Auf deinem eigenen PC oder Laptop entspricht der RAM (Random Access Memory) dem Speicherwerk.

Definition: Arbeitsweise des Speicherwerks

Definition

Das Speicherwerk kann den Wert einer adressierten Speicherstelle auslesen und zur Verfügung stellen oder andersherum einen zur Verfügung gestellten Wert in einer adressierten Speicherstelle ablegen.

Lesen oder Schreiben?

Diese Definition der Arbeitsweise ist an einer Stelle noch etwas ungenau. Woher weiß das Speicherwerk, ob es die adressierte Zelle auslesen oder überschreiben soll?

Das folgende Video geht näher darauf ein.

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/pefqk2C7wgw

Med. 8: Arbeitsweise des Speicherwerks (02:06)
http://youtu.be/pefqk2C7wgw
CC-BY

Aufgabe 1

Aufgabe

Aufg. 2: Speicherzelle 3

Zu Beginn des Videos empfängt das Speicherwerk über den Adressbus die Adresse "00000011". Warum wird damit Speicherzelle 3 angesprochen?

(Entschuldigung! Diese Frage ist für Studierende im ersten Semester bestimmt. Falls Du bereits in einem höheren Semester bist, ist sie natürlich viel zu leicht.) ;-)

Aus dem Video folgt noch eine Definition:

Definition: Steuerbus

Definition

Ein Steuerbus ist ein Bus, bei dem die parallel übertragene Gruppe von Bits als Steuerinformation zu interpretieren ist.

Die Breite des Steuerbusses kann sich deutlich von der Breite des Adress- oder Datenbusses unterscheiden. In der Regel ist die Breite des Steuerbusses geringer.

Das im Video erläuterte Speicherwerk mit dem erweiterten Bus-System zeigt die folgende Abbildung:

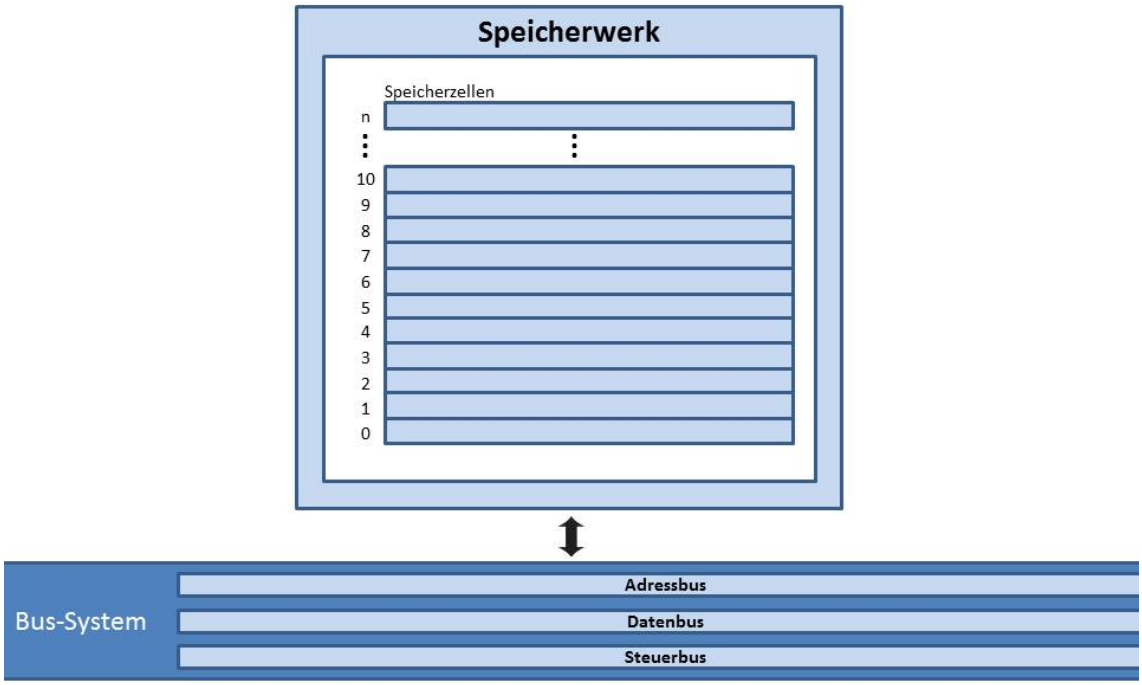


Abb. 13: Speicherwerk mit Bus-System
CC-BY

Aufgabe 2

Aufgabe

Aufg. 3: Von dezimal zu binär

Im Bild oben siehst du das Speicherwerk mit seinen von 0 bis n nummerierten Speicherzellen. Diese Nummerierung ist angegeben in **dezimalen** Zahlen, das Speicherwerk verarbeitet aber in der Realität nur **binär** angegebene Adressen. Ändere

deshalb die Nummerierung der einzelnen Speicherzellen in eine binäre Schreibweise ab!

Interessant ist, welche binäre Adresse du der Speicherzelle **n** gibst. Entscheide dich für eine konkrete binäre Adresse und erläutere deine Entscheidung in deiner Lerngruppe! (Es gibt für Speicherzelle n nicht *"die eine richtige"* binäre Adresse. Es kommt aber darauf an, eine sinnvolle und nachvollziehbare Begründung für die getroffene Entscheidung zu geben.)

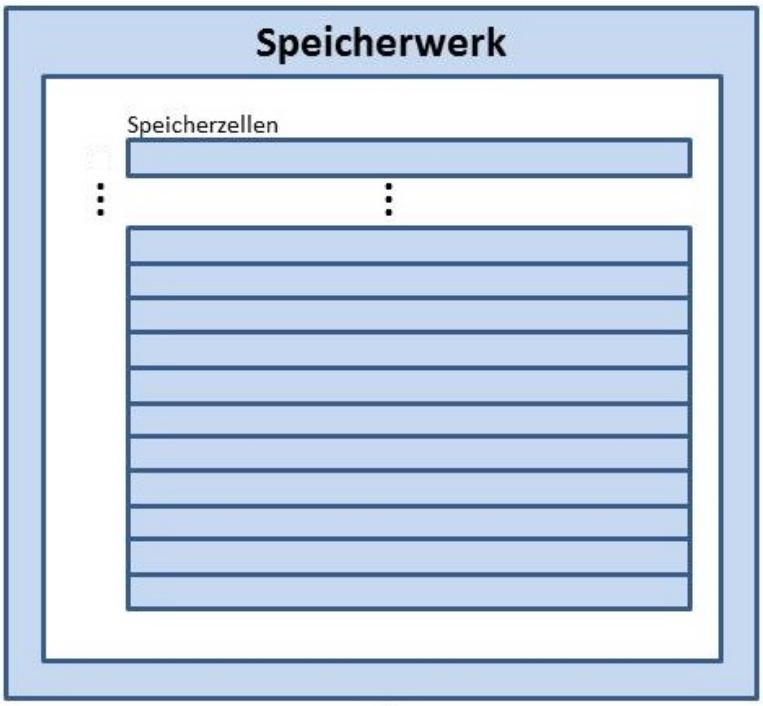


Abb. 14: Abbildung zu Aufgabe 2
CC-BY

Aufgabe 3

Aufgabe

Aufg. 4: Die 4 GiB-Grenze

In der Vergangenheit war immer wieder folgender Satz zu hören: *"Ein 32-Bit-Betriebssystem kann maximal 4 GiB Arbeitsspeicher (RAM) verwalten"*. Gehen wir für unser Speicherwerk davon aus, dass der Adressbus eine Breite von 32 Bit besitzt. Damit können dann auch maximal 4 GiB im Speicherwerk angesprochen werden.

- Wie viele Adressen können mit 32 Bit unterschieden werden? (Das Ergebnis gibt dann gleichzeitig die Anzahl der Speicherzellen im Speicherwerk an.)

- Wie viele Bit besitzt eine einzige Speicherzelle unter Berücksichtigung der 4 GiB Gesamtspeichermenge und dem Hinweis zur Bedeutung von '"GiB"'?
- Gehen wir weiter davon aus, dass der Datenbus ebenfalls 32 Bit breit ist.
  Beim Eintreffen einer Adresse über den Adressbus am Speicherwerk und dem Befehl "Lesen" auf dem Steuerbus wird das Speicherwerk über den Datenbus genau 32 Bit zurücksenden. Der Speicher kann dann in einer Skizze mit Zellen gezeichnet werden, die jeweils eine Breite von 32 Bit besitzen.
  Wie lautet die korrekte Adressierung dieser 32 Bit breiten Zellen? (Gib die Adressierung sowohl in dezimaler, als auch in binärer Schreibweise an!)

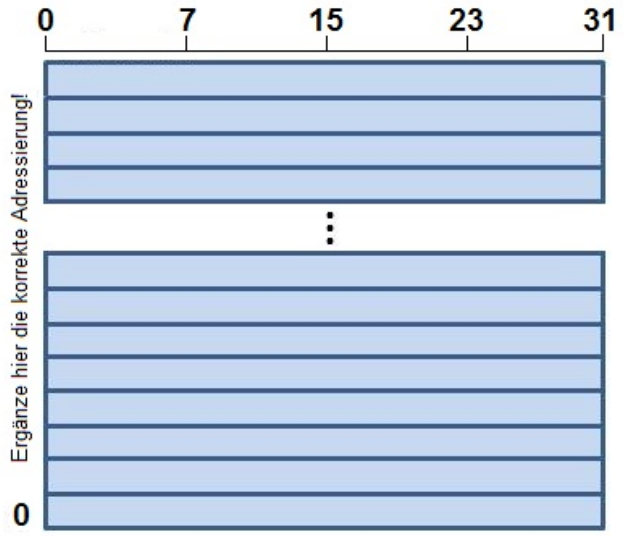


Abb. 15: Abbildung zu Aufgabe 4
CC-BY

- Wenn eine Speicherzelle in der Abbildung eine Breite von 32 Bit besitzt, warum wird oben im Bild nur bis 31 gezählt?

Memory Address Register und Memory Data Register

Um die Zusammenarbeit zwischen dem Speicherwerk und der CPU bzw. dem Steuerwerk zu vereinfachen, werden auf der CPU oftmals spezielle Register verwendet, welche ausschließlich für die Kommunikation mit dem Speicherwerk zuständig sind. Dies sind das **Speicheradressregister** (**Memory Address Register**, kurz **MAR**), sowie das **Speicherdatenregister** (**Memory Data Register**, kurz **MDR**).

Im MAR legt das Steuerwerk jeweils die Adresse ab, welche im Speicherwerk angesprochen werden soll. Bei einem Lesezugriff auf die Speicherzelle wird der vom Speicherwerk über den Datenbus bereitgestellte Wert im MDR abgelegt und kann von hier

aus weiter verarbeitet werden. Bei einem Schreibzugriff muss sich im MDR der zu schreibende Wert befinden, so dass er über den Datenbus an das Speicherwerk übermittelt werden kann.

Hinweis

Die Existenz der beiden Register (MAR und MDR) macht deutlich mehr Sinn, wenn du dir vorstellst, dass es (neben dem "normalen" Bus der Von-Neumann-Architektur) noch einen speziellen weiteren Bus zwischen eben diesen beiden Registern und dem Speicherwerk gibt. Auf das Einzeichnen dieses zusätzlichen Busses wird in diesem Modul der Einfachheit halber jedoch verzichtet.

Aufgabe 4

Aufgabe

Aufg. 5: Der Weg der Daten

Betrachte die Situation, in der die momentan im PC (Befehlszähler) gespeicherte Adresse an das Speicherwerk übermittelt wird, um so den nächsten Befehl aus der adressierten Speicherzelle in das IR (Befehlsregister) zu kopieren.

- Welches Signal wird am Steuerbus angelegt?
- Welchen Weg nehmen die jeweiligen Daten, wenn zur direkten Kommunikation zwischen CPU und Speicherwerk nur MAR und MDR eingesetzt werden dürfen? Erläutere deinen Weg in deiner Lerngruppe!

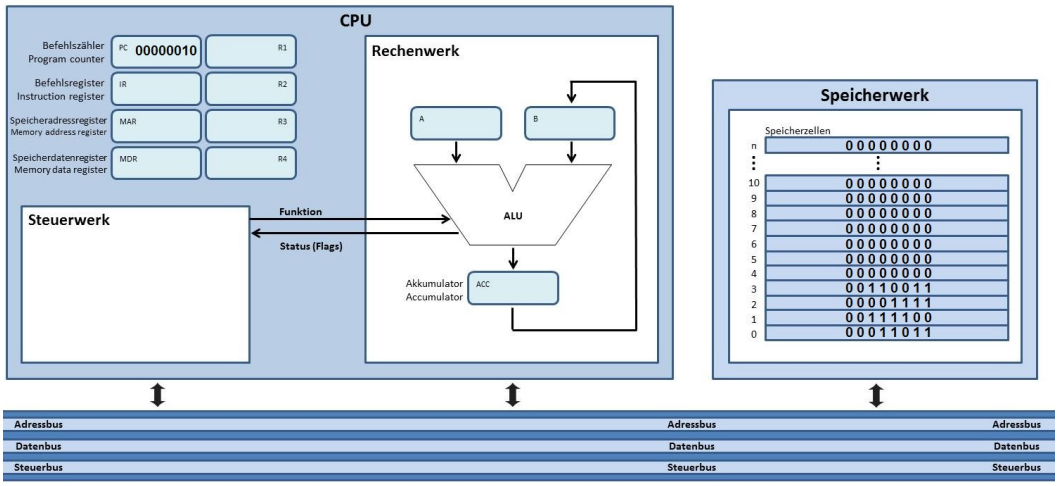


Abb. 16: Abbildung zu Aufgabe 4
CC-BY

2.1.3.3 Ein- / Ausgabewerk

Das Ein-/Ausgabewerk ist in der Von-Neumann-Architektur nicht näher spezifiziert. Man kann es sich vorstellen als eine Art von "Stellvertreter" für alle weiteren Komponenten eines modernen Rechners. Z.B. also für Komponenten wie:

- Monitor mit Grafikkarte
- Festplatte mit Controller
- Netzwerkkarte mit Controller
- Maus mit Controller
- Tastatur mit Controller
- etc.

Ein Controller je Komponente

Wie man sieht, ist für jede Komponente ein eigener Controller vorgesehen, der das Bus-System mit dem eigentlichen Gerät verbindet.

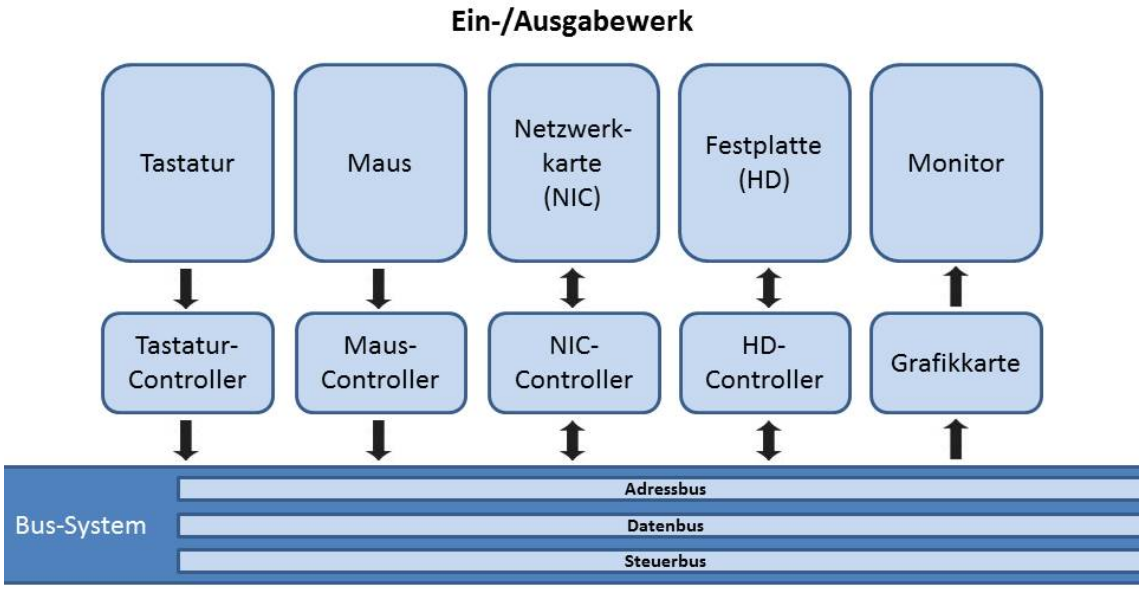


Abb. 17: Ein-/Ausgabewerk
E/A-Komponenten mit ihren Controllern
CC-BY

Aufgabe 1

Aufgabe

Aufg. 6: Pfeilrichtungen

Die Pfeile in der vorangegangenen Abbildung zeigen zum Teil in beide Richtungen, manchmal aber auch nur in eine Richtung. Warum ist das so?

2.1.3.4 Gesamtbild eines Von-Neumann-Rechners

Die einzelnen Komponenten aus der Von-Neumann-Architektur wurden auf den vorangegangenen Seiten vorgestellt. Das Steuerwerk nimmt dabei eine zentrale Rolle ein, es koordiniert die Ausführung aller notwendigen Tätigkeiten.

Damit ergibt sich das typische Gesamtbild eines Von-Neumann-Rechners:

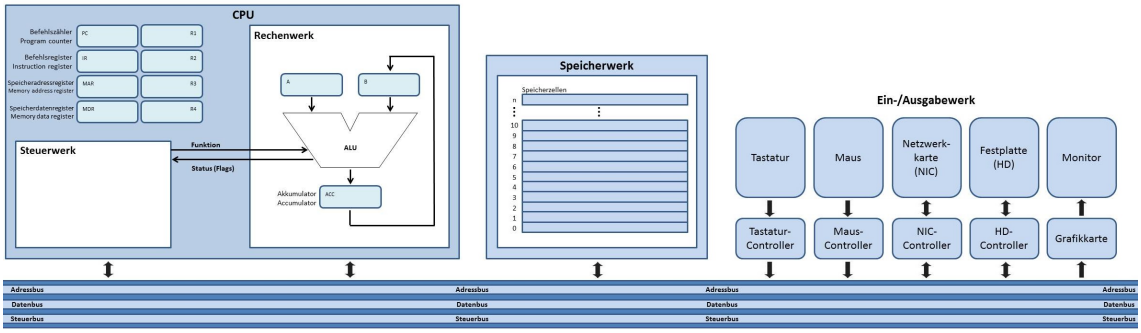


Abb. 18: Gesamtbild eines Von-Neumann-Rechners
CC-BY

Wie wäre es jetzt mit einem kleinen Wettbewerb?

Aufgabe 1

Aufgabe

Aufg. 7: Wettbewerb in deiner Lerngruppe

Triff dich mit deiner Lerngruppe. Egal, ob persönlich oder online. Einer gibt das Startsignal und dann habt ihr genau 5 Minuten Zeit, um aus dem Gedächtnis eine Skizze des oben gezeigten Gesamtbilds zu zeichnen.

- Wer schafft innerhalb der Zeit die meisten Details?
- Und wer hat geschummelt, weil er/sie oben abgeguckt hat? (Pfui!)

2.1.3.5 Von-Neumann-Zyklus

Die detailliertere Sicht auf die Arbeitsweise des Steuerwerks ist ja bereits bekannt:

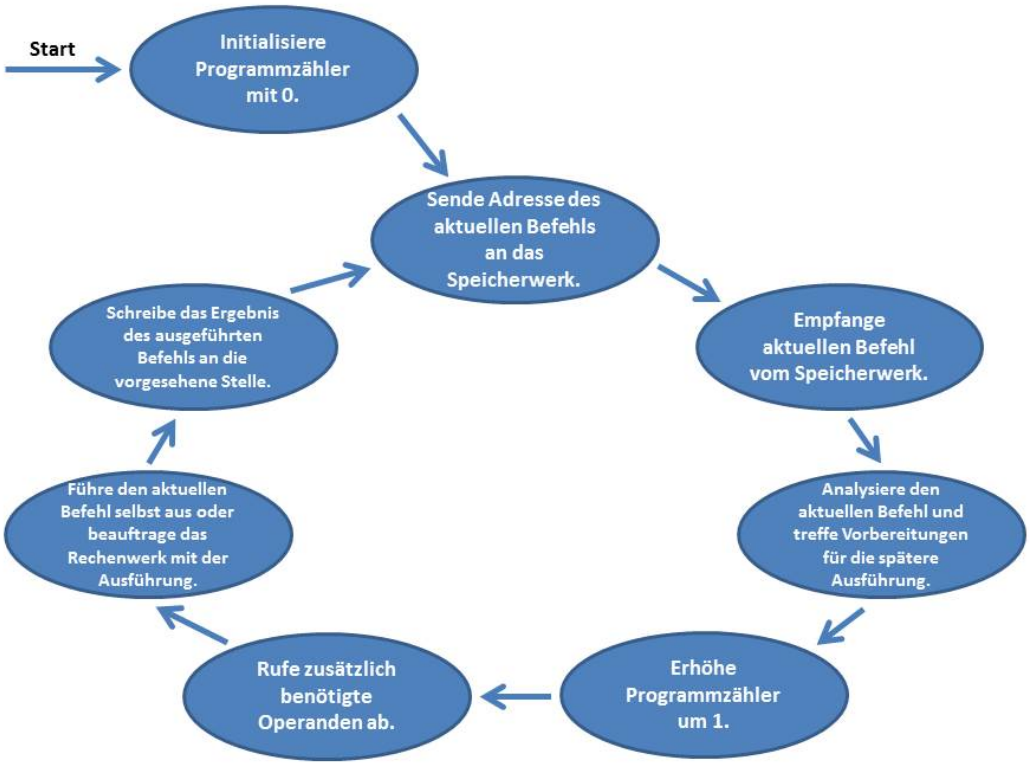


Abb. 19: Arbeitsweise des Steuerwerks II
Eine etwas detailliertere Sicht.

In der detaillierteren Sicht lässt sich der Von-Neumann-Zyklus erkennen, der hier zunächst definiert wird:

Definition: Von-Neumann-Zyklus

Definition

Der Von-Neumann-Zyklus besteht aus den fünf Phasen:

1. Fetch - Hole den nächsten Befehl.
2. Decode - Dekodiere den Befehl.
3. Fetch Operands - Hole benötigte Operanden.
4. Execute - Führe den Befehl aus.
5. Write back - Schreibe das Ergebnis zurück.

Im folgenden Video wird der Zusammenhang erläutert:

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/LBagfbOy_gs

Med. 9: Von-Neumann-Zyklus und die Arbeitsweise des Steuerwerks (02:22)
http://youtu.be/LBagfbOy_gs

CC-BY

Zusammenhang

Als Ergebnis der Herleitung im Video wird hier der Zusammenhang zwischen der detaillierteren Sicht der Arbeitsweise des Steuerwerks sowie dem Von-Neumann-Zyklus wiedergegeben:

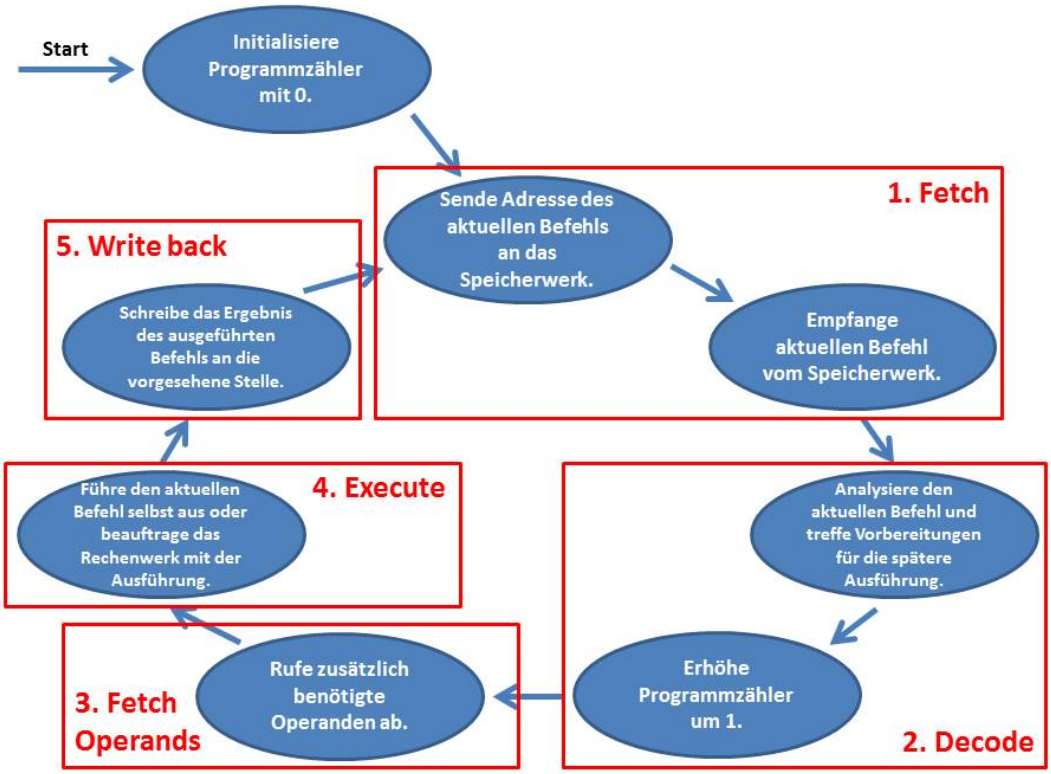


Abb. 20: Von-Neumann-Zyklus in der Arbeitsweise des Steuerwerks
CC-BY

Abbildung des Von-Neumann-Zyklusses

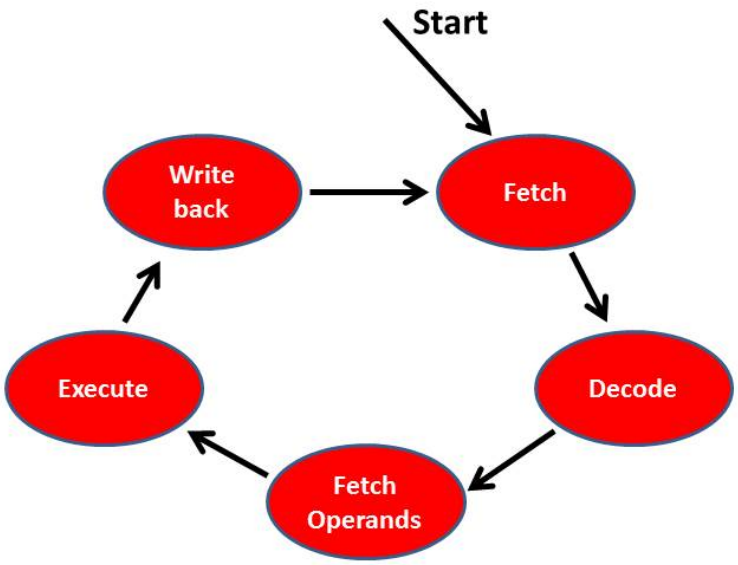


Abb. 21: Von-Neumann-Zyklus
CC-BY

2.1.3.6 Animation der Zusammenarbeit

Prof. Dr. Hans Heinrich Heitmann von der Hochschule für Angewandte Wissenschaften Hamburg hat eine kleine Animation online gestellt, welche die Arbeitsweise einer CPU im Zusammenspiel mit Adress- sowie Datenbus und dem Hauptspeicher zeigt. Insbesondere ist hier der Von-Neumann-Zyklus erkennbar.

Hinweis

Hinweis!
http://tiserver02.cpt.haw-hamburg.de/htm/gt/cpuanimation/cpu.html
Leider ist die Webseite nicht mehr erreichbar, auf die Animation kann deshalb nicht mehr direkt zugegriffen werden. Das folgende Video (https://youtu.be/77oRSDhnBg4) zeigt jedoch eine Aufzeichnung dieser Animation. (Außer den Klick-Geräuschen der Mausklicks hat dieses Video keinen weiteren Ton.)

Video

An dieser Stelle befindet sich online ein YouTube-Video.
https://youtu.be/77oRSDhnBg4

Med. 10: Animation der Zusammenarbeit (01:02)

https://youtu.be/77oRSDhnBg4 - Screencast einer Animation von Prof. Heitmann, HAW Hamburg
CC-BY

Aufgabe 1

Aufg. 8: Animation

Betrachte die Animation und spiele den Ablauf des Beispielprogramms durch. Erläutere den Ablauf den anderen Mitgliedern deiner Lerngruppe!

Aufgabe 2

Aufg. 9: lda und sta

Innerhalb der Animation werden u.a. die folgenden beiden Befehle ausgeführt:

- lda 202
- sta 1002

Wofür stehen die Abkürzungen "lda" sowie "sta"?

Aufgabe 3

Aufg. 10: Hellseher

Betrachte erneut die beiden Befehle:

- lda 202
- sta 1002

Eigentlich sollte es bei der Ausführung dieser beiden Befehle ein Problem geben, oder kann der Speicher hellsehen? Was ist das Problem? Was fehlt in der Animation? Diskutiere das Problem in deiner Lerngruppe!

Aufgabe 4

Aufg. 11: See How The CPU Works In One Lesson

Eine sehr schöne (englischsprachige) Erläuterung zur Arbeitsweise einer CPU bietet auch das folgende Video (20:42) auf YouTube:

See How The CPU Works In One Lesson
http://www.youtube.com/watch?v=cNN_tTXABUA

Schau es dir an!

Aufgabe 5

Aufgabe

Aufg. 12: Visual 6502

Der MOS Technology 6502 ist ein 8-Bit-Mikroprozessor, der in den 1970er und 1980er-Jahren in vielen Systemen zum Einsatz kam (u.a. im Atari 800 XL, im Apple I sowie im Apple II). Wikipedia liefert weitere Hintergrundinformationen zu dieser CPU:
https://de.wikipedia.org/wiki/MOS_Technology_6502

Auf der Webseite http://visual6502.org findet sich eine "Visual Transistor-level Simulation of the 6502 CPU".

Schau dir die Simulation an!
http://visual6502.org/JSSim/index.html

2.1.4 Eigenschaften eines Von-Neumann-Rechners

Nachdem der Aufbau und die Arbeitsweise eines Von-Neumann-Rechners bekannt sind, können hier einige seiner Eigenschaften zusammengefasst werden.

Verschiedene Programme

Zusammen-fassung

Ein Von-Neumann-Rechner kann verschiedene Programme in seinem Speicherwerk ablegen.

Allerdings geht die bisherige Betrachtungsweise immer davon aus, dass nur genau ein Programm zur Zeit im Speicherwerk liegt, welches direkt nach dem Start des Rechners zunächst geladen werden muss. Dieses Programm kann ganz einfach dadurch ausgetauscht werden, dass beim nächsten Start des Rechners ein Anderes geladen wird.

Funktionalität abhängig vom Programm

Zusammenfassung

Die Funktionalität eines Von-Neumann-Rechners ist von seinem Programm abhängig. Insbesondere bestimmt das Programm die Funktionalität.

Damit bildet ein Von-Neumann-Rechner eine Universalmaschine, denn sein Programm ist nicht "fest verdrahtet", sondern kann je nach Problemstellung variieren.

Auch das Ablegen mehrerer Programme zur gleichen Zeit im Speicherwerk ist denkbar, es wird aber erst später im Kapitel Mehrere Programme gleichzeitig im Speicher näher darauf eingegangen, da zuvor noch einige Vorbedingungen erläutert werden müssen.

Daten und Programm zusammen im Speicher

Zusammenfassung

Daten werden zusammen mit dem Programm im Speicherwerk abgelegt.

Es erfolgt keine Trennung zwischen Programm und Daten. Beides muss von außen in das Speicherwerk gelangen und wird gemeinsam in diesem abgelegt. Dies ist beispielsweise ein Unterscheidungskriterium zwischen der Von-Neumann-Architektur und der (hier nicht näher betrachteten) Harvard-Architektur .

2.1.5 Digitale Schaltungen

Dieses Kapitel betrachtet einzelne Bestandteile aus dem Gesamtbild eines Von-Neumann-Rechners. Das Augenmerk liegt dabei auf der technischen Realisierung dieser Bestandteile mit Hilfe von digitalen Schaltungen.

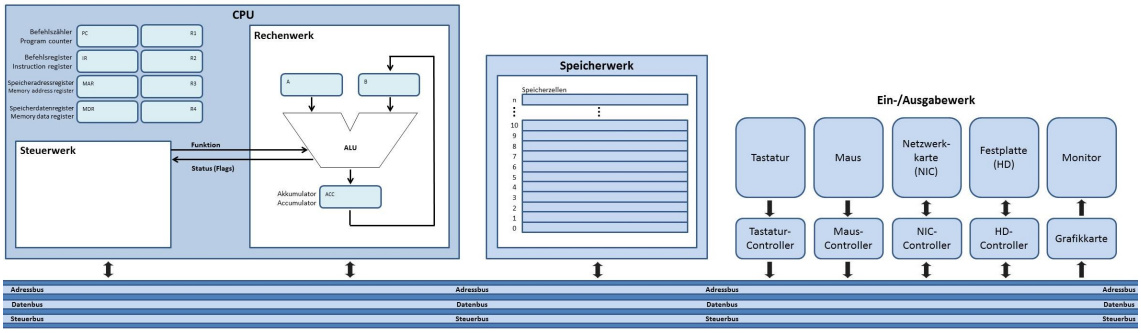


Abb. 22: Gesamtbild eines Von-Neumann-Rechners

So geht es weiter:

Gliederung

2.1.5 Digitale Schaltungen
2.1.5.1 Aufbau und Arbeitsweise eines Registers

2.1.5.2 Aufbau und Arbeitsweise des Speicherwerks
2.1.5.3 Aufbau und Arbeitsweise der ALU
2.1.5.4 Aufbau und Arbeitsweise eines Busses

2.1.5.1 Aufbau und Arbeitsweise eines Registers

Register sind gemäß ihrer Definition kleine Speichereinheiten, die sich direkt auf der CPU befinden. Den internen Aufbau und die Arbeitsweise eines 8-Bit-Registers erläutert das folgende Video.

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/CUxnaaZozVM

Med. 11: Aufbau und Arbeitsweise eines Registers (04:40)
http://youtu.be/CUxnaaZozVM
CC-BY

Ein Register besteht aus einzelnen Speicherzellen. Jede Speicherzelle kann dabei den Wert von genau einem Bit aufnehmen. Ein Register der Breite acht besitzt demnach acht Speicherzellen.

Aufbau einer Speicherzelle

Jede Speicherzelle verfügt über eine separate Input- sowie Output-Leitung. Die Select- und Write-Leitung steht einmal für alle Speicherzellen zur Verfügung.

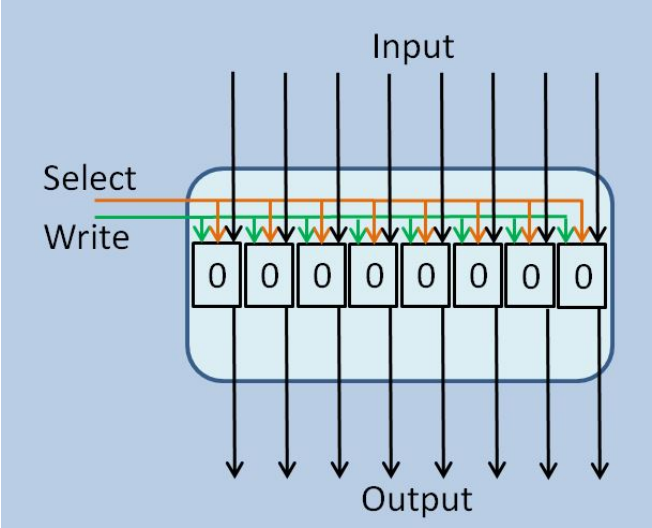


Abb. 23: Register mit 8 Speicherzellen
CC-BY

Der interne Aufbau einer einzelnen Speicherzelle sieht wie folgt aus:

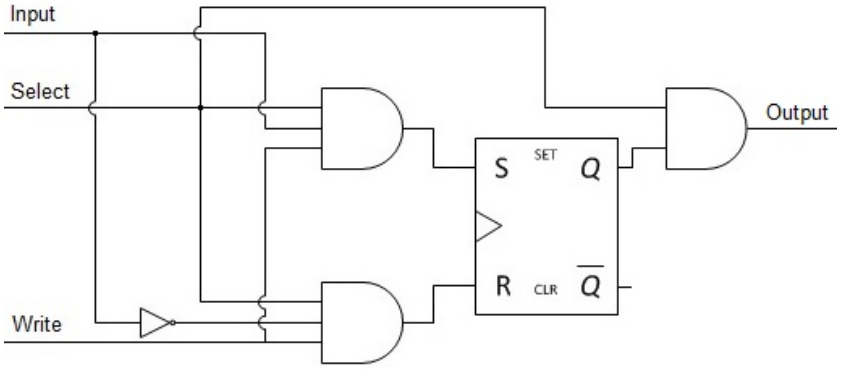


Abb. 24: Aufbau einer einzelnen Speicherzelle
CC-BY

Die Speicherzelle besteht aus einem sogenannten RS-Flip-Flop, drei UND-Gattern, einem NICHT-Gatter sowie der notwendigen Verdrahtung.

Arbeitsweise einer Speicherzelle

Das folgende Video erklärt die Arbeitsweise der Speicherzelle:

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/W1fCLMbYrnw

Med. 12: Arbeitsweise einer Speicherzelle (04:49)
http://youtu.be/W1fCLMbYrnw
CC-BY

Wahrheitstafeln

Hier folgen die im Video gezeigten Wahrheitstafeln der UND-Gatter:

UND-Gatter (2 Eingänge)

Eingänge		Ausgang
A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

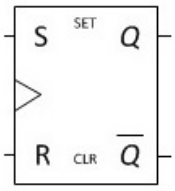
UND-Gatter (3 Eingänge)

Eingänge			Ausgang
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Abb. 25: Wahrheitstafeln von UND-Gattern mit zwei bzw. drei Eingängen
CC-BY

Neben dem hier gezeigten UND-Gatter gibt es noch eine Reihe weiterer Gatter. Diese werden treffenderweise im Kapitel Gatter beschrieben.

Auch das Verhalten des RS-Flip-Flop lässt sich mit einer Wahrheitstafel beschreiben:



S	R	Q	$\overline{Q}$	
0	0	Q*	$\overline{Q}$*	(store)
0	1	0	1	
1	0	1	0	
1	1	0	0	(forbidden)

Abb. 26: RS-Flip-Flop und die zugehörige Wahrheitstafel
CC-BY

Aufgabe 1

Aufgabe

Aufg. 13: Flip-Flop I

Im RS-Flip-Flop ist der Wert ***Null*** gespeichert. Die Eingangsleitungen besitzen folgende Werte:

- Input = 1
- Select = 1
- Write = 0

Welcher Wert wird unter diesen Voraussetzungen für den Ausgang (Output) der Speicherzelle ermittelt?

Aufgabe 2

Aufgabe

Aufg. 14: Flip-Flop II

Im RS-Flip-Flop ist der Wert ***Eins*** gespeichert. Die Eingangsleitungen besitzen folgende Werte:

- Input = 1
- Select = 1
- Write = 0

Welcher Wert wird unter diesen Voraussetzungen für den Ausgang (Output) der Speicherzelle ermittelt?

(Die Werte für Input, Select und Write sind identisch mit denen aus Aufgabe 1, jedoch unterscheidet sich der im Flip-Flop gespeicherte Wert!)

Aufgabe 3

Aufgabe

Aufg. 15: Flip-Flop III

Welchen Wert gibt das RS-Flip-Flop am Ausgang Q aus, wenn beide Eingänge gleich Null sind (S=0, R=0)?

Aufgabe 4

Aufgabe

Aufg. 16: Simulationsframework Hades

Norman Hendrich von der Universität Hamburg hat mit Hades ein Simulationsframework bereitgestellt, welches u.a. den internen Aufbau und die Arbeitsweise eines RS-Flip-Flops per Applet im Browser veranschaulicht. Probiere es aus unter:

http://tams-www.informatik.uni-hamburg.de/applets/hades/webdemos/16-flipflops/10-srff/srff.html

Du erkennst hier: Auch ein RS-Flip-Flop ist nichts anderes als eine digitale Schaltung bestehend aus einfachen Gattern.

Hinweis

<p> Diese Seite war zuletzt nicht mehr direkt erreichbar. Allerdings existiert noch eine statische Kopie beim Internet Archive (ohne Applets, aber mit Wahrheitstafeln): https://web.archive.org/web/20110830134111/http://tams-www.informatik.uni-hamburg.de/applets/hades/webdemos/16-flipflops/10-srff/srff.html

Aufgabe 5

Aufgabe

Aufg. 17: Forbidden (RS-Flip-Flop)

In den im Applet auf der Hades-Seite aus der vorangegangenen Aufgabe angegebenen Wahrheitstafeln gibt es jeweils den Hinweis "forbidden".

- Was ist mit "forbidden" gemeint?

- Welche Voraussetzung muss an den Eingangsleitungen erfüllt sein, damit der mit "forbidden" gekennzeichnete Zustand eintritt?
- Wie realistisch schätzt du das Eintreten dieser Voraussetzung ein?

Hinweis

Weiterführende Literatur

Die hier verlinkte Online-Ausgabe eines Lehrtextes der Otto-Friedrich-Universität Bamberg liefert in **Kapitel 5.2 und 5.3** detailliertere Informationen zum Aufbau eines Registers und seiner Speicherzellen. In weiteren Kapiteln finden sich darüber hinaus ergänzende Erläuterungen zum Themengebiet. Die Lektüre dieser Quelle sei unter Beachtung der geltenden Lizenz ausdrücklich empfohlen.

Autoren: Martin Eisenhardt, Andreas Henrich, Stefanie Sieber
'''Rechner- und Betriebssysteme, Kommunikationssysteme, Verteilte Systeme'''

Dieses Werk steht unter der Creative Commons BY-NC-ND-Lizenz
http://creativecommons.org/licenses/by-nc-nd/2.0/de/

2.1.5.2 Aufbau und Arbeitsweise des Speicherwerks

Dieses Thema wird in der weiterführenden Literatur behandelt:

Hinweis

Weiterführende Literatur

Die hier verlinkte Online-Ausgabe eines Lehrtextes der Otto-Friedrich-Universität Bamberg liefert in **Kapitel 5.4 "Realisierung des Hauptspeichers"** detaillierte Informationen zum Aufbau des Speicherwerks. In weiteren Kapiteln finden sich darüber hinaus ergänzende Erläuterungen zum Themengebiet. Die Lektüre dieser Quelle sei unter Beachtung der geltenden Lizenz ausdrücklich empfohlen.

Autoren: Martin Eisenhardt, Andreas Henrich, Stefanie Sieber
'''Rechner- und Betriebssysteme, Kommunikationssysteme, Verteilte Systeme'''

Dieses Werk steht unter der Creative Commons BY-NC-ND-Lizenz
http://creativecommons.org/licenses/by-nc-nd/2.0/de/

2.1.5.3 Aufbau und Arbeitsweise der ALU

Dieses Thema wird in der weiterführenden Literatur behandelt:

Hinweis

Weiterführende Literatur

Die hier verlinkte Online-Ausgabe eines Lehrtextes der Otto-Friedrich-Universität Bamberg liefert in Kapitel 5.5 "**Die arithmetisch-logische Einheit**" detaillierte Informationen zur ALU. In weiteren Kapiteln finden sich darüber hinaus ergänzende Erläuterungen zum Themengebiet. Die Lektüre dieser Quelle sei unter Beachtung der geltenden Lizenz ausdrücklich empfohlen.

Autoren: Martin Eisenhardt, Andreas Henrich, Stefanie Sieber
'''Rechner- und Betriebssysteme, Kommunikationssysteme, Verteilte Systeme'''

Dieses Werk steht unter der Creative Commons BY-NC-ND-Lizenz
http://creativecommons.org/licenses/by-nc-nd/2.0/de/

2.1.5.4 Aufbau und Arbeitsweise eines Busses

Dieses Thema wird in der weiterführenden Literatur behandelt:

Hinweis

Weiterführende Literatur

Die hier verlinkte Online-Ausgabe eines Lehrtextes der Otto-Friedrich-Universität Bamberg liefert in Kapitel 5.6 "**Integration der Komponenten**" detaillierte Informationen zum parallelen Bus. In weiteren Kapiteln finden sich darüber hinaus ergänzende Erläuterungen zum Themengebiet. Die Lektüre dieser Quelle sei unter Beachtung der geltenden Lizenz ausdrücklich empfohlen.

Autoren: Martin Eisenhardt, Andreas Henrich, Stefanie Sieber
'''Rechner- und Betriebssysteme, Kommunikationssysteme, Verteilte Systeme'''

Dieses Werk steht unter der Creative Commons BY-NC-ND-Lizenz
http://creativecommons.org/licenses/by-nc-nd/2.0/de/

2.1.6 Gatter

Im Kapitel Aufbau und Arbeitsweise eines Registers wird der Aufbau und die Arbeitsweise einer einzelnen Speicherzelle eines Registers der CPU erklärt (siehe Video: Arbeitsweise einer Speicherzelle). Dabei kommen u.a. UND-Gatter zum Einsatz.

Gatter-Typen

Die folgende Abbildung zeigt weitere definierte Gatter-Typen:

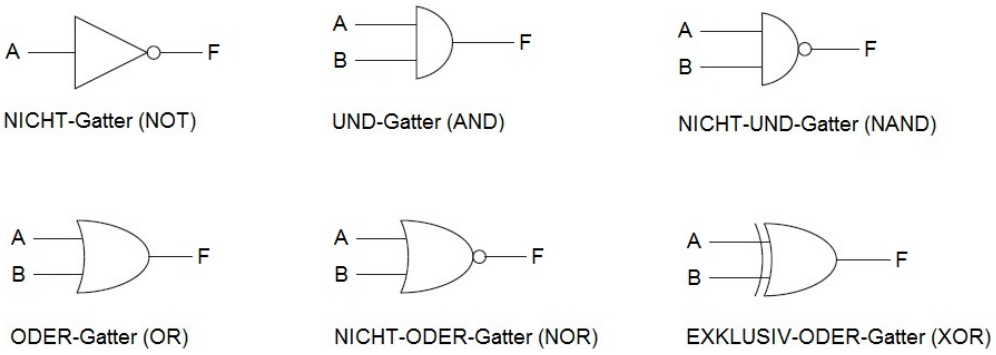


Abb. 27: Unterschiedliche Gatter
Symbole gemäß IEEE Standard 91a-1991
CC-BY

Wahrheitstafeln

Die für die Gatter geltenden Ausgangswerte in Abhängigkeit der Eingangswerte zeigen die folgenden Wahrheitstafeln:

NICHT (NOT)

Eingang	Ausgang
A	F
0	1
1	0

UND (AND)

Eingänge		Ausgang
A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

NICHT-UND (NAND)

Eingänge		Ausgang
A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

ODER (OR)

Eingänge		Ausgang
A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

NICHT-ODER (NOR)

Eingänge		Ausgang
A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

EXKLUSIV-ODER (XOR)

Eingänge		Ausgang
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Abb. 28: Wahrheitstafel für Gatter
CC-BY

Zusammenhang

Das folgende Video verdeutlich den Zusammenhang des jeweiligen Gatters mit seiner Wahrheitstafel.

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/-MXBdtikhzw

Med. 13: Gatter und ihre Wahrheitstafeln (02:56)
http://youtu.be/-MXBdtikhzw
CC-BY

Zusätzliche Bescheibungen, auch zum internen Aufbau eines einzelnen Gatters, liefert die weiterführende Literatur:

Hinweis

Weiterführende Literatur

Die hier verlinkte Online-Ausgabe eines Lehrtextes der Otto-Friedrich-Universität Bamberg liefert in **Kapitel 4: Schaltwerke** detailliertere Informationen zum Aufbau eines Registers und seiner Speicherzellen. Die Lektüre dieser Quelle sei unter Beachtung der geltenden Lizenz ausdrücklich empfohlen.

Autoren: Martin Eisenhardt, Andreas Henrich, Stefanie Sieber
'''Rechner- und Betriebssysteme, Kommunikationssysteme, Verteilte Systeme'''
http://www.uni-bamberg.de/fileadmin/uni/fakultaeten/wiai_lehrstuehle/medieninformatik/Dateien/Publikationen/2007/eisenhardt-rbkvs-1.0.pdf

Dieses Werk steht unter der Creative Commons BY-NC-ND-Lizenz
http://creativecommons.org/licenses/by-nc-nd/2.0/de/

Transistoren, Gatter und Addierer auf der CPU

Quelle im Web

Eine sehr schöne (englischsprachige) Erläuterung des Zusammenhangs zwischen Transistoren, Gatter und Addierer auf einer CPU bietet das folgende Video (14:26) auf YouTube:

See How Computers Add Numbers In One Lesson
http://www.youtube.com/watch?v=VBDoT8o4q00

Schau es dir an!

2.2 Prozessoren und ihre Befehle

Dieses Kapitel widmet sich den Prozessoren und ihren Befehlen. Das folgende Video gibt einen kurzen Überblick:

Video

An dieser Stelle befindet sich online ein YouTube-Video.
https://youtu.be/grZ798STphE

Med. 14: Überblick: Prozessoren und ihre Befehle (01:15)
http://youtu.be/grZ798STphE
CC-BY

Hinweis

Weiterführende Literatur

Brinkschulte et al. 2010 und *Wuest 2011* erläutern viele Hintergründe zum Thema (Mikro-) Prozessoren. Diese Bücher seien als Begleitlektüre ausdrücklich empfohlen.

Studierende sind oftmals berechtigt, eine PDF-Version dieser Bucher ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

So geht es weiter:

Gliederung

2.2.1 Vom Quellcode zum Prozessor

Hier sieht man den Quellcode eines ganz einfachen Programms, jeweils in den Hochsprachen Java, C und Pascal:

Quellcode in Java

Code

```
public class Addierer {
   public static void main(String[] args) {
      int x = 2;
      int y = 5;
      int z = x + y;
   }
}
```

<hr>

Listing 1: Java

Quellcode in C

Code

```
void main(void) {
   int x = 2;
   int y = 5;
   int z = x + y;
}
```

<hr>

Listing 2: C

Quellcode in Pascal

Code

```
PROGRAM Addierer;
VAR
  x, y, z: Integer
BEGIN
  x := 2;
  y := 5;
  z := x + y;
END.

<hr>
```

Listing 3: Pascal

Dieses Programm ist sehr einfach zu verstehen:

- Es werden drei Variablen deklariert.
- Der ersten Variablen x wird der Wert 2 zugewiesen.
- Der zweiten Variablen y wird der Wert 5 zugewiesen.
- Die dritte Variable z bekommt ihren Wert zugewiesen als Ergebnis der Addition der Werte der Variablen x und y.

In C oder Pascal wird nun der Compiler auf den Quellcode angesetzt und heraus kommt eine ausführbare Datei. Bei dem Java-Quelltext sieht es etwas anders aus, da hier der Compiler nicht direkt eine ausführbare Datei erzeugt, sondern zunächst nur eine Datei mit sogenannten Bytecode, der später interpretiert wird. Wir beschränken uns in der weiteren Betrachtung auf die direkt vom Compiler erzeugte ausführbare Datei.

Nach dem Start der ausführbaren Datei wird der sogenannte Programmtext in den Hauptspeicher (Speicherwerk) geladen. Der Programmtext enthält die tatsächlich von der CPU ausführbaren Befehle, es handelt sich um die sogenannte Maschinensprache.

Definition: Maschinensprache

Definition

Unter **Maschinensprache** oder **Maschinencode** versteht man eine Folge von Einsen und Nullen, die einen oder mehrere Befehle repräsentieren, die auf einer CPU ausgeführt werden können.

Unterschiedliche CPUs unterstützen üblicherweise eine unterschiedliche Anzahl an Befehlen, auch die Notation von evtl. gleichbedeutenden Befehlen kann bei unterschiedlichen CPUs variieren.

Beispiel Maschinencode

Hier sieht man ein Beispiel für Maschinencode:

Code

```
00000000110000100000000100001101000000001100010100000001000011100000 0010001101000000011000111000000001000011110000001110000000
```

Listing 4: Maschinencode (Programmtext)

Und an genau dieser Stelle beginnen die Probleme für den Menschen. Eine sehr lange Reihe von Einsen und Nullen ist nicht wirklich dafür geeignet, dass der Mensch sie problemlos versteht.

Das folgende Video bringt etwas Licht ins Dunkel und erläutert die Bedeutung dieser Reihe.

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/cX5XLc9e_g4

Med. 15: Vom Quellcode zum Prozessor (14:09)
http://youtu.be/cX5XLc9e_g4
CC-BY

Am Ende des Videos wird auf eine SWF-Animation verwiesen. Das Bearbeiten der Animation stellt eine sinnvolle Fortsetzung des Videos dar, weshalb die folgende Aufgabe dieses aufgreift.

Aufgabe 1

Aufgabe

Aufg. 18: Sum Program

Bearbeite die SWF-Animation unter
http://courses.cs.vt.edu/csonline/MachineArchitecture/Lessons/CPU/sumprogram.html
und vertiefe damit das Verständnis für die Ausführung von Maschinensprache auf einer CPU.

Die Animation kann sowohl für den Ablauf mit Maschinensprache, als auch mit Assembler konfiguriert werden. Bedenke, dass Assembler nur eine Vereinfachung für den Menschen darstellt. Auf der CPU werden immer binär codierte Befehle (--> Maschinensprache) ausgeführt.
Falls dein Gerät die SWF-Wiedergabe nicht unterstützt, findest du hier ein aufgenommenes Video der Animation:
Sum program (Assembler) --> http://youtu.be/_HwoiEkW8nI (03:33)
Sum program (Maschinensprache) --> http://youtu.be/i2sREE1aAOc (03:23)
Diese Videos sind ohne Ton aufgezeichnet.

Befehle in Maschinencode

Die oben gezeigte Reihe aus Einsen und Nullen besteht in diesem Beispiel also aus insgesamt acht Befehlen, von denen jeder aus genau 16 Bit besteht.

</> Code

```
0000000011000010
0000000100001101
0000000011000101
0000000100001110
0000000010001101
0000000110001110
0000000100001111
0000001110000000

<hr>
```

Listing 5: Acht Maschinensprachebefehle, bestehend aus je 16 Bit

Assemblercode

Diese Maschinensprachebefehle lassen sich für Menschen besser als Assemblercode darstellen:

</> Code

```
LOAD #2
STORE 13
LOAD #5
STORE 14
LOAD 13
ADD 14
```

```
STORE 15
HALT
<hr>
```

Listing 6: Assembler

Wie die verschiedenen auf dieser Seite angegebenen Videos zeigen, ist anhand des Assemblercodes sehr leicht nachvollziehbar, dass der Programmtext auf der betrachteten Beispiel-CPU tatsächlich eine Übersetzung des oben auf dieser Seite gegebenen Quellcodes in einer der Hochsprachen ist.

Hinweis

Hinweis

Der hier verwendete Begriff *Assembler* wird beispielhaft anhand einiger sehr einfach gehaltener Befehle erläutert. Es soll an dieser Stelle darauf hingewiesen werden, dass es für jeden Computertyp eine spezielle, auf den Befehlssatz des Prozessors zugeschnittene Assemblersprache gibt.

Eine tiefergehende Auseinandersetzung mit Assembler soll an dieser Stelle nicht erfolgen. Bei Bedarf können weitere Hintergründe zu Assemblersprachen in einem gleichnamigen Wikipedia-Artikel nachgelesen werden.

In diesem Zusammenhang veranschaulicht die verlinkte Auflistung von ''Hallo-Welt-Programmen'' in verschiedenen Assemblersprachen deutliche Unterschiede.

Ein paar zusätzliche Aufgaben sollen das Thema noch vertiefen.

Aufgabe 2

Aufgabe

Aufg. 19: Count Program

Betrachte folgende Internet-Seite, auf der ganz unten das *Count program* mit SWF-Animation zu finden ist:
http://courses.cs.vt.edu/csonline/MachineArchitecture/Lessons/CPU/Lesson.html
Starte die SWF-Animation zum *Count program* und beobachte den Ablauf.

Falls dein Gerät die SWF-Wiedergabe nicht unterstützt, findest du hier ein aufgenommenes Video der Animation:
Count program (Assembler) --> http://youtu.be/IwjR83896p0 (09:37)
Count program (Maschinensprache) --> http://youtu.be/zIDj4FlOV7U (09:32)
Diese Videos sind ohne Ton aufgezeichnet.

Aufgabe 3

Aufgabe

Aufg. 20: EQUAL-Befehl

Wie funktioniert der im *Count program* enthaltene EQUAL-Befehl?

Aufgabe 4

Aufgabe

Aufg. 21: JUMP-Befehl

Wie funktioniert der im *Count program* enthaltene JUMP-Befehl?

Arbeitet JUMP auch auf dem Akkumulator?

Aufgabe 5

Aufgabe

Aufg. 22: Assembler-2-Hochsprache

Der Assemblercode des *Count program* sieht wie folgt aus:

```
LOAD #5
STORE 15
LOAD #0
EQUAL 15
JUMP #6
HALT
ADD #1
JUMP #3
```

Erarbeite einen Vorschlag für ein äquivalentes Programm in einer Hochsprache wie Java, C, Pascal oder ähnlich. Diskutiere deinen Vorschlag in deiner Lerngruppe! (Keine Idee? Dann frag in deiner Lerngruppe mal nach einem Tipp!)

Aufgabe 6

Aufgabe

Aufg. 23: Speicherzellen für die Befehle

Weise jedem Assembler-Befehl aus der vorangegangenen Aufgabe eine Speicherzelle in der folgenden Abbildung des Speicherwerks zu.

- Ist das Assembler-Programm so noch lauffähig?

- Erkläre den anderen Mitgliedern deiner Lerngruppe den Ablauf im Zusammenspiel zwischen CPU und Speicherwerk!

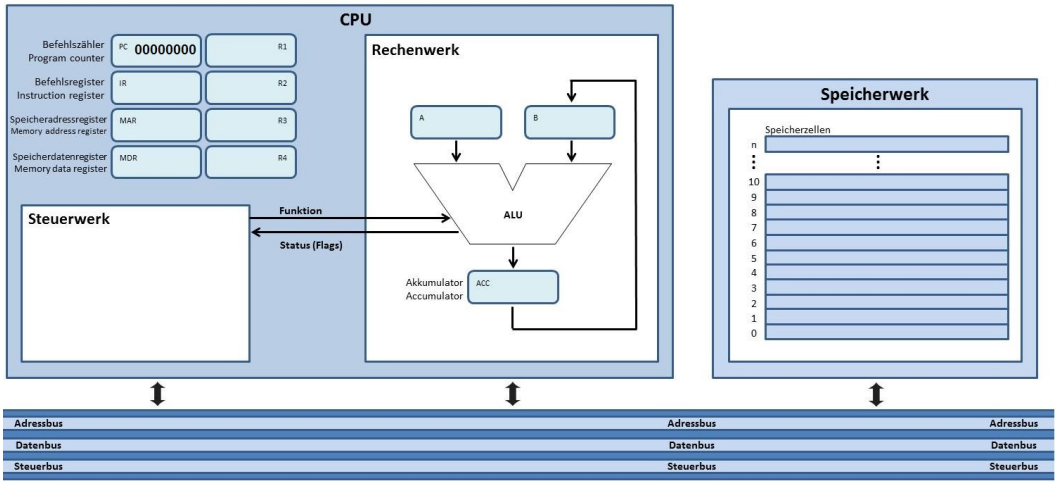


Abb. 29: Abbildung zu Aufgabe 6
CC-BY

2.2.2 Befehlssatz

Definition: Befehlssatz

Definition

Unter dem **Befehlssatz einer CPU** versteht man die Menge der von einer CPU unterstützten Befehle.

Beispiel Befehlssatz

Ein Beispiel für einen ganz einfachen Befehlssatz ist in dem bereits bekannten Video Vom Quellcode zum Prozessor zu sehen:

000 → NOOP
001 → LOAD
010 → STORE
011 → ADD
100 → SUB
101 → EQUAL
110 → JUMP
111 → HALT

Abb. 30: Einfacher Befehlssatz
CC-BY

Die Auflistung der einzelnen Befehle erfolgt in dem vorangegangenen Bild jeweils in Maschinencode (3 Bit) als auch als Assemblerbefehl.

Befehlsarten

Innerhalb des Befehlssatzes lassen sich einzelne **Befehlsarten** unterscheiden. Zu den Befehlsarten zählen unter anderem:

- Datenbewegungsbefehle (z.B. LOAD, STORE)
- Arithmetisch-logische Befehle (z.B. ADD, SUB)
- Programmsteuerbefehle (z.B. JUMP, EQUAL)
- Systemsteuerbefehle (z.B. HALT)

Hinweis

Weiterführende Literatur

Brinkschulte et al. 2010 erläutern in Kapiteln 2.1.4 (Befehlssatz) weitere Hintergründe zum Thema. Die Lektüre dieser Quelle sei ausdrücklich empfohlen.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

2.2.3 Befehlsformat

Neben der Festlegung der Namen der einzelnen Befehle (bzw. deren binärer Codierung), ist die Festlegung des sogenannten **Befehlsformats** eine wichtige Entscheidung bei der Entwicklung einer CPU.

Das **Befehlsformat** definiert für jeden einzelnen Befehl, wie dieser codiert ist. Im Video Vom Quellcode zum Prozessor entspricht dies dem aus drei Teilen bestehende Format:

<Befehl><Num><Operand>

Dabei wird <Befehl> mit drei Bit codiert, <Num> mit einem Bit und <Operand> mit sechs Bit. Die im Video gezeigte Reserve dient lediglich dazu, die gesamte Befehlslänge auf ein Vielfaches eines Bytes (= acht Bit) zu ergänzen. Die Befehlslänge aus dem Beispiel im Video beträgt somit für alle Befehle 16 Bit.

Definition: Opcode

Definition

Unter dem **Opcode** eines Befehls versteht man eine binäre Codierung, aus der sowohl der Befehl, als auch zusätzlich benötigte Steuerinformationen hervorgehen.

Der Opcode für das obige Beispiel zum Einadressformat besteht aus <Befehl> und <Num>, insgesamt also aus vier Bit. Das eine Bit von <Num> ermöglicht die Unterscheidung von zwei Steueroptionen.

Klassifizierungen

Bei den Befehlsformaten werden verschiedene Klassifizierungen unterschieden, *Brinkschulte et al. 2010* gibt diese Klassen ausführlich an. An dieser Stelle werden lediglich drei unterschiedliche Varianten betrachtet:

- Einadressformat
- Zweiadressformat
- Dreiadressformat

Aufgabe 1

Aufg. 24: Klassifizierungen

Welche Klassifizierungen bzgl. der Befehlsformate unterscheiden *Brinkschulte et al. 2010* ?

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

2.2.3.1 Einadressformat

Das **Einadressformat** entspricht dem Format aus dem Video Vom Quellcode zum Prozessor . Hier wird für die einzelnen Befehle nur der *Opcode* und die *Adresse eines Operanden* (deshalb *Einadressformat*) angegeben.

<Opcode><Operand1>

Bedeutung des Akkumulators

Wird ein zweiter Operand benötigt, so wird vorausgesetzt, dass dieser sich im Register Akkumulator befindet. Das bei der Abarbeitung des Befehls berechnete Ergebnis wird per Definition wieder im Akkumulator gespeichert.

Betrachte für die folgenden Aufgaben die Befehle NOOP, LOAD, STORE, ADD, SUB, EQUAL, JUMP, HALT aus dem Video Vom Quellcode zum Prozessor .

Aufgabe 1

Aufgabe

Aufg. 25: Operand oder nicht?

Welche dieser Befehle benötigen einen Operanden, welche benötigen keinen Operanden?

Aufgabe 2

Aufgabe

Aufg. 26: Wenn der Operand keine Adresse ist

Bei Befehlen mit einem Operanden muss der Operand nicht zwingend die Adresse einer Speicherzelle repräsentieren. Im Video gibt es eine zweite Interpretationsmöglichkeit des Operanden (Stichwort: # im Video).

- Was ist gemeint?
- Und wie wird bei der Repräsentation in Maschinensprache (--> Einsen und Nullen!) dafür gesorgt, dass beide Interpretationsmöglichkeiten unterschieden werden können?

2.2.3.2 Zweiadressformat

Beim **Zweiadressformat** besteht ein kompletter Befehl aus dem *Opcode*, einer *Adresse des ersten Operanden* und einer *Adresse des zweiten Operanden*. Per Definition wird das bei der Abarbeitung des Befehls berechnete Ergebnis unter der Adresse eines der beiden Operanden abgelegt. Es bedarf hierfür der Festlegung, an der Adresse welches Operanden das Ergebnis später zu finden ist, üblicherweise wird hierfür die Adresse des ersten Operanden gewählt.

<Opcode><Operand1/Ergebnis><Operand2>

Die Doppelbelegung <Operand1/Ergebnis> kann bei Nutzung des Dreiadressformats vermieden werden.

2.2.3.3 Dreiadressformat

Beim **Dreiadressformat** besteht ein kompletter Befehl aus dem *Opcode*, *einer Adresse des ersten Operanden*, einer *Adresse des zweiten Operanden*, sowie einer *Adresse, an der das Ergebnis gespeichert wird*.

Üblicherweise ergibt sich in der Notation folgende Reihenfolge:

<Opcode><Ergebnis><Operand1><Operand2>

Diese Reihenfolge spiegelt sich beispielsweise auch im C-Quelltext aus dem Beispiel im Kapitel Vom Quellcode zum Prozessor wider. Hier der entscheidende Ausschnitt:

</>
Code

```
int z = x + y;
```

Z nimmt das Ergebnis auf und steht zuerst, x und y sind die beiden Operanden und folgen.

2.2.3.4 Das Adressformat und der Von-Neumann-Zyklus

Der Vom-Neumann-Zyklus wurde ja bereits im gleichnamigen Kapitel erläutert. Hier noch einmal die Abbildung dazu:

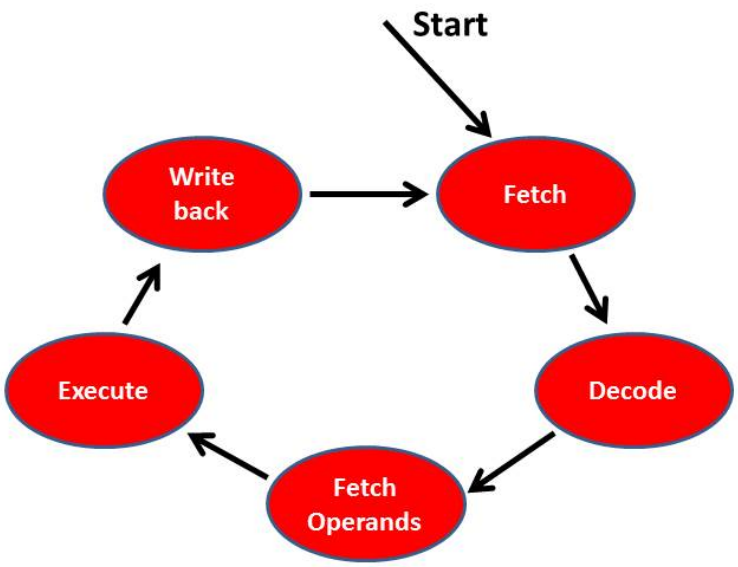


Abb. 31: Von-Neumann-Zyklus
CC-BY

Beim Zweiadressformat und beim Dreiadressformat hat der Zyklus mit seinen 5 Phasen auch nach wie vor seine Gültigkeit.

Beim Einadressformat hingegen kann man auch einen **vereinfachten Zyklus** betrachten. Dieser besteht dann nur noch aus den drei Phasen: **Fetch - Decode - Execute**. Dies begründet sich in der Tatsache, dass die Operanden hier i.d.R. bereits in Registern auf der CPU bereitstehen und auch das Ergebnis wieder in ein Register der CPU eingetragen wird. Es entfallen damit in den meisten Fällen die (verhältnismäßig lange dauernden) Phasen des "Fetch Operands" und des "Write back".

2.2.3.5 Aufgaben & Co. zum Befehlsformat

Betrachte für die folgenden Aufgaben das Beispiel aus dem Video Vom Quellcode zum Prozessor und den dort gegebenen Befehlssatz im Einadressformat:

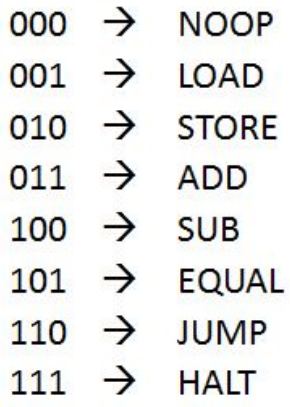


Abb. 32: Einfacher Befehlssatz
CC-BY

Aufgabe 1

Aufgabe

Aufg. 27: Entwicklung eines Befehlssatzes

Ein kompletter Befehl (Opcode und Operand) vom Typ NOOP, LOAD, STORE, ADD, SUB, EQUAL, JUMP und HALT besteht derzeit (wie im Video erläutert) aus insgesamt 16 Bit, wobei die ersten sechs Bit als Reserve nicht genutzt werden.

Entwickle einen Befehlssatz,

- der maximal 256 Befehle umfassen kann, aber zunächst nur die bereits bekannten Befehle NOOP, LOAD, STORE, ADD, SUB, EQUAL, JUMP und HALT definiert,
- der maximal 64 Steueroptionen unterscheiden kann,
  - bislang waren es nur zwei Steueroptionen, siehe <Num> im Video ,

    - die hier angegebene Anzahl ist "*insgesamt für den Befehl*" gemeint, und **nicht** "*pro Operand*",
- der je sechs Bit für die Adressierung von <Ergebnis>, <Operand1> und <Operand2> vorsieht (Dreiadressformat),
- dessen Befehlslänge ein Vielfaches von acht Bit sein soll.

Aus wie vielen Bit besteht damit ein kompletter Befehl (Opcode und Operand) dieses neuen Befehlssatzes mindestens?

Gebe das SUM-Program aus dem Video damit an! Herauskommen sollte eine Übersicht wie diese, angepasst an den neuen Befehlssatz:

Reserve	Befehl	Num	Operand	Assembler
000000	001	1	000010	LOAD #2
000000	010	0	001101	STORE 13
000000	001	1	000101	LOAD #5
000000	010	0	001110	STORE 14
000000	001	0	001101	LOAD 13
000000	011	0	001110	ADD 14
000000	010	0	001111	STORE 15
000000	111	0	000000	HALT

Befehlscodierung

000	→	NOOP
001	→	LOAD
010	→	STORE
011	→	ADD
100	→	SUB
101	→	EQUAL
110	→	JUMP
111	→	HALT

Num (Nummer-Bit)

0	→	Operand ist Speicheradresse
1	→	Operand ist Zahl (#)

Abb. 33: Abbildung zu Aufgabe 1
CC-BY

Erwartet wird also, dass das SUM-Programm sowohl in Maschinensprache (Einsen und Nullen!), als auch in Assembler angegeben wird. Der Additionsbefehl mit drei Operanden könnte in Assembler so aussehen:

ADD ACC, ACC, 14

"ACC" bezeichnet das Register Akkumulator, die Bedeutung des Befehls ist damit: *Addiere den im Akkumulutar gespeicherten Wert und den in Speicherzelle 14 gespeicherten Wert, und lege das Ergebnis im Akkumulator ab*.

Wie löst du das Problem, dass in der binären Codierung (Maschinensprache) unterschieden werden muss, ob ein Operand ein Register, eine Speicherzelle oder gar einen konkreten Wert bezeichnet? (Stichwort: # im Video . Falls du hier Hilfe benötigst, so diskutiere Lösungsansätze in deiner Lerngruppe!)

Aufgabe 2

Aufgabe

Aufg. 28: Nur sechs Bit

Wie viele Speicherzellen können bei sechs Bit für die Adressierung von <Ergebnis>, <Operand1> und <Operand2> maximal unterschieden werden?

2.2.4 Adressierungsarten

Eine CPU kann verschiedene Adressierungsarten unterscheiden.

Definition: Adressierungsart

Definition

Unter einer Adressierungsart versteht man eine Möglichkeit, wie die CPU einen in einem Befehl angegebenen Operanden interpretiert.

Das Ziel bei der Interpretation eines Operanden während der Ausführung eines Befehls auf der CPU ist zunächst herauszufinden,

- ob der Operand eine konkrete Zahl (Konstante) darstellt,
- ob der Operand ein konkretes Register auf der CPU bezeichnet,
- oder ob mit ihm die sogenannte *effektive Adresse* ermittelt werden kann.

Definition: Effektive Adresse

Definition

Unter der **effektiven Adresse** versteht man die tatsächliche Adresse einer Speicherzelle im Hauptspeicher. Man nennt diese auch die **physikalische Adresse**.

Verschiedene Adressierungsarten

Es gibt eine Reihe unterschiedlicher Adressierungsmöglichkeiten. *Brinkschulte et al. 2010* erläutern elf verschiedene Varianten, von denen auf den kommenden Seiten nur die Folgenden vorgestellt werden:

- unmittelbare Adressierung
- Registeradressierung
- direkte/absolute Adressierung
- registerindirekte Adressierung
- indizierte Adressierung mit Verschiebung

Aufgabe 1

Aufg. 29: Adressierungsarten

Informiere dich bei *Brinkschulte et al. 2010* in Kapitel 2.1.6 (Adressierungsarten) über alle dort erläuterten Adressierungsarten.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

2.2.4.1 Unmittelbare Adressierung

Die **unmittelbare Adressierung** ist bereits aus dem Video Vom Quellcode zum Prozessor bekannt. In diesem Fall steht *unmittelbar* im Befehl eine konkrete Zahl (Konstante).

Beispiele aus dem Video :

LOAD #2

LOAD #5

2.2.4.2 Registeradressierung

Eine **Registeradressierung** liegt vor, wenn der Operand direkt ein Register der CPU bezeichnet. Diese Variante ist ebenfalls bereits aus dem Video Vom Quellcode zum Prozessor bekannt, sie kommt dort aber nur "unsichtbar" vor.

Die im Video gezeigten (Einadress-) Befehle arbeiten auf dem Register Akkumulator. Somit liegt hier eine Registeradressierung vor. Unsichtbar ist sie deshalb, da kein Operand dafür zu sehen ist, es wird per Definition immer auf den Akkumulator zugegriffen.

In wird die Registeradressierung dann deutlich sichtbar. Der dort angegebene Befehl

ADD ACC, ACC, 14

spricht das Register Akkumulator als ersten Operanden und als Ziel für die Aufnahme des Additionsergebnisses an.

Hinweis

Wenn du bereits sinnvoll bearbeitet hattest, dann hast du eine Reihe von weiteren Befehlen entwickelt, die ebenfalls die Registeradressierung verwenden.

2.2.4.3 Direkte/absolute Adressierung

Bei der **direkten** oder **absoluten Adressierung** steht die effektive Adresse direkt im Befehl. (Monopoly-Spieler würden sagen: *Gehe **direkt** zu dieser Adresse, denn es ist **absolut** die richtige Adresse!*)

Beispiele aus dem Video Vom Quellcode zum Prozessor :

LOAD 13
ADD 14
STORE 15

Es ergibt sich in der Praxis aber recht schnell ein Problem. In dem im Video gezeigten Beispiel stehen nur sechs Bit für die Codierung der angesprochenen Speicherzelle im Hauptspeicher zur Verfügung, insgesamt lassen sich damit also nur $2^6 = 64$ Speicherzellen unterscheiden, viel zu wenig für die Speichergrößen in heutigen Rechnern. Selbst eine Erweiterung auf 12 oder 24 Bit pro Operand bringt noch keine Lösung des Problems.

Aufgabe 1

Aufgabe

Aufg. 30: Finde es heraus

Finde heraus, ob die auf den folgenden Seiten beschriebenen Adressierungsarten das Problem lösen!

2.2.4.4 Registerindirekte Adressierung

Bei der **registerindirekten Adressierung** bezeichnet der **Operand** ein **Register** auf der CPU, der in diesem Register abgelegte Wert bezeichnet die effektive Adresse der Speicherzelle.

Handelt es sich um ein 32-Bit-Register, so stehen damit auch 32 Bit für die Adressierung einer Speicherzelle im Hauptspeicher zur Verfügung. Bei einem 64-Bit-Register entsprechend 64 Bit.

Aus Aufgabe 1 der Seite ist bereits die Schreibweise

ADD ACC, ACC, 14

bekannt. Hier erfolgt zweimal die Registeradressierung des Akkumulator. Um jetzt eine **registerindirekte Adressierung** anzuzeigen, wird die Schreibweise verändert:

ADD ACC, (ACC), 14

Die hier eingeführten **Klammern** um das bezeichnete Register zeigen die Verwendung der registerindirekten Adressierung an.

Die Bedeutung dieses Befehls ist damit: Addiere den Wert der Speicherzelle im Hauptspeicher, die durch den im Register (ACC) gespeicherten Wert addressiert wird, und den in Speicherzelle 14 gespeicherten Wert. Das Ergebnis schreibe in den Akkumulator.

Falls also im Akkumulator der Wert 80 gespeichert ist, so wird auf Speicherzelle 80 zugegriffen. Zu deren Wert wird der in Speicherstelle 14 hinterlegte Wert addiert und das Ergebnis in den Akkumulator geschrieben.

Aufgabe 1

Aufgabe

Aufg. 31: Adressierungsarten

Welche Bedeutungen haben die folgenden Befehle:

- **ADD** R1, ACC, 8
- **SUB** ACC, 10, (R2)
- **JUMP** (R3)
- **ADD** ACC, (ACC), #5

Diskutiere die Bedeutungen in deiner Lerngruppe!

(R1, R2 und R3 bezeichnen Register auf der CPU, siehe Gesamtbild eines Von-Neumann-Rechners.)

2.2.4.5 Indizierte Adressierung mit Verschiebung

Unter der **indizierten Adressierung mit Verschiebung** versteht man eine Erweiterung der direkten/absoluten Adressierung. Dabei wird zu dem im Befehl angegebenen Operanden (Verschiebung) noch der Wert eines Registers hinzuaddiert. Die Summe entspricht dann der effektiven Adresse der anzusprechenden Speicherzelle.

Auch dazu ein Beispiel anhand der aus dem Video Vom Quellcode zum Prozessor bekannten Ausgangslage:

Man nehme den Befehl "LOAD 13" aus dem Video . 13 ist damit die effektive Adresse der anzusprechenden Speicherzelle bei Verwendung der direkten bzw. absoluten Adressierung. Um es etwas deutlicher zu machen, betrachtet man diesen Befehl im Zweiadressformat:

LOAD ACC, 13

Der Inhalt der Speicherzelle 13 wird in das Register Akkumulator geladen. Bis hierher ist es immer noch die direkte bzw. absolute Adressierung.

Der Schritt zur **indizierten Adressierung mit Verschiebung** besteht nun darin zu erkennen, dass die "Verschiebung" 13 aus dem Beispielbefehl immer von Speicherzelle 0 aus geschieht. Man könnte also sagen: "0+13" ist die effektive Adresse der anzusprechenden Speicherzelle.

Statt der Addition von Null in "0+13" könnte aber auch eine Addition mit dem Inhalt eines Registers stattfinden. Abhängig vom Inhalt dieses Registers geschieht dann eine Verschiebung um 13. Diesen Befehl schreibt man dann als:

LOAD ACC, 13(R2)

Die Bedeutung ist: Lade den Inhalt der Speicherzelle (R2+13) in den Akkumulator. Ist in R2 der Wert 123 gespeichert, so wird auf die effektive Adresse 123+13 = 136 zugegriffen.

Die **indizierte Adressierung mit Verschiebung** war in der Vergangenheit ein bedeutender Meilenstein. Damit wurde eine wichtige Voraussetzung geschaffen, um mehrere Programme gleichzeitig im Hauptspeicher zu halten.

2.2.5 Vom Programm zum Prozess

Bislang wurde in diesem Modul bereits mehrmals der Begriff "Programm" verwendet. Da beim aktuellen Stand der Betrachtung aber bereits ein Rechner beschrieben ist, der ein Programm ausführen kann, wird nun eine Abgrenzung der Begriffe **Programm** und **Prozess** nötig.

Die folgende Definition ist angelehnt an eine deutsche Übersetzung der Definition des englischen Begriffs "computer program" aus:
ISO/IEC 2382-1:1993 Information technology - Vocabulary - Part 1: Fundamental terms ,
ebenso in einer überarbeiteten Version dieses Standards:
ISO/IEC 2382:2015(en) Information technology — Vocabulary .

Definition: Programm

Definition

Ein **Programm** oder **Computerprogramm** ist eine den Regeln einer bestimmten Programmiersprache genügende Folge von Anweisungen (bestehend aus Deklarationen und Instruktionen), mit der auf einem Computer eine bestimmte Funktionalität bereitgestellt wird, bzw. eine bestimmte Aufgaben- oder Problemstellung bearbeitet oder gelöst werden kann.

Nach dieser Definition ist beispielsweise der auf der Seite Vom Quellcode zum Prozessor angegebene Quelltext ein Programm. Dies gilt sowohl für den Quelltext in den Hochsprachen (C, Pascal, Java), als auch für den Assemblercode. Sogar der aus Einsen und Nullen bestehende Maschinencode ist ein Programm im Sinne dieser Definition.

Definition: Prozess

Definition

Ein **Prozess** ist ein Programm in Ausführung.

Diese zweite Definition betont den Begriff "Ausführung". Sobald also ein Programm in den Hauptspeicher eines Computers geladen wurde und der Prozessor bereit zur Ausführung dieses Programms ist, wird aus dem Programm ein Prozess.

Aufgabe 1

Aufgabe

Aufg. 32: Programm-2-Prozess

Ein Programm wurde in den Hauptspeicher eines Computers geladen, die CPU ist bereit zur Ausführung dieses Programms.

Was bedeutet dies für die CPU? Welche Vorarbeiten sind auf der CPU zu erledigen, bevor der erste Befehl des Programms aus dem Speicher geholt und ausgeführt werden kann? Welche Vorarbeiten sorgen also dafür, dass aus dem Programm ein Prozess wird?

2.2.6 Gesamtbild der Programmausführung

An dieser Stelle sei noch einmal darauf hingewiesen, dass das bisher erarbeitete Bild eines Rechners **nur einen Prozess zur Zeit** im Hauptspeicher zulässt. Der Ablauf gestaltet sich wie folgt:

1. Starte den Rechner.
2. Lade ein bestimmtes Programm in den Hauptspeicher.
3. Bereite die CPU zur Ausführung vor. (Dadurch wird das Programm zum Prozess.)
4. Starte in Speicherzelle 0 mit der Ausführung des ersten Befehls.
5. Führe entsprechend dem Ablaufe alle weiteren Befehle aus.
6. Sobald die Ausführung des Prozesses beendet ist (HALT-Befehl), kann der Rechner ausgeschaltet werden.

Beim nächsten Einschalten des Rechners kann wieder neu entschieden werden, welches Programm zum Einsatz kommt.

Anmerkung

Natürlich wirkt dieses ständige Ein- und Ausschalten des Rechners befremdlich für dich als erfahrenen Anwender (bzw. Anwenderin). Schließlich arbeitest du jeden Tag gleichzeitig mit mehreren Programmen (eigentlich: Prozessen!) auf deinem PC oder Notebook. Aber mit dem bis jetzt erarbeiteten Stand eines Rechners ist es noch nicht möglich, dieses zu tun.

Wir nähern uns aber diesem Ziel, es dauert nicht mehr lange. Halte durch!

2.2.7 Aufgaben & Co. zu Prozessoren

Einige weitere Aufgaben runden das Thema ab:

Aufgabe 1

Aufgabe

Aufg. 33: VNA vs. Harvard

Erkundige dich nach den Unterschieden zwischen der **Von-Neumann-Architektur** und der **Harvard-Architektur**!

- Stelle die Harvard-Architektur und die Von-Neumann-Architektur in einer Skizze gegenüber und erläutere die Unterschiede!

- Nutze dafür nicht nur Wikipedia, auch *Brinkschulte et al. 2010* und *Boettcher 2006* geben Erläuterungen dazu.
- Was meint beispielsweise *Boettcher 2006* wenn er in Kapitel 7.2.1 von einer "Pseudo-Harvard-Architektur" schreibt?
- Und welchen Vorteil bieten die in Kapitel 7.2 von *Boettcher 2006* beschriebenen Cache-Speicher?
- Integriere die Cache-Speicher in deine Skizze der Von-Neumann-Architektur!

Studierende sind oftmals berechtigt, eine PDF-Version der genannten Bücher ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Aufgabe 2

Aufg. 34: RISC vs. CISC

RISC (Reduced Instruction Set Computer) und **CISC** (Complex Instruction Set Computer) sind zwei unterschiedliche Designphilosophien aus dem Bereich der Prozessortechniken.

Recherchiere in einem (oder mehreren) Fachbuch/Fachbüchern alles Wissenswerte zu diesen Technologien! Gib dabei die benutzen Fachbücher als Quelle an.

Google und Wikipedia sind sicherlich ein guter Startpunkt für eine Recherche, aber beides sind keine Fachbücher, weshalb sie als offizielle Quelle nicht in Frage kommen. Falls du wissenswerte Fakten "irgendwo im Internet" gefunden hast, so besteht die Kunst dieser Aufgabe darin, nur genau die Fakten in die Antwort dieser Aufgabe aufzunehmen, die ebenfalls in den zur Verfügung stehenden Fachbüchern zu finden sind. Das Literaturverzeichnis listet ausreichend Fachbücher auf. Falls du Zugriff auf weitere Fachbücher hast, kannst du diese selbstverständlich auch benutzen. Aber keine Vorlesungsskripte aus anderen Veranstaltungen deines Studiums.

Du kannst dich an folgenden Fragen orientieren:

- Was versteht man unter RISC bzw. CISC?
- Wie ist die Entstehungsgeschichte? (Was war zuerst da? Was folgte? Warum?)
- Was sind Gemeinsamkeiten und worin unterscheiden sie sich?
- Was sind die jeweiligen Vor- bzw. Nachteile?
- Denke an die Angabe der Fachbücher, die als Quelle gedient haben!

2.3 Weitere Komponenten der Computerarchitektur

Hier werden weitere (Hardware-) Komponenten betrachtet, die das bisher erarbeitete Gesamtbild eines Rechners erweitern. Insbesondere handelt es sich um Komponenten, die für den Bereich Betriebssysteme von besonderer Bedeutung sind.

Bitte beachte

Hinweis

Aktuell haben wir es aufgrund der Erläuterungen aus den vorangegangenen Kapiteln mit einem ganz einfachen Rechner zu tun, der noch weit entfernt ist von der Technik und der Leistungsfähigkeit eines Computers, mit dem jeder von uns tagtäglich arbeitet.

Stell dir einfach vor, dass wir bislang nur einen einfachen Taschenrechner beschrieben haben - mehr noch nicht. Und jetzt geht es in diesem Kapitel weiter damit, zusätzliche Hardware-Komponenten einzuführen und zu verstehen.

Auch die Notwendigkeit eines Betriebssystems war bislang noch nicht gegeben (weil wir immer nur von einem Prozess im Hauptspeicher ausgegangen sind). Es wird also Zeit, dass wir uns dem aktuellen Stand der Technologie weiter annähern.

Abb. 34: Der aktuelle Betrachtungsstand: Nicht mehr als ein einfacher Taschenrechner

CC-BY

So geht es weiter:

Gliederung

2.3 Weitere Komponenten der Computerarchitektur
2.3.1 Stackregister
2.3.2 Basisregister
2.3.3 Limitregister zum Speicherschutz
2.3.4 Interrupt-Controller
2.3.5 DMA-Controller
2.3.6 MMU - Memory Management Unit

2.3.7 Moderne Bussysteme

2.3.1 Stackregister

Das **Stackregister** befindet sich auf der CPU. Sein Wert beziffert eine bestimmte Speicherzelle im Hauptspeicher, den sogenannten **Top of Stack**, also das **obere Ende des Stacks**.

Push oder Pop auf die Datenstruktur

Ein Stack ist eine Datenstruktur, kann also Daten aufnehmen und wieder abgeben, wobei der Zugriff nur über die Befehle **PUSH** (Daten auf dem oberen Ende des Stacks ablegen) sowie **POP** (Daten vom oberen Ende des Stacks entfernen) möglich ist.

Stackregister anschaulich

Anschaulich erklärt wird die Bedeutung des Stackregisters und die Arbeitsweise des Stacks an sich in dem folgenden Video:

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/NjSc5ovr5Tw

Med. 16: Stackregister und Arbeitsweise des Stacks (04:12)
http://youtu.be/NjSc5ovr5Tw
CC-BY

Das Ablegen oder Entfernen von Daten vom Stack kann sowohl aus einem laufenden Prozess heraus erfolgen, oder auch auf Anweisung des Steuerwerks. Letzteres bietet viele Vorteile, wie auf den kommenden Seiten noch zu sehen sein wird.

Stackregister auf der CPU

Das folgende Bild zeigt das Stackregister auf der CPU. In vorangegangenen Bildern zur CPU war an dieser Stelle das Register R4 angesiedelt. Man kann sich gerne vorstellen, dass das Register R4 weiterhin auf der CPU existiert, nur eben aus Platzgründen nicht mehr eingezeichnet ist.

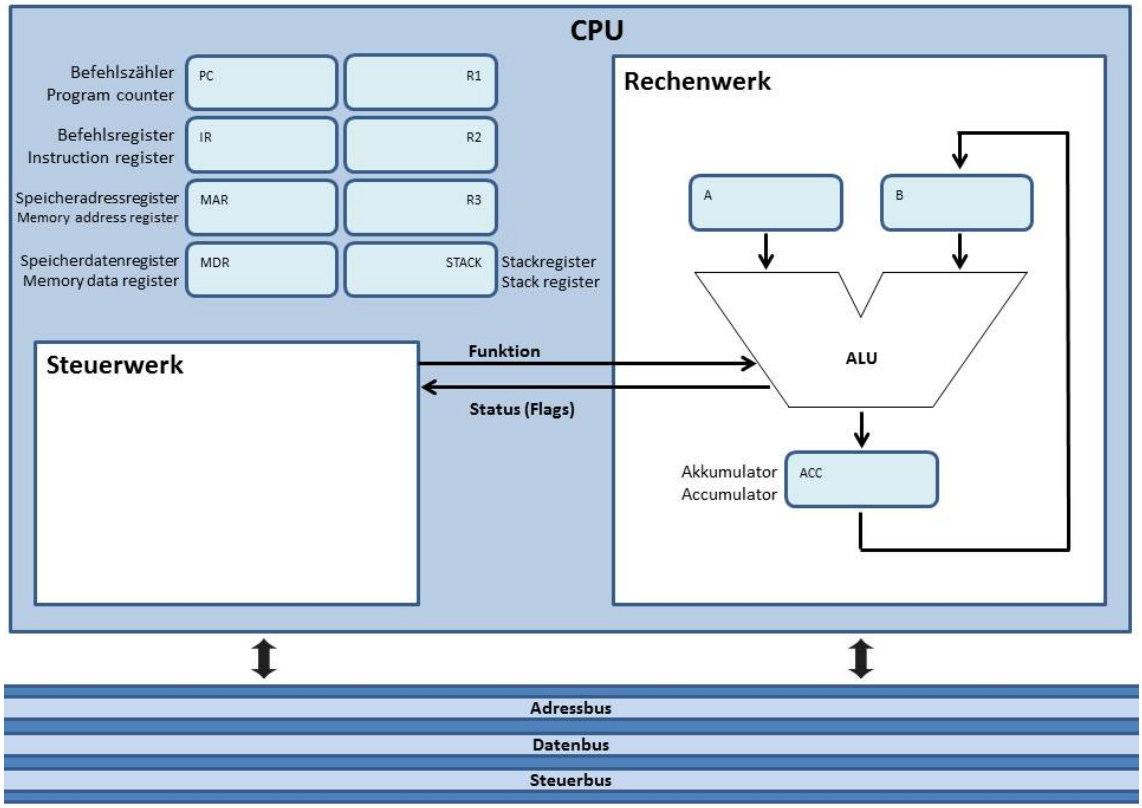


Abb. 35: Stackregister auf der CPU
CC-BY

2.3.2 Basisregister

Das **Basisregister** (engl. **base register**) ist ein spezielles Register auf der CPU. Es wurde in früheren Rechnern eingesetzt, da damit (unter anderem) zwei Probleme sehr einfach gelöst werden konnten. (In heutigen Rechnern findet es üblicherweise keinen Einsatz mehr, es wurde durch die MMU abgelöst.)

Bei den beiden erwähnten Problemen handelt es sich um:

- Wie können mehrere Prozesse gleichzeitig im Hauptspeicher platziert werden?
- Wie können Prozesse temporär aus dem Hauptspeicher ausgelagert werden? (Üblicherweise geschieht die Auslagerung auf die Festplatte, von der sie zu einem späteren Zeitpunkt wieder eingelagert werden.)

Beide Fälle werden auf den kommenden Seiten ausführlich erläutert, zunächst jedoch ein wenig Information vorab.

Definition: Basisregister

Definition

Das Basisregister befindet sich auf der CPU und enthält als Wert die Adresse der Speicherstelle mit dem ersten Befehl des aktiven Prozesses.

Solange sich nur ein einzelnes Programm im Hauptspeicher befindet, ist der Wert des Basisregisters gleich Null, denn ab Speicherstelle 0 beginnt das Programm. In den folgenden Unterkapiteln wird beschrieben, wie auch mehrere Programme gleichzeitig im Hauptspeicher Platz finden. In dieser Situation kann sich der Wert des Basisregisters ändern.

Indizierte Adressierung mit Verschiebung

Das Basisregister wird zur bereits bekannten indizierten Adressierung mit Verschiebung genutzt. Dabei wird diese Adressierungsart "versteckt" eingesetzt, d.h. per Definition werden die Operanden aller Befehle immer als Verschiebung betrachtet, bei der Befehlsausführung wird automatisch der Inhalt des Basisregisters hinzuaddiert. Eine Ausnahme bilden lediglich die unmittelbar adressierten Operanden, da diese keine Speicherzelle adressieren, sondern als konkrete Zahl (Konstante) interpretiert werden müssen.

Das Basisregister auf der CPU

Die folgende Abbildung zeigt die CPU mit dem neu eingeführten Basisregister. In vorangegangenen Bildern zur CPU war an dieser Stelle das Register R3 angesiedelt. Man kann sich gerne vorstellen, dass das Register R3 weiterhin auf der CPU existiert, nur eben aus Platzgründen nicht mehr eingezeichnet ist.

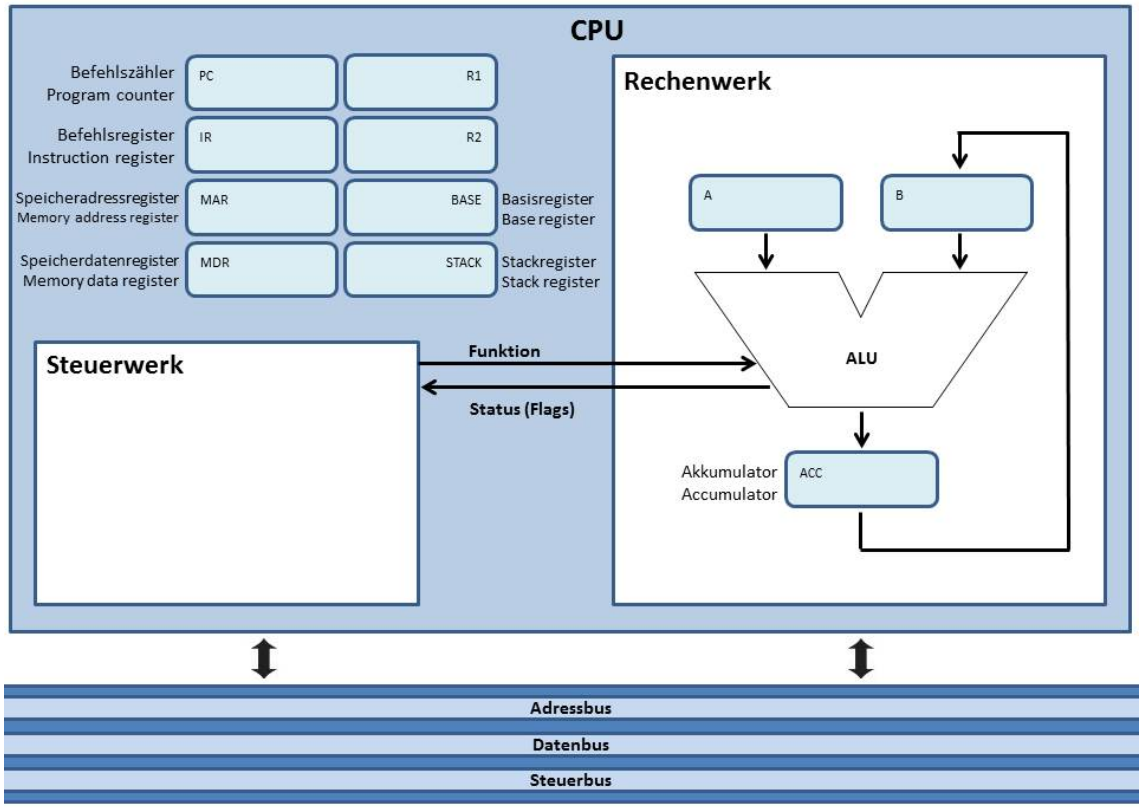


Abb. 36: CPU mit Basisregister
CC-BY

2.3.2.1 Mehrere Prozesse gleichzeitig im Speicher

Zunächst sei an das bisher erarbeitete Gesamtbild der Programmausführung erinnert. Dieser Ablauf wird nun derart erweitert, dass **mehrere** Programme geladen und ausgeführt werden können:

1. Starte den Rechner.
2. Lade nacheinander **mehrere** Programme in den Hauptspeicher. Die Befehle des ersten Programms werden zusammenhängend ab Speicherzelle 0 abgelegt, die Befehle der weiteren Programme jeweils zusammenhängend in freien Speicherbereichen danach.
3. Treffe für jedes Programm die nötigen Vorarbeiten für seine spätere Ausführung. (Dadurch werden nacheinander alle Programme zu Prozessen!)
4. Starte in Speicherzelle 0 mit der Ausführung des ersten Befehls des ersten Prozesses.
5. Führe entsprechend des Ablaufs alle weiteren Befehle dieses ersten Prozesses aus.
6. Sobald die Ausführung des ersten Prozesses beendet ist (HALT-Befehl), wird die CPU auf die Ausführung des zweiten Prozesses vorbereitet.

7. Starte mit dem ersten Befehl des zweiten Prozesses.
8. usw.

Aufgabe 1

Aufgabe

Aufg. 35: Welche Situation gilt?

Welche Situation wird hier beschrieben:

1. Es werden mehrere Prozesse im Speicher gehalten und alle Prozesse laufen parallel ab.

   oder

2. Es werden mehrere Prozesse im Speicher gehalten, aber erst wenn der erste Prozess beendet wurde, startet der zweite Prozess. Und nach dem Ende des zweiten Prozesses startet der dritte Prozess. Usw.

Zusammenspiel mit dem Basisregister

Das folgende Video geht näher auf diese Situation ein und erläutert den Ablauf insbesondere im Zusammenspiel mit dem Basisregister.

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/uBcR9kPIbg8

Med. 17: Basisregister als Voraussetzung für die Unterbringung mehrerer Programme im Hauptspeicher (03:10)

http://youtu.be/uBcR9kPIbg8

CC-BY

Die hier skizzierte Arbeitsweise stammt noch von den Großrechnern aus den Anfängen des Computerzeitalters. Damals war es üblich, beim Start eines Rechners eine Reihe von Programmen (und Daten!) zum Beispiel über Lochkarten einlesen zu lassen. Die Programme liefen dann sequentiell ab, d.h. erst nach Beendigung des ersten Prozesses, startete der zweite Prozess, usw.

Anmerkung

Mit deiner eigenen Erfahrung kannst du abschätzen, wie sehr sich doch das Zusammenspiel von Hardware und Software von den Anfängen bis heute verändert hat.

Anordnung mehrerer Programme im Hauptspeicher

Das folgende Video erläutert, wie mehrere Programme mit ihren Daten im Hauptspeicher angeordnet sind.

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/n3SKQt-yHHg

Med. 18: Speicheraufteilung bei mehreren Programmen im RAM (03:08)
http://youtu.be/n3SKQt-yHHg
CC-BY

Zusammenhängende Speicherbelegung

Die hier gezeigte Speicheraufteilung verdeutlicht, dass ein Programm und seine Daten **zusammenhängend** im Hauptspeicher abgelegt werden. Dieser Umstand vereinfacht viele Dinge, auf die in den nächsten Seiten noch näher eingegangen wird.

Der Grund für diese zusammenhängende Unterbringung im Hauptspeicher ist ganz simpel: es ist die einfachste Variante, bei der auftretende Probleme (auch diese werden in Kürze aufgezeigt) ohne großen Aufwand gelöst werden können.

Anmerkung

Bedenke:
Die hier gezeigten Dinge stammen aus der Zeit der Großrechner. PCs und Laptops wie wir sie heute kennen waren damals noch sehr weit entfernt. Die Entwicklung startete mit diesen einfachen Verfahren, steigerte ihre Komplexität aber im Laufe der Jahrzehnte. Die heute üblichen Verfahren der Speicherbelegung werden später in diesem Modul noch erläutert.

Nötige Vorarbeiten

Man betrachte noch einmal den oben auf dieser Seite beschriebenen Punkt:
"Treffe für jedes Programm die nötigen Vorarbeiten für seine spätere Ausführung."

Genaugenommen ist dieser Punkt etwas schwammig formuliert, denn:

Frage

WER oder **WAS** trifft für jedes Programm die nötigen Vorarbeiten?

Spätestens an dieser Stelle zeigt sich erstmals die *Notwendigkeit einer "Verwaltungseinheit"*, und aus dieser Notwendigkeit heraus entwickelten sich im Laufe der Jahre verschiedene Betriebssysteme.

Notwendigkeit einer Verwaltungseinheit

Wichtig

Wir brauchen ein Betriebssystem!
Und eine der Aufgaben dieses Betriebssystems wird die Verwaltung mehrerer Prozesse sein.

2.3.2.2 Swapping: Aus- und Einlagern von kompletten Prozessen

Nachdem nun bekannt ist, dass es mit Hilfe des Basisregisters möglich ist, mehrere Programme zur gleichen Zeit im Hauptspeicher zu halten, ist weiterhin vorstellbar, dass so viele Programme gleichzeitig in den Hauptspeicher aufgenommen werden sollen, dass der verfügbare Hauptspeicher nicht mehr ausreicht.

Praxissituation

Eine Praxissituation, in der dieser Umstand in früheren Jahren aufgetreten sein könnte, ist:

Beispiel

Kurz vor Feierabend lässt der Operateur eines Großrechners diesen viele Programme mit ihren zugehörigen Daten einlesen. Die Programme sollen anschließend nacheinander ablaufen und ihre Daten verarbeiten. Dies geschieht während der Nacht. Am nächsten Morgen zu Dienstbeginn prüft dann der Operateur die Ergebnisse. Damit der Großrechner möglichst die ganze Nacht ausgelastet ist, werden entsprechend viele Programme und Daten von Lochkarten eingelesen. Diese passen in ihrer Gesamtheit nicht in den Hauptspeicher, einige werden deshalb ausgelagert und erst im Verlaufe der Nacht automatisch wieder eingelagert.

Auch hier kann man fragen:

Frage

WER oder **WAS** lagert Prozesse aus oder ein?

Die Antwort ist wieder:

Wichtig

Wir brauchen ein Betriebssystem!
Und eine der Aufgaben des Betriebssystems wird das Aus- und Einlagern von Prozessen sein.

Das folgende Video erläutert die Zusammenhänge:

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/uzLIsaf-INg

Med. 19: Swapping (02:05)
http://youtu.be/uzLIsaf-INg
CC-BY

Definition: Swapping

Definition

Unter **Swapping** versteht man das Aus- bzw. Einlagern eines **kompletten** Prozesses.

Entscheidend ist hier der Begriff **kompletter Prozess**, also sowohl der Programmtext, als auch **alle** zugehörigen Daten. Dabei sind nicht nur die Daten aus dem Datensegment im Hauptspeicher zu berücksichtigen, sondern insbesondere auch alle zu diesem Prozess gehörigen Daten aus den Registern der CPU.

Aufgabe 1

Aufgabe

Aufg. 36: Zusammenhang mit Basisregister

Im Video ist zu sehen, dass "Prozess C" vor dem Auslagern einen anderen Teil des Hauptspeichers belegt, als nach dem Wiedereinlagern. Erläutere wie dieses in Zusammenhang mit dem Basisregister steht!

Aufgabe 2

Aufgabe

Aufg. 37: Fragmentierter Hauptspeicher

Am Ende des Videos ist ein fragmentierter Hauptspeicher zu erkennen.

- Recherchiere: Was versteht man unter einem *fragmentierten Hauptspeicher*?
- Erläutere: Welche Vor- oder Nachteile hat diese Fragmentierung?

2.3.3 Limitregister zum Speicherschutz

Dass es nun möglich ist, mehrere Prozesse gleichzeitig im Hauptspeicher zu verwalten, bringt nicht nur Vorteile. Es erfordert auch eine Berücksichtigung von Sicherheitsbelangen.

Insbesondere muss sichergestellt werden, dass ein Prozess nicht auf Speicherzellen im Hauptspeicher zugreift, die einem anderen Prozess zugeordnet sind. Dies kann durch Einsatz des Limitregisters erreicht werden.

Definition: Limitregister

Definition

Das **Limitregister** befindet sich auf der CPU. Es enthält als Wert die Größe des **zusammenhängenden** Speicherbereichs des aktiven Prozesses im Hauptspeicher.

Limitregister auf der CPU

Wie aus folgender Abbildung ersichtlich, ist das Limitregister auf der CPU angesiedelt. In vorangegangenen Bildern zur CPU war an dieser Stelle das Register R2 angesiedelt. Man kann sich gerne vorstellen, dass das Register R2 weiterhin auf der CPU existiert, nur eben aus Platzgründen nicht mehr eingezeichnet ist.

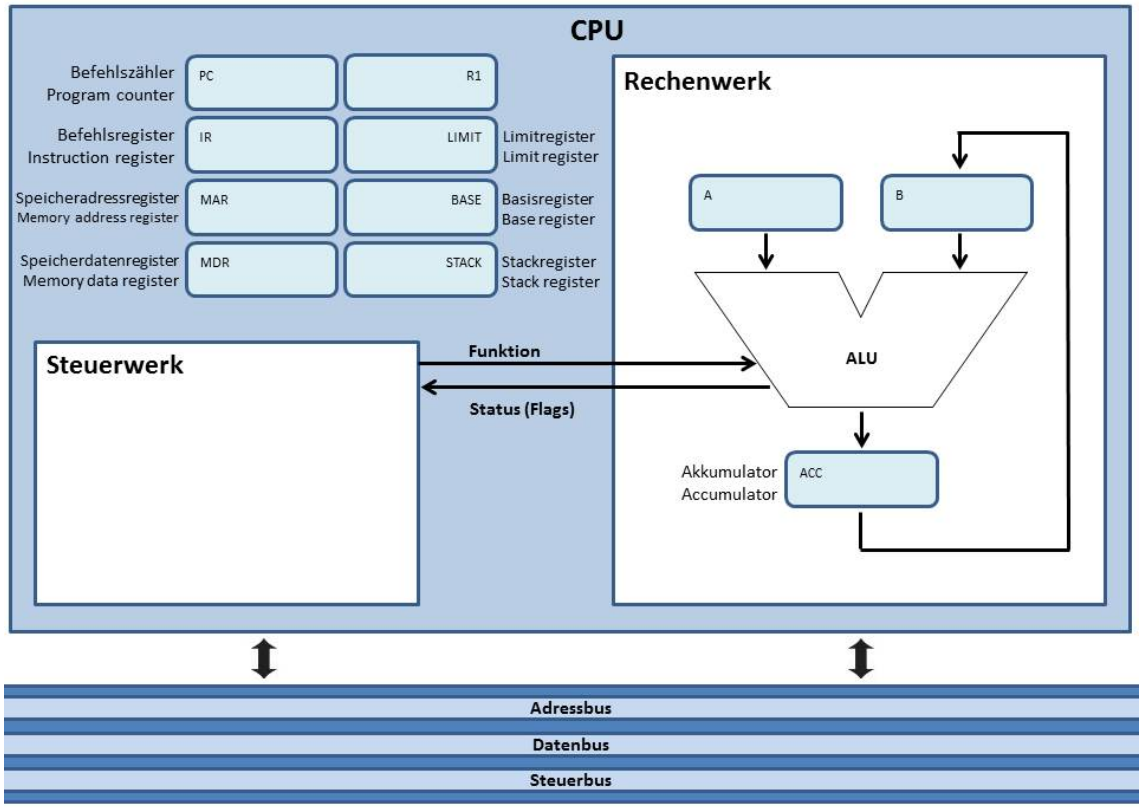


Abb. 37: CPU mit Limitregister
CC-BY

Limitregister im Zusammenhang

Das folgende Video erläutert die Zusammenhänge:

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/2XrQD5q7STQ

Med. 20: Limitregister zum Speicherschutz (02:33)
http://youtu.be/2XrQD5q7STQ
CC-BY

Im Video wird erläutert, dass das Steuerwerk mit Hilfe des Limitregisters in die Lage versetzt wird, Zugriffe auf fremde Speicherbereiche zu bemerken, bevor sie vollzogen werden. Als Reaktion darauf erzeugt das Steuerwerk einen Interrupt, der die Ausführung des aktuellen Prozesses abbricht.

Auf die genauen Abläufe beim Auftreten von Interrupts geht das Kapitel Interrupt-Controller näher ein.

Aufgabe 1

Aufgabe

Aufg. 38: Zugriff auf freien Speicherbereich

Im Video wird erläutert, wie mit Hilfe des Limitregisters verhindert wird, dass Prozess A auf einen Speicherbereich zugreift, der Prozess B zugeordnet ist. Prozess A wird bei auftreten dieser Speicherschutzverletzung beendet.

Wie sollte reagiert werden, wenn Prozess A stattdessen auf den noch freien Bereich im Hauptspeicher zugreift? Den Bereich also, der keinem Prozess zugeordnet ist? Begründe deine Antwort kurz.

Aufgabe 2

Aufgabe

Aufg. 39: Speicherbereich eines Prozesses vergrößern

Mit Hilfe des Basis- und des Limitregisters kann für jeden Prozess festgelegt werden, welchen Bereich des Hauptspeichers dieser nutzen darf. Ein Nachteil ist dabei, dass bereits bei der Erzeugung des Prozesses die maximale Größe dieses Speicherbereichs festgelegt werden muss.

Beschreibe zwei Möglichkeiten, wie zur Laufzeit eines Prozesses der ihm zugeordnete Speicherbereich vergrößert werden kann!

- Die eine Möglichkeit sollte ohne Swapping auskommen.
- Die andere Möglichkeit darf Swapping nutzen!

- Berücksichtige bei deinen Möglichkeiten:
  - Wie wirkt es sich aus, wenn sich mehrere Prozesse den Hauptspeicher teilen müssen?
  - Welche Probleme können bei der Vergrößerung des Speicherbereichs auftreten?
  - Wie können diese Probleme gelöst werden?
- Was könnte(n) der Grund (die Gründe) für die Vergrößerung des Speicherbereichs sein?
- Wer oder was regelt die Durchführung der Vergrößerung des Speicherbereichs?

2.3.4 Interrupt-Controller

Der **Interrupt-Controller** ist ein weiterer Hardware-Baustein in der Architektur eines Computers. Seine Aufgabe besteht darin, Interrupt-Signale von verschiedenen Komponenten des Rechners entgegen zu nehmen, und die CPU über das Vorliegen von einem (oder mehreren) Interrupts zu informieren.

Die CPU ist dann für die Abarbeitung des Interrupts zuständig. Dies geschieht, indem eine sogenannte Interruptbehandlungsroutine aufgerufen wird.

Definition: Interruptbehandlungsroutine

Definition

Unter einer **Interruptbehandlungsroutine** (ISR, Interrupt Service Routine) versteht man eine Reihe von Anweisungen, die einem bestimmten Interrupt zugeordnet ist und deren Anweisungen auf einer CPU ausgeführt werden können.

Interrupt bedeutet Unterbrechung

Die deutsche Übersetzung des englischen Begriffs "Interrupt" ist "Unterbrechung". Und genau diese Unterbrechung eines laufenden Prozesses passiert in dem Moment, in dem sich die CPU entschließt, einen vom Interrupt-Controller angezeigten Interrupt zu bearbeiten.

Den vereinfachten Ablauf zeigt das folgende Video:

An dieser Stelle befindet sich online ein YouTube-Video.

Video

https://youtu.be/ML9mwQ5TyzI

Med. 21: Interrupt-Controller (01:30)
http://youtu.be/ML9mwQ5TyzI
CC-BY

Im Video ist erkennbar, dass der Interrupt-Controller als eigenständige Hardware-Komponente über den Systembus mit anderen Komponenten und insbesondere mit der CPU kommuniziert.

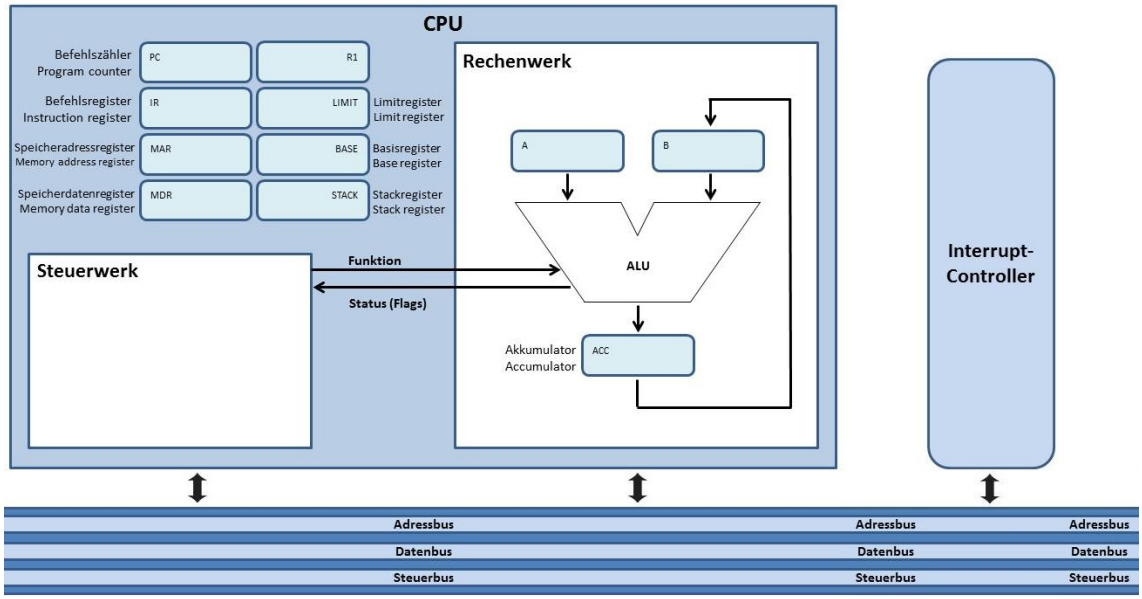


Abb. 38: CPU mit Interrupt-Controller
CC-BY

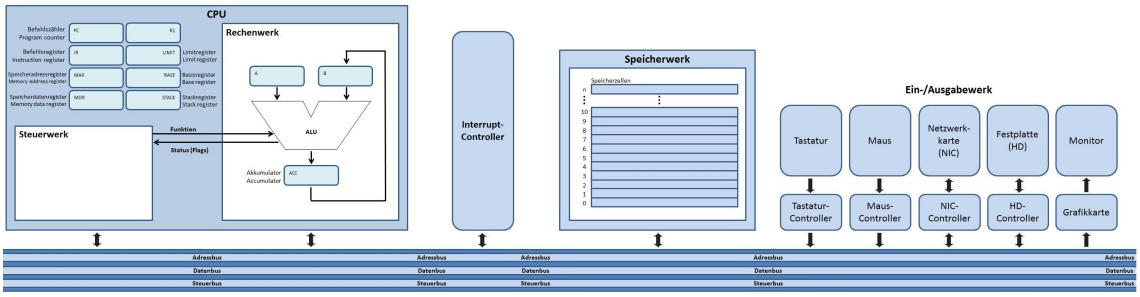


Abb. 39: Interrupt-Controller im Gesamtbild
CC-BY

Hinweis

Weiterführende Literatur

Wuest 2011 erläutert in den Kapiteln 8.1 (Interrupts) und 8.2 (Ausnahmen) weitere Hintergründe zu den genannten Themen. Die Lektüre dieser Quelle sei ausdrücklich empfohlen.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Hinweis

Weiterführende Literatur

Mandl 2013 erläutert in Kapitel 3.1 (Interrupts) weitere Hintergründe zum Thema. Die Lektüre dieser Quelle sei ausdrücklich empfohlen.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Es ist an dieser Stelle wichtig zu verstehen was genau passiert, wenn ein Interrupt auftritt und von der CPU bearbeitet wird. Die Beschreibung oben auf dieser Seite ist noch recht allgemein gehalten, weshalb jetzt einige entscheidende Details betrachtet werden sollen.

Keine negative Beeinträchtigung

Ein entscheidendes Kriterium bei der Abarbeitung eines Interrupts durch die CPU ist, dass der durch den Interrupt unterbrochene Prozess später **ohne negative Beeinträchtigung** weiter ausgeführt werden kann. Das vom Prozess erarbeitete Ergebnis darf sich nicht unterscheiden, egal, ob während der Abarbeitung ein (oder mehrere) Interrupt(s) aufträt(en), oder nicht. Erreicht wird dieses durch die Einhaltung der Bedingungen einer präzisen Unterbrechung.

Definition: Präzise Unterbrechung

Definition

Einen Interrupt nennt man eine **präzise Unterbrechung** (precise interrupt), falls alle der folgenden Bedingungen erfüllt sind:

1. Der Programmzähler des unterbrochenen Prozesses wird an einer bekannten Stelle gesichert.
2. **Alle** Befehle des unterbrochenen Prozesses, die **vor** dem Befehl ausgeführt werden müssen, auf den der Programmzähler zeigt, sind vollständig abgearbeitet.
3. **Kein** Befehl des unterbrochenen Prozesses, der **nach** dem Befehl ausgeführt werden muss, auf den der Programmzähler zeigt, ist bereits abgearbeitet.
4. Der Ausführungszustand des Befehls des unterbrochenen Prozesses, auf den der Programmzähler zeigt, ist bekannt.

Definition: Unpräzise Unterbrechung

Definition

Einen Interrupt nennt man eine **unpräzise Unterbrechung** (Imprecise interrupt), falls mindestens eine der für einen präzisen Interrupt genannten Bedingungen nicht erfüllt ist.

Was bei einem Interrupt passiert

Das folgende Video zeigt ein Beispiel, anhand dessen nachvollzogen werden kann, was genau bei einem Interrupt passiert, und ob die Bedingungen für eine präzise Unterbrechung erfüllt werden.

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/_wVNpUW3kdM

Med. 22: Was passiert bei einem Interrupt? (02:14)

http://youtu.be/_wVNpUW3kdM

CC-BY

Mit Hilfe dieses Videos ist es nun möglich, die folgenden Aufgaben zu bearbeiten:

Aufgabe 1

Aufgabe

Aufg. 40: Prüfe die Bedingungen!

Prüfe anhand des Videos für jede der vier Bedingungen einer präzisen Unterbrechung, ob diese **tatsächlich erfüllt** ist!

- Ist Bedingung 1 erfüllt? Erläutere warum!
- Ist Bedingung 2 erfüllt? Erläutere warum!
- Ist Bedingung 3 erfüllt? Erläutere warum!
- Ist Bedingung 4 erfüllt? Erläutere warum!

Aufgabe 2

Aufgabe

Aufg. 41: Schätze die Folgen ab!

Beschreibe ein Szenario, bei dem jeweils eine der vier Bedingungen einer präzisen Unterbrechung **nicht erfüllt** ist.

- Was sind die Folgen, wenn Bedingung 1 nicht erfüllt ist?

- Was sind die Folgen, wenn Bedingung 2 nicht erfüllt ist?
- Was sind die Folgen, wenn Bedingung 3 nicht erfüllt ist?
- Was sind die Folgen, wenn Bedingung 4 nicht erfüllt ist?

Aufgabe 3

Aufgabe

Aufg. 42: Wer wenn nicht das Steuerwerk?

Im Video wird gezeigt, wie der aktuelle Wert des Befehlszählers bei einer Unterbrechung in einem anderen Register gesichert wird. Alle weiteren Register oder die Statusflags werden hingegen nicht gesichert.

- Wenn das Steuerwerk nicht für die Sicherung dieser Werte zuständig ist, wer oder was ist es dann?

  (Auf verschiedenen anderen Seiten dieses Moduls ist die Antwort auf *Wer-oder-was-Fragen* stets *"Das Betriebssystem"*, aber diese Antwort ist an dieser Stelle nicht zugelassen, sie ist viel zu allgemein.)
- Wo können diese Werte gesichert werden?
- Welche Vor- und/oder Nachteile hat diese Verfahrensweise?

Aufgabe 4

Aufgabe

Aufg. 43: Vier Bit

Im Video werden in den Registern PC und IR jeweils Werte abgelegt, die aus vier Bit bestehen.

- Wie realistisch sind diese *vier Bit*?
- Wieviele Bit erwartest du hier bei deinem eigenen PC oder Laptop?

2.3.4.1 Gründe für eine Interrupt-Auslösung

Es gibt eine ganze Reihe an Gründen für eine Interrupt-Auslösung. Näher betrachtet werden hier lediglich die Folgenden:

- Auslösung aufgrund einer Speicherschutzverletzung.

- Auslösung durch einen Hardware-Taktgeber zur quasi-gleichzeitigen Ausführung mehrerer Prozesse.
- Auslösung durch ein E/A-Gerät während der Kommunikation zwischen CPU und E/A-Gerät.

In den folgenden Unterkapiteln werden diese Punkte näher erläutert.

2.3.4.2 Speicherschutzverletzung

Stellt das Steuerwerk während der Abarbeitung eines Prozesses fest, dass der aktive Prozess auf einen Speicherbereich im Hauptspeicher zugreifen möchte, der ihm selbst nicht zugeordnet ist, so handelt es sich um eine Speicherschutzverletzung, die durch einen Interrupt angezeigt wird.

Reaktion auf Speicherschutzverletzung

Die einfachste Reaktion auf eine Speicherschutzverletzung ist, dass der die Speicherschutzverletzung verursachende Prozess beendet wird, seine zu diesem Zeitpunkt nicht dauerhaft gespeicherten Daten gehen unwiederbringlich verloren.

Aufgabe 1

Aufgabe

Aufg. 44: Weiter laufen lassen?

Wäre es möglich, den verursachenden Prozess trotz Speicherschutzverletzung weiter laufen zu lassen? Welche Konsequenzen hätte dies? Erläutere Vor- und Nachteile!

Aufgabe 2

Aufgabe

Aufg. 45: Ausnahme und Interrupt

Wuest 2011 spricht in Kapitel 8.2 beim Auftreten eines "unerlaubten Speicherzugriffs" von einer **Ausnahme** (**Exception** oder **Trap**) und damit nicht von einem *Interrupt*.

1. Woran unterscheidet *Wuest 2011* zwischen *Ausnahme* und *Interrupt*?
2. Welche Situationen führen nach *Wuest 2011* zu einem Interrupt?
3. Und welche Situationen führen nach *Wuest 2011* zu einer Ausnahme?

Studierende sind oftmals berechtigt, eine PDF-Version des genannten Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

2.3.4.3 Quasi-gleichzeitige Ausführung mehrerer Prozesse

Die quasi-gleichzeitige Ausführung mehrerer Prozesse stellt einen besonderen Meilenstein dar zwischen dem bisher erarbeiteten Bild der Ausführung mehrerer Prozesse im Hauptspeicher und dem, was ein normaler Anwender heute von seinem PC oder Laptop gewohnt ist.

Der bisherige Stand

Zunächst sei an den bisher erarbeiteten technischen Stand erinnert:

Auf einer (hier betrachteten) ganz einfachen CPU kann immer nur **genau ein Prozess zur Zeit** aktiv sein. Eine tatsächlich parallele Ausführung mehrerer Prozesse ist deshalb technisch bedingt nicht möglich.

Es kann aber zu einer "quasi-gleichzeitigen" Ausführung mehrerer Prozesse kommen, indem sich die betreffenden Prozesse in kleinen Zeiteinheiten auf der CPU abwechseln. Sind die "kleinen Zeiteinheiten" hierbei genügend klein gewählt, so erhält ein Anwender vor dem System den Eindruck, als ob alle Prozesse parallel ablaufen.

Aus der Praxis

Du kennst genau dieses Verhalten aus deiner täglichen Arbeit mit deinem PC oder Laptop. Alle gestarteten Programme wie beispielsweise Browser, E-Mail-Client, Textverarbeitung, etc. scheinen parallel zu laufen.

Offene Fragen

Es ergeben sich einige offene Fragen:

Frage

1. **WER** oder **WAS** bestimmt, wann es Zeit ist, einen Prozess zu unterbrechen und die CPU einem anderen Prozess zuzusprechen?

2. **WER** oder **WAS** bestimmt, welcher Prozess als nächstes die CPU bekommt?

Frage 2 lässt sich ganz einfach mit folgender Forderung beantworten:

Wichtig

Wir brauchen ein Betriebssystem!
Und eine der Aufgaben dieses Betriebssystems wird die Auswahl des jeweils nächsten Prozesses für die CPU sein.

Aber bei Frage 1 wird es schwieriger. Hier kann das Betriebssystem nicht helfen, denn das Betriebssystem ist Software, während seiner Ausführung also nichts weiter als ein Prozess.

Wichtige Erkenntnis

Wichtig

Dies ist eine sehr wichtige Erkenntnis für das Zusammenspiel von Hardware, Betriebssystem und sonstiger Software! Ein Betriebssystem ist nichts anderes als Software. Sobald das Betriebssystem in den Hauptspeicher geladen und seine Ausführung gestartet wird, ist das Betriebssystem ein Prozess. Und der Betriebssystem-Prozess wechselt sich fortan mit den weiteren Prozessen im Hauptspeicher auf der CPU ab.

Wenn jetzt Prozess A auf der CPU läuft, dann muss es ein Hardware-gesteuertes Ereignis geben, welches dafür sorgt, dass der laufende Prozess A unterbrochen wird und das Betriebssystem (als weiterer Prozess) die CPU übernimmt.

Hardware-Taktgeber

Zu diesem Zweck gibt es einen **Hardware-Taktgeber**, der in sehr kleinen zeitlichen Abständen einen Interrupt auslöst. Bei der Abarbeitung dieses Interrupts wird der laufende Prozess A unterbrochen, und die zugehörige Interruptbehandlungsroutine ausgeführt.

Die Antwort auf Frage 1 ist somit:

Wichtig

Die Hardware unseres Rechners muss um einen Taktgeber erweitert werden!
Und dieser Taktgeber sorgt unabhängig von allen Prozessen und in kleinen zeitlichen Abständen für ein Signal, welches einen Interrupt auslöst. Die zugehörige Interruptbehandlungsroutine kann dann entscheiden, ob es Zeit für einen Prozesswechsel auf der CPU ist.

Die Interruptbehandlungsroutine kann nun Bestandteil des Betriebssystems sein, wodurch der Interrupt letztlich dafür sorgt, dass das Betriebssystem zur Ausführung auf der CPU kommt. Wird nun entschieden, dass es Zeit für einen Prozesswechsel auf der CPU ist, so kann ein anderer Teil des Betriebssystems aufgerufen werden, der den nächsten Prozess auswählt und ihm die CPU übergibt.

Aufgabe 1

Aufgabe

Aufg. 46: Integration des Taktgebers

Du erinnerst dich an die folgende Abbildung:

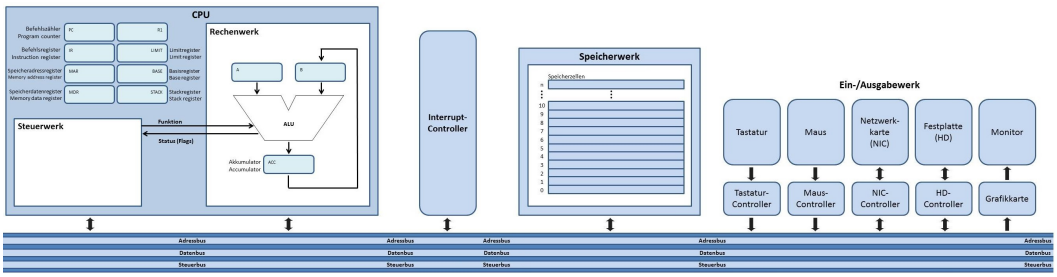


Abb. 40: Abbildung zu Aufgabe 1
CC-BY

Wie integrierst du den Hardware-Taktgeber in dieses Bild?

Aufgabe 2

Aufgabe

Aufg. 47: Betriebssystem und Prozesse im Wechsel

Nach dem Start eines Rechners mit integriertem Hardware-Taktgeber werden insgesamt drei Prozesse gestartet:

1. Prozess BS: Das Betriebssystem.
2. Prozess X: Eine beliebige Anwendung.
3. Prozess Y: Eine weitere beliebige Anwendung.

Prozess BS beginnt ab Speicherstelle 0 im Hauptspeicher, damit wird das Betriebssystem zuerst gestartet.

Erläutere was passiert, wenn Prozess BS sich nun um die quasi-gleichzeitige Ausführung aller Prozesse auf der CPU kümmert. In welcher Weise wechseln sich BS, X und Y auf der CPU ab? Wodurch wird dieser Wechsel veranlasst?

2.3.4.4 Kommunikation mit E/A-Geräten

Im bisherigen Verlauf dieses Moduls sind bereits viele Beispiele gezeigt worden, bei denen das Steuerwerk der CPU mit Registern oder dem Hauptspeicher kommuniziert. Da ein Computer aber noch weitere Komponenten besitzt, muss auch die Kommunikation der CPU mit den sogenannten Peripheriegeräten möglich sein.

Peripheriegeräte sind beispielsweise:

- Festplatte

- Monitor
- Maus
- Tastatur
- etc.

Genaugenommen kommuniziert die CPU dann nicht direkt mit diesen Komponenten, sondern mit einem Controller, der speziell für die betreffende Komponente zuständig ist.

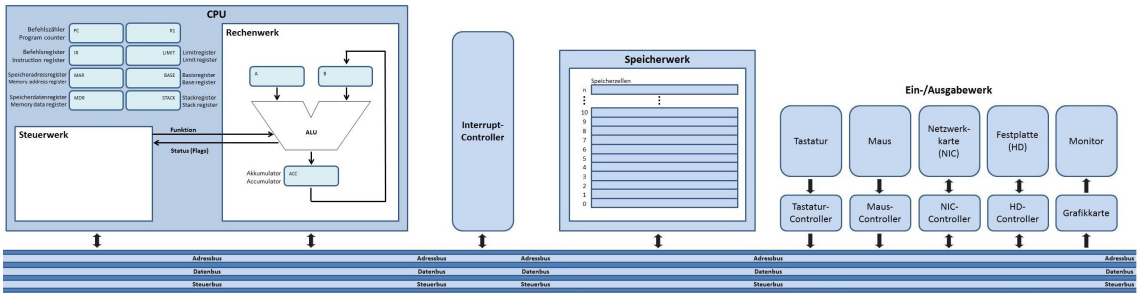


Abb. 41: CPU, Interrupt-Controller, Speicherwerk und weitere Controller mit ihren Komponenten CC-BY

So geht es weiter:

Gliederung

2.3.4.4 Kommunikation mit E/A-Geräten
2.3.4.4.1 Allgemeiner Aufbau eines Controllers
2.3.4.4.2 Zeit und Kosten machen den Unterschied
2.3.4.4.3 Datentransfer und Interrupts

2.3.4.4.1 Allgemeiner Aufbau eines Controllers

Genau wie die CPU besitzt auch der Controller eines Peripheriegeräts verschiedene Register. Üblich sind:

- **Steuerregister**
  Hier kann über das Bussystem ein Steuerbefehl an den Controller übergeben werden (ähnlich dem Befehlsregister auf der CPU).
- **Datenregister**
  Hier kann ein Datenwort hinterlegt, oder ein vom Peripheriegerät über den Controller bereitgestelltes Datenwort ausgelesen werden.
- **Zustandsregister**
  Hier hinterlegt der Controller jeweils Werte, die über den aktuellen Zustand des Controllers (oder Peripheriegeräts) Auskunft geben.

Anmerkung

In dem (oder den) Zustandsregister(n) können verschiedene Zustände angezeigt werden, beispielsweise

- **Ready**: Controller ist bereit für den nächsten Steuerbefehl.
- **Busy**: Controller ist noch mit der Ausführung des aktuellen Steuerbefehls beschäftigt.
- **Read Error**: Das angeforderte Datenwort kann nicht vom Peripheriegerät gelesen werden.
- **Write Error**: Das im Datenregister befindliche Datenwort kann nicht auf dem Peripheriegerät gespeichert werden.
- **Out of paper**: Der an diesen Controller angeschlossene Drucker hat kein Papier mehr.
- etc.

Die folgende Abbildung zeigt den allgemeinen Aufbau eines Controllers. Dabei ist von jeder Registerart nur ein einziges eingezeichnet, es ist jedoch auch denkbar, dass der Controller über mehrere Register je Art verfügt.

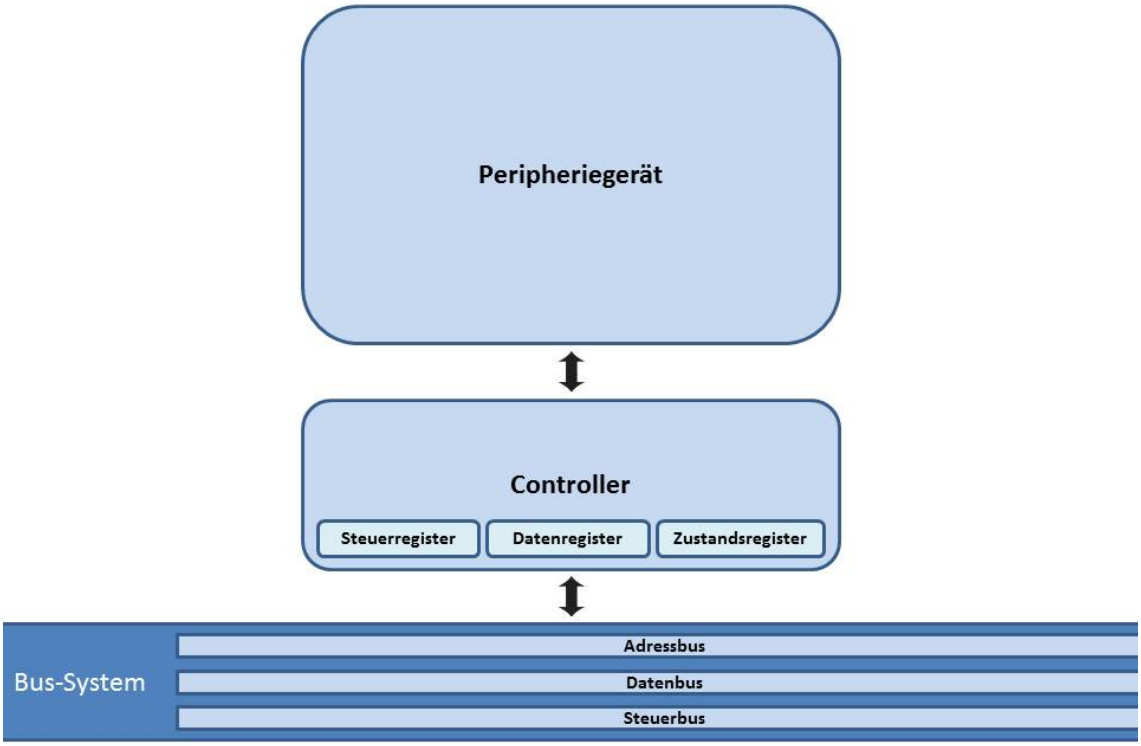


Abb. 42: Allgemeiner Aufbau eines Controllers
CC-BY

Die bislang betrachtete Menge an Peripheriegeräten verfügt jeweils über einen eigenen Controller, wobei jeder Controller mit dem entsprechenden Registersatz ausgestattet ist:

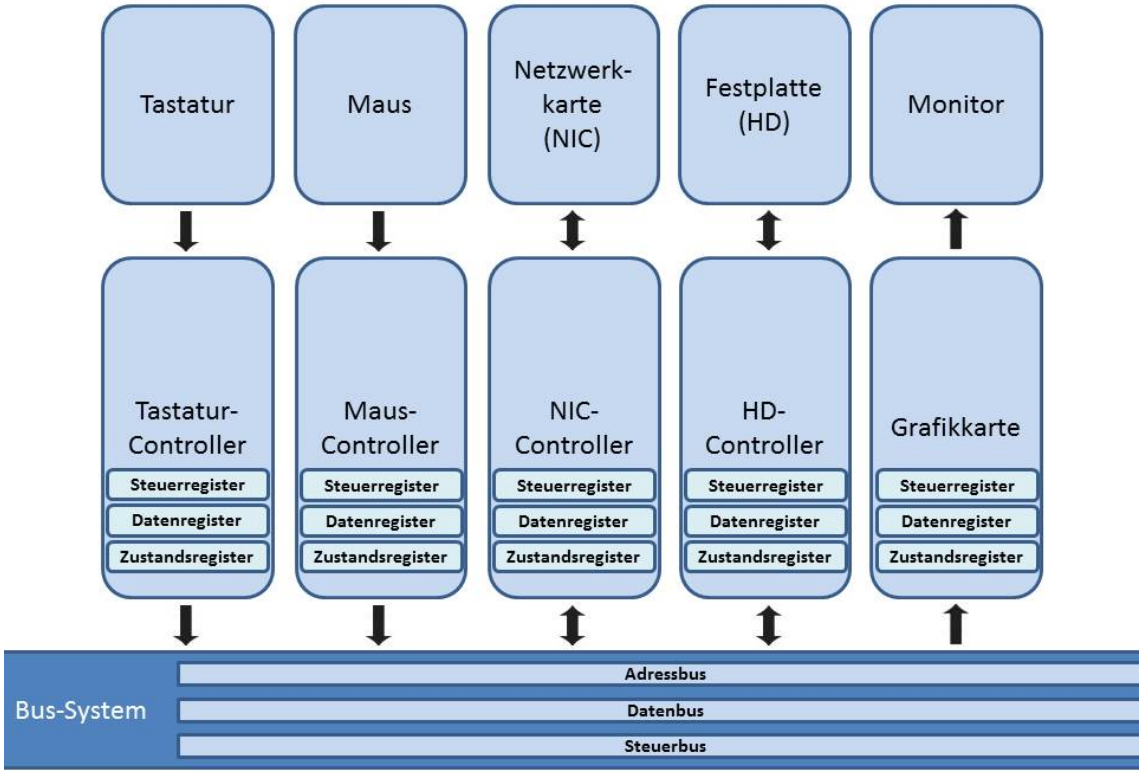


Abb. 43: Verschiedene Controller mit eigenen Registersätzen
CC-BY

Aufgabe 1

Aufgabe

Aufg. 48: Lesen und/oder schreiben?

Auf das Datenregister eines Controllers kann die CPU sowohl lesend, als auch schreibend zugreifen. Wie verhält es sich aber beim Steuer- und beim Zustandsregister? Ist der Zugriff hier *nur lesend*, *nur schreibend* oder *lesend und schreibend* möglich?

2.3.4.4.2 Zeit und Kosten machen den Unterschied

Zwei entscheidende Unterschiede bei der Kommunikation zwischen dem Steuerwerk und Registern, Hauptspeicher sowie den Controllern der weiteren Komponenten bestehen einerseits in der dafür benötigten Zeit und andererseits in den Kosten, die eine technische Realisierung der betreffenden Komponenten verursacht.

Die Kommunikation zwischen Steuerwerk und Registern ist mit Abstand am schnellsten möglich. Beides ist auf der CPU angesiedelt, die Wege sind kurz und effizient ausgebaut. Nachteilig ist lediglich, dass alle Register der CPU zusammen nur recht wenig Speicherplatz bieten. Dies liegt an entsprechend hohen Kosten, welche die Integration (vieler) weiterer Register auf der CPU nach sich ziehen würde.

Anmerkung

Wunschvorstellung

Wäre es nicht ideal, wenn alle Programme und Daten, die du normalerweise auf deiner Festplatte gespeichert hast, direkt in Registern der CPU abgelegt wären? Du bräuchtest dann weder Festplatte noch Hauptspeicher und dein System würde konkurrenzlos schnell arbeiten!

Du kannst dir ausmalen, dass das technisch nicht so einfach möglich ist, oder?

Die Kommunikation zwischen Steuerwerk und Hauptspeicher ist - verglichen mit den Registern - bereits deutlich langsamer, jedoch immer noch recht schnell. Der Hauptspeicher ist speziell für eine schnelle Arbeitsweise konzipiert, er bietet auch deutlich mehr Platz als die Menge der CPU-Register. Nachteilig ist hier, dass der Inhalt des Hauptspeichers verloren geht, sobald der betreffende Computer ausgeschaltet wird.

Abhilfe schafft bei diesem Nachteil beispielsweise die Festplatte. Sie bietet üblicherweise deutlich mehr Speicherplatz als der Hauptspeicher und kann ihre Daten zudem dauerhaft (also auch über ein Ausschalten und einen Neustart des betreffenden Rechners hinaus) speichern. Leider ist die Kommunikation zwischen CPU und Festplatten-Controller erheblich langsamer als zwischen CPU und Hauptspeicher. Dafür liegen die Kosten für eine Festplatte (in Bezug zur Speicherkapazität) deutlich unter den Kosten für Hauptspeicher.

Anmerkung

Eine Festplatte kostet pro Gigabyte Speicherkapazität deutlich weniger, als ein Gigabyte Hauptspeicher. Dafür arbeitet der Hauptspeicher deutlich schneller.

Anmerkung

Tanenbaum 2009 nennt folgende näherungsweisen Vergleichswerte für die Zugriffe auf Daten:

- Die CPU kann 10 x schneller auf ein Register zugreifen, als auf den Arbeitsspeicher.

- Die CPU kann 1 000 x schneller auf den Arbeitsspeicher zugreifen, als auf eine Festplatte.
- Ein Registerzugriff ist damit 10 000 x schneller, als ein Festplattenzugriff.

Auch für die Kommunikation zwischen der CPU und allen weiteren Controllern der sonstigen Komponenten gilt: Es ist langsam.

2.3.4.4.3 Datentransfer und Interrupts

Das folgende Video erläutert die Bedeutung von Interrupts bei der Kommunikation zwischen CPU und Ein-/ Ausgabegeräten. Beispielhaft wird die Festplatte als E/A-Gerät betrachtet.

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/nOEW4I_QX2c

Med. 23: Ein- und Ausgabe mit Festplatte und Interrupts (04:00)
http://youtu.be/nOEW4I_QX2c
CC-BY

Im Video wird deutlich, dass **für jedes einzelne Datenwort** folgende Schritte erforderlich sind:

- Die CPU sendet die Adresse des gewünschten Datenwortes an den Festplatten-Controller.
- Der Festplatten-Controller besorgt das Datenwort von der Festplatte und stellt es in seinem Datenregister zur Verfügung.
- Der Festplatten-Controller sendet einen Interrupt zum Interrupt-Controller.
- Der Interrupt-Controller nimmt den Interrupt entgegen und verwaltet ihn.
- Der Interrupt-Controller informiert die CPU über den Interrupt des Festplatten-Controllers.
- Die CPU startet die zugehörige Interruptbehandlungsroutine.
- Die Behandlungsroutine kopiert das Datenwort in ein Register auf der CPU.
- Die Behandlungsroutine sendet das Datenwort von der CPU zum Hauptspeicher.
- Der Hauptspeicher legt das Datenwort in der gewünschten Speicherzelle ab.
- Die Interruptbehandlungsroutine informiert den Interrupt-Controller darüber, dass der Interrupt jetzt fertig behandelt ist.

Ganz schön viel Aufwand, oder? (Und dabei ist diese Auflistung noch nicht einmal ganz komplett, aber damit beschäftigst du dich gleich noch in den Aufgaben weiter unten ;-)

Für eine Handvoll Bits...

Wichtig

Es ist an dieser Stelle wichtig zu erkennen, was für ein hoher Aufwand betrieben wird, nur um ein einzelnes Datenwort von der Festplatte in den Hauptspeicher zu kopieren.

Wie hoch der Aufwand tatsächlich ist, wird vermutlich erst mit der Beantwortung der folgenden Fragen deutlich:

Frage

Wie groß ist ein einzelnes Datenwort eigentlich?

Antwort (Falls du nicht selbst drauf kommst)

Bist du echt nicht selbst darauf gekommen?

Ein einzelnes Datenwort hat die Größe der Registerbreite des Datenregisters auf dem Festplatten-Controller, oder alternativ die Größe der Busbreite des Datenbusses, über das es übertragen wird. Mit hoher Wahrscheinlichkeit haben Register und Bus dieselbe Breite, je nach Rechner zum Beispiel 32 oder 64 Bit (und auf neueren Systemen irgendwann mal 128 Bit).

Frage

Wie groß ist heute eine übliche Datei, die beispielsweise ein Anwendungsprogramm enthält?

Antwort (Falls du nicht selbst drauf kommst)

Bist du echt nicht selbst darauf gekommen?

Eine übliche Datei kann schnell mehrere Megabyte groß sein. Sie enthält damit ganz schön viele Datenwörter...

Hinweis

Weiterführende Literatur

Wuest 2011 erläutert in Kapitel 5 (Ein- und Ausgabe) weitere Hintergründe zum Thema. Die Lektüre dieser Quelle sei ausdrücklich empfohlen.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Aufgabe 1

Aufgabe

Aufg. 49: Alternative zum Warten?

Im Video wird gezeigt, wie die CPU wartet, und wartet, und wartet. Das ist natürlich keine gute Idee im Hinblick auf die Geschwindigkeit des Gesamtsystems.

- Was könnte die CPU alternativ tun, während sie auf den Interrupt wartet?
- Eigentlich ist das Betriebssystem gefordert, hier andere Tätigkeiten zu veranlassen. Welche fallen dir ein?

Aufgabe 2

Aufgabe

Aufg. 50: Interrupt und die Register

Im Video wird darauf verzichtet zu zeigen, wie sich Werte beispielsweise in den Registern PC oder IR (usw.) ändern, und welche Werte davon über das Bussystem übertragen werden.

- Diskutiere dieses in deiner Lerngruppe!
- Erklärt euch gegenseitig das Zusammenspiel aller Komponenten und Register.
- Findest du auch ein Beispiel, welches die ALU einbezieht?
- Was passiert mit PC und IR bei einem Interrupt?
- Wie werden diese Register beim Abarbeiten der Interruptbehandlungsroutine belegt?
- Was passiert mit diesen Registern am Ende der Abarbeitung der Interruptbehandlungsroutine (wenn im Video das ACK-IRQ-Signal zum Interrupt-Controller gesendet wird)?

Aufgabe 3

Aufgabe

Aufg. 51: Zwei Bit?

Das Zustandsregister eines Controllers kann (u.a.) ein READY anzeigen, im Video ist das zu sehen. Es reicht ein Bit, um den Zustand READY zu kennzeichnen. Ist dieses Bit gleich 1, so gilt: der Controller ist READY für den nächsten Steuerbefehl. Gleiches gilt für den Zustand BUSY. Auch hierfür reicht ein Bit.

Für READY und BUSY werden im Zustandsregister demnach zwei Bit benötigt.

- Kannst du dem zustimmen?

- Oder gibt es eine sinnvolle Alternative?
- Welche Alternative fällt dir ein?

Aufgabe 4

Aufgabe

Aufg. 52: Mehrere Register je Typ

Im Text dieser Seite wird erwähnt, dass die im Bild gezeigten Controller nur ein Register je Typ enthalten, dass es aber auch möglich ist, dass ein Controller mehrere Register eines Typs enthält.

- Entwickle ein Szenario, in dem es sinnvoll mehrere Register eines Typs gibt!

Aufgabe 5

Aufgabe

Aufg. 53: Mal wieder das Betriebssystem

Kombiniere das Wissen aus dem Video mal mit deinen Ideen, was die CPU statt warten so tun könnte, und mit dem Wissen aus dem Kapitel Quasi-gleichzeitige Ausführung mehrerer Prozesse. Es könnte dann vorkommen, dass mehrere Prozesse "quasi-gleichzeitig" auf ein E-/A-Gerät (wie z.B. die Festplatte) zugreifen möchten.

WER oder **WAS** sorgt dann dafür, dass hierbei keine Kollision auf dem Controller des E-/A-Geräts passiert?

Bitte entschuldige, dass die Antwort gleich unterhalb dieser Frage steht ;-)

Wichtig

Wir brauchen ein Betriebssystem!

Und eine der Aufgaben des Betriebssystems wird die Verwaltung der Ein- und Ausgabegeräte sein.

Speziell beim Thema "Datei auf der Festplatte" im Video ergibt sich eine weitere Fragestellung, auf die im Video nicht näher eingegangen wird.

WER oder **WAS** sorgt dafür, dass Verzeichnisse und Dateien auf einem Datenspeicher (Festplatte, CD, DVD, USB-Stick, etc.) angelegt, wiedergefunden, gelesen, bearbeitet oder gelöscht werden können?

Wichtig

Wir brauchen ein Betriebssystem!

Und eine der Aufgaben des Betriebssystems wird die Datei- und Verzeichnisverwaltung auf Datenträgern sein.

2.3.5 DMA-Controller

Ein **DMA-Controller** ist eine Hardware-Komponente, die zum Ziel hat, die Geschwindigkeit des Gesamtsystems zu erhöhen. Erreicht werden soll dies u.a. durch eine Reduzierung der Anzahl an Interrupts, die bei der Kommunikation zwischen der CPU und den E/A-Geräten ausgelöst werden.

Wichtige Voraussetzung!

Wichtig

Du erinnerst dich doch noch an den **enormen Aufwand**, der betrieben wurde, um ein einzelnes Datenwort von der Festplatte, über die CPU in den Hauptspeicher zu kopieren, oder?

Und daran, dass **für jedes einzelne Datenwort ein Interrupt** ausgelöst wurde, oder?

Falls nicht: Lies dir erst das Kapitel Kommunikation mit E/A-Geräten noch einmal durch, und schau dir das Video an!

Mit der Bearbeitung der folgenden Aufgabe 1 reflektierst du noch einmal diesen wichtigen Sachverhalt.

Aufgabe 1

Aufgabe

Aufg. 54: Datentransfer (noch) ohne DMA

Erläutere anhand der folgenden (vereinfachten) Abbildung, wie der Transfer von Daten zwischen Festplatte und RAM abläuft. Wann treten dabei Interrupts auf?

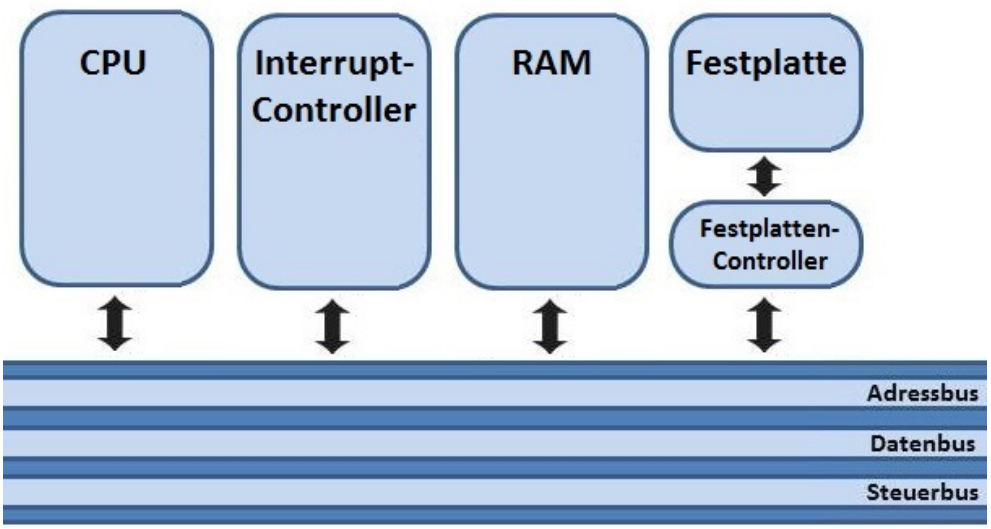


Abb. 44: Vereinfachter Rechner, noch ohne DMA-Controller
CC-BY

Direct Memory Access

Die Abkürzung DMA steht für **Direct Memory Access**, auf Deutsch also: **direkter Speicherzugriff**, und beschreibt die Möglichkeit, Datenwörter direkt zwischen dem Hauptspeicher und einem DMA-fähigen Peripheriegerät auszutauschen, ohne dass daran die CPU beteiligt ist.

Die CPU wird also durch einen vorhandenen DMA-Controller entlastet. Sie muss lediglich zu Beginn und am Ende der Übertragung eingreifen.

Allgemeiner Ablauf

Der allgemeine Ablauf des DMA-Verfahrens ist wie folgt:

1. Daten sollen zwischen dem Hauptspeicher und einem weiteren DMA-fähigen Peripheriegerät übertragen werden.
2. Die CPU versorgt den DMA-Controller mit allen notwendigen Informationen:
   → Beteiligte Komponenten
   → Start- und Zieladresse
   → Anzahl zu übertragener Datenwörter
   → Übertragungsrichtung
3. Die CPU erteilt dem DMA-Controller die Erlaubnis, mit der Datenübertragung zu beginnen.
4. Der DMA-Controller steuert die Übertragung aller Datenwörter zwischen den beteiligten Betriebsmitteln. Die Übertragung geschieht über das Bus-System.

5. Sobald alle Daten übertragen sind, informiert der DMA-Controller mit einem Interrupt über den Abschluss des Datentransfers.
6. Der Interrupt-Controller empfängt den Interrupt und leitet ihn an die CPU weiter.
7. Auf der CPU wird die zugehörige Interruptbehandlungsroutine ausgeführt.
8. Die Datenübertragung per DMA ist abgeschlossen.

Reduzierung der Anzahl der Interrupts

An der vorangegangenen Auflistung ist erkennbar, dass bei Einsatz des DMA-Controllers die Anzahl der Interrupts deutlich reduziert wird. Die CPU muss somit sehr viel weniger Interruptbehandlungsroutinen ausführen, die so gewonnene CPU-Zeit kommt anderen Prozessen zu Gute.

Wichtig

Bei einem Datentransfer ohne DMA-Controller ereignete sich **ein Interrupt pro Datenwort**.

Mit DMA-Controller ist es nur noch **ein Interrupt für die gesamte Übertragung**.

Aufgabe 2

Aufgabe

Aufg. 55: Datentransfer mit DMA

Erläutere anhand der folgenden (vereinfachten) Abbildung, wie der Transfer von Daten zwischen Festplatte und RAM abläuft, wenn ein DMA-Controller zur Verfügung steht. Wann und wie oft muss die CPU auf ein Interruptsignal reagieren? (Gemeint sind nur die Interrupts, welche durch den Datentransfer ausgelöst werden.)

Welche Änderungen ergeben sich zu Aufgabe 1 oben?

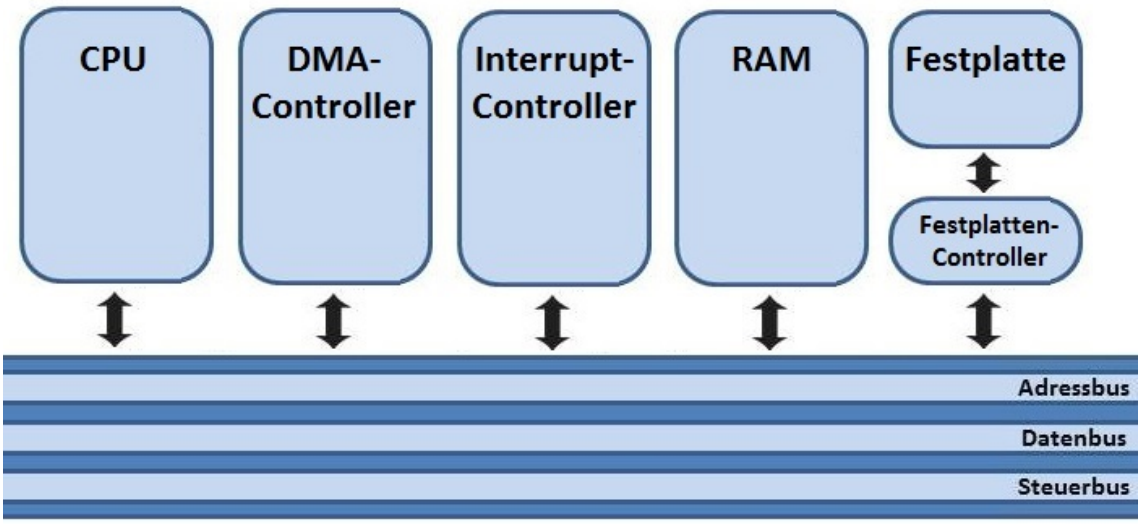


Abb. 45: Vereinfachter Rechner, mit DMA-Controller
CC-BY

Der DMA-Controller macht den Unterschied

Die Einsparung von Interrupts führt unweigerlich zu einer Beschleunigung des Gesamtsystems, da die Abarbeitung jedes einzelnen Interrupts sehr aufwendig ist. Der Abschnitt Datentransfer und Interrupts hat dies bereits gezeigt.

Ein weiterer Grund für einen Geschwindigkeitsvorteil ist, dass ein DMA-Controller von seiner Bauart her auf Datenübertragungen spezialisiert ist. Er kann diese Tätigkeiten deshalb mit einer höheren Geschwindigkeit ausführen, als eine CPU, die universell für viele verschiedene Tätigkeiten entwickelt wurde.

Brinkschulte et al. 2010 erläutern beispielsweise, dass eine Datenübertragung auf der CPU mittels einer in Software programmierten Schleife erfolgt, während auf einem spezialisierten DMA-Controller jegliche Steuerung mittels Hardware erfolgt.

Hinweis

Weiterführende Literatur

Brinkschulte et al. 2010 erläutern in Kapitel 4.6 (DMA) weitere Hintergründe zum Thema. Die Lektüre dieser Quelle sei ausdrücklich empfohlen.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Aufgabe 3

Aufgabe

Aufg. 56: DMA und CPU

Der Einsatz eines DMA-Controllers hat nicht nur Vorteile. Ein Nachteil ist beispielsweise, dass Datenbus und Adressbus während eines Datentransfers per DMA nicht der CPU zur Verfügung stehen.

- Welche Auswirkungen hat dies auf die Arbeitsweise der CPU?
- Was kann die CPU während der Zeit des DMA-Transfers erledigen, und was geht nicht mehr?

Aufgabe 4

Aufg. 57: Arten des DMA-Transfers

Brinkschulte et al. 2010 beschreiben in Kapitel 4.6 drei Arten des DMA-Transfers.

- Welche drei Arten werden beschrieben?
- Worin unterscheiden sie sich?
- Welche Auswirkungen hat dies auf die Arbeit der CPU?
- Bei einer der beschriebenen Arten hat das beteiligte E/A-Gerät die Möglichkeit zu bestimmen, wann der Bus für den DMA-Transfer reserviert wird.
  Erläutere, warum dies Sinn macht!
  Berücksichtige dabei die unterschiedlichen Geschwindigkeiten, mit denen Hauptspeicher und E/A-Geräte arbeiten.

2.3.5.1 Aufbau und Arbeitsweise eines DMA-Controllers

Die folgende Abbildung zeigt den (vereinfachten) Aufbau eines DMA-Controllers.

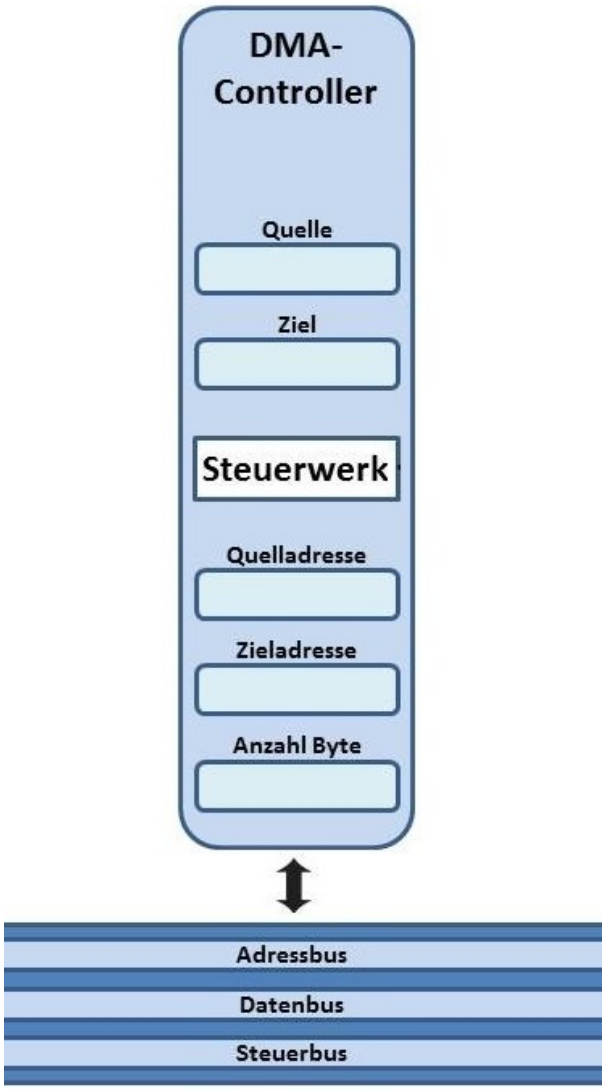


Abb. 46: Aufbau eines DMA-Controllers
CC-BY

Initialisierung des DMA-Controllers

Bevor die Datenübertragung starten kann, muss der DMA-Controller von der CPU initialisiert werden, d.h. die notwendigen Informationen rund um die zu tätigende Übertragung erhalten. Dies sind:

- Quelle:
  Von diesem Gerät sollen Daten übertragen werden.
- Ziel:
  Zu diesem Gerät hin sollen die Daten übertragen werden.
- Startadresse:
  Ab dieser Adresse auf der Quelle sollen Daten übertragen werden.

- Zieladresse:
  Zu dieser Adresse auf dem Ziel sollen die Daten übertragen werden.
- Anzahl Byte:
  Gibt die Anzahl der Bytes an, die ab Startadresse auf der Quelle zur Zieladresse auf dem Ziel übertragen werden sollen.

Datentransfer als Schleife

Sobald der DMA-Controller initialisiert ist, kann die Übertragung starten. Dabei wird immerfort eine Schleife durchlaufen, bis alle Byte übertragen sind.

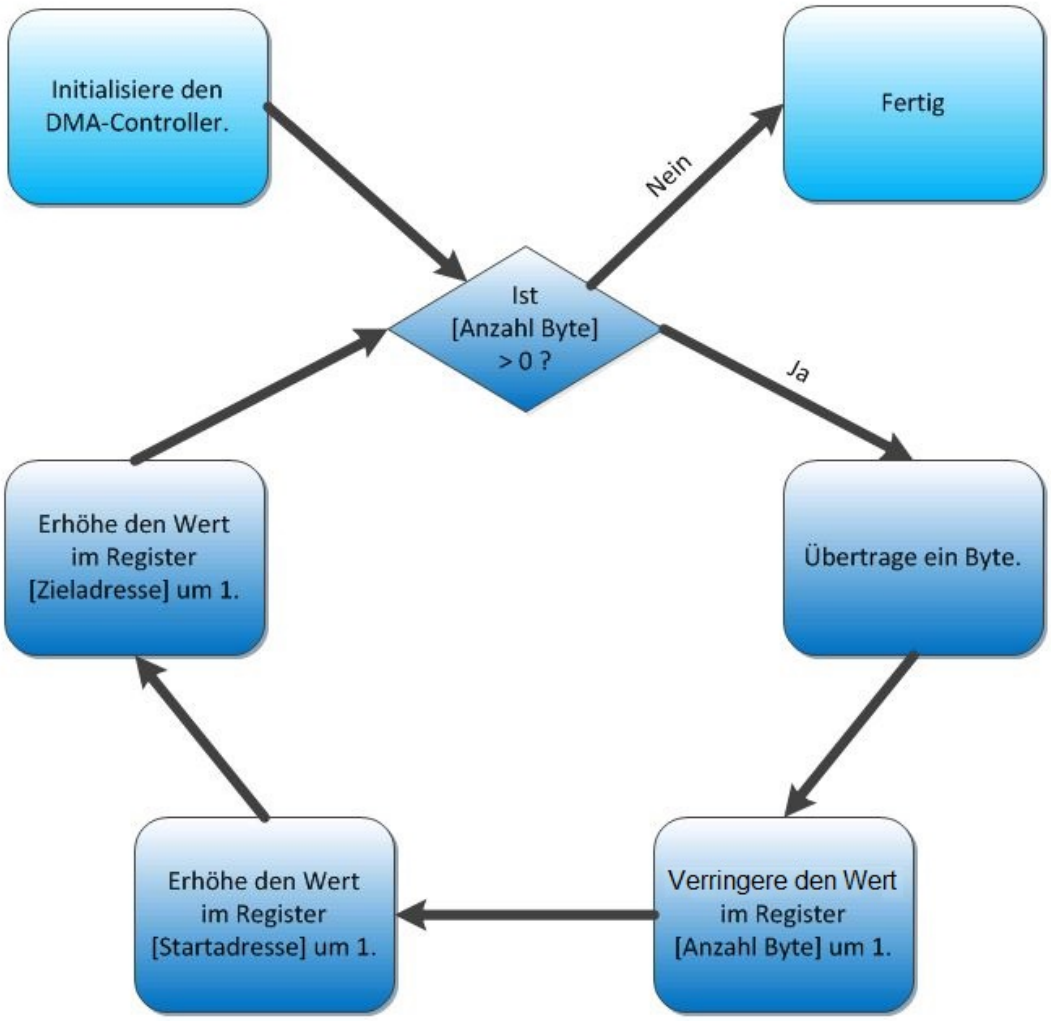


Abb. 47: DMA-Datentransfer als Schleife
CC-BY

Aufgabe 1

Aufgabe

Aufg. 58: Start- und Zieladresse ändern sich

Warum werden beim Datentransfer sowohl die Startadresse als auch die Zieladresse nach jedem übertragenen Byte verändert?

Aufgabe 2

Aufgabe

Aufg. 59: DMA mit RAM und RAM?

Quelle und Ziel der Datenübertragung per DMA könnten beispielsweise Festplatte und RAM sein.

Aber ist auch denkbar, dass RAM und nochmal RAM die Quelle und das Ziel der Übertragung bilden?

Aufgabe 3

Aufgabe

Aufg. 60: Datentransfer mit DMA

Brinkschulte et al. 2010 zeigen in Kapitel 4.6 mit Abb. 4.53 den grundsätzlichen Aufbau eines DMA-Controllers.

Erläutere in deiner Lerngruppe anhand der Abb. 4.53 die aufgelisteten Schritte des Datentransfers mit DMA.

- Welche Informationen werden in welchen Registern abgelegt?
- Wann werden welche Signale/Informationen über welche Leitungen gesendet?

Intel 8257 & 8237 Programmable DMA Controller

Aus der Praxis

Der Intel 8257 ist ein bekannter Vertreter aus der Reihe der DMA-Controller. Sein Datenblatt ist noch heute im Internet abrufbar.

Eine Weiterentwicklung ist der Intel 8237 (Datenblatt), der bereits im Jahre 1981 im ursprünglichen IBM PC verbaut war.

2.3.5.2 Direkt kann wirklich direkt bedeuten

Ohne den DMA-Controller wurden Daten immer über den "Umweg CPU" transportiert. Dieses Video hat den Ablauf beispielhaft gezeigt:

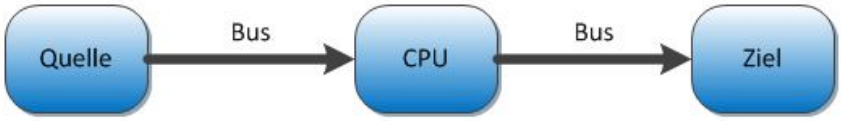


Abb. 48: Daten müssen den Umweg über die CPU nehmen
CC-BY

Ohne Umweg, also direkt

Mit DMA-Controller können die Daten den direkten Weg nehmen, wie das folgende Beispiel zeigt:

1. Der DMA-Controller fordert Daten beim Festplatten-Controller an.
2. Der Festplatten-Controller ist mit dem (Daten-) Bus verbunden und legt die Daten auf selbigen.
3. Der Hauptspeicher ist ebenfalls mit dem (Daten-) Bus verbunden und kann die Daten sofort entgegennehmen, dabei sorgt der DMA-Controller über Steuer- und Adressbus dafür, dass der Hauptspeicher erfährt, wohin die Daten gespeichert werden müssen.

Es kommt also auf das Zusammenspiel zwischen dem DMA-Controller, dem Bussystem, dem Hauptspeicher und dem weiteren beteiligten DMA-fähigen Gerät an. Um einen möglichst schnellen Datentransfer zu erreichen macht es allemal Sinn, die Daten ohne Umweg zu transportieren.

Hinweis

Du hast bereits bemerkt, dass an dieser Stelle eine stark vereinfachte Darstellung des Sachverhalts erfolgt. Generell wird hier davon ausgegangen,

- dass das Bussystem für die Datenübertragung per DMA zur Verfügung steht,
- dass sich CPU und DMA-Controller "irgendwie" einigen, wer gerade den Bus benutzen darf,
- dass Datenkollisionen auf dem Bus "irgendwie" vermieden werden.

Auf eine tiefere Auseinandersetzung mit den angedeuteten Details wird an dieser Stelle bewusst verzichtet, da es den Rahmen dieses Lernmoduls sprengen würde.

Es geht nicht immer direkt

Der beschriebene direkte Weg der Daten von der Quelle zum Ziel ist natürlich der bevorzugte Weg. Jedoch ist dieser nicht immer möglich. Die folgende Aufgabe beschäftigt sich mit einem entsprechenden Szenario.

Aufgabe 1

Aufgabe

Aufg. 61: Umweg über DMA-Controller

Angenommen Quelle und Ziel bei einer Übertragung per DMA sind beide der Hauptspeicher. Dann sollen also Daten von einer Stelle des Hauptspeichers an eine andere Stelle des Hauptspeichers kopiert werden.

- Erläutere warum in diesem Fall keine direkte Übertragung von der Quelle über den Bus zum Ziel möglich ist.

Ersatzweise springt der DMA-Controller ein:

Abb. 49: Umweg über DMA-Controller
CC-BY

- Aber dem DMA-Controller aus der folgenden Abbildung fehlt für diesen Fall noch etwas, oder? Was ist gemeint?

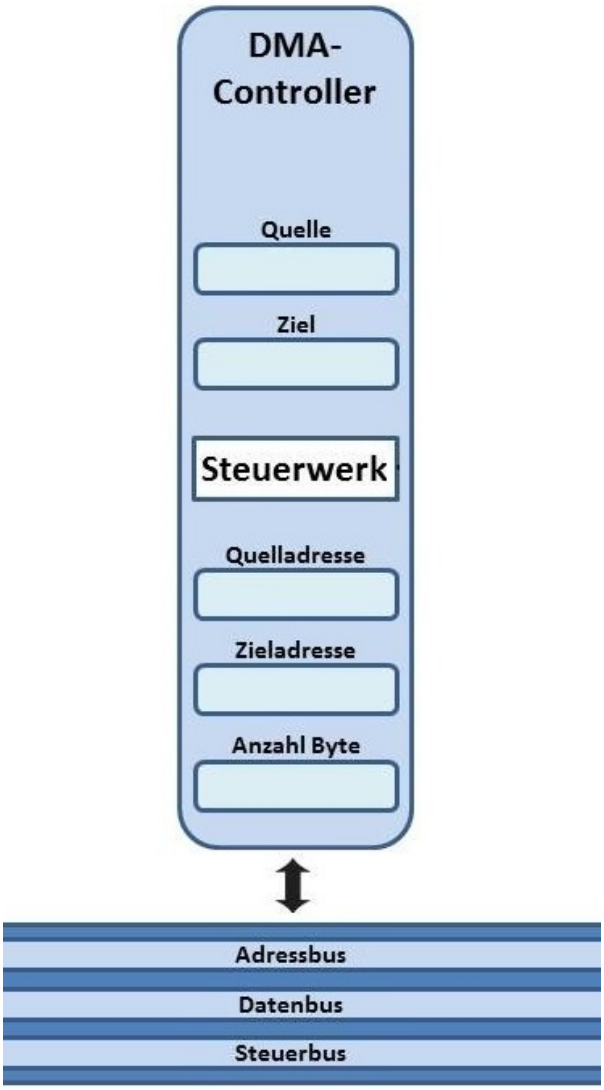


Abb. 50: Aufbau eines DMA-Controllers
CC-BY

2.3.6 MMU - Memory Management Unit

Die **Memory Management Unit**, kurz **MMU**, ist eine Hardware-Komponente, welche üblicherweise direkt auf der CPU beheimatet ist. Sie unterstützt das Betriebssystem bei der Verwaltung des Hauptspeichers, und trägt so zu einer größeren Flexibilität und besseren Ausnutzung des vorhandenen physischen Speichers (RAM) bei.

Wie es bisher ohne MMU ist

Bevor die konkrete Arbeitsweise der MMU erläutert wird, sei kurz an den bisherigen Stand der Hauptspeicherverwaltung erinnert:

Wichtig

Die bisher betrachtete Hauptspeicherverwaltung hat mit Hilfe des Basisregisters jeweils **zusammenhängende** Speicherbereiche für jeden im RAM eingelagerten Prozess zugreifbar gemacht. Der Speicherschutz wurde mit Hilfe des Limitregisters realisiert.

Nachteilig, weil wenig flexibel, ist hierbei der Zwang zum zusammenhängenden Speicherbereich. Es muss bereits ganz zu Beginn festgelegt werden, wie groß dieser Speicherbereich ist. Nachträgliche Änderungen dieser Größe (d.h. zur Laufzeit des betreffenden Prozesses) sind zwar nicht unmöglich, aber i.d.R. sehr zeitintensiv und damit in der Durchführung nicht zu empfehlen.

Um eine Flexibilisierung der Speicherverwaltung zu erreichen, wurden in Betriebssystemen Konzepte einer virtuellen Speicherverwaltung integriert. Die bereits von vorangegangenen Seiten bekannte Forderung wird hier wiederholt:

Wichtig

Wir brauchen ein Betriebssystem!
Und eine der Aufgaben des Betriebssystems wird die Verwaltung des Hauptspeichers und die Versorgung aller Prozesse mit benötigten Teilen des Hauptspeichers sein.

Virtuelle Speicherverwaltung

Die virtuelle Speicherverwaltung kann sehr kompliziert erscheinen, wenn man sie allein durch Worte und ein paar Abbildungen beschreiben, bzw. verstehen soll. Glücklicherweise gibt es das folgende Video, welches die Grundlagen anschaulich zeigt.

Video

An dieser Stelle befindet sich online ein YouTube-Video.
https://youtu.be/PpyWObQw70o

Med. 24: Grundlagen virtueller Speicherverwaltung mit MMU (04:47)
http://youtu.be/PpyWObQw70o
CC-BY

Aus dem Video folgen einige Definitionen:

Definition: Physischer Speicher

Definition

Unter dem **physischen Speicher eines Computers** versteht man den tatsächlich in dieses Gerät verbauten Speicher, soweit er direkt von der CPU oder der MMU angesprochen werden kann.

Definition: Virtueller Speicher

Definition

Unter dem **virtuellen Speicher eines Prozesses** versteht man den Speicherbereich, der einem Prozess durch das Betriebssystem zur Verfügung gestellt wird.

Man bemerke hier eine entscheidende Kleinigkeit in den Definitionen:

Wichtig

Der physische Speicher bezieht sich auf den Computer, während der virtuelle Speicher auf einen Prozess bezogen wird!

Da bei den vorangegangenen Definitionen der Speicher an sich unterschieden wurde, gibt es auch bzgl. der Speicheradressen eine Unterscheidung:

Definition: Physische Speicheradresse

Definition

Unter einer **physischen Speicheradresse** versteht man eine Adresse innerhalb des physischen Speichers eines Rechners.

Definition: Virtuelle Speicheradresse

Definition

Unter einer **virtuellen Speicheradresse** versteht man eine Adresse innerhalb des virtuellen Speichers eines Prozesses.

Aufgabe 1

Aufgabe

Aufg. 62: Eindeutig oder mehrdeutig?

Eine physische Adresse ist eindeutig, d.h. es gibt sie nur einmal pro Rechner. Wie ist das bei einer virtuellen Adresse? Ist diese auch eindeutig, oder ist sie mehrdeutig? Erläutere!

Was sich durch die MMU ändert

Wie im Video zu sehen ist, wird durch die Einführung einer Memory Management Unit das Betriebssystem bei der Umrechnung von virtuellen in physische Speicheradressen unterstützt. Da die MMU als Hardware auf genau diese Tätigkeit optimiert wurde, kann sie diese Umrechnung sehr viel schneller durchführen, und so bei jedem einzelnen Hauptspeicherzugriff einen Geschwindigkeitsvorteil erzielen.

Basis- und Limitregister fallen durch den Einsatz der MMU weg. Sie werden nicht mehr benötigt.

Eine etwas detailliertere Betrachtung der virtuellen Speicherverwaltung geschieht später im Kapitel Betriebssysteme.

Aufgabe 2

Aufgabe

Aufg. 63: Umrechnung und was noch?

Eine der Aufgaben der MMU ist die Umrechnung von virtuellen in physische Speicheradressen. Aber das kann noch nicht alles sein. Denk' mal daran, was durch die MMU auf der CPU alles weggefallen ist.

Was ist also eine weitere Aufgabe der MMU?

Aufgabe 3

Aufgabe

Aufg. 64: Virtuelle Größen

Ein Computer besitzt 2 GiB physischen Speicher. Jeder gestartete Prozess besitzt 4 GiB virtuellen Speicher. Die typische Größe einer einzelnen virtuellen Seite bei Verwendung der virtuellen Speicherverwaltung beträgt 4 KiB.

- Wie groß ist dann ein einzelner Seitenrahmen des physischen Speichers?
- In wieviele Seiten ist der gesamte virtuelle Speicher eines Prozesses unterteilt?
- Wieviele Seitenrahmen gibt es im physischen Speicher insgesamt?
- Wieviele Speicherzellen zu je 8 Bit (= 1 Byte) besitzt ein einzelner Seitenrahmen des physischen Speichers?
- Wieviele Speicherzellen zu je 8 Bit (= 1 Byte) besitzt eine einzelne Seite des virtuelle Speichers?

Gib jeweils kurz den Rechenweg mit an!
Hier findest du Hinweise zu den Schreibweisen GiB und KiB.

Aufgabe 4

Aufgabe

Aufg. 65: Swapping bei virtueller Speicherverwaltung?

Swapping kennst du bereits. Es bezeichnet das Aus- und Einlagern eines **kompletten** Prozesses. Diskutiere die folgenden Fragen in deiner Lerngruppe:

- Funktioniert Swapping deiner Meinung nach auch bei der virtuellen Speicherverwaltung?
- Falls ja: Sollten dann die leeren Seiten auch ausgelagert werden?
- Ist es bei der virtuellen Speicherverwaltung vielleicht auch möglich, dass nur einzelne Seitenrahmen aus- und später wieder eingelagert werden, während die restlichen Seitenrahmen die ganze Zeit im physischen Speicher verbleiben?
  (Das wäre dann eine große Änderung zum Swapping, da hier immer der **komplette** Prozess ausgelagert werden musste.)

2.3.7 Moderne Bussysteme

Das bis jetzt erarbeitete Bild eines Computersystems enthält immer noch einen einzelnen Bus, der lediglich in Adressbus, Datenbus und Steuerbus aufgeteilt ist. Bereits im Kapitel Von-Neumann-Flaschenhals wurde anschaulich gezeigt, dass diese Form des Busses sehr schnell überlastet sein kann und sich deshalb nicht zum Bau eines schnellen Gesamtsystems eignet.

Man kann sich leicht vorstellen, dass moderne Rechner ein deutlich aufwendigeres und auf Geschwindigkeit optimierteres Bussystem besitzen. Im Zuge der Entwicklung wurden verschiedene Ansätze umgesetzt, denen allen gemein ist, dass sie nicht nur einen einzelnen Bus besitzen (wie die Von-Neumann-Architektur), sondern eine Reihe verschiedener Busse, die mit unterschiedlichen Geschwindigkeiten arbeiten.

Front Side Bus und Chipsatz mit North- sowie Southbridge

Boettcher 2006 zeigt in Kapitel 1.1.1, Abbildung 1.1, den Chipsatz, bestehend aus Northbridge und Southbridge, über den alle Komponenten eines Rechners über verschiedene Busse miteinander verbunden sind. Ein ähnlicher Aufbau ist bei *Mandl 2013* in Kapitel 1.1, Abbildung 1-4, zu sehen.

Die CPU ist über den Front Side Bus mit der Northbridge verbunden. Diese bedient alle schnell arbeitenden Komponenten, wie Hauptspeicher, AGP-Grafikkarte und Gigabit-Ethernet-Karte. Weiterhin steht sie in Verbindung mit der Southbridge.

Die Southbridge bedient alle langsamer arbeitenden Komponenten. Sie ist der Knotenpunkt aller mit geringerer Geschwindigkeit arbeitenden Busse, an die z.B. Festplatten und CD/DVD-Laufwerke angeschlossen sind, ebenso gibt es Verbindungen zu den Peripheral-Component-Interconnect-Steckplätzen (kurz: PCI-Slots) und dem Universal-Serial-Bus mit seinen USB-Ports.

Die folgende Abbildung zeigt die North- und die Southbridge als zentrale Verbindungspunkte aller Komponenten.

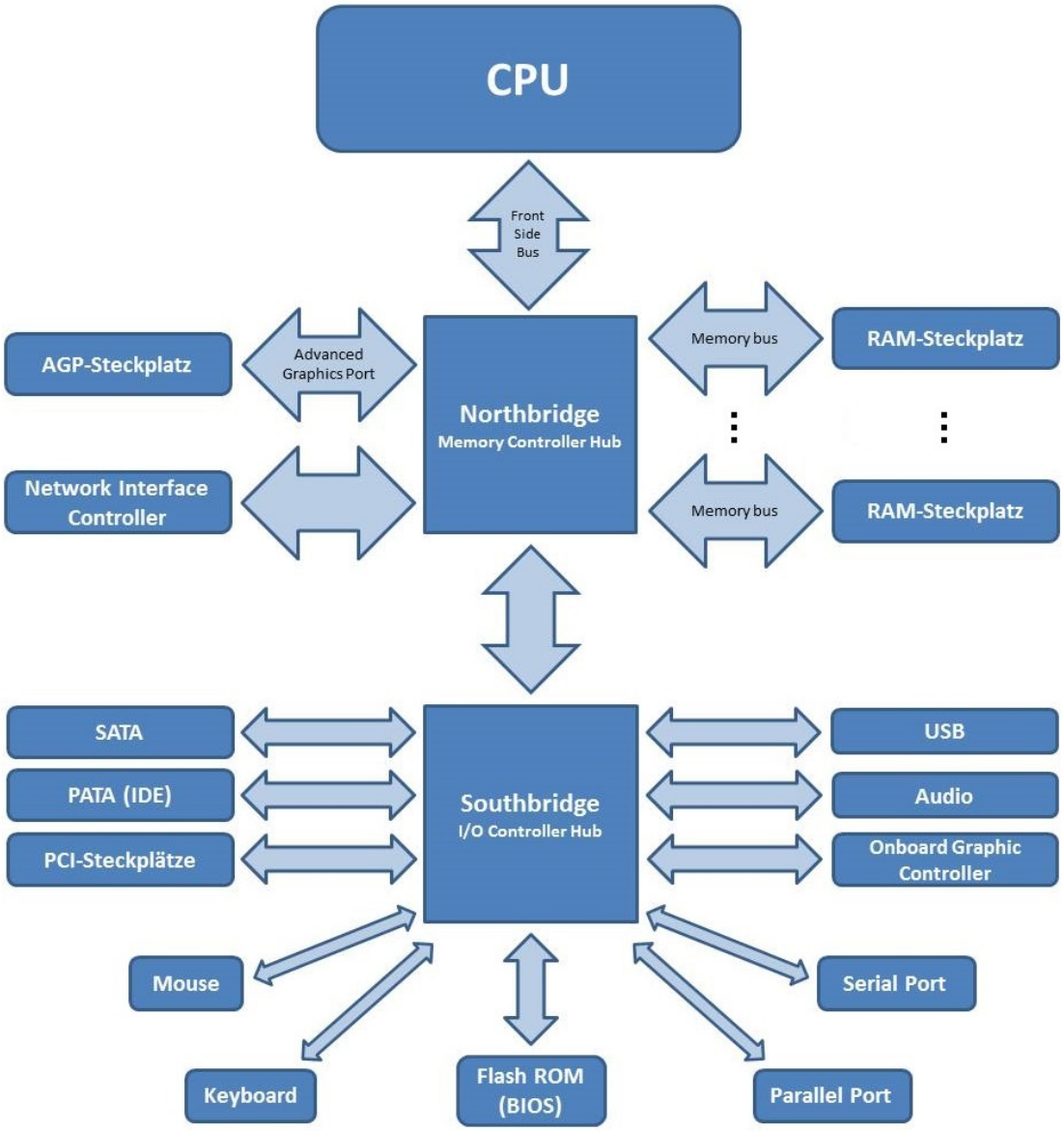


Abb. 51: North- und Southbridge als zentrale Verbindungspunkte
CC-BY

QuickPath Interconnect statt Front Side Bus

Auf Wikipedia ist eine Abbildung des ASUS-Mainboards P6T Deluxe zu sehen. Die zugehörige Wiki-Seite benennt viele der auf dem Mainboard integrierten Komponenten, u.a. auch eine North- und Southbridge.

Asus selbst hat die Spezifikation dieser Hauptplatine ebenfalls im Internet bereitgestellt. Im Handbuch (downloadbar unter "Manual") wird Intel(R) QuickPath Interconnect (kurz: QPI) als Verbindung von der CPU zur Northbridge erwähnt. QPI hat damit

den Front Side Bus abgelöst. Eine Einführung in QuickPath Interconnect stellt Intel zum Download bereit.

HyperTransport als Alternative zu QuickPath Interconnect

Eine Alternative zu QPI stellt HyperTransport (HT) dar. Es wurde (zeitlich noch vor QPI) als offener Standard von einem herstellerübergreifenden Konsortium, dem u.a. die Firmen Advanced Micro Devices (AMD), International Business Machines (IBM) und Apple angehören, entwickelt und standardisiert.

HyperTransport wurde eingeführt, um

- mehrere Prozessoren miteinander zu verbinden,
- Prozessoren mit Coprozessoren zu verbinden,
- Prozessoren mit sonstigen I/O-Komponenten (z.B. North- / Southbridge) zu verbinden.

Northbridge in die CPU integriert

Die Intel Corporation ist ein US-amerikanischer Halbleiterersteller und bekannt für die Produktion von (u.a.) CPUs und Chipsätzen. Eine Produktreihe aus 2012/13 zeigt hier beispielhaft die aktuelle Tendenz in der Weiterentwicklung der Bus-Technologien.

Intel(R) zeigt in Technical Product Specification, Server Board 2600GZ/GL in Kapitel 3 (Product Architecture Overview), Figure 9, den Aufbau einer Hauptplatine mit zwei CPUs. Beide Prozessoren sind miteinander über QuickPath-Interconnect (QPI) verbunden. Weiterhin haben sie eine direkte Anbindung an die Speicherbänke des Hauptspeichers.

Die erste CPU ist direkt mit dem Ethernet-Controller I350 verbunden. Im I350 Reference Design wird auf Seite 3 der Aufbau dieses Controllers gezeigt. Er verfügt über vier Ethernet-Ports (RJ45), PCIE-G2 ist die Verbindung zur CPU.

Eine separate Northbridge ist damit entfallen, ihre Funktionen wurden direkt in die CPU (Intel^{<small>(R)</small>} Xeon^{<small>(R)</small>} E5-2600) intergriert, was einen Geschwindigkeitsgewinn erwarten lässt.

Eine Southbridge ist mit dem Intel ^{<small>(R)</small>} C602 weiterhin vorhanden und stellt die erwarteten Verbindungen bereit.

Hinweis

Auf Abbildungen wurde an dieser Stelle aus Gründen des Copyrights bewusst verzichtet. Die oben im Text genannten Links führen aber direkt zu Dokumenten oder Webseiten, welche entsprechende Grafiken enthalten. Das eigenständige Abrufen der Abbildungen über die entsprechenden Links sei an dieser Stelle dem Leser überlassen.

Hinweis

Weiterführende Literatur

Huette 2012 zählen in Kapitel (J) 7.1.5 einige Bussysteme auf und geben jeweils in kompakter Form ergänzende Informationen dazu. Die Lektüre dieser Quelle sei empfohlen.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

IDE, SATA, USB & Co

Hinweis

Weiterführende Literatur

Huette 2012 zählen in Kapitel (J) 7.1.6 einige periphere Busse (u.a. IDE, USB) auf und geben jeweils in kompakter Form ergänzende Informationen dazu. Die Lektüre dieser Quelle sei empfohlen.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

2.4 Fazit Computerarchitektur

Das Kapitel Computerarchitektur neigt sich damit dem Ende zu.

Es zeigte eine Rundreise, die mit dem Aufschrauben eines "normalen" Computers begann. Die identifizierten Komponenten wurden in das grundlegende Schema der Von-Neumann-Architektur eingebettet und ihre Aufgaben, Funktionen und Arbeitsweisen erläutert.

Die CPU als Herz eines Computers wurde anschließend studiert. Ihr Aufbau, die Arbeitsweise und insbesondere das Zusammenspiel mit einem (oder mehreren) Programmen standen im Fokus.

Schließlich wurden weitere Hardware-Komponenten eingeführt und erläutert, die einerseits die Leistungsfähigkeit eines Computers Stück für Stück erhöhten, andererseits aber auch immer wieder eine zentrale Forderung begründeten:

Wichtig

Wir brauchen ein Betriebssystem!

Denn nur das Zusammenspiel von Hard- **und** Software ergibt am Ende ein Hochleistungssystem, welches gleichzeitig flexibel und bedienbar ist.

Brechen wir also auf zum nächsten großen Kapitel: Betriebssysteme.

3 Betriebssysteme

Im Kapitel Weitere Komponenten der Computerarchitektur ergab sich bereits mehrfach die Forderung:

Wichtig

Wir brauchen ein Betriebssystem!

Und dessen Aufgaben sind:

- die Verwaltung mehrerer Prozesse,
- das Aus- und Einlagern von Prozessen,
- die Verwaltung des Hauptspeichers,
- die Versorgung aller Prozesse mit benötigten Teilen des Hauptspeichers,
- die Auswahl des jeweils nächsten Prozesses für die CPU,
- die Datei- und Verzeichnisverwaltung auf Datenträgern,
- die Verwaltung der Ein- und Ausgabegeräte.

Die identifizierten Aufgabengebiete eines Betriebssystems werden in den folgenden Unterkapiteln näher beleuchtet.

So geht es weiter:

Gliederung

3 Betriebssysteme
3.1 Einführung Betriebssysteme
3.2 Prozessverwaltung
3.3 Speicherverwaltung
3.4 Geräteverwaltung
3.5 Dateiverwaltung

Hinweis

Weiterführende Literatur

Die hier verlinkte Online-Ausgabe eines Lehrtextes der Otto-Friedrich-Universität Bamberg liefert in Teil III "**Betriebssysteme**" (Kapitel 8 bis 11) detaillierte Informationen zu Betriebssystemen. Die Lektüre dieser Quelle sei unter Beachtung der geltenden Lizenz ausdrücklich empfohlen.

Autoren: Martin Eisenhardt, Andreas Henrich, Stefanie Sieber
'''Rechner- und Betriebssysteme, Kommunikationssysteme, Verteilte Systeme'''

Dieses Werk steht unter der Creative Commons BY-NC-ND-Lizenz
http://creativecommons.org/licenses/by-nc-nd/2.0/de/

Hinweis

Weiterführende Literatur

Mandl 2013 erläutert im kompletten Buch ausführlich das Thema Betriebssysteme. Es sei als Begleitlektüre ausdrücklich empfohlen.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Alternative Webquelle zum Thema

Quelle im Web

Operating Systems: Introduction

http://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/1_Introduction.html

Dr. John T. Bell

Department of Computer Science

University of Illinois, Chicago

Quelle im Web

Operating Systems: Operating-System Structures

http://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/2_Structures.html

Dr. John T. Bell

Department of Computer Science

University of Illinois, Chicago

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.1 Einführung Betriebssysteme

Eisenhardt et.al. 2007 unterscheiden in Kapitel 8 zwischen Systemprogrammen und Anwendungsprogrammen, um anschließend den Begriff "Betriebssystem" zu definieren: </p>

Definition: Anwendungsprogramm

Definition

Unter einem **Anwendungsprogramm** versteht man ein Computerprogramm, welches ein oder mehrere Probleme löst, die von Benutzern eines Computers gestellt werden.

Klassische Anwendungsprogramme sind beispielsweise:

- Textverarbeitung

- E-Mail-Client
- Browser
- etc.

Definition: Systemprogramm

Definition

Unter einem Systemprogramm versteht man ein Computerprogramm, welches der Verwaltung des Betriebs eines Computers dient.

Da im Kapitel Weitere Komponenten der Computerarchitektur eine Reihe von nötigen Verwaltungsaufgaben identifiziert wurden, ist bereits an dieser Stelle klar, dass ein Betriebssystem ein spezielles Systemprogramm darstellt.

Definition: Betriebssystem

Definition

Ein Betriebssystem ist ein Programm, das dem Benutzer und Anwendungsprogrammen elementare Dienste bereitstellt. Das Betriebssystem steuert und überwacht die Abwicklung von Programmen und regelt den Betrieb des Rechnersystems. (Nach *Eisenhardt et.al. 2007*)

Viele verschiedene Betriebssysteme

Im Laufe der Jahrzehnte wurden bemerkenswert viele verschiedene Betriebssysteme entwickelt. Einen umfassenden Überblick gibt ein Wikipedia-Artikel:
http://de.wikipedia.org/wiki/Liste_von_Betriebssystemen

So geht es weiter:

Gliederung

3.1 Einführung Betriebssysteme
3.1.1 Geschichtlicher Überblick zu Betriebssystemen
3.1.2 Nur ein Prozessor mit einem Kern
3.1.3 Zwischen Benutzer und Hardware
3.1.4 Betriebsmittel
3.1.5 Betriebsmittel sind Prozessen zugeordnet
3.1.6 Zentrale Aufgabe eines Betriebssystems
3.1.7 Betriebssystemarchitekturen
3.1.8 Betriebssystemarten
3.1.9 Vom Batch-Job zum Multitasking
3.1.10 Kernel-Mode, User-Mode und Systemaufrufe

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.1.1 Geschichtlicher Überblick zu Betriebssystemen

Auf die Angabe eines geschichtlichen Überblicks zum Thema Betriebssysteme wird an dieser Stelle bewusst verzichtet. Die weiterführende Literatur liefert vielfältige Zusammenfassungen. </p>

Weiterführende Literatur

Mandl 2013 gibt in den Kapiteln 1.2.3 (Historische Entwicklung), 1.2.4 (Geschichte von Microsoft Windows), 1.2.5 (Geschichte von Unix) und 1.2.6 (Geschichte von Linux) eine Zusammenfassung zum Thema.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Es findet sich ebenfalls ein geschichtlicher Überblick bei Wikipedia .

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.1.2 Nur ein Prozessor mit einem Kern

Alle Ausführungen und weiteren Betrachtungen in den Unterkapiteln zum Abschnitt Betriebssysteme gehen jeweils davon aus, dass der/die zu Grunde gelegten Rechner nur eine einzige CPU besitzen, die wiederum nur über einen einzelnen Prozessorkern verfügt. </p>

Unter dieser Voraussetzung ist klar definiert, dass immer nur genau ein Befehl zur Zeit auf der betreffenden CPU ausgeführt werden kann. Eine gleichzeitige (parallele) Ausführung mehrerer Befehle ist also technisch bedingt nicht möglich.

Kein Multicore, kein Hyper-Threading!

Auch wenn Multicore- und Hyper-Threading-Prozessoren heute üblich sind, und in den meisten handelsüblichen Personal Computern und Laptops verbaut sind, so betrachten wir hier ausdrücklich nur die einfacher aufgebauten CPUs, bei denen tatsächlich nur ein einziger Befehl zur Zeit ausgeführt werden kann!

Weblinks

- Erläuterungen zu Multicore-Prozessoren bei Wikipedia .
- Erläuterungen zu Hyper-Threading bei Wikipedia .

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.1.3 Zwischen Benutzer und Hardware

Die folgende Abbildung verdeutlicht, wie Anwendungsprogramm(e) und Betriebssystem zwischen dem Benutzer und der Hardware vermitteln: </p>

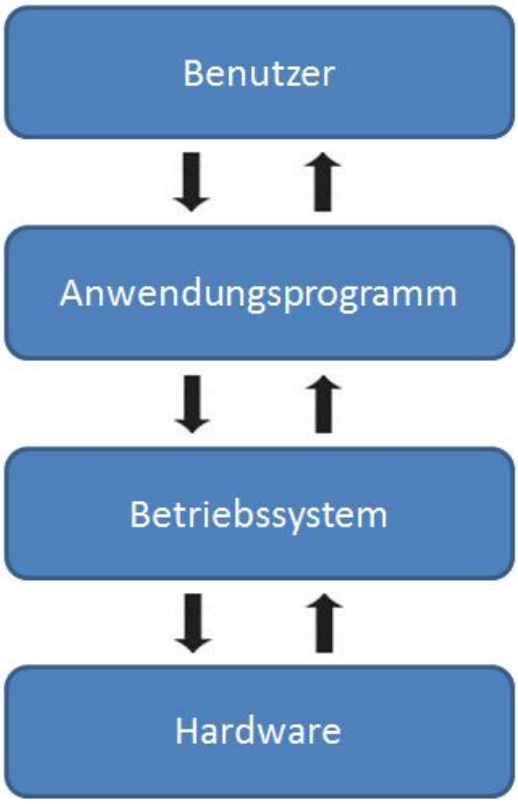


Abb. 52: Betriebssystem und Anwendungprogramm(e) vermitteln zwischen Benutzer und Hardware CC-BY

Der Benutzer (Anwender) bedient ein (oder mehrere) Anwendungsprogramm(e).

Die Anwendungsprogramme geben Rückmeldungen an den Anwender, oder nutzen vom Betriebssystem bereitgestellte Funktionen, um darüber mit der Hardware zu kommunizieren.

Das Betriebssystem empfängt über seine bereitgestellten Funktionen Aufträge von den Anwendungsprogrammen. Es kann die Rechtmäßigkeit dieser Aufträge prüfen und sie im positiven Fall an die Hardware weiterreichen.
Die Anwendungsprogramme erhalten Rückmeldungen vom Betriebssystem über die abgelehnten, sowie die ganz oder teilweise ausgeführten Aufträge.

Die Hardware führt Befehle aus, die ihr vom Betriebssystem in Auftrag gegeben wurden. Das Betriebssystem erhält Rückmeldungen der Hardware zu den ganz oder teilweise ausgeführten Befehlen. Auch bei auftretenden Hardware-Fehlern wird das Betriebssystem (wenn möglich) informiert.

Aufgabe 1

Aufgabe

Aufg. 66: Was wäre wenn... (I)

Was wäre, wenn es keine Betriebssysteme gäbe?

Die Abbildung oben bestünde dann nur aus den Ebenen *Benutzer*, *Anwendungsprogramm* und *Hardware*.

- Was bedeutet dies für die "normalen Nutzer" der Anwendungssoftware?
  (z.B. für den Studierenden, der seine/ihre Abschlussarbeit mit einer Textverarbeitung erstellt? Oder für den Manager, der seine E-Mails abruft?)
- Was bedeutet dies für die Programmierer der Anwendungssoftware?
- Und welche Auswirkungen hätte dies auf die Sicherheit des Systems, insbesondere wenn mehrere Programme quasi-parallel ausgeführt werden?

Diskutiere diese Fragen in deiner Lerngruppe!

Aufgabe 2

Aufgabe

Aufg. 67: Was wäre wenn... (II)

Was wäre, wenn es keine Betriebssysteme und keine Anwendungsprogramme gäbe?

Die Abbildung oben bestünde dann nur aus den Ebenen *Benutzer* und *Hardware*.

- Was würden Studierende dann in Fächern wie "Grundlagen der Programmierung" lernen müssen?

Eine seriöse Auseinandersetzung mit den hier gestellten Fragen sollte unweigerlich zu folgender Erkenntnis führen:

Wichtig

Wenn es keine Betriebssysteme gäbe,
dann müsste man sie so schnell wie möglich erfinden!

Aufgabe 3

Aufgabe

Aufg. 68: Treiber

Aus deiner Erfahrung im täglichen Umgang mit dem Computer kennst du vermutlich Treiber für Hardware. Erläutere:

- Was versteht man unter einem Hardware-Treiber?
- Wo in der obigen Abbildung integrierst du die Treiber?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.1.4 Betriebsmittel

Es folgen einige wichtige Definitionen rund um Betriebsmittel: </p>

Definition: Betriebsmittel

Definition

Unter einem **Betriebsmittel** (oder allgemein einer **Ressource**) eines Rechners versteht man eine beliebige Hardware- oder Software-Ressource.

Definition: Hardware-Ressource

Definition

Unter einer **Hardware-Ressource** eines Rechners versteht man eine einzelne Hardware-Komponente dieses Rechners.

Beispiele für Hardware-Ressourcen sind:

- CPU

  (sofern Sie nur über einen einzelnen Kern verfügt; bei Mehrkern-CPUs kann jeder einzelne Kern auch als einzelne Hardware-Ressource verstanden werden; beachte dazu die Hinweise im Kapitel Nur ein Prozessor mit einem Kern)

- Arbeitsspeicher (RAM)
- beliebige Geräte-Controller
  (und über die Controller werden die dazugehörigen Geräte auch gerne als Hardware-Ressourcen verstanden; z.B. Festplatte, Drucker, Monitor, Tastatur, Maus, ...)

Definition: Software-Ressource

Definition

Unter einer **Software-Ressource** versteht man einen Prozess oder eine Datei auf einem beliebigen Datenträger.

Man beachte an dieser Stelle den bereits bekannten Unterschied zwischen einem Programm und einem Prozess.

Hinweis

Als versierter Anwender weißt du, dass es prinzipiell möglich ist, **ein Programm** auf einem PC oder Laptop **mehrfach zu starten**. Aus einem einzelnen Programm können so mehrere Prozesse resultieren!

Das Betriebsmittel *Prozess* umfasst neben dem Programmtext auch alle zugehörigen Daten im Hauptspeicher (z.B. Stack, Heap, etc.), sowie auf der CPU (z.B. die Registerinhalte).

Bei dem Betriebsmittel *Datei* kann es sich sowohl um eine ausführbare Programmdatei handeln, als auch um eine beliebige Art von Datendatei.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.1.5 Betriebsmittel sind Prozessen zugeordnet

Prozesse benötigen während ihrer Abarbeitung verschiedene Betriebsmittel, um alle anfallenden Aufgaben erledigen zu können. Damit sind jedem gestarteten Prozess eine Menge von Betriebsmitteln zugewiesen. </p>

Aus der Praxis

- Ein Prozess benötigt (mindestens) die Betriebsmittel *Festplatte*, *ausführbare Datei X* sowie *RAM*, um aus der Datei X auf der Festplatte in den Arbeitsspeicher geladen zu werden.
- Ein Prozess benötigt das Betriebsmittel *CPU* um befehlsweise abgearbeitet zu werden.
- Ein Prozess benötigt das Betriebsmittel *Grafikkarte* um eine Ausgabe auf dem Bildschirm zu erzeugen.

- etc.

Es sind jedoch nicht alle Betriebsmittel während der gesamten Laufzeit mit dem Prozess verbunden. Es ist effizienter (gerade wenn viele Prozesse existieren), wenn Betriebsmittel nur dann angefordert werden, wenn sie tatsächlich benötigt werden, und wieder freigegeben werden, wenn sie nicht mehr benötigt werden.

Aus der Praxis

Wenn du mit dem Auto von Hamburg nach München fährst, dann ist es für die Gesamtmenge an Autofahrern nicht vorteilhaft, wenn das Betriebsmittel *Autobahn* während der gesamten Fahrt nur exklusiv für dich reserviert wäre. (Auch wenn du selbst das sicherlich sehr angenehm finden würdest.)

Klassifizierungen

Man betrachte die im Folgenden definierten Klassifizierungen:

Definition: Entziehbares Betriebsmittel

Definition

Unter einem **entziehbaren Betriebsmittel** versteht man ein Betriebsmittel, das einem Prozess bei dessen Abarbeitung **zu einem beliebigen Zeitpunkt** und **ohne negative Folgen** entzogen werden kann.

Ein Prozessor (CPU) ist ein entziehbares Betriebsmittel. Im Abschnitt Interrupt-Controller wurde bereits erläutert, dass keine negativen Folgen auftreten dürfen, wenn die CPU einem Prozess entzogen wird.

Definition: Nicht entziehbares Betriebsmittel

Definition

Unter einem **nicht entziehbaren Betriebsmittel** versteht man ein Betriebsmittel, das einem Prozess so lange zur Verfügung stehen muss, wie dieser es benötigt.

Ein vorzeitiger Entzug eines nicht entziehbaren Betriebsmittels hätte negative Folgen, die i.d.R. vom Anwender nicht gewünscht (und deshalb auch nicht erwartet) werden.

Ein Drucker ist zum Beispiel ein nicht entziehbares Betriebsmittel.

Druckt beispielsweise der Prozess einer Textverarbeitung gerade ein umfangreiches Dokument aus, so führt der Entzug dieses Betriebsmittels mit hoher Wahrscheinlichkeit zu negativen Folgen.

Man stelle sich nur einmal vor, dass während des Ausdrucks eines Briefes der Drucker dem Prozess (Textverarbeitung) entzogen wird, und einem anderen Prozess (E-Mail-Client) zugewiesen wird. Dann könnte auf der oberen Hälfte der Druckseite ein Teil des Briefes gedruckt werden, und auf der unteren Hälfte ein Teil einer zu druckenden E-Mail.

Definition: Exklusiv nutzbares Betriebsmittel

Definition

Unter einem **exklusiv nutzbaren Betriebsmittel** versteht man ein Betriebsmittel, das zu einem beliebigen Zeitpunkt nur maximal einem Prozess zugeordnet sein darf.

Entweder ist ein exklusiv nutzbares Betriebsmittel also gar keinem Prozess zugeordnet, oder es ist genau einem Prozess zugeordnet.

Exklusiv nutzbare Betriebsmittel können prinzipiell *entziehbar* oder *nicht entziehbar* sein.

- Ein Prozessor (CPU) mit nur einem Kern ist nur exklusiv nutzbar (und entziehbar).
- Ein Drucker ist nur exklusiv nutzbar (und nicht entziehbar).

Definition: Gemeinsam nutzbares Betriebsmittel

Definition

Unter einem **gemeinsam nutzbaren Betriebsmittel** versteht man ein Betriebsmittel, das quasi-gleichzeitig von mehreren Prozessen genutzt werden kann.

Eine Festplatte ist ein Beispiel für ein gemeinsam nutzbares Betriebsmittel.

Mehrere Prozesse können zur gleichen Zeit verschiedene Dateien auf der Festplatte geöffnet halten. Lediglich ein *gleichzeitiger Zugriff* der beteiligten Prozesse auf mehrere Dateien ist technisch bedingt nicht möglich.

Aufgabe 1

Aufgabe

Aufg. 69: Datei nur exklusiv oder gemeinsam nutzbar?

Ob eine einzelne Datei ein *exklusiv* oder ein *gemeinsam nutzbares Betriebsmittel* ist, kommt darauf an.

- Erläutere: Worauf kommt es an?
- Finde ein Beispiel, bei dem eine Datei nur exklusiv nutzbar ist.

- Finde ein weiteres Beispiel, bei dem eine Datei von mehreren Prozessen gemeinsam nutzbar ist.

Aufgabe 2

Aufgabe

Aufg. 70: Welche weiteren Betriebsmittel?

Oben auf dieser Seite heißt es:

"*Ein Prozess benötigt (mindestens) die Betriebsmittel Festplatte, ausführbare Datei X sowie RAM, um aus der Datei X auf der Festplatte in den Arbeitsspeicher geladen zu werden.*"

Die Ergänzung *mindestens* weist darauf hin, dass noch weitere Betriebsmittel nötig sein könnten, um aus der Datei in den Arbeitsspeicher geladen zu werden.
Welche weiteren Betriebsmittel könnten das sein?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.1.6 Zentrale Aufgabe eines Betriebssystems

Es ist an der Zeit, die zentrale Aufgabe eines Betriebssystems zu benennen: </p>

Definition: Zentrale Aufgabe eines Betriebssystems

Definition

Die **zentrale Aufgabe** eines Betriebssystems ist die **Betriebsmittelverwaltung**.

(Und da die Begriffe *Betriebsmittel* und *Ressource* synonym verwendet werden können, spricht man auch von der **Ressourcenverwaltung** als zentraler Aufgabe.)

Diese Definition liefert mit der *Betriebsmittelverwaltung* einen zentralen Oberbegriff zu den bislang bereits identifizierten Aufgaben eines Betriebssystems:

- Verwaltung mehrerer Prozesse
- Aus- und Einlagern von Prozessen
- Verwaltung des Hauptspeichers
- Versorgung aller Prozesse mit benötigten Teilen des Hauptspeichers
- Auswahl des jeweils nächsten Prozesses für die CPU

- Datei- und Verzeichnisverwaltung auf Datenträgern
- Verwaltung der Ein- und Ausgabegeräte

Hinweis

Selbstverständlich gibt es noch eine Reihe weiterer Aufgaben, die ein Betriebssystem übernehmen kann. Wir beschränken uns in dieser Lerneinheit allerdings auf die hier genannten grundlegenden Tätigkeiten.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.1.7 Betriebssystemarchitekturen

Dieses Kapitel wird in der weiterführenden Literatur behandelt: </p>

Hinweis

Weiterführende Literatur
Mandl 2013 erläutert in Kapitel 2.2 verschiedene Betriebssystemarchitekturen. Die Lektüre dieser Quelle sei ausdrücklich empfohlen.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Konkret beschreibt *Mandl 2013* die Architektur des **monolithischen Kernels** und dessen Weiterentwicklung, den **schichtenorientierten Kernel**. Anschließend stellt er beiden Varianten die **Mikrokern-Architektur** gegenüber.

Als konkrete Beispiele beschreibt er in etwas vereinfachter Weise die Architekturen der Betriebssysteme **Unix**, **Linux**, **Android** und **Windows**.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.1.8 Betriebssystemarten

Hinsichtlich der Betriebssystemarten finden sich in der Literatur unterschiedliche Klassifizierungsansätze. </p>

Tanenbaum 2009 , Kapitel 1.4, und *Strelen 2012* , Kapitel 1.3, beschreiben eine ähnliche Artenliste, die sich insbesondere an der Hardware orientiert, für welche die jeweilige Betriebssystemart konzipiert wurde.

Mandl 2013 klassifiziert in den Kapiteln 2.3 bis 2.7 eher nach dem Einsatzszenario, unter dem das jeweilige Betriebssystem nach seiner Installation betrieben werden soll.

Für einen erweiterten Einblick in die Thematik seien die oben genannten Quellen als Lektüre empfohlen.

Im Rahmen dieses Lernmoduls ist eine Beschränkung auf die folgenden Betriebssystemarten ausreichend:

- Betriebssysteme für Großrechner
- Betriebssysteme für Server
- Betriebssysteme für Laptops und Personal Computer

Betriebssysteme für Großrechner

Großrechner (Mainframes) verarbeiten i.d.R. eine sehr hohe Zahl an Anfragen in sehr kurzer Zeit. Sie sind oftmals optimiert auf eine sehr hohe E/A-Rate und verfügen über eine sehr große Speicherkapazität auf den angeschlossenen Festplatten.

Mainframe-Betriebssysteme arbeiten üblicherweise im Stapelbetrieb (Batch-Jobs) oder sind transaktionsorientiert. Letzteres bedeutet, dass eine sehr große Zahl an Transaktionen in sehr geringer Zeit ausgeführt werden kann.

Ein Beispiel für ein Mainframe-Betriebssystem ist z/OS von IBM. Es ist u.a. auf Großrechnern bei Banken und Versicherungen im Einsatz.

Betriebssysteme für Server

Server werden überwiegend über das Netzwerk von vielen verschiedenen Clients angesprochen. Die Anwender an den Clients erwarten eine möglichst schnelle Reaktionszeit des Server-Betriebssystems, d.h. die angeforderten Daten oder Informationen sollen mit nur minimaler Verzögerung bereitgestellt werden.

Übliche Vertreter für Server-Betriebssysteme sind Windows Server 20xx, Unix oder Linux. Eine grafische Oberfläche am Server-Betriebssystem ist möglich, aber nicht erforderlich.

Die Einsatzzwecke für Server-Betriebssysteme liegen beispielsweise bei:

- File-Server
- Print-Server
- E-Mail-Server
- Terminal-Server

Betriebssysteme für Laptops und Personal Computer

Mit dieser Art Rechnersystem dürften die meisten Leserinnen und Leser dieses Lernmoduls täglich in Berührung kommen. Üblicherweise werden hier durch einen einzigen Nutzer verschiedene Programme gestartet, die in der Folge quasi-parallel genutzt werden.

Es handelt sich also um eine überwiegend dialogorientierte Nutzungsart: Der Anwender tätigt eine Eingabe (z.B. mit Maus und/oder Tastatur) und das Betriebssystem reagiert (möglichst schnell) darauf.

In der Praxis werden heute überwiegend Betriebssysteme mit grafischer Oberfläche eingesetzt, z.B. Windows 10/11, MacOS X oder Linux mit KDE oder Gnome.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.1.9 Vom Batch-Job zum Multitasking

Der Weg von den Batch-Jobs zum Multitasking gleicht einer Zeitreise. </p>

Batch-Jobs

In den Anfangsjahren der Rechnerentwicklung wurden auszuführende Programme als sogenannte Batch-Jobs verarbeitet (Stapelbetrieb). Auf diese Verfahrensweise wurde in den Kapiteln Gesamtbild der Programmausführung und Mehrere Prozesse gleichzeitig im Speicher eingegangen.

Jeder Prozess (Batch-Job) wurde nach dem Start komplett bis zu seinem Ende bearbeitet, erst im Anschluß startete der nächste Prozess (Batch-Job) und belegte die CPU ebenfalls bis zu seiner Terminierung. Entsprechend der Anzahl der Batch-Jobs wurde dieses Prinzip wiederholt, der letzte Prozess in der Reihe musste damit auch am längsten warten, bis er endlich gestartet wurde.

Multitasking

Im Kapitel Quasi-gleichzeitige Ausführung mehrerer Prozesse wurde erläutert, wie mehrere Prozesse sich bei ihrer Abarbeitung auf der CPU abwechseln können. Dieses Verfahren wird **Multitasking** genannt, oder auf deutsch **Mehrprogrammbetrieb**.

Diese geänderte Vorgehensweise markiert einen gewaltigen Meilenstein in der Informatik, da sich hierdurch - neben einer besseren Auslastung des Rechners - ganz andere Möglichkeiten in der Interaktion zwischen dem Computersystem und dem Anwender auftun.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.1.10 Kernel-Mode, User-Mode und Systemaufrufe

Solange auf einem Rechner nur ein einzelner Prozess im Hauptspeicher gehalten wurde und auf dessen CPU ablief, war alles noch ganz einfach: Ein Betriebssystem wurde nicht benötigt und über Sicherheit musste man sich keine Gedanken machen. </p>

Sobald aber mehrere Prozesse quasi-parallel auf einem System ablaufen, wird es komplexer:

- Jeder einzelne Prozess besitzt eigene Daten, und es muss dafür gesorgt werden, dass nur der berechtigte Prozess Zugriff auf diese Daten hat. Für alle anderen Prozesse muss sichergestellt sein, dass sie keinen unberechtigten Zugriff auf fremde Daten besitzen.
- Mehrere Prozesse müssen sich auch in fairer Weise auf der CPU abwechseln, damit alle ihre Aufgabe erfüllen können. Dabei muss es eine zentrale Stelle geben, die den fairen Wechsel durchsetzt.

Dies sind nur zwei Beispiele, die letztlich zu der Entwicklung von Betriebssystemen geführt haben. Damit ein Betriebssystem aber diese (und weitere) Aufgaben erfüllen kann, benötigt es auf einem Rechnersystem **mehr Privilegien** als jeder normale Prozess (als also jedes Anwendungsprogramm).

Aus diesem Grund werden bei der Abarbeitung von Befehlen auf der CPU **zwei Modi** unterschieden:

1. Kernel-Mode
2. User-Mode

Kernel-Mode

Arbeitet die CPU im Kernel-Mode, so ist jeder beliebige Befehl zur Ausführung zugelassen. Es kann auf sämtliche Speicherbereiche für Daten- und Programmtext, sowie auf alle Betriebsmittel zugegriffen werden. Hier ist alles erlaubt, es bestehen die höchsten Privilegien. (*Mandl 2013* nennt diesen Modus deshalb auch den *privilegierten Modus*.)

Durch ein Steuer- oder Kontrollregister auf der CPU wird der Kernel-Mode angezeigt.

Das Betriebssystem arbeitet üblicherweise im Kernel-Mode und hat somit alle Möglichkeiten, seine definierten Aufgaben zu erfüllen.

User-Mode

Arbeitet die CPU im User-Mode, so ist nur ein eingeschränkter Befehlssatz zur Ausführung zugelassen. Es sind also nicht alle Befehle erlaubt, ebenso kann nicht auf alle Speicherbereiche und auch nicht auf alle Betriebsmittel zugegriffen werden.

Durch ein Steuer- oder Kontrollregister auf der CPU wird der User-Mode angezeigt.

Anwendungsprogramme arbeiten üblicherweise im User-Mode. Diese haben damit nur sehr eingeschränkte Möglichkeiten, das soll auch so sein.

Übergang vom User-Mode in den Kernel-Mode

Durch die Unterteilung in User-Mode und Kernel-Mode muss klar definiert sein, wie ein Übergang von dem einen in den anderen Modus stattfinden kann.

Ein Übergang vom höher privilegierten Kernel-Mode in den User-Mode ist unproblematisch, da hierbei die Rechte eingeschränkt werden, und somit keine Sicherheitsbedenken bestehen. Das Betriebssystem veranlasst diesen Übergang, wenn es dem Prozess eines Anwendungsprogramms die CPU zuteilt.

Anders verhält es sich beim Übergang vom User-Mode in den Kernel-Mode. Der Prozess eines Anwendungsprogramms darf nicht über die umfassenden Rechte des Kernel-Modes verfügen. Er kann sie deshalb nur indirekt bekommen, indem er einen sogenannten Systemaufruf ausführt.

Definition: Systemaufruf

Definition

Unter einem **Systemaufruf** (oder: **Systemcall**, **Syscall**) versteht man den von einem im User-Mode ablaufenden Prozess getätigten Aufruf einer vom Betriebssystem zur Verfügung gestellten Funktion, welche nur im Kernel-Mode ausgeführt werden kann.

Durch den Systemaufruf gibt der im User-Mode ablaufende Prozess die Kontrolle zurück an das Betriebssystem, welches die gewünschte Funktion (nach einem Umschalten in den Kernel-Mode) stellvertretend ausführt.

Das Betriebssystem hat damit die Möglichkeit, durch vorherige Sicherheitsüberprüfungen festzustellen, ob der aufrufende Prozess zur Ausführung der gewünschten Funktion überhaupt berechtigt ist, und ob auch sonst keine anderen Gegebenheiten gegen eine Ausführung sprechen.

Ist die betreffende Funktion ausgeführt (oder wurde sie aufgrund der Überprüfungen verweigert), so wird zur Beendigung des Systemaufrufs wieder in den User-Mode zurückgeschaltet, und das Betriebssystem gibt die Kontrolle zurück an den aufrufenden

Prozess, der i.d.R. ein Ergebnis seines Systemaufrufs empfängt und in der Folge auf das Resultat reagieren kann.

Beispiel

Beispiel

Ein Anwendungsprogramm möchte auf eine Datei zugreifen.

Der im User-Mode ablaufende Prozess eines Anwendungsprogramms möchte auf eine Datei zugreifen. Diese E/A-Handlung ist jedoch nur im Kernel-Mode zugelassen. Das Betriebssystem stellt dem Anwendungsprogramm deshalb eine Reihe von Systemaufrufen zur Verfügung, welche Dateizugriffe ermöglichen, z.B.:

- open
- read
- write
- close

Wird jetzt ein Systemaufruf getätigt, so entspricht dies der Auslösung eines (Software-) Interrupts. Der (im User-Mode) laufende Prozess wird angehalten und der weitere Ablauf ist:

- die zugehörige Interrupt-Service-Routine wird ausgeführt (dadurch wird Betriebssystem-Code ausgeführt),
- es wird in den Kernel-Mode geschaltet,
- notwendige Überprüfungen werden durchgeführt (z.B. dürfen nicht quasi-gleichzeitig mehrere Prozesse in eine Datei schreiben),
- der Dateizugriff wird entweder erlaubt und durchgeführt oder verweigert,
- es wird in den User-Mode zurückgeschaltet,
- der zuvor angehaltene Prozess wird wieder gestartet,
- dabei wird ihm ein Rückgabewert mit dem Ergebnis des Systemaufrufs zur Verfügung gestellt.

Systemaufruf durch Anwendungsprogrammierer

Der folgende JAVA-Quelltext verdeutlicht, wie Systemaufrufe durch einen Anwendungsprogrammierer getätigt werden:

Code

```
import java.io.FileWriter;
import java.io.IOException;

public class Beispiel_Systemaufrufe {
```

```
public static void main(String[] args) {
  FileWriter fw;
  try {
      // Hier folgen drei Systemaufrufe:
      fw = new FileWriter("hallo.txt");
      fw.write("Hallo Systemaufruf!");
      fw.close();
  } catch (IOException e) {
    System.out.println(
      "Es ist ein Fehler aufgetreten: \r\n"
      + e.toString()
    );
  }
}
}
```

Innerhalb des try-Blocks der try-catch-Anweisung

- wird eine Datei zum Schreiben geöffnet,
- wird der Text *Hallo Systemaufruf!* in die geöffnete Datei geschrieben,
- wird die Datei wieder geschlossen.

In jeder dieser drei Zeilen findet ein Systemaufruf statt. Es wird dabei jeweils

- eine von der Java-Laufzeitumgebung bereitgestellte Methode aufgerufen,
- diese Methode ruft intern eine vom Betriebssystem bereitgestellte Funktion auf,
- diese Funktion tätigt den betreffenden Systemaufruf,
- und liefert ggf. einen Rückgabewert bzw. im Fehlerfall eine IOException zurück.

Wichtig

Der Anwendungsprogrammierer muss über Systemaufrufe also gar nichts Näheres wissen. Er nutzt in seinem Quelltext lediglich bereitgestellte Methoden, alles weitere geschieht im Hintergrund.

Aufgabe 1

Aufgabe

Aufg. 71: Liste von Systemaufrufen

Recherchiere im Internet und finde eine Liste mit Systemaufrufen für ein beliebiges Betriebssystem.

Aufgabe 2

Aufgabe

Aufg. 72: Wo ist der Systemaufruf?

Die folgende Abbildung ist ja bereits aus dem Kapitel Zwischen Benutzer und Hardware bekannt.

- Welcher Pfeil in der Abbildung symbolisiert einen Systemaufruf?
- Und wofür steht der Pfeil in Gegenrichtung auf derselben Ebene?

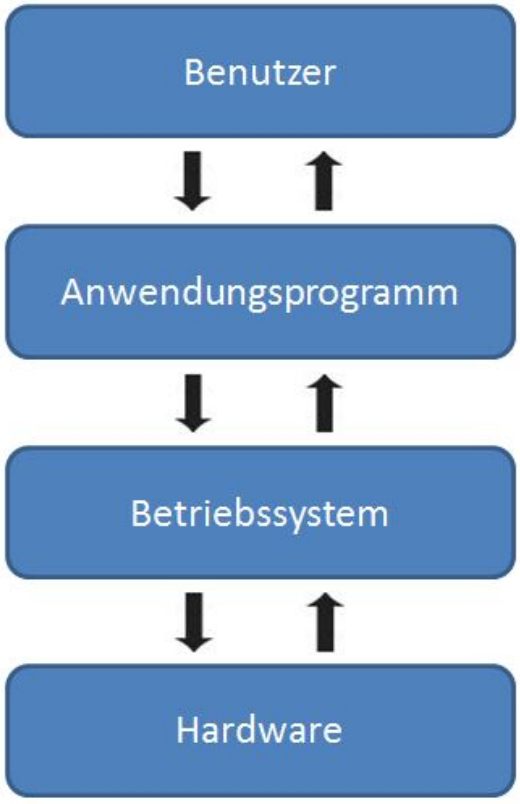


Abb. 53:
CC-BY

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2 Prozessverwaltung

So geht es weiter:

Gliederung

3.2 Prozessverwaltung
3.2.1 Prozess
3.2.2 Prozesskontext
3.2.3 Kontextwechsel
3.2.4 Prozesse erzeugen
3.2.5 Prozesskontrollblock
3.2.6 Prozesstabelle
3.2.7 Prozesszustände
3.2.8 Verwalten von Prozessen
3.2.9 Threads
3.2.10 Scheduling
3.2.11 Synchronisation
3.2.12 Deadlocks
3.2.13 Interprozesskommunikation

Alternative Webquelle zum Thema

Quelle im Web

Operating Systems: Processes

http://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/3_Processes.html

Dr. John T. Bell

Department of Computer Science

University of Illinois, Chicago

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.1 Prozess

Prozess ist der wohl zentralste Begriff beim Thema *Betriebssysteme*. Es wird daher die folgende, aus dem Kapitel Vom Programm zum Prozess bereits bekannte, Definition noch einmal wiederholt: </p>

Definition: Prozess

Definition

Ein **Prozess** ist ein Programm in Ausführung.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.2 Prozesskontext

Definition: Prozesskontext

Unter dem **Prozesskontext** oder **Kontext eines Prozesses** versteht man die Gesamtheit aller Informationen, die der betreffende Prozess während seiner Ausführung auf der CPU benötigt.

Zum Kontext eines Prozesses gehören damit unter anderem:

- Die Werte in den betreffenden Registern der CPU (Program Counter, Instruction Register, Stack Register, Flags, etc.).
- Die Belegung des Caches mit Befehlen und Daten des Prozesses.
- Die Belegung des Hauptspeichers mit Programmtext und Daten des Prozesses.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.3 Kontextwechsel

Zunächst die Definition: </p>

Definition: Kontextwechsel

Unter einem **Kontextwechsel** (engl. context switch) oder genauer einem **Prozess-Kontextwechsel** auf der CPU versteht man alle erforderlichen Tätigkeiten, um einen gerade auf der CPU aktiven Prozess A, durch einen anderen Prozess B zu ersetzen.

Diese Tätigkeiten werden vom Steuerwerk der CPU in Zusammenarbeit mit dem Betriebssystem durchgeführt.

Erforderliche Tätigkeiten

Zu den erforderlichen Tätigkeiten gehören unter anderem:

- Sichere alle notwendigen Registerinformationen des scheidenden Prozesses A an einer bekannten Stelle (damit sie von dort später wiederhergestellt werden können).

- Lade alle notwendigen Registerinformationen des neuen Prozesses B in die entsprechenden Register auf der CPU.
- Lade alle notwendigen Befehle und Daten des neuen Prozesses B in den Cache.

Es ist leicht verständlich, dass jeder Kontextwechsel eine gewisse Zeit für seine Durchführung beansprucht.

Aufgabe 1

Aufgabe

Aufg. 73: Sind Kontextwechsel positiv oder negativ?

Diskutiere in deiner Lerngruppe:

- Ist die in einen (bzw. mehrere) Kontextwechsel investierte Zeit positiv oder negativ im Hinblick auf die Bedürfnisse eines Anwenders an das Gesamtsystem?
- Gibt es verschiedene "Arten von Anwendern", die hier berücksichtigt werden sollten?
  - Welche Arten fallen dir ein?
  - Welche positiven bzw. negativen Aspekte sind für die einzelnen Anwenderarten zu unterscheiden?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.3.1 Die Statistik der Kontextwechsel unter Windows

Um einmal ein Gefühl für Kontextwechsel zu bekommen, insbesondere im Hinblick auf deren Häufigkeit, zeigt das folgende Video einige Hintergründe unter einem handelsüblichen Microsoft Windows 7. </p>

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/43yVqPTmsLo

Med. 25: Kontextwechsel unter Windows 7

http://youtu.be/43yVqPTmsLo

CC-BY

Aufgabe 1

Aufgabe

Aufg. 74: Perfmon unter Windows

Arbeitest du auch gerade mit einem Windows-Betriebssystem?

Dann dürfte auch bei dir das Systemprogramm perfmon.exe verfügbar sein.

(*Perfmon* ist übrigens die Abkürzung für *Performance Monitor*.)

- Starte perfmon.exe (wie im Video gezeigt) und finde heraus, wieviele Kontextwechsel pro Sekunde gerade auf deinem System stattfinden!
- Wie verändert sich die Anzahl der Kontextwechsel pro Sekunde, wenn du mehrere Programme startest und gleichzeitig ein YouTube-Video abspielst oder beispielsweise einen Radio-Stream über den Browser abrufst?

Falls du gerade kein Windows-System zur Hand hast, brauchst du diese Aufgabe natürlich nicht zu bearbeiten ;-)

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.3.2 Die Statistik der Kontextwechsel unter Linux

Auch Linux bietet (mindestens) eine Möglichkeit, um die Anzahl der Kontextwechsel pro Sekunde statistisch zu erfassen. Dies geschieht mit Hilfe des *vmstat*-Kommandos. </p>

Ein Aufruf von

vmstat 1 10

sorgt beispielsweise dafür, dass alle *1* Sekunden eine aktuelle Statistik ausgegeben wird, die Ausgabe wird *10* mal wiederholt.

```
aw@linux:~$  vmstat 1 10
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 1  1    560  34464  79892 379260    0    0     0     5    1    7  0  0 99  0
 0  0    560  34456  79900 379284    0    0     0   184    8   25  0  0 94  6
 0  0    560  34456  79900 379284    0    0     0     0    3    7  0  0 100  0
 0  0    560  34456  79900 379284    0    0     0     0    6   15  0  0 100  0
 0  0    560  34456  79900 379284    0    0     0     0    5    9  0  0 100  0
 0  0    560  34456  79900 379284    0    0     0     0    4   13  0  0 100  0
 0  0    560  34456  79900 379284    0    0     0     0    4    7  0  0 100  0
 0  0    560  34456  79900 379284    0    0     0     0    5   15  0  0 100  0
 0  0    560  34456  79900 379284    0    0     0     0    3    9  0  0 100  0
 0  0    560  34456  79900 379284    0    0     0     0    3   15  0  0 100  0
```

In der Spalte *cs* unter *system* ist die aktuelle Zahl an Kontextwechseln pro Sekunde protokolliert (cs = context switch).

Quelle im Web

Weiterführende Informationen zum *vmstat*-Kommando liefert die zugehörige Manpage:
http://unixhelp.ed.ac.uk/CGI/man-cgi?vmstat

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.4 Prozesse erzeugen

Während der Laufzeit eines Betriebssystems werden ständig Prozesse erzeugt, abgearbeitet, unterbrochen, weiter abgearbeitet und irgendwann schließlich beendet. </p>

Prozess-ID

Jeder Prozess erhält zur Unterscheidung und Verwaltung direkt bei seiner Erzeugung eine eindeutige Prozess-ID (engl.: process identifier, kurz: PID).
Diese PID ist üblicherweise eine ganze Zahl, größergleich Null.

Erzeugen des ersten Prozesses

Wird der Rechner eingeschaltet, so startet üblicherweise das Betriebssystem. Dabei muss es ganz zu Beginn einen Mechanismus geben, welcher den ersten Prozess des Betriebssystems erzeugt. Dieser erhält (zumindest in der Theorie) die Prozess-ID 0 (Null).

Ausgehend vom ersten Prozess starten dann i.d.R. weitere Prozesse, die zum Betriebssystem gehören, und zur Erfüllung der Zentralen Aufgabe eines Betriebssystems benötigt werden.

Später kann der Anwender oder Administrator weitere Anwendungsprogramme starten, die jeweils als eigener Prozess vom Betriebssystem verwaltet werden.

Beispiel

Unter Unix-orientierten Betriebssystemen (wie z.B. Linux) ist der erste Prozess üblicherweise der *init*-Prozess. Dieser erhält die PID 1. Die PID 0 wird hier nicht vergeben.

Unter Windows gibt es einen *System-Idle*-Prozess mit der PID 0. In deutschen Windows-Versionen heißt dieser *Leerlaufprozess*. Dieser Prozess hat einen speziellen Hintergrund: Wenn kein anderer Prozess im Betriebssytem zur Ausführung bereit ist, wird der Leerlaufprozess ausgeführt. Er nutzt damit ungenutze Rechenzeit auf der CPU.

Allgemeine Erzeugung eines Prozesses

Ganz allgemein hängt es vom Betriebssystem ab, wie genau die Erzeugung eines Prozesses erfolgt. Unter Unix/Linux gibt es dazu einen Systemaufruf *fork*, unter Windows heißt dieser Systemaufruf *CreateProcess*. Beide Varianten arbeiten sehr unterschiedlich.

So geht es weiter:

Gliederung

3.2.4 Prozesse erzeugen
3.2.4.1 Fork
3.2.4.2 CreateProcess

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.4.1 Fork

Der Systemaufruf *fork* sorgt dafür, dass vom aufrufenden Prozess (Elternprozess) eine exakte Kopie (Kindprozess) erzeugt wird. Der Kindprozess erhält eine eigene Prozess-ID, übernimmt sonst aber alle Informationen des Elternprozesses: Programmtext, Datensegment, Befehlszähler, etc. </p>

Prof. Dr. Carsten Vogt von der TH Köln hat ein Video zum *fork*-Systemaufruf bereitgestellt:

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/aw3u8b3iAxg

Med. 26: UNIX/Linux: Prozesserzeugung mit fork() (12:19)
https://youtu.be/aw3u8b3iAxg

Der Kindprozess wird direkt nach dem *fork* unabhängig vom Elternprozess als eigenständige Instanz auf dem System verwaltet und ausgeführt. Sowohl Eltern- als auch Kindprozess laufen nach dem "klonen" an gleicher Stelle weiter: Der Rückgabewert von *fork* wird geliefert und die direkt auf das *fork* folgende Anweisung wird ausgeführt.

Beispiel

Unter http://de.wikipedia.org/wiki/Fork_%28Unix%29 findet sich ein Beispiel, welches den Aufruf der fork-Funktion zeigt, und bei dem anhand des Rückgabewertes entschieden wird, ob man sich im Elternprozess, oder im Kindprozess befindet.

- Rückgabewert von *fork* > 0: Elternprozess (Der Wert ist die PID des Kindprozesses.)
- Rückgabewert von *fork* = 0: Kindprozess
- Rückgabewert von *fork* < 0: Es ist ein Fehler aufgetreten.

Aufgabe 1

Aufgabe

Aufg. 75: Vater- und Sohn-Variable

Prof. Dr. Carsten Vogt von der TH Köln zeigt und erläutert in seinem Video , wie eine Variable durch ein *fork* sowohl im Vater-, als auch im Sohn-Prozess existiert und unabhängig voneinander mit Werten belegt sein kann.

Schau es dir an!

Aufgabe 2

Aufgabe

Aufg. 76: fork gleich mehrmals

Was passiert eigentlich bei mehrmaligem Aufruf von *fork*?

Betrachte den C-Quelltext aus Listing 1 unten. Was denkst Du: Wieviele Prozesse werden durch die Ausführung des Programms insgesamt erzeugt?

Falls du vorhast den Quelltext auf einem Unix-/Linux-System zu compillieren und auszuführen:

- Erst nachdenken und eine Prognose notieren!
- Dann die Prognosen innerhalb deiner Lerngruppe vergleichen und diskutieren.
- Am Ende schließlich compillieren, ausführen und die Prognose überprüfen.

</> Code

```
1. include <stdio.h>
2. include <unistd.h>
int main ()
{
  int pid;
  pid = fork();
  pid = fork();
```

```
    pid = fork();
    pid = fork();

    printf("Hier ist ein Prozess mit ID %d!\n", getpid());
    return 0;
}

<hr>
```

Listing 1: fork mehrmals nacheinander

Aufgabe 3

Aufgabe

Aufg. 77: Fork und DMA?

Könnte bei Ausführung des Systemaufrufs fork auch der DMA-Controller zum Einsatz kommen? Erläutere warum das Sinn macht!

Aufgabe 4

Aufgabe

Aufg. 78: Ein anderes Programm mittels fork starten

Schaust du dir die Unterschiede zwischen *fork* und *CreateProcess* genauer an, so fällt auf, dass bei *CreateProcess* eine andere Anwendung gestartet wird, während *fork* lediglich eine Kopie eines bereits existierenden Prozesses erzeugt.

Wie kann mittels *fork* ein anderes Programm gestartet werden?
Erläutere mit deinen eigenen Worten die Vorgehensweise dabei!

Achilles 2006 gibt Hinweise dazu in Kapitel 3.2.1 und 3.2.2. (Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.)

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.4.2 CreateProcess

Der Systemaufruf *CreateProcess* unter Windows läuft sehr viel komplexer ab, als ein *fork* unter Unix/Linux. *Solomon et.al. 2006* beschreiben sechs Hauptphasen bei der Prozesserzeugung.

- Phase 1 beginnt mit dem Öffnen der EXE-Datei.
- Die Phasen 2 und 3 erstellen notwendige Verwaltungsobjekte.
- Phase 4 informiert das Windows-Subsystem über den hier neu erstellten Prozess.
- Phase 5 leitet die Ausführung des neu erstellten Prozesses ein.
- Phase 6 nimmt alle abschließenden Initialisierungen vor.

</p>
Anschließend ist der neue Prozess komplett erstellt und wartet auf die Zuteilung der CPU.

Hinweis

Eine detailliertere Beschreibung der einzelnen Phasen findet sich bei *Solomon et.al. 2006* . Der geneigte Leser erhält dort weitere Informationen.

Microsoft stellt unter

http://msdn.microsoft.com/en-us/library/windows/desktop/ms682425%28v=vs.85%29.aspx

Informationen zum Funktionsaufruf von *CreateProcess* bereit und erläutert die zahlreichen Übergabeparameter.

Weiterhin gibt Microsoft unter

http://msdn.microsoft.com/en-us/library/windows/desktop/ms682512%28v=vs.85%29.aspx

ein sehr einfach gehaltenes Beispielprogramm an, welches den Systemaufruf *CreateProcess* durchführt. Ein Anwender mit Windows-Erfahrung kann sich so sicher leicht vorstellen, wie ein Doppelklick auf ein Programmsymbol (oder alternativ auf eine EXE-Datei) zum Aufruf der *CreateProcess*-Funktion führt.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.5 Prozesskontrollblock

Definition: Prozesskontrollblock

Definition

In einem **Prozesskontrollblock** (**Process control block**, kurz **PCB**) fasst das Betriebssystem alle zu einem einzelnen Prozess gehörenden Informationen zusammen.

Sobald also ein neuer Prozess erzeugt wird, legt das Betriebssystem dafür einen Prozesskontrollblock als Verwaltungsstruktur an. Für jeden Prozess existiert somit ein eigener PCB.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.5.1 Prozesskontrollblock unter Windows

Da Windows zur Gattung der "Closed-Source"-Software gehört, ist der Prozesskontrollblock unter Windows nicht offiziell veröffentlicht. </p>
Russinovich et.al. 2012a beschreiben, dass jeder unter Windows erzeugter Prozess durch eine Instanz der Datenstruktur EPROCESS (engl.: executive process structure) repräsentiert wird. Sie geben in einer Übersicht auch die wichtigsten Felder dieser Datenstruktur an, ohne jedoch deren komplette Definition offenzulegen.

Microsoft veröffentlicht nur recht allgemein gehaltene Informationen zu EPROCESS:
http://msdn.microsoft.com/en-us/library/windows/hardware/ff544273%28v=vs.85%29.aspx

Debugging Experiment

Interessant ist, dass *Russinovich et.al. 2012a* als Experiment vorschlagen, die einzelnen Felder der EPROCESS-Datenstruktur mit Hilfe des Windows Kernel Debuggers (WinDbg) zu identifizieren.

Diese Arbeit gemacht hat sich nach eigener Dokumentation der Software Entwickler Nir Sofer . Für Windows Vista hat er verschiedene Datenstrukturen rekonstruiert.

EPROCESS im C/C++-Format:
http://www.nirsoft.net/kernel_struct/vista/EPROCESS.html

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.5.2 Prozesskontrollblock unter Linux

Das folgende Videos zeigt, wie man in den Quelltexten des Linux-Kernels die Deklaration des Linux-Prozesskontrollblocks (*task_struct*) findet, und wie der Zusammenhang mit der Prozesstabelle ist. </p>

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/QeerwZOO9bw

Med. 27: Prozesskontrollblock im Linux-Quellcode
http://youtu.be/QeerwZOO9bw
CC-BY

task_struct: Deklaration des Linux-Prozesskontrollblocks

Der folgende Auszug aus der Quelltext-Datei *sched.h* des Linux-Kernels (Version 3.13.0) zeigt die Deklaration der Datenstruktur *task_struct*. Dabei handelt es sich um den Prozesskontrollblock, wie er von Linux verwendet wird.

```
 1 struct task_struct {
 2     volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */
 3     void *stack;
 4     atomic_t usage;
 5     unsigned int flags;    /* per process flags, defined below */
 6     unsigned int ptrace;
 7
 8 #ifdef CONFIG_SMP
 9     struct llist_node wake_entry;
10     int on_cpu;
11     struct task_struct *last_wakee;
12     unsigned long wakee_flips;
13     unsigned long wakee_flip_decay_ts;
14
15     int wake_cpu;
16 #endif
17     int on_rq;
18
19     int prio, static_prio, normal_prio;
20     unsigned int rt_priority;
21     const struct sched_class *sched_class;
22     struct sched_entity se;
23     struct sched_rt_entity rt;
24 #ifdef CONFIG_CGROUP_SCHED
25     struct task_group *sched_task_group;
26 #endif
27
28 #ifdef CONFIG_PREEMPT_NOTIFIERS
29     /* list of struct preempt_notifier: */
30     struct hlist_head preempt_notifiers;
31 #endif
32
33 #ifdef CONFIG_BLK_DEV_IO_TRACE
34     unsigned int btrace_seq;
35 #endif
36
37     unsigned int policy;
38     int nr_cpus_allowed;
39     cpumask_t cpus_allowed;
```

```
40
41 #ifdef CONFIG_PREEMPT_RCU
42     int rcu_read_lock_nesting;
43     char rcu_read_unlock_special;
44     struct list_head rcu_node_entry;
45 #endif /* #ifdef CONFIG_PREEMPT_RCU */
46 #ifdef CONFIG_TREE_PREEMPT_RCU
47     struct rcu_node *rcu_blocked_node;
48 #endif /* #ifdef CONFIG_TREE_PREEMPT_RCU */
49 #ifdef CONFIG_RCU_BOOST
50     struct rt_mutex *rcu_boost_mutex;
51 #endif /* #ifdef CONFIG_RCU_BOOST */
52
53 #if defined(CONFIG_SCHEDSTATS) || defined(CONFIG_TASK_DELAY_ACCT)
54     struct sched_info sched_info;
55 #endif
56
57     struct list_head tasks;
58 #ifdef CONFIG_SMP
59     struct plist_node pushable_tasks;
60 #endif
61
62     struct mm_struct *mm, *active_mm;
63 #ifdef CONFIG_COMPAT_BRK
64     unsigned brk_randomized:1;
65 #endif
66 #if defined(SPLIT_RSS_COUNTING)
67     struct task_rss_stat    rss_stat;
68 #endif
69 /* task state */
70     int exit_state;
71     int exit_code, exit_signal;
72     int pdeath_signal; /* The signal sent when the parent dies */
73     unsigned int jobctl;    /* JOBCTL_*, siglock protected */
74
75     /* Used for emulating ABI behavior of previous Linux versions */
76     unsigned int personality;
77
78     unsigned did_exec:1;
79     unsigned in_execve:1;    /* Tell the LSMs that the process is doing an
80                   * execve */
81     unsigned in_iowait:1;
82
83     /* task may not gain privileges */
84     unsigned no_new_privs:1;
85
86     /* Revert to default priority/policy when forking */
87     unsigned sched_reset_on_fork:1;
88     unsigned sched_contributes_to_load:1;
89
90     pid_t pid;
91     pid_t tgid;
92
93 #ifdef CONFIG_CC_STACKPROTECTOR
94     /* Canary value for the -fstack-protector gcc feature */
95     unsigned long stack_canary;
96 #endif
```

```
 97     /*
 98      * pointers to (original) parent process, youngest child, younger sibling,
 99      * older sibling, respectively. (p->father can be replaced with
100      * p->real_parent->pid)
101      */
102     struct task_struct __rcu *real_parent; /* real parent process */
103     struct task_struct __rcu *parent; /* recipient of SIGCHLD, wait4() reports
*/
104     /*
105      * children/sibling forms the list of my natural children
106      */
107     struct list_head children;    /* list of my children */
108     struct list_head sibling;    /* linkage in my parent's children list */
109     struct task_struct *group_leader;    /* threadgroup leader */
110
111     /*
112      * ptraced is the list of tasks this task is using ptrace on.
113      * This includes both natural children and PTRACE_ATTACH targets.
114      * p->ptrace_entry is p's link on the p->parent->ptraced list.
115      */
116     struct list_head ptraced;
117     struct list_head ptrace_entry;
118
119     /* PID/PID hash table linkage. */
120     struct pid_link pids[PIDTYPE_MAX];
121     struct list_head thread_group;
122     struct list_head thread_node;
123
124     struct completion *vfork_done;        /* for vfork() */
125     int __user *set_child_tid;        /* CLONE_CHILD_SETTID */
126     int __user *clear_child_tid;        /* CLONE_CHILD_CLEARTID */
127
128     cputime_t utime, stime, utimescaled, stimescaled;
129     cputime_t gtime;
130 #ifndef CONFIG_VIRT_CPU_ACCOUNTING_NATIVE
131     struct cputime prev_cputime;
132 #endif
133 #ifdef CONFIG_VIRT_CPU_ACCOUNTING_GEN
134     seqlock_t vtime_seqlock;
135     unsigned long long vtime_snap;
136     enum {
137         VTIME_SLEEPING = 0,
138         VTIME_USER,
139         VTIME_SYS,
140     } vtime_snap_whence;
141 #endif
142     unsigned long nvcsw, nivcsw; /* context switch counts */
143     struct timespec start_time;         /* monotonic time */
144     struct timespec real_start_time;    /* boot based time */
145 /* mm fault and swap info: this can arguably be seen as either mm-specific or
thread-specific */
146     unsigned long min_flt, maj_flt;
147
148     struct task_cputime cputime_expires;
149     struct list_head cpu_timers[3];
150
151 /* process credentials */
```

```
152     const struct cred __rcu *real_cred; /* objective and real subjective task
153                      * credentials (COW) */
154      const struct cred __rcu *cred;     /* effective (overridable) subjective
task
155                      * credentials (COW) */
156     char comm[TASK_COMM_LEN]; /* executable name excluding path
157                   - access with [gs]et_task_comm (which lock
158                   it with task_lock())
159                   - initialized normally by setup_new_exec */
160 /* file system info */
161     int link_count, total_link_count;
162 #ifdef CONFIG_SYSVIPC
163 /* ipc stuff */
164     struct sysv_sem sysvsem;
165 #endif
166 #ifdef CONFIG_DETECT_HUNG_TASK
167 /* hung task detection */
168     unsigned long last_switch_count;
169 #endif
170 /* CPU-specific state of this task */
171     struct thread_struct thread;
172 /* filesystem information */
173     struct fs_struct *fs;
174 /* open file information */
175     struct files_struct *files;
176 /* namespaces */
177     struct nsproxy *nsproxy;
178 /* signal handlers */
179     struct signal_struct *signal;
180     struct sighand_struct *sighand;
181
182     sigset_t blocked, real_blocked;
183      sigset_t saved_sigmask;     /* restored if set_restore_sigmask() was used
*/
184     struct sigpending pending;
185
186     unsigned long sas_ss_sp;
187     size_t sas_ss_size;
188     int (*notifier)(void *priv);
189     void *notifier_data;
190     sigset_t *notifier_mask;
191     struct callback_head *task_works;
192
193     struct audit_context *audit_context;
194 #ifdef CONFIG_AUDITSYSCALL
195     kuid_t loginuid;
196     unsigned int sessionid;
197 #endif
198     struct seccomp seccomp;
199
200 /* Thread group tracking */
201      u32 parent_exec_id;
202      u32 self_exec_id;
203 /* Protection of (de-)allocation: mm, files, fs, tty, keyrings, mems_allowed,
204  * mempolicy */
205     spinlock_t alloc_lock;
206
```

```
207     /* Protection of the PI data structures: */
208     raw_spinlock_t pi_lock;
209 
210 #ifdef CONFIG_RT_MUTEXES
211     /* PI waiters blocked on a rt_mutex held by this task */
212     struct plist_head pi_waiters;
213     /* Deadlock detection and priority inheritance handling */
214     struct rt_mutex_waiter *pi_blocked_on;
215 #endif
216 
217 #ifdef CONFIG_DEBUG_MUTEXES
218     /* mutex deadlock detection */
219     struct mutex_waiter *blocked_on;
220 #endif
221 #ifdef CONFIG_TRACE_IRQFLAGS
222     unsigned int irq_events;
223     unsigned long hardirq_enable_ip;
224     unsigned long hardirq_disable_ip;
225     unsigned int hardirq_enable_event;
226     unsigned int hardirq_disable_event;
227     int hardirqs_enabled;
228     int hardirq_context;
229     unsigned long softirq_disable_ip;
230     unsigned long softirq_enable_ip;
231     unsigned int softirq_disable_event;
232     unsigned int softirq_enable_event;
233     int softirqs_enabled;
234     int softirq_context;
235 #endif
236 #ifdef CONFIG_LOCKDEP
237 # define MAX_LOCK_DEPTH 48UL
238     u64 curr_chain_key;
239     int lockdep_depth;
240     unsigned int lockdep_recursion;
241     struct held_lock held_locks[MAX_LOCK_DEPTH];
242     gfp_t lockdep_reclaim_gfp;
243 #endif
244 
245 /* journalling filesystem info */
246     void *journal_info;
247 
248 /* stacked block device info */
249     struct bio_list *bio_list;
250 
251 #ifdef CONFIG_BLOCK
252 /* stack plugging */
253     struct blk_plug *plug;
254 #endif
255 
256 /* VM state */
257     struct reclaim_state *reclaim_state;
258 
259     struct backing_dev_info *backing_dev_info;
260 
261     struct io_context *io_context;
262 
263     unsigned long ptrace_message;
```

```
264     siginfo_t *last_siginfo; /* For ptrace use. */
265     struct task_io_accounting ioac;
266 #if defined(CONFIG_TASK_XACCT)
267     u64 acct_rss_mem1;    /* accumulated rss usage */
268     u64 acct_vm_mem1;    /* accumulated virtual memory usage */
269     cputime_t acct_timexpd;    /* stime + utime since last update */
270 #endif
271 #ifdef CONFIG_CPUSETS
272     nodemask_t mems_allowed;    /* Protected by alloc_lock */
273     seqcount_t mems_allowed_seq;    /* Seqence no to catch updates */
274     int cpuset_mem_spread_rotor;
275     int cpuset_slab_spread_rotor;
276 #endif
277 #ifdef CONFIG_CGROUPS
278     /* Control Group info protected by css_set_lock */
279     struct css_set __rcu *cgroups;
280     /* cg_list protected by css_set_lock and tsk->alloc_lock */
281     struct list_head cg_list;
282 #endif
283 #ifdef CONFIG_FUTEX
284     struct robust_list_head __user *robust_list;
285 #ifdef CONFIG_COMPAT
286     struct compat_robust_list_head __user *compat_robust_list;
287 #endif
288     struct list_head pi_state_list;
289     struct futex_pi_state *pi_state_cache;
290 #endif
291 #ifdef CONFIG_PERF_EVENTS
292     struct perf_event_context *perf_event_ctxp[perf_nr_task_contexts];
293     struct mutex perf_event_mutex;
294     struct list_head perf_event_list;
295 #endif
296 #ifdef CONFIG_NUMA
297     struct mempolicy *mempolicy;    /* Protected by alloc_lock */
298     short il_next;
299     short pref_node_fork;
300 #endif
301 #ifdef CONFIG_NUMA_BALANCING
302     int numa_scan_seq;
303     unsigned int numa_scan_period;
304     unsigned int numa_scan_period_max;
305     int numa_preferred_nid;
306     int numa_migrate_deferred;
307     unsigned long numa_migrate_retry;
308     u64 node_stamp;            /* migration stamp */
309     struct callback_head numa_work;
310 
311     struct list_head numa_entry;
312     struct numa_group *numa_group;
313 
314     /*
315      * Exponential decaying average of faults on a per-node basis.
316      * Scheduling placement decisions are made based on the these counts.
317      * The values remain static for the duration of a PTE scan
318      */
319     unsigned long *numa_faults;
320     unsigned long total_numa_faults;
```

```
321
322     /*
323      * numa_faults_buffer records faults per node during the current
324      * scan window. When the scan completes, the counts in numa_faults
325      * decay and these values are copied.
326      */
327     unsigned long *numa_faults_buffer;
328
329     /*
330      * numa_faults_locality tracks if faults recorded during the last
331      * scan window were remote/local. The task scan period is adapted
332      * based on the locality of the faults with different weights
333      * depending on whether they were shared or private faults
334      */
335     unsigned long numa_faults_locality[2];
336
337     unsigned long numa_pages_migrated;
338 #endif /* CONFIG_NUMA_BALANCING */
339
340     struct rcu_head rcu;
341
342     /*
343      * cache last used pipe for splice
344      */
345     struct pipe_inode_info *splice_pipe;
346
347     struct page_frag task_frag;
348
349 #ifdef    CONFIG_TASK_DELAY_ACCT
350     struct task_delay_info *delays;
351 #endif
352 #ifdef CONFIG_FAULT_INJECTION
353     int make_it_fail;
354 #endif
355     /*
356      * when (nr_dirtied >= nr_dirtied_pause), it's time to call
357      * balance_dirty_pages() for some dirty throttling pause
358      */
359     int nr_dirtied;
360     int nr_dirtied_pause;
361     unsigned long dirty_paused_when; /* start of a write-and-pause period */
362
363 #ifdef CONFIG_LATENCYTOP
364     int latency_record_count;
365     struct latency_record latency_record[LT_SAVECOUNT];
366 #endif
367     /*
368      * time slack values; these are used to round up poll() and
369      * select() etc timeout values. These are in nanoseconds.
370      */
371     unsigned long timer_slack_ns;
372     unsigned long default_timer_slack_ns;
373
374 #ifdef CONFIG_FUNCTION_GRAPH_TRACER
375     /* Index of current stored address in ret_stack */
376     int curr_ret_stack;
377     /* Stack of return addresses for return function tracing */
```

```
378     struct ftrace_ret_stack    *ret_stack;
379     /* time stamp for last schedule */
380     unsigned long long ftrace_timestamp;
381     /*
382      * Number of functions that haven't been traced
383      * because of depth overrun.
384      */
385     atomic_t trace_overrun;
386     /* Pause for the tracing */
387     atomic_t tracing_graph_pause;
388 #endif
389 #ifdef CONFIG_TRACING
390     /* state flags for use by tracers */
391     unsigned long trace;
392     /* bitmask and counter of trace recursion */
393     unsigned long trace_recursion;
394 #endif /* CONFIG_TRACING */
395 #ifdef CONFIG_MEMCG /* memcg uses this to do batch job */
396     struct memcg_batch_info {
397         int do_batch;    /* incremented when batch uncharge started */
398         struct mem_cgroup *memcg; /* target memcg of uncharge */
399         unsigned long nr_pages;    /* uncharged usage */
400         unsigned long memsw_nr_pages; /* uncharged mem+swap usage */
401     } memcg_batch;
402     unsigned int memcg_kmem_skip_account;
403     struct memcg_oom_info {
404         struct mem_cgroup *memcg;
405         gfp_t gfp_mask;
406         int order;
407         unsigned int may_oom:1;
408     } memcg_oom;
409 #endif
410 #ifdef CONFIG_UPROBES
411     struct uprobe_task *utask;
412 #endif
413 #if defined(CONFIG_BCACHE) || defined(CONFIG_BCACHE_MODULE)
414     unsigned int    sequential_io;
415     unsigned int    sequential_io_avg;
416 #endif
417 };
```

Das folgende Bild wurde im Video erläutert:

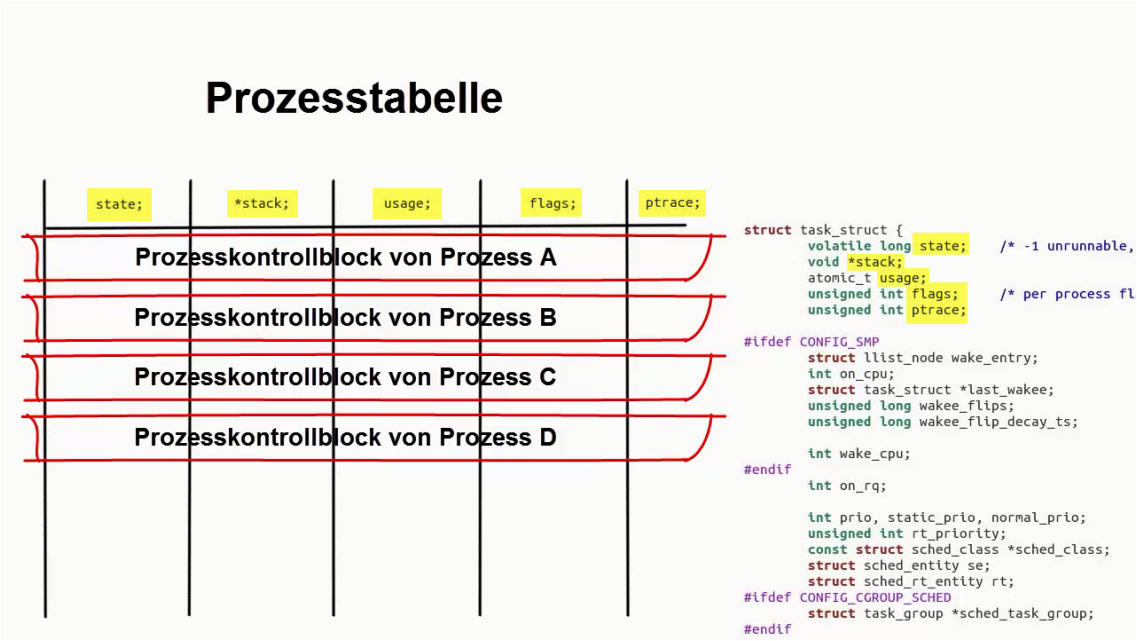


Abb. 54:
Der Zusammenhang zwischen task_struct und den Spalten der Prozesstabelle
CC-BY

Aufgabe 1

Aufgabe

Aufg. 79: Die Spalten der Prozesstabelle

Im Video wurde der Zusammenhang zwischen der Datenstruktur *task_struct* und den Spalten der Prozesstabelle erläutert.

Was schätzt du:
Aus wievielen Spalten besteht in etwa die Prozesstabelle?

- Aus 5 bis 10 Spalten.
- Aus 25 bis 30 Spalten.
- Aus mehr als 50 Spalten.

Hinweis

Aus wievielen Zeilen Quelltext besteht die Deklaration des *task_struct*?

Beispiel: Prozesskontrollblock unter Linux

Hinweis

Weiterführende Literatur

Achilles 2006 zeigt in Kapitel 3.1 den *Linux Process Control Block*. Die Lektüre dieser Quelle sei ausdrücklich empfohlen.

> Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.6 Prozesstabelle

Definition: Prozesstabelle

Definition

> In der Prozesstabelle fasst das Betriebssystem alle Informationen aller erzeugter Prozesse zusammen.

In der Praxis kann die Prozesstabelle ganz einfach als Liste aller Prozesskontrollblöcke realisiert werden.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.7 Prozesszustände

Wird ein Prozess innerhalb eines Betriebssystems erzeugt, so bedeutet dies noch nicht, dass er auch sofort auf der CPU ausgeführt wird. Vielmehr werden verschiedene Zustände eines Prozesses unterschieden. </p>
Die einfachste Betrachtungsweise geht von nur drei möglichen Prozesszuständen aus: *Bereit*, *Rechnend* und *Blockiert*. Die folgende Abbildung zeigt zusätzlich die mögliche Übergänge zwischen diesen Zuständen:

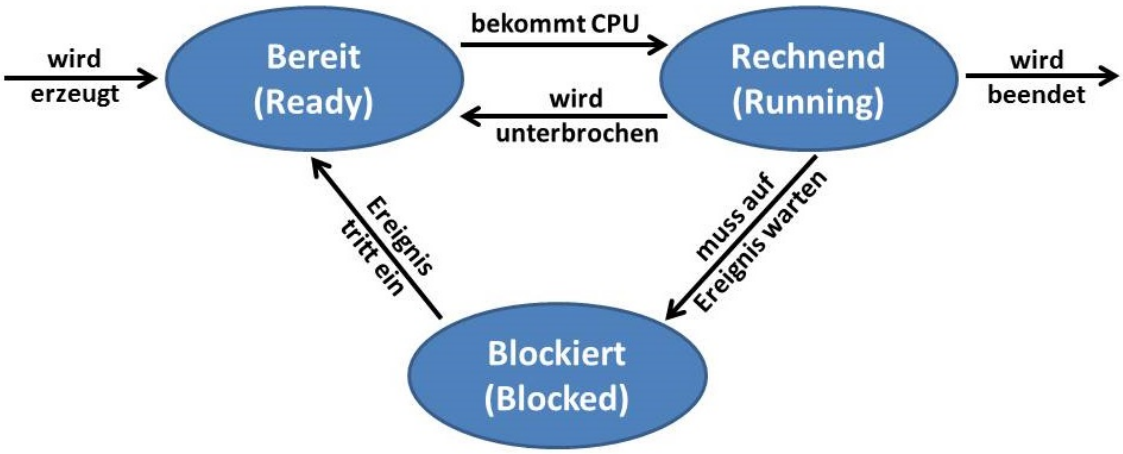


Abb. 55: Prozesszustände mit Übergängen
CC-BY

Hinweis

Sowohl Unix/Linux, als auch Windows definieren für ihre Prozesse weitere Zustände, beide unterscheiden sich darin sehr. Dies wird im Rahmen dieses Lernmoduls aber nicht näher betrachtet.

Direkt nach seiner Erzeugung befindet sich jeder Prozess im Zustand *Bereit* und wartet auf die Zuteilung der CPU. Sobald er die CPU bekommt, wechselt er in den Zustand *Rechnend*.

Aufgrund der im Kapitel Nur ein Prozessor mit einem Kern erläuterten Voraussetzung kann sich immer nur ein einziger Prozess zur Zeit im Zustand *Rechnend* befinden.

Einem Prozess im Zustand *Rechnend* kann das Betriebsmittel CPU auch wieder entzogen werden, er wechselt dann zurück in den Zustand *Bereit*.

Ein Prozess wechselt vom Zustand *Rechnend* in den Zustand *Blockiert*, wenn er einen Befehl ausführt, dessen Ergebnis noch etwas auf sich warten lässt. Beispielsweise sind E/A-Geräte im Vergleich zur CPU nur sehr langsam arbeitende Komponenten. Ein E/A-Befehl zur Kommunikation mit einem E/A-Gerät wird deshalb oftmals einen Prozess "blockieren".

Tritt das gewünschte Ereignis ein, meldet also beispielsweise das zuvor angesprochene E/A-Gerät, dass das Ergebnis des gewünschten Befehls nun verfügbar ist, so wechselt der betreffende Prozess vom Zustand *Blockiert* in den Zustand *Bereit*. Hier wartet er wieder auf die Zuteilung der CPU.

Aufgabe 1

Aufgabe

Aufg. 80: Merke dir den Zustand!

Wo wird festgehalten, in welchem Zustand sich ein Prozess aktuell befindet?
Wie (bzw. wo) verwaltet das Betriebssystem also diese Information?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.8 Verwalten von Prozessen

Eine der Aufgaben eines Betriebssystems ist ja die Verwaltung der erzeugten Prozesse. Dabei kommen Prozesskontrollblock und Prozesstabelle zum Einsatz. </p>

So geht es weiter:

Gliederung

3.2.8 Verwalten von Prozessen
3.2.8.1 Prozessverwaltung aus Admin-Sicht unter Windows
3.2.8.2 Prozessverwaltung aus Admin-Sicht unter Linux

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.8.1 Prozessverwaltung aus Admin-Sicht unter Windows

Das folgende Video zeigt Tools unter Windows 7, mit denen sich der Administrator die aktuell gestarteten Prozesse ansehen kann. Auch viele Informationen aus dem Prozesskontrollblock eines Prozesses werden hier zur Laufzeit des Prozesses sichtbar. </p>

Video

An dieser Stelle befindet sich online ein YouTube-Video.
https://youtu.be/QuPBAAnBCM4

Med. 28: Prozessverwaltung unter Windows
http://youtu.be/QuPBAAnBCM4
CC-BY

Hier der Link zu dem im Video gezeigten *Process Explorer*:
http://technet.microsoft.com/de-de/sysinternals/bb896653.aspx

Aufgabe 1

Aufgabe

Aufg. 81: Process Explorer unter Windows

Arbeitest du auch gerade mit einem Windows-Betriebssystem?
Dann lade dir den Process Explorer herunter und schaue dir an, welche Prozesse auf deinem Windows gerade ausgeführt werden.

- Starte einige neue Prozesse und beende einige alte, und sieh' dir jeweils an, welche Änderungen sich in der Anzeige des Process Explorer ergeben.
- Welche Prozess-ID hat der neu von dir gestartete Prozess zugewiesen bekommen?
- Wenn du mehrere Prozesse nacheinander startest: Welchen Zusammenhang gibt es zwischen der Startreihenfolge und der zahlenmäßigen Größe der Prozess-ID?
- Starte einen neuen Prozess (wie wäre es z.B. mit MS Paint?). Finde heraus, wie du die Prozess-ID des Elternprozesses ermitten kannst!

Falls du gerade kein Windows-System zur Hand hast, brauchst du diese Aufgabe natürlich nicht zu bearbeiten ;-)

Die Jagd kann beginnen

Aus der Praxis

'''Mal 'was Praktisches: Jagd auf Viren, Trojaner & Co '''
Für Windows-Administratoren ist der Process Explorer (neben einigen weiteren Tools) übrigens ein hervorragendes Hilfsmittel, um Viren, Trojaner und andere Schadsoftware auf einem Windows-Betriebssystem zu identifizieren, zu beobachten und zu entfernen.

Falls du auch auf Virenjagd gehen möchtest, schau dir dieses Video an:
http://channel9.msdn.com/Events/TechEd/NorthAmerica/2014/DCIM-B368#fbid=

[archivierte version: https://web.archive.org/web/20210314171402/http://channel9.msdn.com/Events/TechEd/NorthAmerica/2014/DCIM-B368]
Vortrag vom 15. Mai 2014 auf der TechEd North America 2014:
Malware Hunting with Mark Russinovich and the Sysinternals Tools

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.8.2 Prozessverwaltung aus Admin-Sicht unter Linux

Unter Linux gibt es (auf der Kommandozeile) verschiedene Programme, mit deren Hilfe der Administrator (eigentlich sagt man *Superuser* oder kurz *root*) Aufschluss über die aktuell gestarteten Prozesse bekommt. </p>

- ps
- pstree
- top
- htop

3.2.9 Threads

Threads haben sehr viel Ähnlichkeit mit Prozessen. Zunächst die Definition: </p>

Definition: Thread (Leichtgewichtiger Prozess)

Definition

Unter einem **Thread** oder **Leichtgewichtigen Prozess** versteht man einen Teil eines Prozesses, der einen unabhängigen Kontrollfluss repräsentiert.

Ein Prozess kann aus mehreren Threads bestehen, somit ergeben sich mehrere voneinander unabhängige, nebenläufige Kontrollflüsse. Vereinfachend kann man sich vorstellen, dass ein Thread so etwas wie "*ein Prozess in einem Prozess*" ist. Bei mehreren Threads innerhalb eines Prozesses hat man somit "*mehrere Prozesse innerhalb eines Prozesses*". Jedoch gibt es unterschiedliche Verfahrensweisen bei der Nutzung von Betriebsmitteln, was weiter unten auf dieser Seite noch erläutert wird.

Beispiel

Man betrachte zunächst ein einfaches Beispiel für einen Prozess mit mehreren Threads:

Beispiel

Ein Textverarbeitungsprogramm wird gestartet. Somit existiert auf dem Computersystem ein Textverarbeitungs-Prozess. Dieser Prozess startet intern mehrere Threads, die jeweils bestimmte Aufgaben übernehmen:

- Thread 1:
  realisiert den Texteditor. Er reagiert also auf Eingaben des Users mit Tastatur oder Maus.
- Thread 2:

realisiert eine "Alle 10 Minuten automatisch im Hintergrund speichern"-Funktion. D.h. dieser Thread wartet 10 Minuten, speichert dann den aktuellen Inhalt des Texteditors in einer (temporären) Datei auf der Festplatte, wartet wieder 10 Minuten, speichert wieder... usw.

- Thread 3:
  realisiert die automatische Rechtschreibprüfung. D.h. in kleinen zeitlichen Abständen werden eingegebene Wörter geprüft und ggf. rot unterstrichen.

Das Thread-Konzept wird heute von den meisten relevanten Betriebssystemen unterstützt. Threads können deshalb auf Kernel-Ebene realisiert sein. Falls das Betriebssystem keine Threads unterstützt, so können Threads immer noch auf User-Ebene realisiert werden. Denkbar sind auch Mischformen: Das Betriebssystem unterstützt Threads auf Kernel-Ebene und zusätzlich ist das Thread-Konzept noch auf User-Ebene implementiert.

Threads auf Kernel-Ebene

Unterstützt das Betriebssystem Threads, so kann auch das Betriebssystem deren Verwaltung übernehmen. Dies geschieht sehr ähnlich zur Verwaltung von Prozessen, beispielsweise existiert i.d.R. ein Threadkontrollblock (Thread Control Block, TCB), analog zum Prozesskontrollblock.

Threads auf User-Ebene

Unterstützt das Betriebssystem keine Threads, so kann auf der User-Ebene ein spezielles Programm (*Mandl 2013* nennt dies in Kapitel 4.2.2 die *Threadbibliothek*) deren Bereitstellung übernehmen. Ein Beispiel dafür ist die Java-Laufzeitumgebung (Java Runtime Environment, kurz JRE) , welche eine eigene Threadverwaltung implementiert, und somit auch Threads auf Nicht-Thread-unterstützenden Betriebssystemen ermöglicht.

Beispiel einer Mischform

Wird die Java-Laufzeitumgebung auf einem Betriebssystem installiert, welches bereits Threads unterstützt, so können Threads sowohl auf Kernel-Ebene (also vom Betriebssystem) verwaltet werden, als auch auf User-Ebene (also von der JRE).

Hinweis

Bei Systemen ohne jegliche Thread-Unterstützung
Angenommen, du hast es mit einem System zu tun, welches gar keine Threads unterstützt, also weder auf Kernel-Ebene, noch auf User-Ebene. Dann ist ein Prozess praktisch gleichzusetzen mit einem Thread.

> Der Prozess ist in diesem Fall ein Thread, und der Thread ist ein Prozess!

Hinweis

Wenn ein Prozess nur einen Thread besitzt

Angenommen, dein System unterstützt Threads, aber der betrachtete Prozess besteht nur aus einem einzelnen Thread. Dann ist dieser betrachtete Prozess praktisch gleichzusetzen mit seinem Thread.

Falls du bereits einmal eine Anwendung programmiert hast, dabei von dir aber nicht ausdrücklich mehrere Threads implementiert wurden, so bestand deine Anwendung nach dem Start nur aus einem einzelnen Thread.

Falls du bislang vielleicht noch gar nichts über Threads wusstest, so haben deine selbstprogrammierten Anwendungen vermutlich immer nur aus einem einzigen Thread bestanden.

Weiterhin nur eine CPU

Es sei nochmals verwiesen auf die Ausführungen im Kapitel Nur ein Prozessor mit einem Kern. Damit wurde bereits deutlich, dass immer nur ein Prozess zur Zeit auf der CPU ausgeführt werden kann. Besteht dieser Prozess nun aus mehreren Threads, so ist klar, dass davon nur ein einziger Thread aktiv auf der CPU ausgeführt werden kann. Die Threads eines Prozesses müssen sich auf der CPU also ebenso abwechseln, wie die verschiedenen Prozesse.

In Bezug auf Prozesse wurde bereits der Begriff Kontextwechsel erläutert. Analog lässt sich der Begriff Thread-Kontextwechsel definieren:

Definition: Thread-Kontextwechsel

Definition

Unter einem **Thread-Kontextwechsel** auf der CPU versteht man alle erforderlichen Tätigkeiten, um einen gerade auf der CPU aktiven Thread A, durch einen anderen Thread B zu ersetzen.

Dabei kann stillschweigend davon ausgegangen werden, dass beide Threads demselben Prozess zugeordnet sind, denn falls die Threads A und B unterschiedlichen Prozessen zugeordnet wären, so handelt es sich offensichtlich um einen (Prozess-) Kontextwechsel.

Vorteile von Threads

- Ein Thread-Kontextwechsel ist einfacher, und damit schneller durchführbar, als ein Prozess-Kontextwechsel.
- Alle Threads eines Prozesses haben Zugriff auf alle Betriebsmittel, welche diesem Prozess zugeordnet sind. (Dies ist gleichzeitig ein Vor- und ein Nachteil!)
- Der Anwendungsprogrammierer kann die Funktionalität der Gesamtanwendung in unterschiedliche Threads aufteilen, welche jeder für sich einfacher zu implementieren ist. (Ein Beispiel für die in der Informatik gerne verwendete Teile-und-herrsche-Strategie .)

Nachteile von Threads

- Alle Threads eines Prozesses haben Zugriff auf alle Betriebsmittel, welche diesem Prozess zugeordnet sind. (Dies ist gleichzeitig ein Vor- und ein Nachteil!)
- Anwendungsprogrammierer müssen über spezielle Kenntnisse bei der Programmierung von Threads verfügen. (Andernfalls kann es zu unerwünschten Nebeneffekten kommen.)

Aufgabe 1

Aufg. 82: Geschwindigkeitsvorteil

Warum benötigt ein Thread-Kontextwechsel weniger Zeit als ein Prozess-Kontextwechsel? Erläutere!

Aufgabe 2

Aufg. 83: Vor- und Nachteil des Betriebsmittelzugriffs

Warum ist die Zugriffsmöglichkeit auf alle Betriebsmittel eines Prozesses durch die verschiedenen Threads dieses Prozesses gleichzeitig ein Vor- und ein Nachteil? Erläutere!

Aufgabe 3

Aufg. 84: Threadzustände

Da der Begriff *Kontextwechsel* sich sowohl auf Prozesse, wie auch auf Threads beziehen lässt, stellt sich hier die Frage, ob Zustände analog für Prozesse (siehe Prozesszustände) und für Threads definiert werden können?

Achte beim Durcharbeiten der folgenden beiden Kapitel mal darauf!

Aufgabe 4

Aufgabe

Aufg. 85: Unerwünschte Nebeneffekte bei mehreren Threads

Bei den Nachteilen von Threads ist von "*unerwünschten Nebeneffekten*" die Rede. Überlege, recherchiere und diskutiere in deiner Lerngruppe: Finde zwei mögliche negative Effekte, die auftreten können, wenn ein Prozess aus mehreren Threads besteht!

Alternative Webquelle zum Thema

Quelle im Web

Operating Systems: Threads

http://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/4_Threads.html

Dr. John T. Bell

Department of Computer Science

University of Illinois, Chicago

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.9.1 Java-Beispiel mit Threads

Der folgende Quellcode zeigt ein Java-Programm mit zwei Threads. </p>

```
1 public class Beispiel_mit_Threads {
2
3 public static class Erster_Thread extends Thread {
4  public void run() {
5  System.out.println("Ich bin der erste Thread!");
6  System.out.println("Ich zähle von 1 bis 100.");
7  System.out.println("Es geht los:");
8  for (int i = 1; i <= 100; i++) {
9  System.out.println(i);
10     }
11  }
12 }
13
14 public static class Zweiter_Thread extends Thread {
```

```
15  public void run() {
16  System.out.println("Hier ist der zweite Thread!");
17  System.out.println("Das ABC ist ganz einfach:");
18  for (int i = 1; i <= 26; i++) {
19  System.out.println("ABCDEFGHIJKLMNOPQRSTUVWXYZ".charAt(i-1));
20  }
21  }
22 }
23
24 public static void main(String[] args) {
25  Thread e = new Erster_Thread();
26  Thread z = new Zweiter_Thread();
27  e.start();
28  z.start();
29 }
30
31 }
```

List. 1: Beispiel: Java-Threads
Ein Java-Programm mit zwei Threads.

Nach dem Start des Programms werden in der main-Methode beide Threads erzeugt und anschließend gestartet. Der erste Thread zählt von eins bis 100 und gibt jede Zahl einzeln auf der Console aus. Der zweite Thread buchstabiert das Alphabet und gibt jeden einzelnen Buchstaben aus.

Aufgabe 1

Aufgabe

Aufg. 86: Starte die Threads!

Übertrage den Java-Quellcode in eine Entwicklungsebene deiner Wahl, kompiliere und starte ihn.

- Was passiert?
- Welche Veränderung kannst du bei der Ausgabe erkennen, wenn du das Programm mehrmals startest?

Falls du keine Veränderung erkennst: Sorge auf deinem Rechner mal für etwas mehr Arbeitslast: Starte viele Programme, lass ein Video laufen und gleichzeitig Musik abspielen. Und starte immer wieder das Programm.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.9.2 Prozesse und Threads unter Windows

Dieses Kapitel wird in der weiterführenden Literatur behandelt: </p>

Hinweis

Weiterführende Literatur

Mandl 2013 erläutert in Kapitel 4.4.1 das Konzept von Prozessen und Threads in Windows. Die Lektüre dieser Quelle sei ausdrücklich empfohlen.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.9.3 Prozesse und Threads unter Unix und Linux

Dieses Kapitel wird in der weiterführenden Literatur behandelt: </p>

Hinweis

Weiterführende Literatur

Mandl 2013 erläutert in Kapitel 4.4.2 das Konzept von Prozessen und Threads in Unix und Linux. Die Lektüre dieser Quelle sei ausdrücklich empfohlen.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.10 Scheduling

Beim Scheduling geht es um die Zuteilung des Betriebsmittels *CPU* zu den einzelnen Prozessen. </p>

Definition: Scheduling

Definition

Unter **Scheduling** (genauer: **Prozess-Scheduling** oder **CPU-Scheduling**) versteht man die Tätigkeit des Aufteilens der verfügbaren Prozessorzeit auf alle Prozesse.

Es sei an die im Kapitel Nur ein Prozessor mit einem Kern genannte Bedingung erinnert.

Definition: Scheduler

Definition

Unter einem **Scheduler** (genauer: **Prozess-Scheduler** oder **CPU-Scheduler**) versteht man den Teil des Betriebssystems, welcher die Scheduling-Tätigkeit durchführt.

Der Scheduler ist somit dafür zuständig zu entscheiden, welcher Prozess als nächstes die CPU zugeteilt bekommt. Die Umsetzung dieser Entscheidung obliegt dem Dispatcher.

Definition: Dispatcher

Definition

Unter einem **Dispatcher** versteht man den Teil eines Betriebssystems, welcher bei einem Kontextwechsel dem derzeit aktiven Prozess die CPU entzieht, um sie anschließend dem nächsten Prozess zuzuteilen.

Die folgende Abbildung zeigt den zeitlichen Verlauf auf der CPU: Bei jedem Kontextwechsel entscheidet der Scheduler darüber, welcher Prozess als nächstes die CPU zugeteilt bekommt, der Dispatcher führt anschließend den Wechsel vom "alten" zum "neuen" Prozess auf der CPU durch.

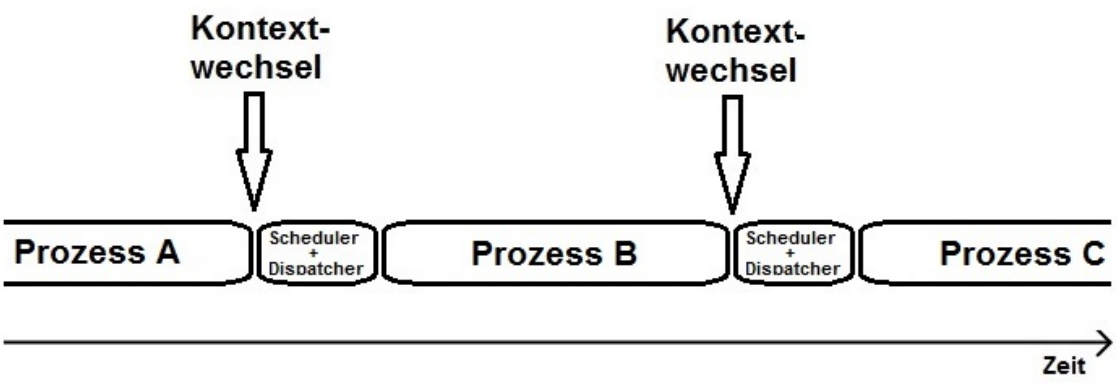


Abb. 56: Scheduler und Dispatcher werden bei einem Kontextwechsel aktiv
CC-BY

Man kann sich leicht vorstellen, dass der Dispatcher bei einem Kontextwechsel eine Reihe von kleinen Dingen erledigen muss.

Aufgabe 1

Aufgabe

Aufg. 87: Ein Dispatcher muss tun, was ein Dispatcher tun muss...
Überlege, recherchiere und diskutiere in deiner Lerngruppe, um eine Liste mit kleinen Tätigkeiten zusammenzustellen, welche der Dispatcher bei einem Kontextwechsel zu erledigen hat!

Bei der Bearbeitung dieser Aufgabe ist es ausdrücklich erwünscht, dass in der erstellten Liste Fachbegriffe verwendet werden. Hier ein Beispiel für eine kleine Tätigkeit mit Fachbegriffen:

- Der Dispatcher ändert innerhalb der Prozesstabelle im Prozesskontrollblock des von der CPU scheidenden Prozesses den Zustand von *Rechnend* auf *Bereit* bzw. auf *Blockiert*.

Definition: Non-preemptive Scheduling

Definition

Man spricht von **non-preemptive Scheduling** (oder auf Deutsch: **nicht-unterbrechendes Scheduling** oder **nicht-verdrängendes Scheduling**), wenn ein Prozess die exklusive Nutzung der CPU nur dann aufgeben muss, wenn er

- in den Zustand *Blockiert* wechselt,
- freiwillig und von selbst die CPU abgibt, oder
- sich selbst beendet.

Hierbei ist es also denkbar, dass ein Prozess, sobald er die CPU bekommt, diese erst wieder bei seiner Terminierung abgibt, egal wie lange es bis dahin dauert. Alle weiteren Prozesse müssen sich unter Umständen also sehr lange gedulden.

Definition: Preemptive Scheduling

Definition

Man spricht von **preemptive Scheduling** (oder auf Deutsch: **unterbrechendes Scheduling** oder **verdrängendes Scheduling**), wenn einem Prozess die exklusive Nutzung der CPU zu einem beliebigen Zeitpunkt entzogen werden kann, um sie einem anderen Prozess zu übertragen.

Üblicherweise wird die Prozessorzeit dabei in kleine Zeiteinheiten eingeteilt und (spätestens) am Ende einer Zeiteinheit entscheidet der Scheduler neu darüber, welcher Prozess als nächstes die CPU bekommt.

Diese *kleine Zeiteinheit* ist in der Literatur unter verschiedenen Namen anzutreffen:

Definition: Zeitscheibe (Quantum)

Definition

Unter einer **Zeitscheibe** oder einem **Quantum** versteht man den (i.d.R. sehr kleinen) zusammenhängenden Zeitraum, innerhalb dessen ein Prozess auf einem preemptiven Betriebssystem die CPU nutzen darf, ohne dabei von einem anderen Prozess verdrängt zu werden.

Innerhalb eines Quantums ist es möglich, dass der auf der CPU aktive Prozess durch Interrupts unterbrochen wird. Jedoch bekommt er nach der Abarbeitung der jeweiligen Interruptbehandlungsroutine die CPU zurück und darf diese weiter nutzen; maximal bis das Quantum aufgebraucht ist. (Siehe hierzu auch die folgende Aufgabe:)

Aufgabe 2

Aufgabe

Aufg. 88: Früher als spätestens

Bei den Erläuterungen zum preemptiven Scheduling heisst es: "*(spätestens) am Ende einer Zeiteinheit entscheidet der Scheduler neu*".

Nenne mindestens zwei Gründe, warum der Scheduler auch schon früher als "*am Ende der Zeiteinheit*" eine Scheduling-Entscheidung treffen könnte/müsste.

Aufgabe 3

Aufgabe

Aufg. 89: Der Nächste ist der Vorherige?

Ist es beim unterbrechenden Scheduling denkbar, dass am Ende der Zeitscheibe der Scheduler dem gerade aktiven Prozess erneut die CPU zuteilt, ohne dass zwischendurch ein anderer Prozess an der Reihe war?

Diskutiere mögliche dafür- oder dagegen-sprechende Gründe in deiner Lerngruppe!

Aufgabe 4

Aufgabe

Aufg. 90: Wenn ein Prozess blockiert

Denke dir folgende Situation: Der gerade auf der CPU aktive Prozess blockiert.

- Gib drei mögliche Gründe an, warum der Prozess blockiert.

In dieser Situation kommen Scheduler und Dispatcher ins Spiel. Aber in welcher Reihenfolge?

- Erst der Scheduler und dann der Dispatcher?
- Oder erst der Dispatcher und dann der Scheduler?
- Oder ist der Dispatcher ein Teil des Schedulers und wird aus diesem heraus aufgerufen?
- Oder ist der Scheduler ein Teil des Dispatchers und wird aus diesem heraus aufgerufen?
- Oder fällt dir noch eine andere Variante ein?

Diskutiere die Varianten in deiner Lerngruppe und begründe deine Entscheidung!

So geht es weiter:

Gliederung

3.2.10 Scheduling
3.2.10.1 Scheduling-Ziele
3.2.10.2 Scheduling-Verfahren
3.2.10.3 Scheduling in gängigen Betriebssystemen
3.2.10.4 Vergleichskriterien

Alternative Webquelle zum Thema

Quelle im Web

Operating Systems: CPU Scheduling

http://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/6_CPU_Scheduling.html

Dr. John T. Bell
Department of Computer Science
University of Illinois, Chicago

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.10.1 Scheduling-Ziele

Bevor in den folgenden Abschnitten verschiedene Scheduling-Verfahren erläutert werden, können zunächst einige Ziele identifiziert werden, deren Umsetzung vom Scheduler so weit wie möglich angestrebt werden sollte. </p>

Aufgabe 1

Aufgabe

Aufg. 91: Scheduling-Ziele

Mandl 2013 erläutert in Kapitel 5.1 (Scheduling-Kriterien) verschiedene Scheduling-Ziele.

- Welche Ziele sind das?
- Erläutere jedes Ziel kurz.
- Wie steht es mit der gleichzeitigen Erfüllung aller Ziele?

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.10.2 Scheduling-Verfahren

Auf den folgenden Unterseiten werden eine Reihe unterschiedlicher Scheduling-Verfahren behandelt: </p>

Gliederung

3.2.10.2 Scheduling-Verfahren
3.2.10.2.1 First Come First Serve
3.2.10.2.2 Shortest Job First
3.2.10.2.3 Shortest Remaining Time Next
3.2.10.2.4 Round Robin
3.2.10.2.5 Priority Scheduling
3.2.10.2.6 Weitere Verfahren

Alle Verfahren haben eine gewisse Berechtigung ihrer Existenz.

Hinweis

Stell dir vor, der Scheduler eines Betriebssystems arbeitet nach dem jeweils erläuterten Verfahren. Finde heraus:

- Wie unterscheiden sich die beschriebenen Verfahren?
- Was sind die Auswirkungen auf das Gesamtsystem?
- Was bedeutet das jeweilige Scheduling-Verfahren für die beteiligten Prozesse?
- Welche Verfahren sind praxistauglich? Und welche sind eher theoretischer Natur?

Es sei an dieser Stelle an die verschiedenen bereits beschriebenen Betriebssystemarten erinnert.

Wichtig

Wichtig

Studierende sind oftmals geneigt, ein Scheduling-Verfahren nur auf das Betriebssystem ihres eigenen PCs oder Laptops zu beziehen. Denke daran: es gibt unterschiedliche Betriebssysteme für unterschiedliche Einsatzzwecke.

Betrachte die einzelnen Scheduling-Verfahren immer im Hinblick auf alle Betriebssystemarten!

Wichtige Frage

Auch die Antwort auf die folgende Frage ist wichtig für das Verständnis dieses Kapitels:

Frage

Es gibt viele verschiedene Scheduling-Verfahren. Aber **wieviele** dieser Verfahren müssen in einem Betriebssystem **mindestens** implementiert sein, damit der Scheduler dieses Betriebssystems seine Aufgabe erfüllen kann?

Wieviele es sind

Wieviele es sind findest du ganz bestimmt heraus, wenn du selbst darüber nachdenkst und diese Frage in deiner Lerngruppe diskutierst!

Animation von Scheduling-Verfahren

Einige der auf den kommenden Seiten beschriebenen Scheduling-Verfahren werden in dieser Animation der Universität Osnabrück behandelt:

AnimOS CPU-Scheduling - A project by Gregor Kotainy and Prof. Dr.-Ing. Olaf Spinczyk
https://ess.cs.uni-osnabrueck.de/software/AnimOS/CPU-Scheduling/index.html

Auch auf der Webseite der University of Texas gab es einst eine Animation zu den Scheduling-Verfahren. Die Webseite ist mittlerweile nicht mehr online, aber die Animation ist in folgenden YouTube-Video festgehalten:

Aufgabe 1

Aufgabe

Aufg. 92: Ein Hauch von Texas

Starte die Animation und verfolge den Ablauf parallel, wenn du die Beschreibung des jeweiligen Verfahrens auf den kommenden Seiten durcharbeitest!

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/eI2pPS6rUJw

Med.:

3.2.10.2.1 First Come First Serve

First Come First Serve (kurz: FCFS) arbeitet die Prozesse in der Reihenfolge ihres Starts ab. Der zuerst erzeugte Prozess darf auch als erstes in den Zustand *Rechnend* wechseln. </p>

Aufgabe 1

Aufgabe

Aufg. 93: FCFS animiert

FCFS wird in dieser Animation behandelt. Probiere es aus!

Achte darauf, dass unter 'Scheduling strategy' der richtige Scheduling-Algorithmus ausgewählt ist. Und finde heraus, welche Bedeutung 'CPU burst' und 'IO burst' in dieser Animation haben.

Während seiner Rechenzeit kann ein Prozess durch einen Interrupt unterbrochen werden. Direkt nach dem Interrupt setzt er seine Arbeit auf der CPU fort.

Weiterführende Literatur

FCFS wird u.a. behandelt bei:

- *Eisenhardt et.al. 2007* , Kapitel 9.2.3
- *Glatz 2010* , Kapitel 3.4.3
- *Mandl 2013* , Kapitel 5.2
- *Strelen 2012* , Kapitel 4.1
- *Tanenbaum 2009* , Kapitel 2.4.2

und kann dort - je nach Verfügbarkeit der Quellen - nachgelesen werden.

Aufgabe 2

Aufgabe

Aufg. 94: Interessante FCFS-Situation

Interessant wird das Verhalten der FCFS-Strategie in der Situation, in der ein Prozess vom Zustand *Rechnend* in den Zustand *Blockiert* übergeht.

Überlege, recherchiere und diskutiere in deiner Lerngruppe:
Nenne mindestens zwei Möglichkeiten, wie die CPU anschließend vom Scheduler zugeteilt werden kann.

Aufgabe 3

Aufgabe

Aufg. 95: FCFS auf deinem Rechner

Wir können sicherlich voraussetzen, dass du Besitzerin oder Besitzer eines handelsüblichen PCs oder Laptops mit einem Betriebssystem mit grafischer Oberfläche bist. Und du ahnst bestimmt schon, dass das Betriebssystem deines Computers nicht nach der FCFS-Scheduling-Strategie arbeitet, oder?

Welche Auswirkungen wären zu befürchten, wenn der Scheduler deines Betriebssystems plötzlich auf FCFS umstellen würde?

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.10.2.2 Shortest Job First

Shortest Job First (kurz: SJF) kann kaum treffender als durch seinen Namen beschrieben werden: Von allen erzeugten Prozessen auf dem System bekommt derjenige als erstes die CPU, der die kürzeste Laufzeit besitzt. </p>

In einigen Quellen wird dieses Verfahren auch Shortest Process Next (kurz: SPN) genannt. Beide Begriffe werden synonym verwendet.

Aufgabe 1

Aufgabe

Aufg. 96: SPN animiert

SPN wird in dieser Animation behandelt. Probiere es aus!

Achte darauf, dass unter 'Scheduling strategy' der richtige Scheduling-Algorithmus ausgewählt ist. Und finde heraus, welche Bedeutung 'CPU burst' und 'IO burst' in dieser Animation haben.

Weiterführende Literatur

SJF bzw. SPN wird u.a. behandelt bei:

- *Eisenhardt et.al. 2007* , Kapitel 9.2.3
- *Glatz 2010* , Kapitel 3.4.3
- *Mandl 2013* , Kapitel 5.2
- *Strelen 2012* , Kapitel 4.1
- *Tanenbaum 2009* , Kapitel 2.4.2

und kann dort - je nach Verfügbarkeit der Quellen - nachgelesen werden.

Aufgabe 2

Aufgabe

Aufg. 97: Kurz oder lang

Auf einem System existieren bereits mehrere Prozesse. Einige sind *kurz*, andere *lang*. Welche Auswirkung hat es auf die *langen Prozesse*, wenn auf diesem System ständig neue *kurze Prozesse* hinzukommen?

Aufgabe 3

Aufgabe

Aufg. 98: Blockieren und Interrupts

Wie sollte sich der Scheduler bei Anwendung der SJF-Strategie verhalten, wenn der auf der CPU aktive Prozess blockiert?

Und was passiert mit dem auf der CPU aktiven Prozess bei einem Interrupt?

Aufgabe 4

Aufgabe

Aufg. 99: Erzeugt oder was?

Oben ist zu lesen: "*Von allen erzeugten Prozessen...*".
Aber das Wort *erzeugt* ist dabei eigentlich nur unter einer Bedingung oder Einschränkung richtig. Was ist gemeint?

Tipp

Denk mal an die Prozesszustände und an die möglichen Zustandsübergänge.

Aufgabe 5

Aufgabe

Aufg. 100: SJF auf deinem Rechner

Was denkst du, wie sich SJF auf deinem PC oder Laptop mit grafischer Oberfläche auswirkt? Nenne mindestens zwei Auswirkungen! Sind diese Auswirkungen positiv oder negativ?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.10.2.3 Shortest Remaining Time Next

Shortest Remaining Time Next (kurz: SRTN) ist eine Abwandlung des SJF-Verfahrens. Hierbei bekommt immer derjenige Prozess die CPU, welcher die kürzeste Restlaufzeit besitzt. </p>

Aufgabe 1

Aufgabe

Aufg. 101: SRTN animiert

SRTN wird in dieser Animation behandelt. Probiere es aus!

Achte darauf, dass unter 'Scheduling strategy' der richtige Scheduling-Algorithmus ausgewählt ist. Und finde heraus, welche Bedeutung 'CPU burst' und 'IO burst' in dieser Animation haben.

Weiterführende Literatur

SRTN wird u.a. behandelt bei:

- *Glatz 2010* , Kapitel 3.4.3
- *Mandl 2013* , Kapitel 5.2
- *Strelen 2012* , Kapitel 4.1
- *Tanenbaum 2009* , Kapitel 2.4.2

und kann dort - je nach Verfügbarkeit der Quellen - nachgelesen werden.

Aufgabe 2

Aufgabe

Aufg. 102: SJF vs. SRTN

Was ist der Unterschied zwischen dem SJF- und dem SRTN-Verfahren? Warum werden diese beiden Verfahren also gerne getrennt genannt?

Tipp

Was war doch gleich der Unterschied zwischen non-preemptive und preemptive Scheduling?

(Natürlich hätte ich die beiden Fachbegriffe hier direkt mit ihren Definitionen verlinken können, aber: Nein! Wenn du die Bedeutung der Fachbegriffe nicht kennst, dann musst du sie mühsam suchen! Denk' dir an dieser Stelle das gehässige Lachen des Autors...)

Aufgabe 3

Aufgabe

Aufg. 103: Wie kurz bist du?

Scheduler, die nach dem SJF- oder SRTN-Prinzip arbeiten, müssen herausfinden können, wie kurz die (Rest-) Laufzeit jedes Prozesses (noch) ist.

Überlege, recherchiere und diskutiere in deiner Lerngruppe:
Wie macht der Scheduler das?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.10.2.4 Round Robin

Round Robin (RR) ist ein Scheduling-Verfahren, welches die zur Verfügung stehende CPU-Zeit in kleine Zeitscheiben einteilt. Alle Prozesse werden in eine initiale Reihenfolge gebracht und der erste Prozess bekommt die CPU. Er darf diese genau bis zum Ablauf seines Zeit-Quantums nutzen, anschließend wird er unterbrochen und der nächste Prozess in der Reihenfolge ist am Zug. Auch hier findet wieder eine Unterbrechung am Ende des Quantums statt, usw. Nach dem letzten Prozess ist wieder der Erste an der Reihe. </p>

Hinweis

In gewisser Weise ergänzt RR das FCFS-Verfahren um eine Zeitscheibe. Alle Prozesse werden gleich behandelt.

Aufgabe 1

Aufgabe

Aufg. 104: RR animiert

RR wird in dieser Animation behandelt. Probiere es aus!

Achte darauf, dass unter 'Scheduling strategy' der richtige Scheduling-Algorithmus ausgewählt ist. Und finde heraus, welche Bedeutung 'CPU burst' und 'IO burst' in dieser Animation haben.

Optimale Länge einer Zeitscheibe

Es drängt sich die Frage nach der optimalen Länge einer Zeitscheibe auf. Sie soll nicht zu lang und nicht zu kurz sein. (Was eine herrlich diplomatische Antwort darstellt.)

Aufgabe 2

Aufgabe

Aufg. 105: Zu kurz oder zu lang

Es ist denkbar, dass das Betriebssystem deines PCs oder Laptops nach dem RR-Verfahren arbeitet. Was ist zu befürchten,

- wenn das Quantum zu kurz gewählt ist?
- wenn das Quantum zu lang gewählt ist?

Zeitscheiben in der Praxis

Tanenbaum 2009 und *Strelen 2012* geben an, dass Zeitscheiben in einer Größenordnung von 10 bis 50 ms verbreitet seien. *Eisenhardt et.al. 2007* sehen typische Längen zwischen 10 und 100 ms, und bei *Mandl 2013* ist ein Intervall von 10 bis 200 ms zu finden. (Zumindest in der unteren Grenze sind sich alle einig.)

In einem Knowledge Base Artikel von Microsoft (KB 259025) finden sich einige Hinweise zur Länge der verwendeten Zeitscheiben bei Windows 2000 und Windows XP. Je nach Prozessor und Konfiguration von Windows gehen die Werte aber stark auseinander.

Die folgende Abbildung stammt aus einem Windows 7 Professional und zeigt den Ausschnitt eines Konfigurationsfensters. Durch ändern der Option *Optimale Leistung anpassen für* wird die Länge des Quantums für (Vordergrund-) Programme und Hintergrunddienste verändert, ohne dass man jedoch die tatsächliche Länge in Millisekunden beeinflussen könnte. (Das ist wohl auch besser so.)

Abb. 57: Windows 7 Prozessorzeitplanung
Zu finden unter: Systemsteuerung / System / Erweiterte Systemeinstellungen / Erweitert / Leistung / Einstellungen
CC-BY

Ist das Betriebssystem auf einem Computer installiert, der vorrangig im Dialogbetrieb mit einem User agiert, so ist die Option *Optimale Leistung anpassen für* ***Programme*** richtig gewählt. Fungiert der Computer hingegen als Server und bedient die Anfragen verschiedener Clients, so ist *Optimale Leistung anpassen für* ***Hintergrunddienste*** als Einstellung empfehlenswert. Es verwundert daher nicht, dass auf einem Windows Server 2008 R2 das gleiche Konfigurationsfenster zu finden ist, wobei jedoch die alternative Option ausgewählt ist.

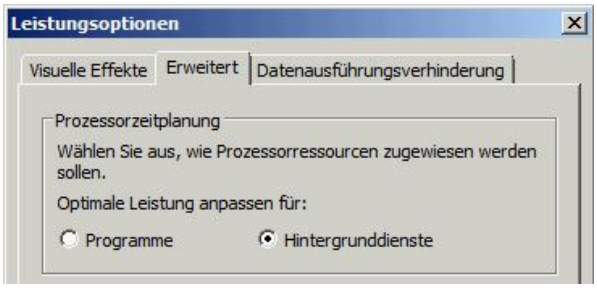


Abb. 58: Windows Server 2008 R2 Prozessorzeitplanung
Zu finden unter: Systemsteuerung / System / Erweiterte Systemeinstellungen / Erweitert / Leistung / Einstellungen
CC-BY

Hinweis

Auch wenn viele aktuelle Betriebsysteme Zeitscheiben verwenden, so bedeutet dies nicht, dass auch das ursprüngliche Round Robin-Verfahren, wie es hier beschrieben ist, eingesetzt wird. In gewisser Weise kann man aber sicherlich von einem *stark weiterentwickelten und optimierten RR-Verfahren* sprechen.

Weiterführende Literatur

RR wird u.a. behandelt bei:

- *Eisenhardt et.al. 2007* , Kapitel 9.2.4
- *Glatz 2010* , Kapitel 3.4.3

- *Mandl 2013* , Kapitel 5.2
- *Strelen 2012* , Kapitel 4.1
- *Tanenbaum 2009* , Kapitel 2.4.3

und kann dort - je nach Verfügbarkeit der Quellen - nachgelesen werden.

Aufgabe 3

Aufgabe

Aufg. 106: RR und E/A-lastige Prozesse

Dem RR-Verfahren wird nachgesagt, dass es E/A-lastige Prozesse benachteiligt.

Überlege, recherchiere und diskutiere in deiner Lerngruppe:

- Was versteht man unter E/A-lastigen Prozessen?
- Warum kommt es zu einer Benachteiligung?
- Macht es einen Unterschied, ob gerade sehr viele oder nur sehr wenige Prozesse auf dem System gestartet sind?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.10.2.5 Priority Scheduling

Beim Priority Scheduling (kurz: PS) wird jedem Prozess eine Priorität zugewiesen. Der Prozess mit der höchsten Priorität bekommt als erstes die CPU. </p>

Aufgabe 1

Aufgabe

Aufg. 107: Verschiedene Prozesse mit gleicher Priorität

Wie soll der Scheduler vorgehen, wenn es mehrere Prozesse mit gleicher Priorität gibt? Z.B. mehrere Prozesse mit höchster Priorität?

Weiterführende Literatur

PS wird u.a. behandelt bei:

- *Eisenhardt et.al. 2007* , Kapitel 9.2.3
- *Glatz 2010* , Kapitel 3.4.3 (siehe dort unter *ML-Strategie*)
- *Mandl 2013* , Kapitel 5.2

- *Strelen 2012* , Kapitel 4.1
- *Tanenbaum 2009* , Kapitel 2.4.3

und kann dort - je nach Verfügbarkeit der Quellen - nachgelesen werden.

Aufgabe 2

Aufgabe

Aufg. 108: Was passiert bei niedriger Priorität?

Auf einem System warten mehrere Prozesse mit hoher Priorität, mehrere mit mittlerer Priorität und auch mehrere mit niedriger Priorität auf die Zuteilung der CPU.

Solange sich mindestens ein Prozess mit hoher Priorität im Zustand *Bereit* befindet: welche Auswirkungen hat dies auf die CPU-Zuteilung für Prozesse mit mittlerer und niedriger Priorität?

Aufgabe 3

Aufgabe

Aufg. 109: Erweitere PS

Erweitere das PS-Verfahren um folgende Tätigkeit:

Bei jedem Kontextwechsel wird die Priorität des von der CPU scheidenden Prozesses um eine Stufe herabgesetzt (sofern er sich nicht bereits in der niedrigsten Stufe befindet).

Welche Auswirkungen auf alle Prozesse mit unterschiedlichen Prioritäten ergeben sich jetzt?

- Erläutere die Auswirkungen anhand eines selbstgewählten Beispiels!
- Orientiere dein Beispiel an den Beispielen der bekannten Animation . Denke dabei an die Angabe der Prioritäten.

Hast du dein Beispiel erstellt? Prima. Dann führen wird das Gedankenspiel noch einmal in eine entgegengesetzte Richtung:

Aufgabe 4

Aufgabe

Aufg. 110: Herauf statt herunter!

Statt die Prioritäten von Prozessen dynamisch herabzusetzen, könnte man sie auch heraufsetzen. Überlege, recherchiere und diskutiere in deiner Lerngruppe:

- Wessen Priorität wird heraufgesetzt?
- Wann genau passiert das?
- Wie groß ist der Aufwand im Vergleich zur oben beschriebenen Strategie mit dem Herabsetzen von Prioritäten?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.10.2.6 Weitere Verfahren

Im Verlaufe der vergangenen Jahrzehnte sind noch eine Reihe weiterer Scheduling-Verfahren entstanden. Einige existieren nur in der Theorie und/oder sind nur im Zuge der Forschung am Thema beschrieben worden, andere wurden auch in der Praxis implementiert. Manchmal aber auch nur unter bestimmten Bedingungen, d.h. zum Beispiel im Zuge einer möglichst optimalen Ausnutzung der von der Hardware vorgegebenen Situation. Sobald die Hardware weiterentwickelt wurde (üblicherweise wird sie dann ja leistungsfähiger), wurden oftmals auch die Scheduling-Algorithmen angepasst und optimiert. </p>

Hinweis

Die geneigte Leserin und der geneigte Leser kann gerne in der auf den vorangegeangenen Seiten immer wieder erwähnten Literatur nach weiteren Verfahren stöbern.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.10.3 Scheduling in gängigen Betriebssystemen

Dieses Kapitel wird in der weiterführenden Literatur behandelt: </p>

Hinweis

Weiterführende Literatur

Mandl 2013 erläutert das Scheduling für Unix, Linux und Windows. Die Lektüre dieser Quelle sei ausdrücklich empfohlen.

- Kapitel 5.3.1: CPU-Scheduling im ursprünglichen Unix
- Kapitel 5.3.2: CPU-Scheduling unter Linux
- Kapitel 5.3.3: CPU-Scheduling unter Windows

Zusätzlich wird auf Scheduling in der Java Virtual Machine eingegangen.

- Kapitel 5.3.4: Scheduling von Threads in Java

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.10.4 Vergleichskriterien

Nachdem die Arbeitsweise verschiedener Scheduling-Verfahren nun bekannt ist, können Kriterien definiert werden, mit deren Hilfe ein Vergleich unterschiedlicher Verfahren vorgenommen werden kann. </p>

Aufgabe 1

Aufgabe

Aufg. 111: Scheduling-Kriterien

Mandl 2013 erläutert in Kapitel 5.3 (Vergleich ausgewählter Scheduling-Verfahren) verschiedene Scheduling-Kriterien.

- Welche Kriterien sind das?
- Erläutere jedes Kriterium kurz.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.11 Synchronisation

Bei der **Synchronisation** (auch: **Prozess-Synchronisation** oder **Thread-Synchronisation**) geht es um Mechanismen zur Vermeidung von Problemen, die bei der nebenläufigen Ausführung von Prozessen oder Threads in Verbindung mit gemeinsam genutzten Betriebsmitteln entstehen können. </p>

Hinweis

Der vorangegangene Satz hat es in sich!

Lies ihn noch einmal in Ruhe durch und überlege, ob dir die Bedeutung aller enthaltenen Fachbegriffe klar ist. Insbesondere solltest du den anderen Mitgliedern deiner Lerngruppe erklären können:

- Was ist ein Prozess?
- Was ist ein Thread?
- Was ist der Unterschied zwischen einem Prozess und einem Thread?
- Was sind Betriebsmittel?
- Und was sind dann *gemeinsam genutzte* Betriebsmittel?

Bislang noch nicht definiert wurde die *nebenläufige Ausführung*, deshalb passiert dies im folgenden Kapitel.

Es sei an die im Kapitel Nur ein Prozessor mit einem Kern genannte Bedingung erinnert.

So geht es weiter:

3.2.11 Synchronisation
3.2.11.1 Grundlegende Begriffsdefinitionen zur Synchronisation
3.2.11.2 Aktives Warten
3.2.11.3 Semaphore
3.2.11.4 Monitore
3.2.11.5 Zusammenfassung Synchronisation
3.2.11.6 Synchronisationstechniken moderner Betriebssysteme
3.2.11.7 Synchronisationsmechanismen in Programmiersprachen

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.11.1 Grundlegende Begriffsdefinitionen zur Synchronisation

Auf den folgenden Seiten werden einige grundlegende Begriffe zum Thema Synchronisation von Prozessen definiert. </p>

So geht es weiter:

3.2.11.1 Grundlegende Begriffsdefinitionen zur Synchronisation
3.2.11.1.1 Nebenläufigkeit
3.2.11.1.2 Race Conditions

3.2.11.1.3 Kritischer Abschnitt

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.11.1.1 Nebenläufigkeit

Zunächst die Definition. </p>

Definition: Nebenläufigkeit

Definition

Unter **Nebenläufigkeit** versteht man die quasi-parallele Ausführung von Befehlen unterschiedlicher Prozesse oder Threads auf einer CPU.

Solange nur eine CPU (mit einem Rechenkern) zur Verfügung steht, ist hier von *quasi-paralleler Ausführung* die Rede. Erst bei mehreren CPUs oder mehreren Rechenkernen auf einer CPU kann Nebenläufigkeit auch mit *echt-paralleler Ausführung* definiert werden.

Definition: Nebenläufige Ausführung

Definition

Unter der *nebenläufigen Ausführung* mehrerer Prozesse oder Threads auf nur *einer* CPU versteht man, dass sich mehrere Prozesse oder Threads bei ihrer Ausführung auf der CPU abwechseln.

Es kommt dabei zwangsläufig immer wieder zu Kontextwechseln auf der CPU, dies ist eine Folge des preemptiven Schedulings.

Wichtig

Für das Verständnis der weiteren Kapitel ist es wichtig, sich einmal über den **Zeitpunkt von Kontextwechseln** während einer nebenläufigen Ausführung von Prozessen oder Threads Gedanken zu machen. Die folgende Aufgabe gibt dazu Gelegenheit.

Aufgabe 1

Aufgabe

Aufg. 112: Wann genau finden Kontextwechsel statt?

Wenn zwei Prozesse A und B nebenläufig ausgeführt werden, ist dann vorherbestimmt, wann genau ein Kontextwechsel zwischen diesen beiden Prozessen erfolgt?

> Wenn diese beiden Prozesse nach ihrer Terminierung immer wieder neu gestartet und nebenläufig ausgeführt werden, finden die Kontextwechsel dann **immer an der gleichen Stelle im Maschinencode** statt?
>
> Diskutiere diese Situation in deiner Lerngruppe!
>
> Wovon hängt es ganz allgemein ab, wann ein Kontextwechsel erfolgt?
> Nenne mindestens drei Bedingungen!

Dass Probleme bei der nebenläufigen Ausführung von Prozessen und Threads auftreten können, und wie diesen entgegnet werden kann, wird auf den folgenden Seiten gezeigt.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.11.1.2 Race Conditions

Bevor auf die eigentliche Definition von Race Conditions eingegangen wird, seien einige Hintergründe anhand eines Beispiels erläutert.

Beispiel

Das folgende Video zeigt ein anschauliches Beispiel für Race Conditions:

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/dlOg4Dz-bgM

> **Med. 30:** Race Conditions (04:50)
> http://youtu.be/dlOg4Dz-bgM
> CC-BY

Quellcode aus dem Video

Hier ist der Quellcode aus dem Video:

```
1 public class Beispiel_Race_Conditions {
2
3 static int counter = 0;
4
5 public static class Counter_Thread_A extends Thread {
6     public void run() {
7         counter = 10;
8         counter++;
```

```
 9          counter++;
10          System.out.println("A-Counter: " + counter);
11      }
12  }
13
14  public static class Counter_Thread_B extends Thread {
15      public void run() {
16          counter = 20;
17          counter++;
18          counter++;
19          counter++;
20          counter++;
21          counter++;
22          counter++;
23          System.out.println("B-Counter: " + counter);
24      }
25  }
26
27  public static void main(String[] args) {
28      Thread a = new Counter_Thread_A();
29      Thread b = new Counter_Thread_B();
30      a.start();
31      b.start();
32  }
33
34  }
```

List. 2: Listing 1: Beispiel für Race Conditions
Ein Java-Programm mit zwei Threads. Bei der Ausführung kommt es zu Race Conditions.

Definition: Race Conditions

Definition

Unter **Race Conditions** (oder einem **kritischem Ablauf**) versteht man Situationen, bei denen zwei oder mehr Prozesse (bzw. Threads) ein oder mehrere Betriebsmittel gemeinsam nutzen, und das Ergebnis der Ausführung von der zeitlichen Reihenfolge der Zugriffe der beteiligten Prozesse oder Threads auf das (bzw. die) Betriebsmittel abhängt.

Man bemerke, dass Race Conditions nicht allein durch nebenläufige Ausführung von Prozessen oder Threads entstehen. Erst wenn *gemeinsam genutzte* Betriebsmittel im Spiel sind, kommt es zu kritischen Abläufen.

Aufgabe 1

Aufgabe

Aufg. 113: Gemeinsam genutzes Betriebsmittel
Was ist das gemeinsam genutzte Betriebsmittel in Listing 1 oben?

Aufgabe 2

Aufgabe

Aufg. 114: Kleiner Fehler bei den Erklärungen im Video

In den Erläuterungen im Video gibt es einen kleinen Fehler (dort, wo beide Threads den Wert 12 ausgeben). Was ist gemeint?

Tipp

Es geht um die Reihenfolge, in der die beiden Threads ihre Ausgaben tätigen.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.11.1.3 Kritischer Abschnitt

Im Quelltext aller Prozesse oder Threads lassen sich Abschnitte identifizieren, welche entweder *kritisch*, oder *unkritisch* im Hinblick auf Race Conditions sind. </p>

Definition: Kritischer Abschnitt

Definition

Unter einem **kritischen Abschnitt** versteht man Programmteile, die während ihrer Ausführung auf der CPU nicht durch kritische Abschnitte anderer Prozesse oder Threads unterbrochen werden dürfen, sofern die beteiligten Prozesse oder Threads auf gemeinsam genutzte Betriebsmittel zugreifen.

Es sei hier klar hingewiesen auf die Tatsache, dass kritische Abschnitte während ihrer Ausführung sehr wohl unterbrochen werden dürfen, und das passiert in der Realität auch häufig, zum Beispiel durch Interrupts.

Es kommt aber immer darauf an, was genau während der Unterbrechung getan wird. Sobald gemeinsam genutzte Betriebsmittel ins Spiel kommen, wird es im wahrsten Sinne des Wortes "kritisch" und ein Unterbrechungsverbot im Sinne der obigen Definition droht.

Definition: Unkritischer Abschnitt

Definition

Unter einem **unkritischen Abschnitt** versteht man jeden Programmteil, der keinen kritischen Abschnitt darstellt.

Als *Programmteil* im Sinne der vorangegangenen Definitionen kann jeder Codeabschnitt mit der geforderten Eigenschaft gelten. Diese Programmteile lassen sich sowohl in einer Hochsprache wie Java, C, C++, Pascal, usw. identifizieren, als auch in Maschinencode oder Assembler.

Hinweis

Du musst dich daran erinnern, dass ein in einer Hochsprache wie Java, C, C++, Pascal, usw. angegebener Befehl in seiner Übersetzung in Maschinencode bzw. Assembler in mehrere kleine Befehle resultieren kann. Falls dir das entfallen war, so schau noch mal auf die Seite Vom_Quellcode_zum_Prozessor.

Ein Kontextwechsel findet immer zwischen zwei Maschinenbefehlen auf der CPU statt!

Beispiel zu kritischen Abschnitten

Das folgende Beispiel zeigt ein Java-Programm mit zwei Threads, bei deren Ausführung es zu Race Conditions kommt. Bei Thread_A werden durch spezielle Methoden kritische und unkritische Abschnitte gekennzeichnet.

```
1 public class Beispiel_Kritischer_Abschnitt {
2
3 static int counter = 0;
4
5 public static class Thread_A extends Thread {
6     public void run() {
7         do_something(); // unkritisch
8         count_from_10(); // kritisch !!!
9         do_something_else(); // unkritisch
10    }
11    private void do_something() {
12        // unkritischer Abschnitt
13        System.out.println("Thread_A: unkritisch");
14    }
15    private void count_from_10() {
16        // Vorsicht: kritischer Abschnitt!
17        counter = 10;
18        counter++;
19        counter++;
20        System.out.println("A-Counter: " + counter);
21    }
22    private void do_something_else() {
23        // unkritischer Abschnitt
24        System.out.println("Thread_A: wieder unkritisch");
25    }
26 }
27
28 public static class Thread_B extends Thread {
29    public void run() {
30        System.out.println("Thread_B ist gestartet.");
```

```
31          counter = 20;
32          counter++;
33          counter++;
34          counter++;
35          counter++;
36          counter++;
37          counter++;
38          System.out.println("B-Counter: " + counter);
39      }
40  }
41
42  public static void main(String[] args) {
43      Thread a = new Thread_A();
44      Thread b = new Thread_B();
45      a.start();
46      b.start();
47  }
48
49  }
```

List. 3: Listing 1: Beispiele für kritische und unkritische Abschnitte
Ein Java-Programm mit zwei Threads. Bei Thread_A wechseln sich ein unkritischer, ein kritischer und noch ein unkritischer Abschnitt ab.

Aufgabe 1

Aufgabe

Aufg. 115: Abschnitte in Thread_B
Wo finden sich in Listing 1 bei Thread_B *kritische* bzw. *unkritische* Abschnitte?

Werden Thread_A und Thread_B nebenläufig ausgeführt, so kann praktisch jederzeit ein Kontextwechsel erfolgen.

Angenommen Thread_B befindet sich innerhalb seines kritischen Abschnitts, und es erfolgt der Kontextwechsel zu Thread_A:

- Werden von Thread_A dann nur Befehle aus unkritischen Abschnitten ausgeführt, so gibt es keine Probleme.
- Falls aber von Thread_A Befehle aus dem kritischen Abschnitt ausgeführt werden, so kommt es zu Race Conditions.

Wichtig

Die Synchronisations-Aufgabe besteht nun darin sicherzustellen, dass sich immer nur ein Prozess oder Thread zur Zeit in seinem kritischen Abschnitt befindet.

Dazu werden auf den folgenden Seiten mehrere Konzepte vorgestellt.

Hinweis

Bei dem hier gezeigten Java-Beispielcode konkurrieren jeweils zwei Threads um das gemeinsame Betriebsmittel einer einfachen Integer-Variable.

Du musst verstehen, dass dies nur ein sehr einfaches Beispiel ist, und dass speziell bei Threads und "selbstgeschaffenen" Betriebsmitteln (wie der Integer-Variable) der Programmierer selbst in der Pflicht zur Synchronisation ist. (Java bietet dafür das Schlüsselwort synchronized .)

Es gibt - neben der durch Programmierer "selbst geschaffenen" Variablen - noch viele andere Betriebsmittel.

Aufgabe 2

Aufgabe

Aufg. 116: Viele andere Betriebsmittel

- Nenne möglichst viele Betriebsmittel!
- Und was war nochmal die zentrale Aufgabe des Betriebssystems?

Durch die Beantwortung der letzten Frage sollte klar werden:

Wichtig

Das Betriebssystem ist für die Synchronisation von Prozessen und Threads zuständig, welche auf gemeinsam genutzte Betriebsmittel zugreifen wollen.

Erinnert sei in diesem Zusammenhang an Systemaufrufe.

Aufgabe 3

Aufgabe

Aufg. 117: Warum Systemaufrufe?

Warum wird hier an Systemaufrufe erinnert? Erläutere den Zusammenhang von Systemaufrufen (wenn Betriebsmittel angesprochen werden) und der Synchronisation von Prozessen bzw. Threads!

Aufgabe 4

Aufgabe

Aufg. 118: Zwei Prozesse und kritische Abschnitte

Im Rahmen dieser Aufgabe existieren zwei Prozesse A und B, welche sich auf der CPU abwechseln.

Prozess A benötigt als Betriebsmittel das DVD-Laufwerk. Es lassen sich also (ein oder mehrere) kritische Abschnitte in Prozess A identifizieren, in denen der Zugriff auf das genannte Betriebsmittel erfolgt.

Prozess B benötigt als Betriebsmittel die Datei foo.txt auf der Festplatte, in die hineingeschrieben wird. Auch dafür lassen sich (ein oder mehrere) kritische Abschnitte identifizieren.

Weitere Betriebsmittel werden von den Prozessen nicht benötigt.

Angenommen Prozess B wird auf der CPU ausgeführt und befindet sich mitten in der Abarbeitung eines kritischen Abschnitts. Es erfolgt der Kontextwechsel zu Prozess A. Dieser arbeitet zunächst einen unkritischen Abschnitt ab, möchte dann aber einen kritischen Abschnitt betreten.

- Darf A seinen kritischen Abschnitt ausführen, obwohl der zuvor unterbrochene Prozess B sich noch innerhalb seines kritischen Bereichs befindet? (Dann wären beide Prozesse gleichzeitig in ihren kritischen Abschnitten.)
- Oder muss A zunächst warten, bis B seinen kritischen Abschnitt verlassen hat?

Begründe deine Entscheidung! (Sind alle Mitglieder deiner Lerngruppe auch dieser Meinung?)

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.11.2 Aktives Warten

Aktives Warten, manchmal auch als **Geschäftiges Warten** oder auf Englisch **Busy Waiting** bezeichnet, ist eine recht einfache Technik, um Prozesse oder Threads zu synchronisieren. </p>

Definition: Aktives Warten

Definition

Unter **aktivem Warten** versteht man nach *Mandl 2013* das ständige Abfragen eines Sperrkennzeichens am Eingang eines kritischen Abschnitts.

Polling

Die Technik des *ständigen Abfragens* ist in der Informatik auch als *Polling* bekannt. Das genannte *Sperrkennzeichen* kann eine von mehreren Prozessen oder Threads gemeinsam genutzte Variable sein, man spricht dann von einer *Sperrvariable*. Im folgenden Kapitel gibt es dazu ein Beispiel.

So geht es weiter:

Gliederung

3.2.11.2 Aktives Warten
3.2.11.2.1 Aktives Warten mit while
3.2.11.2.2 Das Problem des ungünstigsten Moments
3.2.11.2.3 Aktives Warten mit TSL

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.11.2.1 Aktives Warten mit while

Der Code des bereits bekannten Beispiels zu kritischen Abschnitten wird um eine (globale) Sperrvariable erweitert, beide Threads haben damit Zugriff auf diese Variable:

Code

```
static int lock = 0;
```

Besitzt lock den Wert Null, so definiert man, dass sich aktuell kein Prozess oder Thread in seinem kritischen Abschnitt befindet. Der kritische Abschnitt darf also betreten werden, dabei muss lock auf Eins gesetzt werden.

Direkt bevor der kritische Abschnitt wieder verlassen wird, wird lock auf Null zurück gesetzt.

Alle beteiligten Prozesse oder Threads müssen beim Betreten eines kritischen Bereichs den Wert von lock prüfen, und gegebenenfalls warten.

Code

Beim Betreten des kritischen Abschnitts:

```
while (lock == 1);
lock = 1;
```

Code

Direkt vor dem Verlassen des kritischen Abschnitts:

```
lock = 0;
```

Aufgabe 1

Aufgabe

Aufg. 119: Die Bedeutung des Semikolons!

Die Codezeile

while (lock == 1);

ist sehr wichtig. Hier wird der Wert von lock geprüft und gegebenenfalls gewartet. Die Sache mit dem Warten kannst du aber nur verstehen, wenn du die Bedeutung des Semikolons ganz am Ende der Codezeile kennst.

Erläutere:
Welche Bedeutung hat das Semikolon in Verbindung mit dem while?

Und was würde passieren, wenn das Semikolon versehentlich fehlt:

while (lock == 1) // Hier fehlt das Semikolon, Programmierfehler!
lock = 1;

(PS.: Die while-Schleife ist auch der Grund für den großen Nachteil des aktiven Wartens, siehe letzte Aufgabe ganz unten auf dieser Seite!)

Nachdem nun bekannt ist, welcher Code zur Implementierung des aktiven Wartens notwendig ist, kann dieser an den entsprechenden Stellen des Beispielprogramms eingepflegt werden.

Aufgabe 2

Aufgabe

Aufg. 120: Aktives Warten implementieren

Tu es! Füge den für das aktive Warten nötigen Code dem Beispielprogramm hinzu.

Lösung

```
1 public class Beispiel_Aktives_Warten {
2
3 static int counter = 0; // gemeinsam genutztes Betriebsmittel
4 static int lock = 0; // Sperrvariable
5
6 public static class Thread_A extends Thread {
7     public void run() {
8         do_something(); // unkritisch
9         count_from_10(); // kritisch !!!
10        do_something_else(); // unkritisch
```

```
11      }
12      private void do_something() {
13          // unkritischer Abschnitt
14          System.out.println("Thread_A: unkritisch");
15      }
16      private void count_from_10() {
17          // Vorsicht: kritischer Abschnitt!
18          while (lock == 1); // Semikolon beachten!
19          lock = 1;
20
21          counter = 10;
22          counter++;
23          counter++;
24          System.out.println("A-Counter: " + counter);
25
26          lock = 0;
27      }
28      private void do_something_else() {
29          // unkritischer Abschnitt
30          System.out.println("Thread_A: wieder unkritisch");
31      }
32  }
33
34  public static class Thread_B extends Thread {
35      public void run() {
36          System.out.println("Thread_B ist gestartet.");
37          while (lock == 1); // Semikolon beachten!
38          lock = 1;
39
40          counter = 20;
41          counter++;
42          counter++;
43          counter++;
44          counter++;
45          counter++;
46          counter++;
47          System.out.println("B-Counter: " + counter);
48
49          lock = 0;
50      }
51  }
52
53  public static void main(String[] args) {
54      Thread a = new Thread_A();
55      Thread b = new Thread_B();
56      a.start();
57      b.start();
58  }
59
60  }
```

List. 4: Beispiel: Aktives Warten
Ein Java-Programm mit zwei Threads. Die Sperrvariable soll ein gleichzeitiges Betreten der kritischen Abschnitte durch beide Threads verhindern.

Hinweis

In dieser Lösung wird wieder gezeigt, wie zwei Threads durch die Sperrvariable beeinflusst werden. Es hat den Anschein, als ob ein Anwendungsprogrammierer sich darum kümmern müsste. (Das muss er nur bei den von ihm selbst geschaffenen "Betriebsmitteln". Es sei erneut auf das für diese Zwecke in Java integrierte Schlüsselwort synchronized verwiesen.)

Du musst gedanklich hier wieder den Sprung zu Prozessen schaffen, und zu den vielen Betriebsmitteln eines Computersystems, und zum Betriebssystem, und zu Systemaufrufen, über die letztlich das aktive Warten durch das Betriebssystem realisiert werden kann.

Dann verstehst du, dass sich der **Programmierer des Betriebssystems** darum kümmern muss!

Aufgabe 3

Aufgabe

Aufg. 121: Warte aktiv!

Spiele in deiner Lerngruppe das Verhalten der beiden Threads aus der Lösung zu Aufgabe 2 durch. Bedenke:

- Es gibt nur eine CPU mit einem Kern. Es kann also nur ein Prozess zur Zeit ausgeführt werden, es kommt zu Quasi-Parallelität.
- Interessant ist die Situation, bei der sich bereits ein Prozess in seinem kritischen Abschnitt befindet, und die Sperre gesetzt hat.
- Der Kontextwechsel erfolgt, und der andere Prozess muss dann an der while-Schleife warten. Tut er das?

Aufgabe 4

Aufgabe

Aufg. 122: Der Nachteil des aktiven Wartens

Welchen großen Nachteil besitzen alle Verfahren, die nach dem Prinzip des aktiven Wartens verfahren?

Tipp

Gemeint ist hier ein *zeitlicher Nachteil*. Denk mal daran, wie sinnvoll die CPU beschäftigt wird. Wie wirkt sich das deiner Meinung nach auf die Performance des Gesamtsystems aus?

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.11.2.2 Das Problem des ungünstigsten Moments

Man betrachte erneut die mit einer while-Schleife realisierte Sperre des aktiven Wartens, welche zu Beginn eines kritischen Abschnitts durchlaufen werden muss:

Code

```
while (lock == 1);
lock = 1;
```

<hr>

Listing 1: Sperre zu Beginn des kritischen Abschnitts

Eine Übersetzung des Java-Quellcodes aus Listing 1 in Assembler könnte (in Anlehnung an die Assembler-Befehle aus dem bekannten Sum- bzw. Count-Program) in etwa so aussehen:

Code

```
10: LOAD 25        ; Wert aus Speicherzelle 25 in Akkumulator kopieren
11: EQUAL #1       ; Prüfe: Ist ACC == 1? (Eigentlich: ist lock == 1?)
12: JUMP 14        ; Prüfung ergab FALSE
13: JUMP 10        ; Prüfung ergab TRUE
14: LOAD #1        ; Der neue Wert der lock-Variablen...
15: STORE 25       ; ...wird in die Speicherzelle 25 geschieben
```

<hr>

Listing 2: Sperre in Assembler

Aus zwei Befehlen des Java-Quellcodes sind sechs Assembler-Befehle geworden. In Speicherzelle 25 steht der aktuelle Wert der lock-Variablen. Der gesamte Codeabschnitt steht in den Speicherzellen 10 bis 15, das heißt ab Speicherzelle 16 geht es weiter mit dem ersten Befehl des kritischen Abschnitts. Die konkreten Speicheradressen sind in diesem Beispiel zufällig gewählt, es obliegt dem Compiler bei der Übersetzung diese virtuellen Adressen festzulegen.

Hinweis

Es sei an dieser Stelle daran erinnert, dass direkt nach jedem Assembler-Befehl ein Kontextwechsel auf der CPU stattfinden könnte.

Ungünstigster Moment

Falls der Kontextwechsel zufällig im *ungünstigsten Moment* erfolgt, so könnte sich ein Problem ergeben. Dieser *ungünstigste Moment* liegt direkt nach dem EQUAL-Befehl, sofern die lock-Variable noch den Wert Null hat (es befindet sich also noch kein Prozess in seinem kritischen Bereich). Die Prüfung der Bedingung ist gerade erfolgt, aber der Wert der lock-Variablen (Speicherzelle 25) wurde noch nicht verändert, d.h. die Sperre wurde noch nicht gesetzt.

Code

```
10: LOAD 25         ; Wert aus Speicherzelle 25 in Akkumulator kopieren
11: EQUAL #1        ; Prüfe: Ist ACC == 1? (Eigentlich: ist lock == 1?)
          ; --> Ungünstigster Moment,
          ; --> falls direkt hier ein Kontextwechsel erfolgt!
12: JUMP 14         ; Prüfung ergab FALSE
13: JUMP 10         ; Prüfung ergab TRUE
14: LOAD #1         ; Der neue Wert der lock-Variablen...
15: STORE 25        ; ...wird in die Speicherzelle 25 geschieben
<hr>
```

Listing 3: Ungünstigste Stelle für einen Kontextwechsel

Aufgabe 1

Aufgabe

Aufg. 123: Kontextwechsel im ungünstigsten Moment

Was passiert, wenn ein Kontextwechsel im ungünstigsten Moment erfolgt und der andere Prozess nun ebenfalls seinen kritischen Abschnitt betritt, also den Wert der lock-Variablen prüft?

Aufgabe 2

Aufgabe

Aufg. 124: Mehrere ungünstigste Momente

Analysiert man den Assembler-Code genauer, so fällt auf, dass es insgesamt drei Stellen für den *ungünstigsten Moment* gibt. Die erste Stelle ist direkt nach dem EQUAL-Befehl. Wo sind die anderen beiden Stellen?

Ein Kontextwechsel im ungünstigsten Moment kann also dazu führen, dass zwei (vielleicht auch noch mehr!) Prozesse oder Threads sich gleichzeitig in ihrem kritischen Abschnitt befinden. Zwar ist die Wahrscheinlichkeit, dass dieses passiert, sehr gering, aber selbst ein extrem kleines Risiko ist nicht hinnehmbar.

Aus der Praxis

Selbst das kleinste Risiko ist zu groß!

Denke an eine Steuerungssoftware in einem Kernkraftwerk: Zwei Prozesse gleichzeitig in ihrem kritischen Abschnitt könnten eine Kernschmelze bedeuten.

Oder an einen Bordcomputer im Flugzeug (Stichwort: *Autopilot*): Zwei Prozesse gleichzeitig in ihrem kritischen Abschnitt könnten der Grund für einen Absturz sein.

Allein die Gedankenspiele *Kernkraftwerk* und *Flugzeug* rechtfertigen also, dass ein Mechanismus geschaffen wird, welcher eine Sperrvariable innerhalb einer *atomaren Aktion* abfragen und setzen kann.

Definition: Atomare Aktion

Definition

Unter einer **atomaren Aktion** (oder **atomaren Operation**) versteht man nach *Mandl 2013* Codebereiche, die in einem Stück, also atomar, ausgeführt werden müssen und (logisch) nicht unterbrochen werden dürfen.

Der Begriff *atomar* wird gerne im Sinne von *unteilbar* verwendet. Hier ist eher *ununterbrochen* gemeint, wobei der in Klammern getätigte Einschub *(logisch)* eine Lockerung der Nicht-Unterbrechbarkeit darstellt. Auf diese Lockerung wurde bei der Definition zum kritischen Abschnitt bereits eingegangen.

Nach dem bis jetzt im Rahmen dieses Lernmoduls erlangten Wissensstand ist eine atomare Aktion nichts weiter als ein einzelner Assemblerbefehl, wie er beispielsweise oben in Listing 2 oder Listing 3 zu sehen ist. Denkbar ist aber auch, dass mehrere Assemblerbefehle ununterbrochen ausgeführt werden müssen.

Anmerkung

Das Ziel besteht nun darin, eine Möglichkeit zu schaffen, wie die beiden Zeilen aus Listing 1 zu einer atomaren Aktion verschmelzen können.

Eine Möglichkeit bietet dafür der TSL-Befehl, welcher im folgenden Kapitel erläutert wird.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.11.2.3 Aktives Warten mit TSL

Der TSL-Befehl ist ein spezieller Maschinenbefehl, der von einigen CPUs zur Verfügung gestellt wird. Es handelt sich also um einen Assemblerbefehl, ähnlich denen aus diesem vorangegangenen Listing. </p>

TSL, TAS oder XCHG

TSL steht als Abkürzung für **Test and Set Lock**, in einigen Quellen (oder auf einigen CPUs) heißt dieser Befehl alternativ **TAS**, was für **Test And Set** steht. *Mandl 2013* weist in Kapitel 6.2.1 auch auf den **XCHG**-Befehl hin, der auf Intel-basierten Maschinen mit entsprechender Funktionalität zu finden ist (XCHG als Abkürzung für **Exchange**, siehe Aufgabe 2 unten).

Dieses vorangegangene Listing kann nun durch Einbringung des TSL-Befehls umgestaltet werden. Zur Erinnerung: In Speicherzelle 25 ist der aktuelle Wert der Sperrvariablen lock abgelegt.

</>
Code

```
10: TSL 25        ; Hier passieren zwei Dinge als atomare Aktion:
                  ; --> Wert aus Speicherzelle 25 in Akkumulator kopieren
                  ; --> Zahl 1 in Sp.zelle 25 schreiben (setze lock=1)
11: EQUAL #0      ; Prüfe: Ist ACC == 0? (Eigentlich: ist lock == 0?)
12: JUMP 10       ; Prüfung ergab FALSE
13: ...           ; Prüfung ergab TRUE

<hr>
```

Listing 1: Sperre in Assembler mit TSL-Befehl

Im Vergleich zum vorangegangenen Listing ist Listing 1 hier deutlich kürzer. Für das Verständnis wichtig ist die Arbeitsweise des TSL-Befehls. Die CPU sorgt bei der Ausführung des TSL-Befehls dafür, dass zwei Dinge innerhalb einer Aktion *(atomar!)* passieren, weder ein Kontextwechsel, noch ein Interrupt können den TSL-Befehl unterbrechen:

1. Kopiere den aktuellen Wert der lock-Variablen aus Speicherzelle 25 in das Register R1, und
2. Schreibe in Speicherzelle 25 den Wert 1, setze also lock = 1.

Der EQUAL-Befehl vergleicht anschließend den Wert aus Register 1 mit der Zahl 0. Abhängig vom Ergebnis dieses Vergleichs muss der durchführende Prozess entweder in der Schleife weiter warten, oder darf seinen kritischen Bereich ausführen.

Aufgabe 1

Aufgabe

Aufg. 125: Speicherzelle 13

In Listing 1 oben auf dieser Seite findet sich in Speicherzelle 13 kein Befehl, sondern nur *"drei Punkte"*.

Welcher Befehl sollte in Speicherzelle 13 stehen?
Oder anders gefragt: Was sollen die *"drei Punkte"* andeuten?

Aufgabe 2

Aufgabe

Aufg. 126: XCHG bei Intel

Intel zeigt im Intel® 64 and IA-32 Architectures Optimization Reference Manual in Kapitel 8.4.2, Example 8-4, ein Beispiel für die Verwendung des XCHG-Befehls.

Wie unterscheiden sich die Funktionsweisen des gerade beschriebenen TSL-Befehls und des XCHG-Befehls?

Kann deiner Meinung nach mit beiden Varianten die gleiche Funktionalität erreicht werden? Diskutiere diese Frage in deiner Lerngruppe!

Aufgabe 3

Aufgabe

Aufg. 127: Gilt der Nachteil noch?

Hattest du in dieser Aufgabe den großen Nachteil des aktiven Wartens herausgefunden?

Gilt dieser Nachteil dann auch noch bei Einsatz des TSL-, TAS- oder XCHG-Befehls?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.11.3 Semaphore

Der niederländische Informatiker Edsger Wybe Dijkstra hat das Semaphor-Konzept in den 1960er-Jahren entwickelt, und in seinem Artikel Co-Operating Sequential Processes vorgestellt. Auf Wikipedia findet sich ein Hinweis zur Namensherkunft des Begriffs *Semaphor*.

Wie im weiteren Verlauf dieses Abschnitts noch zu sehen sein wird, können mit Hilfe des Semaphor-Konzepts eine Reihe von Problemen der Prozess-Synchronisation gelöst werden. Als großer Vorteil der Semaphore wird noch zu erkennen sein, dass sie dabei auf aktives Warten verzichten, und somit **nicht** den Nachteil der *Verschwendung von CPU-Zeit* haben.

Zunächst einige Definitionen:

Definition: Semaphor

Definition

Unter einem **Semaphor** versteht man eine Datenstruktur, welche einen ganzzahligen Zähler, sowie eine Warteschlange bereitstellt. Zusätzlich sind zwei atomare Operationen P() und V() auf diese Datenstruktur definiert.

Ein *ganzzahliger Zähler* ist in der Programmierung beispielsweise eine einfache Integer -Variable. Diese kann negative oder positive ganzzahlige Werte annehmen. Der ganzzahlige Zähler einer Semaphore hingegen kann keine negativen Werte annehmen.

Eine *Warteschlange* ist eine klassische Datenstruktur in der Informatik, die nach dem FIFO-Prinzip (First In, First Out) arbeitet. Sie dient hier zur geordneten Aufnahme von Prozessen.

Die Operation P() wird in einigen Quellen auch als down()-Operation betitelt, analog up() anstatt V().

Definition: Binärer Semaphor

Definition

Unter einem **binären Semaphor** versteht man einen Semaphor, dessen ganzzahliger Zähler nur die Werte 0 oder 1 annehmen kann.

Ein binärer Semaphor besitzt gemäß seiner Definition ebenso eine Warteschlange und auch die auf ihm operierenden P()- und V()-Operationen.

Definition: P()-Operation eines Semaphors

Definition

Die P()-Operation eines Semaphors ist als atomare Operation wie folgt definiert:

- Hat die Zählervariable einen positiven Wert (>0), dann verringere den Wert der Zählvariablen um 1.
- Wenn die Zählvariable den Wert 0 hat, dann blockiert die P()-Operation den aufrufenden Prozess und fügt ihn an das Ende ihrer Warteschlange hinzu.

Mit "blockieren" ist hier das Ändern des Prozesszustands des die P()-Operation aufrufenden Prozesses in den Zustand "Blockiert" gemeint.

Definition: V()-Operation eines Semaphors

Definition

Die V()-Operation eines Semaphors ist als atomare Operation wie folgt definiert:

- Wenn die Warteschlange nicht leer ist, dann entblockiere den ersten Prozess der Warteschlange.
- Ist die Warteschlange leer, dann erhöhe den Wert der Zählervariable um 1.

Mit "entblockieren" ist hier das Ändern des Prozesszustands des ersten Prozesses aus der Warteschlange in den Zustand "Bereit" gemeint.

Nach diesen grundlegenden Definitionen folgen mit dem Mutex und dem Zählsemaphor zwei konkretere Konzepte, anhand derer die Einsatzmöglichkeiten von Semaphoren gezeigt werden.

So geht es weiter:

Gliederung

3.2.11.3 Semaphore
3.2.11.3.1 Mutex
3.2.11.3.2 Zählsemaphor

Aufgabe 1

Aufgabe

Aufg. 128: Warum up() und down()?

Erläutere warum es Sinn macht, dass die P()-Operation auch als down()-Operation betitelt wird, und warum analog die V()-Operation als up() betitelt wird.

Aufgabe 2

Aufgabe

Aufg. 129: P() und V() sind atomar

Die P()- und V()-Operation ist jeweils als atomare Operation definiert.

Was bedeutet dies für die Ausführung dieser beiden Operationen?

Aufgabe 3

Aufgabe

Aufg. 130: P() und V() und die Zustände

Wenn die P()-Operation ausgeführt wird, dann ändert sich gegebenenfalls der Zustand desjenigen Prozesses, der die P()-Operation aufgerufen hat.

- Von welchem alten Zustand in welchen neuen Zustand erfolgt hier ggf. die Änderung?

Wie ist das bei der V()-Operation? Ändert sich hier eventuell auch der Zustand des Prozesses, der die V()-Operation aufgerufen hat?

- Falls ja: Von welchem alten Zustand in welchen neuen Zustand erfolgt hier ggf. die Änderung?
- Falls nein: Was ändert sich dann?

Aufgabe 4

Aufgabe

Aufg. 131: V()-Operation und die Warteschlange

In der Definition zur V()-Operation heisst es:

- *Wenn die Zählvariable den Wert Null hat, dann entblockiere den ersten Prozess der Warteschlange.*

Kann es passieren, dass der Wert der Zählvariablen gleich Null ist, aber kein Prozess entblockiert werden kann, weil die Warteschlange leer ist?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.11.3.1 Mutex

Zunächst die Definition:

Definition: Mutex

Definition

Unter einem **Mutex** (als Abkürzung für **MUTual EXclusion**, auf deutsch: **gegenseitiger Ausschluss**) versteht man einen binären Semaphor.

Mit dieser Definition geht einher, dass ein Mutex eine ganzzahlige Variable besitzt, welche nur die Werte 0 und 1 annehmen darf. Ebenso besitzt er eine Warteschlange, sowie die P()- und V()-Operation.

Ein binärer Semaphor oder Mutex ist geeignet, um kritische Abschnitte vor gleichzeitigem Betreten zu sichern. Man spricht in diesem Zusammenhang von einem *wechselseitigen Ausschluss* (auf englisch: *mutual exclusion*) der beteiligten Prozesse.

Der folgende Abschnitt erläutert den wechselseitigen Ausschluss.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.11.3.1.1 Wechselseitiger Ausschluss

Der folgende Quellcode ist bereits aus dem Kapitel Kritischer Abschnitt bekannt: </p>

```
1  public class Beispiel_Kritischer_Abschnitt {
2
3  static int counter = 0;
4
5  public static class Thread_A extends Thread {
6      public void run() {
7          do_something(); // unkritisch
8          count_from_10(); // kritisch !!!
9          do_something_else(); // unkritisch
10     }
11     private void do_something() {
12         // unkritischer Abschnitt
13         System.out.println("Thread_A: unkritisch");
14     }
15     private void count_from_10() {
16         // Vorsicht: kritischer Abschnitt!
17         counter = 10;
18         counter++;
19         counter++;
20         System.out.println("A-Counter: " + counter);
21     }
22     private void do_something_else() {
23         // unkritischer Abschnitt
24         System.out.println("Thread_A: wieder unkritisch");
25     }
26 }
27
28 public static class Thread_B extends Thread {
29     public void run() {
30         System.out.println("Thread_B ist gestartet.");
31         counter = 20;
32         counter++;
33         counter++;
```

```
34          counter++;
35          counter++;
36          counter++;
37          counter++;
38          System.out.println("B-Counter: " + counter);
39      }
40  }
41
42  public static void main(String[] args) {
43      Thread a = new Thread_A();
44      Thread b = new Thread_B();
45      a.start();
46      b.start();
47  }
48
49  }
```

List. 5: Listing 1: Beispiele für kritische und unkritische Abschnitte
Ein Java-Programm mit zwei Threads. Bei Thread_A wechseln sich ein unkritischer, ein kritischer und noch ein unkritischer Abschnitt ab.

Erläuterungen zu Listing 1

Zwei Threads A und B greifen jeweils auf die counter-Variable zu. Die counter-Variable stellt damit ein gemeinsam genutztes Betriebsmittel dar.

In beiden Threads lassen sich kritische Abschnitte identifizieren. Es muss dafür gesorgt werden, dass sich immer nur ein Thread zur Zeit in seinem kritischen Abschnitt befindet.

Dies nennt man den *wechselseitigen Ausschluss*: Wenn sich ein Thread in seinem kritischen Abschnitt befindet, dann muss ausgeschlossen sein, dass auch der andere Thread in seinen kritischen Abschnitt eintritt.

Hinweis

Du kennst bereits das aktive Warten (mit TSL), durch das ein wechselseitiger Ausschluss realisiert werden kann. Jedoch verschwendet das aktive Warten CPU-Zeit.

Ein Mutex kommt nun anstatt des aktiven Wartens zum Einsatz. Somit verschwindet auch der Nachteil der CPU-Zeitverschwendung.

Mutex in Listing 1 einbauen

Um einen Mutex in Listing 1 einzubauen, muss diese Datenstruktur zunächst erzeugt werden, um anschließend die P()- und V()-Operationen an den geeigneten Stellen aufzurufen.

Listing 2 zeigt dieses Vorgehen, direkt darunter gibt es einige Erläuterungen.

```
public class Wechselseitiger_Ausschluss_mit_Mutex {

static int counter = 0;
static Semaphore mutex = new Semaphore(1);

public static class Thread_A extends Thread {
    public void run() {
        do_something(); // unkritisch
        count_from_10(); // kritisch !!!
        do_something_else(); // unkritisch
    }
    private void do_something() {
        // unkritischer Abschnitt
        System.out.println("Thread_A: unkritisch");
    }
    private void count_from_10() {
        // Vorsicht: kritischer Abschnitt!

        P(mutex);

        counter = 10;
        counter++;
        counter++;
        System.out.println("A-Counter: " + counter);

        V(mutex);
    }
    private void do_something_else() {
        // unkritischer Abschnitt
        System.out.println("Thread_A: wieder unkritisch");
    }
}

public static class Thread_B extends Thread {
    public void run() {
        System.out.println("Thread_B ist gestartet.");

        P(mutex);

        counter = 20;
        counter++;
        counter++;
        counter++;
        counter++;
        counter++;
        counter++;
        System.out.println("B-Counter: " + counter);

        V(mutex);
    }
}

public static void main(String[] args) {
    Thread a = new Thread_A();
    Thread b = new Thread_B();
    a.start();
    b.start();
```

```
58 }
59
60 }
```

List. 6: Listing 2: Ergänzt um einen Mutex

Erläuterungen zu Listing 2 mit Mutex

Entscheidend sind hier die Zeilen 4, 19, 26, 38 und 49.

In Zeile 4 wird zunächst eine Variable mit dem Namen mutex vom Datentyp Semaphore deklariert und initialisiert. Hier wird also ein binärer Semaphor (Mutex) erzeugt.

Der kritische Abschnitt von Thread_A liegt in den Zeilen 21 bis 24.
Direkt davor (in Zeile 19) wird die Methode P(mutex); aufgerufen, direkt danach (in Zeile 26) wird die Methode V(mutex); aufgerufen.

Der kritische Abschnitt von Thread_B liegt in den Zeilen 40 bis 47.
Direkt davor (in Zeile 38) wird die Methode P(mutex); aufgerufen, direkt danach (in Zeile 49) wird die Methode V(mutex); aufgerufen.

Mutex-Beispiel mit Threads

Hinweis

Der hier abgedruckte Quellcode zu Listing 2 ist in dieser Form nicht ausführbar. Er soll lediglich beispielhaft verdeutlichen, an welchen Stellen zusätzliche Operationen im Bezug auf den Mutex zur Realisierung des wechselseitigen Ausschlusses eingefügt werden müssen.

Zwar gibt es in Java eine Klasse Semaphore, siehe
http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Semaphore.html
jedoch sind hier die P() und V()-Operationen anders benannt:
acquire() bzw. release()

Unter dem hier verlinkten Dokument findet sich bei Bedarf ein anderes sehr einfaches Java-Beispiel, welches den praktischen Einsatz der Semaphore-Klasse zeigt.

Aufgabe 1

Aufgabe

Aufg. 132: Mutex in Action

Spiele einmal gedanklich den Ablauf der beiden Threads aus Listing 2 durch.

Was passiert, wenn Thread_A bereits seinen kritischen Abschnitt betreten hat, und beispielsweise in Zeile 23 ein Kontextwechsel zu Thread_B erfolgt?

- Thread_B möchte auch seinen kritischen Abschnitt betreten und ruft in Zeile 38 die P()-Operation auf... Und dann?
- Was passiert mit Thread-B?
- Und wie kommt Thread-B da wieder raus?

Aufgabe 2

Aufgabe

Aufg. 133: Aktives Warten vs. Mutex

Vergleiche den obigen Quellcode aus Listing 2 (Mutex) einmal mit dem bereits bekannten Quellcode mit dem Beispiel des Aktiven Wartens (mit while).

```
1  public class Beispiel_Aktives_Warten_mit_while {
2
3  static int counter = 0; // gemeinsam genutztes Betriebsmittel
4  static int lock = 0; // Sperrvariable
5
6  public static class Thread_A extends Thread {
7      public void run() {
8          do_something(); // unkritisch
9          count_from_10(); // kritisch !!!
10         do_something_else(); // unkritisch
11     }
12     private void do_something() {
13         // unkritischer Abschnitt
14         System.out.println("Thread_A: unkritisch");
15     }
16     private void count_from_10() {
17         // Vorsicht: kritischer Abschnitt!
18         while (lock == 1); // Semikolon beachten!
19         lock = 1;
20
21         counter = 10;
22         counter++;
23         counter++;
24         System.out.println("A-Counter: " + counter);
25
26         lock = 0;
27     }
28     private void do_something_else() {
29         // unkritischer Abschnitt
30         System.out.println("Thread_A: wieder unkritisch");
31     }
32 }
33
34 public static class Thread_B extends Thread {
35     public void run() {
36         System.out.println("Thread_B ist gestartet.");
```

```
37          while (lock == 1); // Semikolon beachten!
38          lock = 1;
39
40          counter = 20;
41          counter++;
42          counter++;
43          counter++;
44          counter++;
45          counter++;
46          counter++;
47          System.out.println("B-Counter: " + counter);
48
49          lock = 0;
50      }
51  }
52
53  public static void main(String[] args) {
54      Thread a = new Thread_A();
55      Thread b = new Thread_B();
56      a.start();
57      b.start();
58  }
59
60  }
```

List. 7: Beispiel: Aktives Warten
Ein Java-Programm mit zwei Threads. Die Sperrvariable soll ein gleichzeitiges Betreten der kritischen Abschnitte durch beide Threads verhindern.

Hier ist ein Bild mit beiden Quelltexten nebeneinander:

```java
public class Beispiel_Aktives_Warten_mit_while {

static int counter = 0;        // gemeinsam genutztes Betriebsmittel
static int lock = 0;           // Sperrvariable

public static class Thread_A extends Thread {
        public void run() {
                do_something();          // unkritisch
                count_from_10();         // kritisch !!!
                do_something_else();     // unkritisch
        }
        private void do_something() {
                // unkritischer Abschnitt
                System.out.println("Thread_A: unkritisch");
        }
        private void count_from_10() {
                // Vorsicht: kritischer Abschnitt!
                while (lock == 1);       // Semikolon beachten!
                lock = 1;

                counter = 10;
                counter++;
                counter++;
                System.out.println("A-Counter: " + counter);

                lock = 0;
        }
        private void do_something_else() {
                // unkritischer Abschnitt
                System.out.println("Thread_A: wieder unkritisch");
        }
}

public static class Thread_B extends Thread {
        public void run() {
                System.out.println("Thread_B ist gestartet.");
                while (lock == 1);       // Semikolon beachten!
                lock = 1;

                counter = 20;
                counter++;
                counter++;
                counter++;
                counter++;
                counter++;
                counter++;
                System.out.println("B-Counter: " + counter);

                lock = 0;
        }
}

public static void main(String[] args) {
        Thread a = new Thread_A();
        Thread b = new Thread_B();
        a.start();
        b.start();
}

}
```

```java
public class Wechselseitiger_Ausschluss_mit_Mutex {

static int counter = 0;
static Semaphore mutex = new Semaphore(1);

public static class Thread_A extends Thread {
        public void run() {
                do_something();          // unkritisch
                count_from_10();         // kritisch !!!
                do_something_else();     // unkritisch
        }
        private void do_something() {
                // unkritischer Abschnitt
                System.out.println("Thread_A: unkritisch");
        }
        private void count_from_10() {
                // Vorsicht: kritischer Abschnitt!

                P(mutex);

                counter = 10;
                counter++;
                counter++;
                System.out.println("A-Counter: " + counter);

                V(mutex);
        }
        private void do_something_else() {
                // unkritischer Abschnitt
                System.out.println("Thread_A: wieder unkritisch");
        }
}

public static class Thread_B extends Thread {
        public void run() {
                System.out.println("Thread_B ist gestartet.");

                P(mutex);

                counter = 20;
                counter++;
                counter++;
                counter++;
                counter++;
                counter++;
                counter++;
                System.out.println("B-Counter: " + counter);

                V(mutex);
        }
}

public static void main(String[] args) {
        Thread a = new Thread_A();
        Thread b = new Thread_B();
        a.start();
        b.start();
}

}
```

Mutex mit Prozessen

Die in Listing 2 beispielhaft gezeigte Verwendung eines Mutex bezieht sich im Quellcode auf zwei Threads. Ein Mutex wird in Betriebssystemen aber üblicherweise eingesetzt, um die kritischen Abschnitte zweier Prozesse (mit einem gemeinsam genutzten Betriebsmittel) gegeneinander abzuschotten.

Aufgabe 3

Aufgabe

Aufg. 134: Mutex und Prozesse bei Wikipedia

Wikipedia zeigt auf der Seite

http://de.wikipedia.org/wiki/Semaphor_%28Informatik%29#Anwendungsbeispiele

im Abschnitt *"Einsatz in Konkurrenzsituationen I"* ein sehr einfach gehaltenes Beispiel mit zwei Prozessen und einem Mutex.

Schau' es dir dort an!

Das Erzeugen eines Mutex sowie die P()- und V()-Operation kann man sich bei Betriebssystemen als Systemaufrufe vorstellen. Somit verwaltet das Betriebssystem die Warteschlange eines Semaphors und kann problemlos dafür sorgen, dass die P()- und V()-Operation atomar abläuft. Das Betriebssystem kann dann mit dem Scheduler die wartenden Prozesse in den Status "wartend" versetzen - was nicht möglich wäre, wenn die Programmiersprache selbst die Mutex-Verwaltung durchführen würde.

Aufgabe 4

Aufgabe

Aufg. 135: Applet zum wechselseitigen Ausschluss

An der FH Köln wird ein Semaphor Workshop mit Java-Applets bereitgestellt, anhand derer der wechselseitige Ausschluss mit Hilfe eines binären Semaphors (Mutex) nachvollzogen werden kann. Probiere es aus!

http://www.nt.fh-koeln.de/fachgebiete/inf/diplom/semwork/beispiele/waus/waus.html

Falls das Java-Applet in deinem Browser nicht startet, musst du eventuell die Java-Sicherheitseinstellungen anpassen. Trage dort ein: http://www.nt.fh-koeln.de

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.11.3.1.2 Reihenfolgedurchsetzung

Man spricht von Reihenfolgedurchsetzung, wenn zwei Prozesse zwingend in einer bestimmten Reihenfolge auf ein gemeinsames Betriebsmittel zugreifen sollen: Erst nachdem der erste Prozess seinen Zugriff erledigt hat, darf der zweite Prozess zugreifen. Die Reihenfolge des Zugriffs ist also wichtig.

Aufgabe 1

Aufgabe

Aufg. 136: Applet zur Reihenfolgedurchsetzung

An der FH Köln wird ein Semaphor Workshop mit Java-Applets bereitgestellt, anhand derer die Reihenfolgedurchsetzung mit Hilfe eines binären Semaphors (Mutex) nachvollzogen werden kann. Probiere es aus!

http://www.nt.fh-koeln.de/fachgebiete/inf/diplom/semwork/beispiele/reihenf/reihenf.html

Falls das Java-Applet in deinem Browser nicht startet, musst du eventuell die Java-Sicherheitseinstellungen anpassen. Trage dort ein: http://www.nt.fh-koeln.de

Aufgabe 2

Aufgabe

Aufg. 137: Reihenfolgedurchsetzung vs. Wechselseitiger Ausschluss

In der vorangegangenen Aufgabe kannst du im Applet sehen, wann und in welchem Prozess die P()- und die V()-Operation aufgerufen wird.

Welcher Unterschied ergibt sich dabei zum Aufruf der P()- und der V()-Operation beim wechselseitigen Ausschluss?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.11.3.2 Zählsemaphor

Zunächst die Definition:

Definition: Zählsemaphor

Definition

Unter einem **Zählsemaphor** versteht man einen Semaphor, der den Zugriff auf eine beschränkte Anzahl eines Betriebsmittels regelt.

Ein Zählsemaphor ist geeignet, um eine begrenzte Anzahl **eines** Betriebsmittels zu verwalten. Dieses Betriebsmittel kann entsprechend seiner Anzahl genutzt werden, aber nicht darüber hinaus. Der Zugriff auf das Betriebsmittel stellt deshalb einen kritischen Abschnitt dar und muss geschützt werden.

Beispiel aus dem Studentenleben

Beispiel

In einer Bibliothek stehen beispielsweise fünf Ausgaben des Buchs *Tanenbaum 2009* . Damit können maximal fünf Studierende dieses Buch ausleihen. Das Betriebsmittel ist in diesem Beispiel das Buch. Ein Studierender repräsentiert einen Prozess.

Solange noch Ausgaben des Buchs in der Bibliothek vorhanden sind, kann es ausgeliehen werden. Sind alle Ausgaben verliehen, so muss der nächste ausleihwillige Studierende so lange warten, bis (mindestens) ein Buch abgegeben wurde, der Ausleihvorgang dafür also beendet wurde.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.11.3.2.1 Erzeuger- / Verbraucherproblem

Ein Erzeuger erzeugt etwas, was der Verbraucher verbraucht. Natürlich kann der Verbraucher erst dann etwas verbrauchen, wenn zuvor der Erzeuger auch etwas erzeugt hat. Und ebenso kann der Erzeuger nur soviel erzeugen, bis alle Lagerplätze für das Erzeugnis belegt sind. Dann muss der Verbraucher erst wieder etwas verbrauchen, bis der Erzeuger wieder etwas erzeugen kann.

Alles klar?

Aufgabe 1

Aufgabe

Aufg. 138: Applet zum Erzeuger- / Verbraucherproblem

An der FH Köln wird ein Semaphor Workshop mit Java-Applets bereitgestellt, anhand derer das Erzeuger- / Verbraucherproblem mit Hilfe zweier Zählsemaphore und eines binären Semaphors (Mutex) nachvollzogen werden kann. Probiere es aus!

http://www.nt.fh-koeln.de/fachgebiete/inf/diplom/semwork/beispiele/erzver/erzver.html

Falls das Java-Applet in deinem Browser nicht startet, musst du eventuell die Java-Sicherheitseinstellungen anpassen. Trage dort ein: http://www.nt.fh-koeln.de

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.11.3.2.2 Philosophenproblem

Die Philosophen dieses erdachten Problems können denken, essen oder schlafen. Wobei sie sofort dann einschlafen, wenn sie zwar essen wollen, es aber nicht können (weil eine Gabel fehlt). Wenn sie dann einmal schlafen, können sie nur von einem anderen Philosophen wieder aufgeweckt werden.

Aufgabe 1

Aufgabe

Aufg. 139: Applet zum Philisophenproblem

An der FH Köln wird ein Semaphor Workshop mit Java-Applets bereitgestellt, anhand derer das Philosophenproblem mit Hilfe von insgesamt fünf Semaphoren nachvollzogen werden kann. Probiere es aus!

http://www.nt.fh-koeln.de/fachgebiete/inf/diplom/semwork/beispiele/phil/phil.html

Welcher Semaphor aus dem Applet hat welche Aufgabe?

Falls das Java-Applet in deinem Browser nicht startet, musst du eventuell die Java-Sicherheitseinstellungen anpassen. Trage dort ein: http://www.nt.fh-koeln.de

Wenn nichts mehr geht: Deadlock

Beim Philosophenproblem kann eine Situation eintreten, bei der alle Philosophen einschlafen. Dann ist keiner mehr wach, um die anderen zu wecken. Alle Philosophen schlafen für immer.

Diese Situation nennt man einen Deadlock.

Hinweis

Deadlocks werden später in einem eigenständigen Kapitel detaillierter betrachtet.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.11.4 Monitore

Dieses Kapitel wird in der weiterführenden Literatur behandelt: </p>

Hinweis

Weiterführende Literatur

Mandl 2013 erläutert Monitore in Kapitel 6.2.3. Die Lektüre dieser Quelle sei ausdrücklich empfohlen.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Weblinks

- Sir Charles Antony Richard Hoare ; britischer Informatiker; *11. Januar 1934
- Per Brinch Hansen ; dänischer Informatiker; *13. November 1938 bis †31. Juli 2007

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.11.5 Zusammenfassung Synchronisation

Das Kapitel zur Prozess-Synchronisation lässt sich ganz einfach zusammenfassen: </p>

Zusammenfassung

- Durch nebenläufige Ausführung von Prozessen (oder Threads) können Race Conditions entstehen.
- Durch aktives Warten können kritische Abschnitte geschützt werden. Race Conditions werden dadurch vermieden.
- Um dabei das Problem des ungünstigsten Moments zu lösen, mussten die CPU-Hersteller den TSL-Befehl in den Prozessor integrieren.
- Der Nachteil des aktiven Wartens ist die Verschwendung von CPU-Zeit.
- Diesen Nachteil besitzen Semaphore nicht, denn sie blockieren Prozesse, wodurch diese keine weitere CPU-Zeit mehr zugesprochen bekommen.
- Ein durch einen Semaphor blockierter Prozess muss durch einen anderen Prozess wieder entblockiert werden. Wenn der Programmierer des anderen Prozesses dieses vergisst, so gibt es ein Problem.
- Das größte Problem beim Einsatz von Semaphoren ist also der Mensch (→ der Programmierer).
- Das Monitor-Konzept wurde als Verbesserung des Semaphor-Konzepts entwickelt.
- Dabei wird der Mensch (→ Programmierer) als Fehlerquelle bei der Synchronisation weitgehend ausgeschlossen.

Alternative Webquelle zum Thema

Quelle im Web

Operating Systems: Process Synchronization

http://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/5_Synchronization.html

Dr. John T. Bell

Department of Computer Science

University of Illinois, Chicago

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.11.6 Synchronisationstechniken moderner Betriebssysteme

Dieses Kapitel wird in der weiterführenden Literatur behandelt: </p>

Hinweis

Weiterführende Literatur

Mandl 2013 erläutert das Thema in Kapitel 6.3. Die Lektüre dieser Quelle sei dem geneigten Leser überlassen.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.11.7 Synchronisationsmechanismen in Programmiersprachen

Dieses Kapitel wird in der weiterführenden Literatur behandelt: </p>

Hinweis

Weiterführende Literatur

Mandl 2013 erläutert das Thema in Kapitel 6.4. Die Lektüre dieser Quelle sei dem geneigten Leser überlassen.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.12 Deadlocks

Deadlocks sind eine unangenehme Sache. Sie sollten besser nicht auftreten, aber das kann man leider nicht selbst bestimmen. Zunächst die Definition: </p>

Definition: Deadlock-Zustand

Definition

Eine Menge von Prozessen befindet sich nach *Tanenbaum 2009* in einem **Deadlock-Zustand**, wenn jeder Prozess aus der Menge auf ein Ereignis wartet, das nur ein anderer Prozess aus der Menge auslösen kann.

Wenn sich mehrere Prozesse in einem Deadlock-Zustand befinden, so sagt man auch vereinfachend: Es ist ein Deadlock aufgetreten.

Der englische Betriff *Deadlock* wird auf deutsch gerne mit *Verklemmung* übersetzt.

Eine Analogie aus der Realität

In der realen Welt gibt es eine schöne Analogie zum Deadlock-Zustand von Prozessen:

Aus der Praxis

Wenn du dir vorstellen kannst, dass ein Auto im Straßenverkehr einen Prozess repräsentiert, dann zeigt dieses Bild einen Deadlock-Zustand einer Menge von Autos. (Hier gibt es eine kleine Sammlung mit ähnlichen Fotos.)

In Anbetracht dieser Bilder kannst du überlegen, ob die Menge der Prozesszustände noch um einen ergänzt werden sollte. Welcher Zustand ist damit gemeint?

Lösung

Hupend!

Ein Deadlock beim Philosophenproblem

Im Kapitel zum Philosophenproblem wurde bereits auf die Möglichkeit eines Deadlocks hingewiesen. Die folgenden Aufgaben greifen dieses wieder auf.

Aufgabe 1

Aufgabe

Aufg. 140: Deadlock-Philosophen

Mandl 2013 geht am Ende von Kapitel 6.2.2 auf das Philosophenproblem und eine dabei bestehende Deadlock-Gefahr ein.

Erläutere:

- Unter welcher Bedingung tritt bei den speisenden Philosophen ein Deadlock-Zustand ein?
- Welche Rolle spielt eine dabei?

Aufgabe 2

Aufgabe

Aufg. 141: Applet zum Philisophenproblem

An der FH Köln wird ein Semaphor Workshop mit Java-Applets bereitgestellt, anhand derer das Philosophenproblem mit Hilfe von insgesamt fünf Semaphoren nachvollzogen werden kann.

http://www.nt.fh-koeln.de/fachgebiete/inf/diplom/semwork/beispiele/phil/phil.html

Erzeuge in dem Applet einen Deadlock!

Falls das Java-Applet in deinem Browser nicht startet, musst du eventuell die Java-Sicherheitseinstellungen anpassen. Trage dort ein: http://www.nt.fh-koeln.de

So geht es weiter:

Gliederung

3.2.12 Deadlocks
- 3.2.12.1 Vier Bedingungen nach Coffman
- 3.2.12.2 Deadlocks erkennen
- 3.2.12.3 Deadlocks ignorieren
- 3.2.12.4 Deadlocks vermeiden
- 3.2.12.5 Deadlocks verhindern
- 3.2.12.6 Deadlock-Fazit

Alternative Webquelle zum Thema

Quelle im Web

Operating Systems: Deadlocks

http://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/7_Deadlocks.html

Dr. John T. Bell
Department of Computer Science
University of Illinois, Chicago

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.12.1 Vier Bedingungen nach Coffman

Eine grundlegende Arbeit über *System Deadlocks* veröffentlichten E.G. Coffman, Jr.; M.J. Elphick und A. Shoshani im Jahre 1971 in der Zeitschrift Computing Surveys, Vol. 3, No. 2 ; (hier ist ein alternativer Link zu diesem Dokument).

Sie beschreiben darin vier Bedingungen, welche - sobald sie allesamt zur gleichen Zeit eingetreten sind - einen Deadlock-Zustand verursacht haben:

1. *Mutual exclusion condition*
   Eine Ressource steht einem Prozess nur exklusiv zur Verfügung, sie kann also nicht gleichzeitig von mehreren Prozessen belegt werden.

2. *Wait for condition*
   Prozesse warten und behalten dabei die Kontrolle über bereits zugewiesene Ressourcen solange, bis sie alle Ressourcen zugesprochen bekommen haben, um schließlich ihre Arbeit fortführen zu können.

3. *No preemption condition*
   Zugewiesene Ressourcen können einem Prozess nicht gewaltsam wieder entrissen werden.

4. *Circular wait condition*
   Es gibt eine zyklische Kette von Prozessen, die bereits eine oder mehrere Ressourcen zugewiesen bekommen haben, und die gleichzeitig auf weitere Ressourcen warten, welche bereits dem jeweils nächsten Prozess in der Kette zugesprochen wurden.

Wenn alle vier Bedingungen zur selben Zeit zutreffen, dann liegt ein Deadlock vor.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.12.2 Deadlocks erkennen

Die vierte der gerade beschriebenen Bedingungen (Circular wait condition) liefert eine Möglichkeit, nach der ein bereits eingetretener Deadlock-Zustand auf einem System entdeckt werden kann. Man realisiert dies durch die Konstruktion eines speziellen Graphen, der anschließend untersucht wird.

Betriebsmittelgraph

Das folgende Video veranschaulicht diese beiden Schritte:

- Schritt 1: Konstruktion des Betriebsmittelgraphen

- Schritt 2: Überprüfung des Betriebsmittelgraphen auf Zyklen

Wenn (mindestens ein) Zyklus im Graphen vorhanden ist, dann liegt ein Deadlock vor.

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/mhrohwfGQU0

Med. 31: Deadlocks erkennen mit Hilfe eines Betriebsmittelgraphen (09:46)
http://youtu.be/mhrohwfGQU0
CC-BY

Ein *Betriebsmittelgraph* wird in gängiger Literatur alternativ auch *Belegungs-Anforderungs-Graph* genannt.

Aufgabe 1

Aufgabe

Aufg. 142: Deadlocks erkennen mit Hilfe eines Betriebsmittelgraphen

Mache dich mit dem Tutorial Deadlock-Algorithmen der Universität Oldenburg vertraut und vollziehe insbesondere die beiden Schritte Konstruktion und Überprüfung des Betriebsmittelgraphen nach!

Aufgabe 2

Aufgabe

Aufg. 143: Zeichne den Betriebsmittelgraph und erkenne den Deadlock!

Gegeben seien die Prozesse *P1* bis *P5* und die Ressourcen *Drucker*, *Plotter*, *Modem*, *Magnetbandlaufwerk* sowie *Diskettenlaufwerk* und *CD-ROM-Laufwerk*. Jede Ressource sei genau einmal vorhanden.

Aktuell besteht folgende Ressourcenzuteilung und -anforderung:

- *P1* belegt *Plotter* und fordert *Diskettenlaufwerk* an.
- *P2* belegt *Magnetbandlaufwerk* und fordert *Plotter* und *Modem* an.
- *P3* belegt *Modem* und fordert *Drucker* an.
- *P4* belegt *Drucker* und fordert *CD-ROM-Laufwerk* an.
- *P5* belegt *Diskettenlaufwerk* und fordert *Magnetbandlaufwerk* an.

Zeichne den Betriebsmittelgraph! Achte dabei auf die Pfeilrichtungen!

Offensichtlich liegt hier ein Deadlock vor.

- Woran ist dies im Betriebsmittelgraphen erkennbar?
- Welche Prozesse sind daran beteiligt?

Wie man sieht, hilft der Betriebsmittelgraph tatsächlich bei der Erkennung eines Deadlocks. Aber:

Hinweis

Wenn du die letzte Aufgabe zum Betriebsmittelgraphen durchgeführt hast, dann hast du (**ein Mensch!**) den Deadlock erkannt.

Eigentlich ist es aber die Aufgabe **des Betriebssystems** einen Deadlock zu erkennen!

Die Frage ist also:

Frage

Kann **ein Betriebssystem** auch einen Betriebsmittelgraphen konstruieren und einen Zyklus darin erkennen?

Die Antwort ist: Ja, das geht!

Prof. Dr. Holger Schlingloff vom Institut für Informatik der Humboldt-Universität zu Berlin geht darauf in seinem Aufsatz Zyklensuche (hier auch als PDF) ein.

Hinweis

Es wird an dieser Stelle nicht näher erläutert, wie die Graphenkonstruktion (bzw. -repräsentation) im Rechner, sowie die anschließende Zyklensuche abläuft.

Es sei aber darauf verwiesen, dass diese Tätigkeit eine Reihe anderer Dinge benutzt, die für Studierende an Hochschulen üblicherweise in Vorlesungen zur *Mathematik* oder zu *Algorithmen und Datenstrukturen* behandelt werden.

Falls du diese Vorlesungen bereits gehört hast, kommen dir diese Stichwörter vielleicht bekannt vor: Adjazenzmatrix, Nachbarschaftsliste, Rekursion, Tiefensuche, Breitensuche.

Es ist jetzt also klar, dass ein Betriebssystem die folgenden beiden Dinge tun muss, um einen gegebenenfalls vorhandenen Deadlock-Zustand zu erkennen:

1. Konstruiere den aktuellen Betriebsmittelgraphen,
   d.h. repräsentiere ihn als interne Datenstruktur.

2. Überprüfe den Graphen auf einen Zyklus,
   d.h. überprüfe die interne Datenstruktur entsprechend.

Falls ein Zyklus erkannt wird, so liegt ein Deadlock vor, und es können auch die beteiligten Prozesse und Ressourcen identifiziert werden.

Falls kein Zyklus erkannt wird, so besteht die Gewissheit, dass "alles in Ordnung" ist, d.h. es liegt kein Deadlock vor.

Aufgabe 3

Aufgabe

Aufg. 144: Was tun bei erkanntem Deadlock?

Was sollte das Betriebssystem tun, wenn es einen Deadlock erkannt und die zugehörigen Prozesse und Betriebsmittel identifiziert hat?

Überlege, recherchiere und diskutiere in deiner Lerngruppe! Schätze für jeden Vorschlag auch die möglichen Folgen ab.

Aufgabe 4

Aufgabe

Aufg. 145: Wie oft nach einem Deadlock suchen?

Wie oft sollte ein Betriebssystem die Suche nach einem eventuell existenten Deadlock durchführen?

- Einmal pro Sekunde?
- Einmal pro Minute?
- Einmal pro Stunde?
- Einmal am Tag?
- etc.

Die pragmatischte Antwort lautet immer: "Es kommt darauf an!"

Dann erläutere doch mal: Worauf kommt es an?

- Welche Situationen oder Einsatzzwecke für Betriebssysteme sind vorstellbar, bei denen eine Suche nach Deadlocks in kürzeren oder längeren Zeitabständen durchgeführt werden sollte?
- Welche Vor- oder Nachteile hat eine (zu) häufige oder (zu) seltene Suche?

Vielleicht liefert die Antwort auf die letzte Aufgabe ja auch Gründe für das nächste Thema: Deadlocks ignorieren.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.12.3 Deadlocks ignorieren

Eine durchaus gängige Methode im Umgang mit Deadlocks ist das **Ignorieren**:

Aus der Praxis

"Es gibt keine Deadlocks, weil ich daran glaube, dass es keine Deadlocks gibt!", sagt das Betriebssystem.

Rechtfertigen kann man diese Haltung, wenn die Wahrscheinlichkeit des Auftretens eines Deadlocks gering ist, und gleichzeitig die Folgen eines aufgetretenen Deadlocks "nicht dramatisch" sind (wie immer man "dramatisch" dann auch definieren möchte).

Zum Ignorieren von Deadlocks kommt in Betriebssystemen üblicherweise der Vogel-Strauß-Algorithmus zum Einsatz.

Aufgabe 1

Aufgabe

Aufg. 146: Implementiere!

Implementiere den Vogel-Strauß-Algorithmus in einer beliebigen Hochsprache!
Welcher Quellcode fehlt zwischen den geschweiften Klammern?

```
void ausfuehrenVogelStraussAlgorithmus() {
  // ? ? ?
}
```

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.12.4 Deadlocks vermeiden

Wenn es um die Erkennung von Deadlocks geht, dann impliziert dies immer, dass der Deadlock-Zustand bereits eingetreten ist. Irgendeine Art von negativen Folgen wird sich daraus zwangsläufig ergeben.

Eine andere Strategie ist die **Vermeidung von Deadlocks**. Ein Betriebssystem könnte von vorneherein so konstruiert werden, dass Deadlocks gar nicht möglich sind.

Wichtig

Wenn man ein Betriebssystem entwickelt, welches mindestens eine der vier Bedingungen für Deadlocks **unerfüllbar** macht, dann können Deadlocks überhaupt nicht auftreten!

Man betrachte also einzelne Möglichkeiten in dieser Richtung. Die folgenden Aufgaben beschäftigen sich damit.

Aufgabe 1

Aufg. 147: Spooling

Recherchiere und erläutere:

Was versteht man unter Spooling? (Zum Beispiel bei einem Drucker-Spooler.)

Spooling ist prinzipiell geeignet, um Deadlocks zu vermeiden. Welche der vier Bedingungen macht Spooling unerfüllbar?

Aufgabe 2

Aufg. 148: Überweis' mal was

Denke dir folgende Situation in einer Bank:

Es werden von einem Großrechner der Bank viele Überweisungen durchgeführt. Der Einfachheit halber (für diese Aufgabe) nur innerhalb derselben Bank.

Zu einer Überweisung gehören drei Dinge:

1. Der Überweisungsbetrag.
2. Die Nummer des Kontos, von dem der Betrag abgebucht wird (Konto A).
3. Die Nummer des Kontos, dem der Betrag gutgeschrieben wird (Konto B).

Für jede Überweisung startet das Betriebssystem des Großrechners einen separaten Prozess. Dieser *Prozess zur Durchführung einer einzelnen Überweisung* geht nun wie folgt vor:

- Reserviere Konto A.
- Reserviere Konto B.
- Führe aus: A minus Überweisungsbetrag ("abbuchen").
- Führe aus: B plus Überweisungsbetrag ("gutschreiben").
- Gib Konto B frei.
- Gib Konto A frei.

Die beiden Konten A und B sind damit die vom Überweisungsprozess benötigten Betriebsmittel.

Erläutere eine Situation, bei der es durch die zu erledigenden Überweisungen zu einem Deadlock kommt! Zeichne dazu den Betriebsmittelgraphen.

Aufgabe 3

Aufgabe

Aufg. 149: Überweisen ohne Deadlock?

Es geht wieder um Überweisungen in einer Bank, gemäß der vorangegangenen Aufgabe. Der Prozess zur Durchführung einer einzelnen Überweisung geht diesmal nach einem besonderen Muster vor.

Für jede Überweisung:

- Reserviere zunächst das Konto mit der **kleineren** Kontonummer.
- Reserviere anschließend das Konto mit der **größeren** Kontonummer.
- Führe aus: A minus Überweisungsbetrag ("abbuchen").
- Führe aus: B plus Überweisungsbetrag ("gutschreiben").
- Gebe das Konto mit der **größeren** Kontonummer wieder frei.
- Gebe das Konto mit der **kleineren** Kontonummer wieder frei.

Kann bei dieser Vorgehensweise noch ein Deadlock auftreten?

- Falls ja: Erläutere eine Beispielsituation mit dem Deadlock und zeichne den zugehörigen Betriebsmittelgraphen.
- Falls nein: Welche der vier Bedingungen ist nicht erfüllt?

Noch ein Denkanstoß

Die einfachste Situation besteht aus zwei Prozessen und zwei Konten.

Aber was ist, wenn die zwei Prozesse auf insgesamt drei Konten zugreifen?
Und wenn man noch mehr Prozesse einbezieht, dann könnte auch die Anzahl der beteiligten Konten weiter steigen.

Ist dann ein Deadlock möglich, oder nicht? Erläutere!

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.12.5 Deadlocks verhindern

Bei der Vermeidung von Deadlocks sollte eine Umgebung geschaffen werden, in der Deadlocks gar nicht erst auftreten können. In der Praxis ist das - wenn überhaupt - nur in Einzelfällen (d.h. für bestimmte, aber nicht für alle Betriebsmittel) möglich.

Hinweis

Betriebsmittel werden i.d.R. **nach und nach** von Prozessen **angefordert**. Sie werden deshalb auch **nicht alle auf einmal** dem betreffenden Prozess zugeteilt!

Angenommen einem Prozess sind bereits eine beliebige Anzahl an Betriebsmitteln zugewiesen worden. Dann geht es bei der **Verhinderung von Deadlocks** darum, dem Prozess nur dann eine weitere Ressource zuzusprechen, wenn sichergestellt ist, dass dadurch **in der Zukunft** kein Deadlock entstehen kann.

Bei der Anforderung eines Betriebsmittels muss vom Betriebssystem also eine Entscheidung getroffen werden. *Tanenbaum 2009* bringt es auf den Punkt:

Frage

Es stellt sich also die Frage, ob ein Algorithmus existiert, der Deadlocks zuverlässig verhindern kann, indem er immer die richtige Entscheidung trifft.

Eine Antwort erläutert *Tanenbaum 2009* dazu auch: "ja, aber", wobei das "aber" zu der Erkenntnis führt, dass in der Praxis immer Situationen zu befürchten sind, die nicht von dem betreffenden Algorithmus abgedeckt werden können.

Weitere Details zur Deadlock-Verhinderung können - je nach Verfügbarkeit der Quellen - nachgelesen werden bei:

- *Glatz 2010* , Kapitel 4.6.3
- *Strelen 2012* , Kapitel 5.1.3
- *Tanenbaum 2009* , Kapitel 6.5

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.12.6 Deadlock-Fazit

Zusammenfassung

Allgemein ist davon auszugehen, dass die Erkennung, Vermeidung oder Verhinderung von Deadlocks sehr aufwendig ist. Eine ausnahmlose Abdeckung aller denkbarer Fälle scheint nahezu unmöglich.

</p>

Wenn dann die Wahrscheinlichkeit für das Auftreten eines Deadlocks noch als sehr gering eingeschätzt werden kann, ist schon verständlich, dass Betriebssystem-Hersteller gerne die Strategie des "ignorierens" anwenden. Aber falls der unwahrscheinliche Fall dann doch einmal eintritt, trägt der Anwender die Folgen.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.13 Interprozesskommunikation

Bei der **Interprozesskommunikation** (englisch: **Interprocess Communication**, kurz: **IPC**) geht es um den Austausch von Informationen zwischen zwei (oder mehr) Prozessen bzw. Threads. Damit alle Beteiligten die ausgetauschten Informationen in gleicher Weise verstehen können, sind bestimmte Regeln der Kommunikation einzuhalten, das sogenannte *Protokoll*. </p>

Grundbegriffe der IPC

Die folgende Aufgabe nimmt sich der Grundbegriffe an:

Aufgabe 1

Aufgabe

Aufg. 150: IPC-Grundbegriffe

Mandl 2013 erläutert in Kapitel 6.5.1 einige Grundbegriffe der Kommunikation.

Was versteht man demnach unter:

- Verbindungsorientierter Kommunikation
- Verbindungsloser Kommunikation
- Speicherbasierter Kommunikation
- Nachrichtenbasierter Kommunikation
- Synchroner Kommunikation
- Asynchroner Kommunikation
- Halbduplex-Betrieb
- Vollduplex-Betrieb
- Unicast
- Multicast
- Anycast
- Broadcast

Im Folgenden werden einige Möglichkeiten der Interprozesskommunikation erläutert. Dabei sei es dem geneigten Leser überlassen, für jede Möglichkeit zu überlegen, welche der gerade angesprochenen Grundbegriffe zutreffen.

So geht es weiter:

Gliederung

3.2.13 Interprozesskommunikation
3.2.13.1 Zwei Threads kommunizieren über gemeinsame Variablen
3.2.13.2 Zwei Prozesse kommunizieren über gemeinsame Speicherobjekte
3.2.13.3 Zwei Prozesse kommunizieren über Shared Memory
3.2.13.4 Zwei Prozesse kommunizieren über Pipes
3.2.13.5 Zwei Prozesse kommunizieren über Sockets
3.2.13.6 Interprozesskommunikation-Fazit

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.13.1 Zwei Threads kommunizieren über gemeinsame Variablen

Beispiel

In dem Video zu den Race Conditions wurde Quellcode gezeigt, bei dem zwei Threads auf eine gemeinsame Variable counter zugreifen.

Prozessintern ist es also möglich, dass zwei (oder mehr) Threads sich gegenseitig Informationen über gemeinsame Datenbereiche (→Variablen) bereitstellen.

Eine Synchronisation der beteiligten Threads beim Zugriff auf die gemeinsamen Datenbereiche wird erforderlich, da es hier zu kritischen Abläufen kommt.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.13.2 Zwei Prozesse kommunizieren über gemeinsame Speicherobjekte

Unter *Speicherobjekten* kann man sich beispielsweise *Dateien* vorstellen. </p>

Beispiel

Ein Prozess A erzeugt während seiner Laufzeit Daten und speichert diese in einer Datei ab. Ein zweiter Prozess B liest diese Datei zu einem späteren Zeitpunkt ein und kann so die enthaltenen Informationen weiterverarbeiten.

Auch hier ist eine Synchronisation nötig. Allerdings sorgt das Betriebssystem nur für eine Synchronisation beim *gleichzeitigen Zugriff* der beiden Prozesse auf die Datei (z.B. über Semaphore).

Falls hingegen Prozess B den Dateiinhalt ausliest, lange bevor Prozess A die gewünschten Informationen hineingeschrieben hat, so ist nicht zu erwarten, dass Prozess B das erwartete Ergebnis berechnen kann.

In dieser Situation kann entweder der Mensch eingreifen, indem er Prozess B erst dann startet, wenn Prozess A fertig ist (in Shell-Skripten oder Batch-Dateien kann das sehr einfach über die Reihenfolge der Befehle erfolgen), oder das Betriebssystem stellt einen sogenannten Lock-Mechanismus für den Dateizugriff zur Verfügung.

Aufgabe 1

Aufgabe

Aufg. 151: Lock

Recherchiere: Was versteht man unter einem Lock-Mechanismus im Hinblick auf den schreibenden oder lesenden Zugriff mehrerer Prozesse auf eine Datei?

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.2.13.3 Zwei Prozesse kommunizieren über Shared Memory

Man spricht von Shared Memory (auf Deutsch: gemeinsamer Speicher), wenn ein Teil des Hauptspeichers gemeinsam mehreren Prozessen zur Verfügung gestellt wird. </p>

Aufgabe 1

Aufgabe

Aufg. 152: Shared Memory realisieren

Überlege, recherchiere und diskutiere in deiner Lerngruppe: Wie kann Shared Memory sehr einfach mehreren Prozessen zur Verfügung gestellt werden?

Tipp

Man spricht hier vom *Einblenden* des Shared Memory in mehrere virtuelle Adressräume.

Schau dir das Video Grundlagen virtueller Speicherverwaltung mit MMU noch einmal an und überlege, wie hier der Zusammenhang zwischen einem physikalischen Seitenrahmen (Frame), und dem Einblenden als virtuelle Seite in **mehreren virtuellen Adressräumen** wohl sein wird.

Ein später folgendes Kapitel beschäftigt sich noch einmal detaillierter mit Shared Memory.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.13.4 Zwei Prozesse kommunizieren über Pipes

Mandl 2013 beschreibt Pipes als Einweg-Kommunikationskanäle, die es einem Prozess ermöglichen, Daten bzw. Nachrichten über das Betriebssystem als Datenstrom an einen anderen Prozess zu übertragen. </p>

Beispiel

Ein ganz einfaches Beispiel für eine Pipe liefert der folgende Befehl, der beispielsweise auf einer Linux-Shell eingegeben werden kann:

ls -l | more

Genaugenommen handelt es sich hier um die beiden Programme ls und more.

Das erste Programm wird mit dem Parameter -l aufgerufen und insgesamt ergibt ls -l einen Prozess, der den Inhalt des aktuellen Verzeichnisses in einer ausführlichen Fassung zurückliefert.

Der Prozess des zweiten Programms nimmt eine beliebige Menge an Informationen entgegen und gibt sie seitenweise zurück. Nach jeder Seite wird zunächst ein Tastendruck abgewartet, anschließend wird die nächste Seite geliefert.

Das Zeichen | ist das sogenannte Pipe-Zeichen, es verbindet die Ausgabe des ersten Prozesses mit der Eingabe des zweiten.

Durch den Befehl ls -l | more wird also der Inhalt des aktuellen Verzeichnisses seitenweise auf dem Bildschirm ausgegeben.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.13.5 Zwei Prozesse kommunizieren über Sockets

Sockets sind eine vom Betriebssystem bereitgestellte Kommunikationsmöglichkeit, die üblicherweise auf den Internet-Standardprotokollen TCP (Transmission Control Protocol) und/oder UDP (User Datagram Protocol) basiert. Damit können Informationen mit Hilfe des Internets rund um die Welt zwischen verschiedenen Prozessen auf völlig verschiedenen Rechnern ausgetauscht werden. (Es sei an dieser Stelle erwähnt, dass die Kommunikation aber auch per TCP bzw. UDP zwischen zwei Prozessen auf demselben Rechner erfolgen kann.) </p>

Beispiel

Anschauliche Beispiele zur Socket-Programmierung mit Java liefert Dietmar Abts in seinem Buch Masterkurs Client/Server-Programmierung mit Java .

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Aufgabe 1

Aufgabe

Aufg. 153: Deine tägliche Socket-Dosis

Erläutere ein kleines Beispiel aus deiner täglichen Praxis im Umgang mit dem Computer für eine Kommunikation über Sockets. Welches Programm steckt hinter einem sicher täglich von dir auf deinem Rechner genutzen Prozess, der über Sockets Informationen mit einem anderen Prozess irgendwo auf einem anderen Rechner im weltweiten Internet austauscht? Welches Programm steckt hinter diesem *anderen Prozess*?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.2.13.6 Interprozesskommunikation-Fazit

Die in diesem Kapitel gezeigten Beispiele geben einen Überblick über einige Möglichkeiten der Interprozesskommunikation. </p>
Ein Betriebssystem-Programmierer muss also diese (und eventuell noch weitere) Mechanismen implementieren, damit sein Betriebssystem den Prozessen und Threads entsprechende Kommunikationsmöglichkeiten bereitstellt.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.3 Speicherverwaltung

Die Verwaltung des Hauptspeichers ist eine sehr wichtige Aufgabe des Betriebssystems.

Definition: Speicherverwaltung

Definition

Die **Speicherverwaltung** oder **Hauptspeicherverwaltung** ist ein Teil des Betriebssystems und erledigt alle erforderlichen Arbeiten zur Verwaltung des physikalischen und des virtuellen Speichers eines Computers.

Physikalischer Speicher (RAM) ist in jedem Rechner eingebaut. In diesem Video sind (u.a.) die Speichermodule zu sehen, und in dieser Abbildung erkennt man die RAM-Steckplätze, welche die Verbindung zwischen Speichermodul und Bussystem herstellen.

Ob bei der Arbeit mit einem Computersystem auch virtueller Speicher zum Einsatz kommt, hängt vom verwendeten Betriebssystem ab. Die Speicherverwaltung des Betriebssystems muss die virtuelle Speicherverwaltung implementieren. Andernfalls kommt kein virtueller Speicher zum Einsatz.

Bedenke!

Hinweis

Virtueller Speicher existiert nur in der Vorstellung! Alle Daten müssen entweder im physikalischen Speicher abgelegt werden, oder auf einen Hintergrundspeicher (wie z.B. die Festplatte) ausgelagert sein.
Aber die Implementierung einer virtuellen Speicherverwaltung vereinfacht und flexibilisiert Vieles.

Erinnert sei an dieser Stelle an das Basis- und das Limitregister, welche auf alten CPUs zu finden waren. Hiermit wurde lediglich physikalischer Speicher verwaltet.

Als dann später Basis- und Limitregister durch die Memory Management Unit (MMU) ersetzt wurden, kam die virtuelle Speicherverwaltung ins Spiel.

In beiden Fällen musste das jeweils eingesetzte Betriebssystem mit der erwähnten Hardware (Register oder MMU) zusammenarbeiten.

Hinweis

Betrachtet wird im Rahmen dieses Lernmoduls lediglich die virtuelle Speicherverwaltung mit Hilfe der MMU. Dieses Konzept ist in heutigen Betriebssystemen üblicherweise vorzufinden.

Auf eine Speicherverwaltung ohne MMU (und damit ohne virtuellen Speicher) geht *Tanenbaum 2009* ausführlich ein. Bei Bedarf kann dort nachgelesen werden. Auch *Strelen 2012* widmet diesem Thema einige Kapitel: 6.2 und 6.3.

So geht es weiter:

Gliederung

3.3 Speicherverwaltung
3.3.1 Virtuelle Speicherverwaltung
3.3.2 Swapping und Paging
3.3.3 Shared Memory
3.3.4 Speicherverwaltung moderner Betriebssysteme

Alternative Webquelle zum Thema

Quelle im Web

Operating Systems: Main Memory

http://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/8_MainMemory.html

Dr. John T. Bell
Department of Computer Science
University of Illinois, Chicago

Quelle im Web

Operating Systems: Virtual Memory

http://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/9_VirtualMemory.html

Dr. John T. Bell
Department of Computer Science
University of Illinois, Chicago

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.3.1 Virtuelle Speicherverwaltung

Die Grundlagen der virtuellen Speicherverwaltung mit MMU waren bereits in dem folgenden Video erläutert worden. Es schadet an dieser Stelle nicht, sich die erklärten Inhalte noch einmal in Erinnerung zu rufen. </p>

Grundlagen

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/PpyWObQw70o

Med. 32: Grundlagen virtueller Speicherverwaltung mit MMU (04:47)
http://youtu.be/PpyWObQw70o
CC-BY

Mit den Erläuterungen des Videos ergibt sich eine grundlegende Vorstellung davon, was die virtuelle Speicherverwaltung ausmacht. Im Folgenden geht es um einen tieferen Einblick in die Thematik.

Grundgedanken der virtuellen Speicherverwaltung

Mandl 2013 fasst die Grundgedanken der virtuellen Speicherverwaltung treffend in drei Punkten zusammen:

- Ein Prozess sollte auch dann noch ablaufen können, wenn er nur teilweise im Hauptspeicher ist. Wichtig ist hierbei, dass die Teile des Prozesses (Daten und Code) im physikalischen Speicher sind, die gerade benötigt werden.
- Der Speicherbedarf eines Prozesses sollte größer als der physikalisch vorhandene Hauptspeicher sein können.
- Ein Programmierer sollte am besten nur einen kontinuierlichen (linearen) Speicherbereich, beginnend bei Adresse 0 sehen und sich nicht um die Zerstückelung (Fragmentierung) des Hauptspeichers auf mehrere Benutzer kümmern müssen.

Grundbegriffe

Die virtuelle Speicherverwaltung definiert eine Reihe von Grundbegriffen, welche in den folgenden Erläuterungen immer wieder verwendet werden.

Bereits bekannt sind die Fachbegriffe:

- Physikalischer Speicher
- Physikalische Speicheradresse
- Virtueller Speicher
- Virtuelle Speicheradresse

Gegebenenfalls sollten die Definitionen noch einmal nachgelesen werden.

Zusätzlich werden weitere Fachbegriffe benötigt, die teilweise bereits in dem obigen Video benutzt wurden.

Definition: Seitenrahmen

Definition

Unter einem **Seitenrahmen** (englisch: **Pageframe**, oder kurz: **Frame** bzw. **Rahmen**) versteht man einen zusammenhängenden Block von Speicherzellen des *physikalischen* Speichers.

Typische Größen für einen Seitenrahmen in der Praxis sind 1, 4, 8, 16 oder 64 KiB. Alle Seitenrahmen eines Systems haben stets die gleiche Größe.

Definition: Seite

Definition

Unter einer (virtuellen) **Seite** (englisch: **Page**) versteht man einen zusammenhängenden Block von Speicherzellen des *virtuellen* Speichers. Die Blockgröße einer Seite entspricht immer exakt der Größe eines Seitenrahmens.

Alle Seiten eines Systems haben stets die gleiche Größe.

Aufgabe 1

Aufgabe

Aufg. 154: Rahmen vs. Seite

Was ist der Unterschied zwischen einem *Seitenrahmen* und einer *Seite*?

Anhand des Videos kann nachvollzogen werden, warum die Größe eines *Seitenrahmens* und einer *Seite* stets identisch sein muss. Erläutere warum!

Die folgende Aufgabe ist besonders wichtig, um alle weiteren Details der virtuellen Speicherverwaltung verstehen zu können.

Aufgabe 2

Aufgabe

Aufg. 155: Rechne nach!

Ein Rechnersystem verfügt über 1 GiB physikalischen Speicher. Jedem gestarteten Prozess wird vom Betriebssystem ein virtueller Speicher von 4 GiB zugewiesen. Die Größe eines Seitenrahmens beträgt 64 KiB.

a) Wie viele Seitenrahmen gibt es insgesamt?

b) Wie viele Seiten gibt es insgesamt (pro Prozess)?

Angenommen, ein einzelner Maschinensprachebefehl hat eine Größe von 32 Bit:

c) Wie viele Befehle passen in einen Seitenrahmen?

d) Und wie viele Befehle passen demnach in eine Seite?

Innerhalb eines Seitenrahmens kann jede einzelne Speicherzelle (= 8 Bit) adressiert werden. Demnach beginnt der erste Befehl bei Adresse 0 innerhalb des Rahmens, der zweite Befehl bei Adresse 4, der dritte Befehl bei Adresse 8, usw.

e) Wie viele Adressen existieren pro Seitenrahmen insgesamt?

f) Und wie viele Adressen existieren demnach pro Seite?

g) Wie viele Bit werden benötigt, um diese Adressen angeben zu können?

Merke dir die gerade errechnete Bitanzahl, du wirst sie gleich wieder brauchen!

Oben hattest du bereits die Anzahl der Rahmen und die Anzahl der Seiten berechnet:

h) Wie viele Bit werden benötigt, um die Anzahl der Seitenrahmen damit codieren zu können?

i) Und wie viele Bit benötigt man für die Codierung der Seitenanzahl?

Jetzt kannst du den Schluss ziehen:

j) Aus wie vielen Bit besteht eine physikalische Adresse des betrachteten Rechnersystems?

k) Und aus wie vielen Bit besteht eine virtuelle Adresse?

Tipp

Die Bit-Länge einer *physikalischen* Adresse ist die Anzahl der Bits für die Gesamtzahl an Seitenrahmen addiert mit der Anzahl der Bits für die Gesamtzahl an Adressen innerhalb eines Rahmens.

Die Bit-Länge einer *virtuellen* Adresse ist die Anzahl der Bits für die Gesamtzahl an Seiten addiert mit der Anzahl der Bits für die Gesamtzahl an Adressen innerhalb einer Seite.

Das Endergebnis der vorangegangenen Aufgabe zeigt, dass eine physikalische Adresse eine andere Bit-Länge als eine virtuelle Adresse besitzt. Zusammen mit dem Video wird nun klar, dass eine Umrechnung von virtuellen Adressen in physikalische Adressen erfolgen muss.

Genau dies ist die Aufgabe der Memory Management Unit (MMU).

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.3.1.1 Arbeitsweise der MMU

Die Aufgabe der MMU besteht in der Umrechnung von virtuellen in physikalische Adressen. Die Grundlagen dazu sind bereits bekannt. </p>
Da die MMU als Hardware-Bauteil (i.d.R. ist die MMU Bestandteil der CPU) auf diese Umrechnung hin optimiert wurde, kann sie diese Aufgabe extrem schnell durchführen.

Aufgabe 1

Aufg. 156: Besonders schnell ist besonders wichtig!

Erläutere: Warum ist es **besonders wichtig**, dass die MMU die Umrechnung von virtuellen in physikalische Adressen **besonders schnell** erledigen kann?

Diskutiere diese Frage in deiner Lerngruppe!

Damit die MMU Ihre Aufgabe erfüllen kann, muss sie Zugriff auf sogenannte Seitentabellen besitzen.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.3.1.2 Seitentabellen

Seitentabellen werden benötigt, damit die MMU die Umrechnung einer virtuellen Adresse in eine physikalische Adresse vornehmen kann. Da der Begriff *Seitentabellen* hier bereits in der Mehrzahl benutzt wird, deutet dies darauf hin, dass es nicht nur eine einzige, sondern mehrere Seitentabellen gibt. </p>

Viele Seitentabellen existieren gleichzeitig

Für jeden einzelnen vom Betriebssystem zu verwaltenden virtuellen Adressraum gibt es jeweils eine zugehörige Seitentabelle. Anders ausgedrückt besitzt jeder Prozess seine eigene Seitentabelle, da ja auch jeder Prozess seinen eigenen virtuellen Adressraum besitzt.

Jeder Prozess besitzt seine eigene Seitentabelle!

Je nach Betriebssystem kommen entweder **einstufige Seitentabellen** oder **mehrstufige Seitentabellen** zum Einsatz.

So geht es weiter:

3.3.1.2 Seitentabellen
3.3.1.2.1 Einstufige Seitentabellen
3.3.1.2.2 Mehrstufige Seitentabellen

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.3.1.2.1 Einstufige Seitentabellen

Damit die Hintergründe zu einstufigen Seitentabellen verstanden werden können, sind zunächst einige Umrechnungen nötig. Dies geschieht in der folgenden Aufgabe: </p>

Aufgabe 1

Aufg. 157: Wenn du es umrechnen kannst, dann kannst du es auch verstehen!

Ein Computersystem arbeitet mit der virtuellen Speicherverwaltung. Es gelten folgende Voraussetzungen:

- Physischer Adressraum:
  Der (in den Rechner eingebaute) physische Speicher habe eine Größe von 512 MiB.
- Virtueller Adressraum:
  Der (jedem Prozesess zugeordnete) virtuelle Speicher habe eine Größe von 4 GiB.
- Ein Seitenrahmen (Pageframe) habe eine Größe von 64 KiB.

- Eine (virtuelle) Seite hat per Definition die gleiche Größe wie ein Seitenrahmen, also auch 64 KiB.

Berechne:

- Aus wievielen Seitenrahmen besteht der physische Adressraum?
  Gib die Antwort sowohl als Dezimalzahl, als auch als 2er-Potenz an!
- Aus wievielen Seiten besteht ein virtueller Adressraum?
  Gib die Antwort sowohl als Dezimalzahl, als auch als 2er-Potenz an!
- Wieviele Byte ergeben 64 KiB umgerechnet?
  Gib die Antwort sowohl als Dezimalzahl, als auch als 2er-Potenz an!

Die bei den Umrechnungen ermittelten Werte werden in dem folgenden Video wieder aufgegriffen. Es erklärt die Hintergründe einer einstufigen Seitentabelle und wie die MMU diese nutzt.

Adressumrechnung mit einstufiger Seitentabelle

An dieser Stelle befindet sich online ein YouTube-Video.
https://youtu.be/aSSmsETQARw

Med. 33: MMU: Adressumrechnung mit einstufiger Seitentabelle (06:32)
http://youtu.be/aSSmsETQARw
CC-BY

Das im Video erläuterte Beispiel geht davon aus, dass in den betrachteten Rechner ein physischer Speicher (RAM) von 512 MiB eingebaut ist. Jedem Prozess wird vom Betriebssystem ein virtueller Speicher von 4 GiB zugewiesen. Aus diesen Voraussetzungen ergeben sich die Adresslängen:

- Länge einer virtuelle Adresse: 32 Bit
- Länge einer physische Adresse: 29 Bit

Wichtig

Basiert das betrachtete Rechnersystem auf anderen (physischen und virtuellen) Speichergrößen, so ändern sich auch die Adresslängen!
(Siehe Aufgabe 3 unten.)

Beispiel einer Seitentabelle

In der folgenden Abbildung sieht man die im Video gezeigte Seitentabelle. Sie besitzt insgesamt 65.536 Zeilen.

Seitenrahmen-Nr. (Frame-No.)	Present-/ Absent-Bit
0 1111 0000 0011	1
0 1100 0111 0101	0
1 0101 0011 1110	0
0 1001 1111 1001	1
.	.
.	.
.	.
0 0000 0000 0000	0
1 1111 1111 1111	0
0 0000 1111 1111	0
1 0000 0111 1000	1
1 1001 1001 0000	1
0 0000 0000 0000	0
0 0000 1001 1111	1

Abb. 59: Die im Video gezeigte Seitentabelle
CC-BY

Seitentabelleneintrag

Jede Zeile der Seitentabelle nennt man einen sogenannten **Seitentabelleneintrag**. Nach dem bisherigen Kenntnisstand besteht ein Seitentabelleneintrag also aus einer Seitenrahmen-Nummer und dem Present-/Absent-Bit.

Hinweis

In der Praxis besteht ein Seitentabelleneintrag üblicherweise noch aus einigen weiteren Informationen. Bildlich gesprochen enthält die Seitentabelle also noch weitere Spalten. Wir kommen später darauf zurück.

Aufgabe 2

Aufgabe

Aufg. 158: Eine Seitentabelle hat es in sich

Betrachte die Seitentabelle aus der vorangegangenen Abbildung.

- Warum besitzt die Tabelle genau 65.536 Zeilen?
- Unter welchen Umständen ist es erlaubt, dass eine Seitenrahmen-Nr. mehrfach in der Tabelle vorkommt? (Siehe *0 0000 0000 0000*, es wären aber auch andere doppelte Nummern denkbar.)
- In der Spalte mit dem Present-/Absent-Bit: Wieviele Einsen könnten in dieser Spalte maximal auftreten? Warum nicht mehr?

- Wie realistisch ist es in der Praxis, dass in der Spalte mit dem Present-/Absent-Bit tatsächlich einmal die Maximalzahl an Einsen auftritt? Erläutere!

Aufgabe 3

Aufg. 159: Nur 64 KiB RAM

Ein Rechnersystem besitzt einen physischen Speicher (RAM) von 64 KiB. Das Betriebssystem weist jedem Prozess einen virtuellen Speicher von 128 KiB zu.

- Welche Länge hat eine physische Adresse?
- Welche Länge hat eine virtuelle Adresse?
- Wie ist die einstufige Seitentabelle in diesem Fall aufgebaut?
- Wie viele Zeilen besitzt die Seitentabelle?
- Erkläre an einem Beispiel, wie die Umrechnung einer virtuellen in eine physische Adresse funktioniert.
- Gehe in deinem Beispiel davon aus, dass das zehnte Byte innerhalb der betreffenden virtuellen Seite adressiert wird.

Hinweis:
Wenn du schon ein gewisses Verständnis für dieses Thema entwickelt hast, dann hast du bereits bemerkt, dass innerhalb dieser Aufgabe eine wichtige Vorraussetzung nicht genannt wurde. In dem Video ist sie hingegen genannt!

Denke erst nach, schaue dir das Video nochmals an, und diskutiere in deiner Lerngruppe. Dann brauchst du den folgenden Tipp sicher nicht.

Tipp

HA! Du brauchst den Tipp ja doch!

Du kannst die Aufgabe nur dann vollständig lösen, wenn du die Größe eines Seitenrahmens festlegst. In dem Video wurde das getan, aber der dort verwendete Wert ist für diese Aufgabe viel zu groß. Wähle lieber einen sinnvollen kleineren Wert.

Bedenke auch, dass die Größe eines Seitenrahmens identisch mit der Größe einer virtuellen Seite sein muss.

Mandl 2013 zeigt in Kap. 7.2.1, Abb 7-14, ein Beispiel für eine Adressumsetzung mit Hilfe einer einstufigen Seitentabelle. Die folgende Aufgabe fordert Ähnliches:

Aufgabe 4

Aufgabe

Aufg. 160: Du bist jetzt die MMU

Betrachte die einstufige Seitentabelle oben.

Rechne mit Hilfe der Seitentabelle die folgenden virtuellen Adressen in physische Adressen um:

- 0000 0000 0000 0010 1111 0000 0101 0000
- 0000 0000 0000 0011 1000 0110 0111 0111
- 0000 0000 0000 0101 1100 0100 1110 1111

Hinweis:
Gehe davon aus, dass die Zeilen in der Seitentabelle **von unten nach oben** durchnummeriert sind:

- Die Zeile ganz **unten** ist die *Zeile Nr. 0*.
- Die Zeile ganz **oben** ist die *Zeile Nr. 65.535*.

Wie bereits erwähnt können alternativ zu *einstufigen* Seitentabellen auch *mehrstufige* Seitentabellen zum Einsatz kommen.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.3.1.2.2 Mehrstufige Seitentabellen

Mehrstufige Seitentabellen sind zu Optimierungszwecken entwickelt worden. An dieser Stelle wird jedoch nicht näher darauf eingegangen, es sei auf die weiterführende Literatur verwiesen. </p>

Hinweis

Weiterführende Literatur

Mandl 2013 erläutert in Kapitel 7.2.1 den Aufbau und die Funktionsweise von mehrstufigen Seitentabellen. Die Lektüre dieser Quelle sei dem geneigten Leser überlassen.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.3.2 Swapping und Paging

Der Begriff *Swapping* ist bereits aus einem vorangegangenen Kapitel bekannt. Seine Definition sei hier kurz wiederholt: </p>

Definition: Swapping

Definition

Unter **Swapping** versteht man das Aus- bzw. Einlagern eines **kompletten Prozesses**.

Es soll an dieser Stelle klar werden, dass Swapping rein gar nichts mit der virtuellen Speicherverwaltung zu tun hat.

Hinweis

Swapping kommt nur bei Betriebssystemen zum Einsatz, die **keine virtuelle Speicherverwaltung** unterstützen!

Kommt in einem Betriebssystem die virtuelle Speicherverwaltung mit Hilfe der MMU zum Einsatz, so gibt es eine alternative Technik, das sogenannte **Paging**, welches wesentlich flexibler als Swapping agiert.

Definition: Paging

Definition

Unter **Paging** versteht man das Ein- bzw. Auslagern von **Teilen eines Prozesses**.

Aus den vorangegangenen Kapiteln ist ja bereits bekannt, dass man es bei der virtuellen Speicherverwaltung mit (virtuellen) Seiten bzw. (physikalischen) Seitenrahmen zu tun hat.

Eine Seite enthält einen bestimmten Teil eines Prozesses. Die Gesamtheit aller Seiten eines Prozesses repräsentiert somit den kompletten Prozess.

Eine (virtuelle) Seite kann einerseits in einem (physikalischen) Seitenrahmen eingelagert sein. Dann befindet sich die Seite im RAM und der zugehörige Prozess kann auf die Befehle und/oder Daten innerhalb der Seite zugreifen. Zum Einsatz kommt hier die Adressumrechnung mit Hilfe von MMU und Seitentabelle.

Andererseits kann eine (virtuelle) Seite auf einen Hintergrundspeicher (wie beispielsweise die Festplatte) ausgelagert sein. In diesem Fall steht die Seite nicht im RAM zur Verfügung, ein Zugriff des zugehörigen Prozesses auf Befehle/Daten schlägt fehl, es kommt zu einem sogenannten **Page fault** (**Seitenfehler**).

Die ausgelagerte Seite muss nun zunächst vom Hintergrundspeicher in einen Seitenrahmen eingelagert werden. Anschließend kann die Adressumrechnung mit Hilfe von MMU und Seitentabelle erfolgreich durchgeführt werden.

Wenn in der Definition des Begriffs Paging von *Teilen eines Prozesses* die Rede ist, dann ist klar, dass:

Hinweis

Beim Paging werden virtuelle Seiten in (physikalische) Seitenrahmen eingelagert, oder aus Seitenrahmen in den Hintergrundspeicher ausgelagert.

Bei der Durchführung des Paging können die bereits erwähnten **Seitenfehler** (**Page faults**) auftreten. Weiterhin wird ggf. ein sogenanntes **Seitenersetzungsverfahren** angewendet. Beides wird in folgenden Kapiteln näher betrachtet.

Zunächst noch einige Aufgaben:

Aufgabe 1

Aufgabe

Aufg. 161: Swapping vs. Paging I

Erläutere die Unterschiede zwischen Swapping und Paging!

- Wann kommt welche Technik zum Einsatz?
- Wie funktioniert die jeweilige Technik?
- Was hat der DMA-Controller mit Swapping bzw. Paging zu tun?
- Was hat Swapping mit einer Seitentabelle zu tun?

Aufgabe 2

Aufgabe

Aufg. 162: Swapping vs. Paging II

Ergänze die folgenden Sätze:

- Ein Prozess, der durch Swapping ausgelagert ist, kann nicht auf der CPU ausgeführt werden, weil...
- Ein Prozess, der durch Paging teilweise ein- und teilweise ausgelagert ist, kann auf der CPU solange laufen, wie nur auf ... zugegriffen wird.

Aufgabe 3

Aufgabe

Aufg. 163: Paging und die Seitentabelle

Eine (virtuelle) Seite befinde sich derzeit in einem Seitenrahmen eingelagert. Das Betriebssystem möchte diese Seite nun in den Hintergrundspeicher auslagern.

- Wie sieht der Seitentabelleneintrag für die betreffende Seite aus, solange die Seite noch eingelagert ist?
- Welche Änderung muss das Betriebssystem an dem betreffenden Seitentabelleneintrag vornehmen, sobald die Seite ausgelagert wurde?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.3.2.1 Page Fault

Ein **Page fault**, oder zu deutsch **Seitenfehler**, tritt auf, falls die MMU bei der Umrechnung einer virtuellen in eine physische Adresse feststellt, dass die benötigte (virtuelle) Seite nicht in einem (physischen) Seitenrahmen eingelagert ist. </p>

Benötigte Seite ist nicht eingelagert

Zu sehen war diese Situation bereits in dem folgenden, schon bekannten Video, ab Minute 05:30:

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/aSSmsETQARw

Med. 34: MMU: Page fault bei der Adressumrechnung (06:32 Gesamtlänge, Page fault ab 05:30)
http://youtu.be/aSSmsETQARw?t=5m30s
CC-BY

Aufgabe 1

Aufgabe

Aufg. 164: Page Fault or No Page Fault?

Erläutere:

Wie oder woran stellt die MMU fest, ob sie einen Page fault auslösen muss, oder nicht?

Am Ende des Videos wird angedeutet, dass ein Page fault einem Interrupt entspricht. In der Folge muss das Betriebssystem dafür sorgen, dass die fehlende virtuelle Seite in einen freien Seitenrahmen des physischen Hauptspeichers eingelagert wird. Anschließend kann die MMU erneut die Adressumrechnung durchführen.

Die folgende Aufgabe gibt Anlass zu einer detaillierteren Beschäftigung mit einem Page fault:

Aufgabe 2

Aufg. 165: Was passiert bei einem Page fault?

Ein Page fault wird ausgelöst , dies wird durch einen Interrupt angezeigt. Was passiert mit diesem Interrupt-Signal?

Oder anders gefragt:
Wie müsste das Video fortgesetzt werden?

Aufgrund deiner Erfahrungen aus den vorangegangenen Kapiteln kannst du eine Prognose zur Fortsetzung des Videos wagen.

- Notiere stichpunktartig den weiteren Verlauf!
- Nutze Fachbegriffe zur Beschreibung!
- Gehe davon aus, dass mehrere Prozesse auf dem Rechner gestartet sind und vom Betriebssystem verwaltet werden.
- Beginne mit dem Moment, in dem die MMU den Page fault auslöst.
- Gehe davon aus, dass ein freier Seitenrahmen zur Verfügung steht.
- Beschreibe bis zu dem Moment, in dem die Adressumrechnung durch die MMU erneut versucht wird und schließlich klappt.

Diskutiere deine Prognose in deiner Lerngruppe! Vervollständigt gemeinsam die Beschreibung und eweitert die Liste der Fachbegriffe.

Erst daran anschließend könnt ihr anhand der folgenden Liste prüfen, ob euch noch Fachbegriffe fehlen, oder ob ihr sogar noch darüber hinaus gehende Begriffe identifiziert habt:

Liste von Fachbegriffen für diese Aufgabe

Folgende Fachbegriffe sollten (mindestens) in deiner Erläuterung verwendet werden. Einige davon auch mehrfach in unterschiedlichem Zusammenhang.

Die Begriffe stehen hier in alphabetischer Ordnung. Achte darauf, dass du die sinnvolle zeitliche Abfolge beschreibst!

Falls dir einige der folgenden Fachbegriffe nicht geläufig sind: Lies ihre jeweilige Definition nach!

- Adressumrechnung
- bereit (Prozesszustand)
- Betriebssystem
- blockiert (Prozesszustand)
- Computer bzw. Rechner
- Dispatcher
- DMA (Direct Memory Access) bzw. DMA-Controller
- Hintergrundspeicher
- Interrupt
- Interruptbehandlungsroutine
- Interrupt-Controller
- Kontextwechsel
- MMU (Memory Management Unit)
- Page fault bzw. Seitenfehler
- physische Adresse
- physischer Speicher (RAM)
- Present/Absent-Bit
- Prozess (Prozess A, Prozess B, ...)
- Prozesskontrollblock
- Prozesstabelle
- Quantum
- rechnend (Prozesszustand)
- Register
- Scheduler
- Seite (virtuell)
- Seitenrahmen (physisch)
- Seitenrahmen-Nr.
- Seitentabelle
- virtuelle Adresse
- virtuelle Speicherverwaltung
- virtueller Speicher

Das Ergebnis der vorangegangenen Aufgabe zeigt den detaillierten Ablauf bei einem **Page fault**. Dabei wird vereinfachend davon ausgegangen, dass im physischen Speicher ein freier Seitenrahmen zur Verfügung steht, in den die virtuelle Seite aus dem Hintergrundspeicher eingelagert werden kann.

In der Praxis ist aber davon auszugehen, dass nicht immer ein freier Seitenrahmen verfügbar ist. In diesem Fall sind also alle Seitenrahmen bereits belegt und bevor die benötigte Seite eingelagert werden kann, muss zunächst (mindestens) ein Seitenrahmen frei gemacht werden.

Anders ausgedrückt geht es darum, eine derzeit in einem Seitenrahmen eingelagerte virtuelle Seite durch die benötigte virtuelle Seite zu ersetzen. Hierzu gibt es verschiedene sogenannte Seitenersetzungsverfahren.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.3.2.2 Seitenersetzung

Tritt bei der Adressumrechnung in der MMU ein Seitenfehler (Page fault) auf, so muss das Betriebssystem dafür sorgen, dass die benötigte virtuelle Seite aus dem Hintergrundspeicher in einen freien Seitenrahmen des physikalischen Speichers eingelagert wird. </p>
Entscheidend ist bei der Einlagerung, ob es einen freien Seitenrahmen gibt, oder nicht.

Falls ja, so kann dieser für die Einlagerung der benötigten Seite genutzt werden und es gibt keine Probleme.

Steht für die Einlagerung jedoch kein freier Seitenrahmen zur Verfügung, so ist es die Aufgabe der **Seitenersetzung** zu entscheiden, welche momentan eingelagerte virtuelle Seite in den Hintergrundspeicher verschoben wird. Dadurch wird dann (mindestens) ein freier Seitenrahmen geschaffen, die benötigte virtuelle Seite kann eingelagert werden, und die MMU kann die zuvor gescheiterte Adressumrechnung erneut durchführen.

Statt des im vorstehenden Absatz erwähnten Begriffs *verschoben* findet oftmals auch das Verb *verdrängt* in gängiger Literatur seine Anwendung. Man spricht hier dann auch von einer **Verdrängungsstrategie** und meint damit wieder die Seitenersetzung.

Hinweis

Für das Verständnis der folgenden Erläuterungen ist es wichtig, sich über die hier verwendeten Begriffe im Klaren zu sein. Es wird unterschieden zwischen:

- (virtuelle) Seite
- Seitenrahmen
- eingelagerte Seite
- benötigte Seite
- zu ersetzende Seite
- modifizierte Seite

Falls erforderlich gibt es hier noch einige

Hinweise

- (virtuelle) Seite:
  Eine beliebige virtuelle Seite.
- Seitenrahmen:
  Ein beliebiger Seitenrahmen des physikalischen Speichers.
- eingelagerte Seite:
  Eine virtuelle Seite, deren Inhalt derzeit in einem Seitenrahmen des physikalischen Speichers vorgehalten wird.
- ausgelagerte Seite:
  Eine virtuelle Seite, deren Inhalt derzeit **nicht** in einem Seitenrahmen des physikalischen Speichers vorgehalten wird.
- benötigte Seite:
  Eine virtuelle Seite, die aktuell **nicht** eingelagert ist. Sie ist der Grund für den aufgetretenen Page fault (Seitenfehler).
- zu ersetzende Seite:
  Eine virtuelle Seite, die derzeit eingelagert ist, und deren zugehöriger Seitenrahmen durch die Seitenersetzung für eine andere (derzeit ausgelagerte) virtuelle Seite vorgesehen ist.
- modifizierte Seite:
  Eine virtuelle Seite, die derzeit eingelagert ist, und deren Inhalt durch die jüngst von der CPU verarbeiteten Befehle verändert wurde.

So geht es weiter:

Gliederung

3.3.2.2 Seitenersetzung
3.3.2.2.1 Was bei der Seitenersetzung passiert

3.3.2.2.2 Das Modifiziert-Bit
3.3.2.2.3 Seitenersetzungsverfahren

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.3.2.2.1 Was bei der Seitenersetzung passiert

Eine einfache Beschreibung des Ablaufs einer Seitenersetzung ist wie folgt: </p>

1. Die MMU stellt fest, dass die benötigte virtuelle Seite B nicht in einem Seitenrahmen eingelagert ist und löst deshalb einen Seitenfehler aus.

2. Es wird festgestellt, dass kein freier Seitenrahmen mehr verfügbar ist, das Seitenersetzungsverfahren wird deshalb gestartet.

3. Die zu ersetzende Seite E wird bestimmt, sie ist derzeit in Seitenrahmen X eingelagert.

4. Die zu ersetzende Seite E wird in den Hintergrundspeicher geschrieben. Damit ist ihr Inhalt gesichert, sie kann später wieder eingelagert werden.

5. Die benötigte Seite B wird eingelagert, d.h. ihr Inhalt wird in den Seitenrahmen X geschrieben. Das Ersetzungsverfahren ist damit abgeschlossen.

Wichtig

Das Schreiben der zu ersetzenden Seite E in den Hintergrundspeicher kostet viel Zeit! Das ist schlecht für die Performance des Gesamtsystems.

Es ist deshalb sinnvoll, eine zu ersetzende Seite nur dann in den Hintergrundspeicher zu kopieren, wenn dies auch tatsächlich notwendig ist.

Programmtext oder Daten

Eine Seite kann beispielsweise den Programmtext oder die Daten eines Prozesses enthalten.

Aufgabe 1

Aufgabe

Aufg. 166: Wenn eine Seite Programmtext enthält

Eine zu ersetzende Seite enthält lediglich Programmtext (also den ausführbaren Maschinencode mit den Befehlen) eines Prozesses. Diese Inhalte dürfen während der Ausführung des zugehörigen Prozesses nicht verändert werden.

Ist es erforderlich, diese Seite in den Hintergrundspeicher auszulagern, oder kann die dafür benötigte Zeit eingespart werden? Erläutere deine Antwort!

Aufgabe 2

Aufgabe

Aufg. 167: Wenn eine Seite Daten enthält

Eine zu ersetzende Seite enthält Daten. Diese Daten können verändert worden sein, sie können aber auch noch unverändert sein.

Ist es erforderlich, diese Seite in den Hintergrundspeicher auszulagern, oder kann die dafür benötigte Zeit eingespart werden? Erläutere deine Antwort!

Betrachte folgende Situationen:

1. Die Inhalte der Seite **wurden verändert**, weil Werte in Speicherzellen des dieser Seite zugeordneten Seitenrahmens geschrieben wurden.

2. Die Inhalte der Seite **wurden nicht verändert**, zudem befindet sich eine Kopie der Seite im Hintergrundspeicher, weil sie zu einem früheren Zeitpunkt bereits einmal ausgelagert war.

Die Antwort zu der vorangegangenen Aufgabe sollte zeigen, dass die Zeit zur Auslagerung einer Seite eingespart werden kann, wenn

- die Seite bereits im Hintergrundspeicher abgelegt ist, und
- die Seite in ihrem Seitenrahmen nicht verändert (modifiziert) wurde.

Jetzt muss das Betriebssystem also nur noch in der Lage sein herauszufinden, ob eine in einem Seitenrahmen befindliche Seite modifiziert wurde, oder nicht. Damit beschäftigt sich das folgende Kapitel.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.3.2.2.2 Das Modifiziert-Bit

Es liegt auf der Hand, dass modifizierte Seiten in ihrem Seitenrahmen nicht einfach ersetzt werden dürfen, bevor sie nicht im Hintergrundspeicher gesichert wurden. Ihre Inhalte wären andernfalls verloren, was negative Auswirkungen auf den betreffenden Prozess nach sich ziehen dürfte. Insbesondere könnten sie zu einem späteren Zeitpunkt nicht wieder eingelagert werden. </p>

Wie kann das Betriebssystem aber feststellen, ob die Inhalte einer derzeit noch eingelagerten Seite modifiziert wurden, und so eine Auslagerung vor dem Ersetzen erforderlich wird?

Es gibt dafür eine ganz einfache Lösung: Das M-Bit.

Eine Spalte voller M-Bits

In den Seitentabellen wird eine zusätzliche Spalte eingeführt, siehe Abbildung unten. Für jede einzelne Seite (d.h. in jedem Seitentabelleneintrag) wird mit Hilfe eines einzelnen Bits festgehalten, ob die betreffende Seite verändert wurde.

Man spricht hier vom sogenannten **Modifiziert-Bit**, oder kurz vom **M-Bit** (in einigen Quellen ist auch vom **Dirty-Bit** die Rede).

- Das M-Bit ist gesetzt, also 1:
  Der Inhalt der zugehörigen Seite wurde modifiziert.
- Das M-Bit ist nicht gesetzt, also 0:
  Der Inhalt der zugehörigen Seite wurde nicht modifiziert.

Seitenrahmen-Nr.	Present-/ Absent-Bit	M-Bit
0 1111 0000 0011	1	0
0 1100 0111 0101	0	0
1 0101 0011 1110	0	0
0 1001 1111 1001	1	1
...	...	...

Abb. 60: Seitentabelle mit zusätzlicher Spalte für M-Bit
CC-BY

Aufgabe 1

Aufg. 168: M-Bit beim Einlagern

> Eine Seite wird in einen freien Seitenrahmen eingelagert. Die Speicherverwaltung des Betriebssystems aktualisiert dann den zugehörigen Seitentabelleneintrag.
>
> Welchen Wert schreibt die Speicherverwaltung in das Feld mit dem M-Bit dieses Seitentabelleneintrags?

Setzen des M-Bits

Die MMU nimmt bei der (erfolgreichen) Umrechnung einer virtuellen in eine physikalische Adresse ein Setzen des Bits im betreffenden Eintrag der Seitentabelle vor, sofern ein schreibender Zugriff auf die betreffende Speicherzelle erfolgen soll.

> Ist das M-Bit einer zu ersetzenden Seite gesetzt (also gleich 1), so muss der Inhalt dieser Seite aus dem Seitenrahmen in den Hintergrundspeicher geschrieben werden.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.3.2.2.3 Seitenersetzungsverfahren

In der vorangegangenen Auflistung findet sich u.a. der Punkt "*Die zu ersetzende Seite E wird bestimmt,...*", und es stellt sich die Frage:

> **Wie** wird die zu ersetzende Seite bestimmt?

Diese Aufgabe wird von einem sogenannten Seitenersetzungsalgorithmus erledigt.

Im Laufe der Entwicklungsgeschichte von Betriebssystemen wurden eine ganze Reihe verschiedener Seitenersetzungsverfahren (oder Verdrängungsstrategien) vorgeschlagen bzw. implementiert, u.a.:

- Optimaler Algorithmus
- NRU - Not Recently Used Algorithmus
- FIFO - First In First Out Algorithmus
- Second Chance Algorithmus
- Clock Page Algorithmus
- LRU - Least Recently Used Algorithmus
- NFU - Not Frequently Used Algorithmus

- Working Set Algorithmus

Hinweis

Weiterführende Literatur

Mandl 2013 erläutert die genannten Algorithmen in den Kapiteln 7.2.3 und 7.2.4. Die Lektüre dieser Quelle sei ausdrücklich empfohlen.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Einige der genannten Algorithmen werden auf den folgenden Seiten behandelt. Die ausführlichere Besprechung aller Verfahren leistet jedoch nur die weiterführende Literatur.

Hinweis

Beim Durcharbeiten der genannten Kapitel bei *Mandl 2013* kann man sehr schön die *Evolution der Algorithmen* verfolgen. D.h. ein späteres Verfahren beruht sehr oft auf einem vorangegangenen Verfahren, wobei jedoch ein vorheriger Nachteil verbessert wurde.
Achte mal darauf!

So geht es weiter:

Gliederung

3.3.2.2.3 Seitenersetzungsverfahren
3.3.2.2.3.1 Optimaler Seitenersetzungsalgorithmus
3.3.2.2.3.2 NRU - Not Recently Used Algorithmus
3.3.2.2.3.3 FIFO - First In First Out Algorithmus
3.3.2.2.3.4 Second Chance Algorithmus
3.3.2.2.3.5 Working Set Algorithmus

Aufgabe 1

Aufgabe

Aufg. 169: Die MMU und Seitenersetzungsverfahren

In diesem Kapitel wird beschrieben, dass ein Seitenersetzungsverfahren zum Einsatz kommt, wenn bei der Umrechnung einer virtuellen Adresse in eine physikalische Adresse auf der MMU ein Seitenfehler (page fault) ausgelöst wird.

Was ist jedoch im anderen Fall:
Eine virtuelle Adresse kann von der MMU erfolgreich in eine physikalische Adresse umgerechnet werden, es tritt also **kein** Seitenfehler auf.

Kommt dann auch ein Seitenersetzungsverfahren zum Einsatz?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.3.2.2.3.1 Optimaler Seitenersetzungsalgorithmus

Der **optimale Algorithmus** wählt unter allen derzeit in einem Seitenrahmen eingelagerten virtuellen Seiten diejenige Seite zur Ersetzung aus, welche **in Zukunft am längsten nicht mehr benötigt** wird. </p>
Der Grund für diese Strategie ist ganz einfach: Hierbei werden in Zukunft am wenigsten weitere Seitenfehler ausgelöst. Und da die Behebung jedes Seitenfehlers Zeit kostet, und so das Gesamtsystem für den Bediener verlangsamt, ergibt sich bei konsequenter Anwendung des optimalen Algorithmuses ein möglichst schnelles System.

Hinweis

Es wäre für die Performance des Systems sehr schlecht, wenn ein Seitenersetzungsalgorithmus genau denjenigen Seitenrahmen für die benötigte Seite frei macht, in dem sich bislang genau die Seite befand, die gleich beim nächsten auf der CPU verarbeiteten Befehl angesprochen werden wird.

Aufgabe 1

Aufgabe

Aufg. 170: Implementiere den optimalen Seitenersetzungsalgorithmus
Hast du eine Idee, wie man den optimalen Algorithmus implementieren könnte?
Seine Arbeitsweise ist doch mit Worten so einfach zu beschreiben!

Diskutiere deine Idee in deiner Lerngruppe.

Aufgabe 2

Aufgabe

Aufg. 171: Optimales M-Bit
Angenommen, ein Betriebssystem arbeitet mit dem optimalen Seitenersetzungsalgorithmus und einstufigen Seitentabellen.

- Wird in den Seitentabellen dann eine Spalte für das M-Bit enthalten sein?
- Wofür wird das M-Bit ggf. benötigt?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.3.2.2.3.2 NRU - Not Recently Used Algorithmus

Der **Not Recently Used Seitenersetzungsalgorithmus**, kurz **NRU**, ersetzt immer eine Seite, auf die *in letzter Zeit* nicht zugegriffen wurde. </p>
Die Idee dahinter ist: Wenn *in letzter Zeit* nicht auf die Seite zugegriffen wurde, dann wird vermutlich auch in (naher) Zukunft nicht darauf zugegriffen werden müssen.

Um dieses Verfahren zu implementieren, müssen zwei Fragen beantwortet, bzw. Probleme gelöst werden:

? **Frage**

1. Wie kann festgestellt werden, ob auf eine Seite zugegriffen wurde?
2. Wie lang ist die mit "*in letzter Zeit*" gemeinte Zeitspanne?

Auf die erste Frage gibt es eine ganz einfache Antwort: Mit Hilfe des R-Bits.

Das Referenziert-Bit

In den Seitentabellen wird eine zusätzliche Spalte eingeführt, siehe Abbildung unten. Für jede einzelne Seite (d.h. in jedem Seitentabelleneintrag) wird mit Hilfe eines einzelnen Bits festgehalten, ob die betreffende Seite referenziert wurde.

Man spricht hier vom sogenannten **Referenziert-Bit**, oder kurz vom **R-Bit**.

- Das R-Bit ist gesetzt, also 1:
  Der Inhalt der Seite wurde referenziert, es wurde also darauf zugegriffen.
- Das R-Bit ist nicht gesetzt, also 0:
  Der Inhalt der Seite wurde nicht referenziert, es wurde also nicht darauf zugegriffen.

Seitenrahmen-Nr.	Present-/ Absent-Bit	R-Bit	M-Bit
0 1111 0000 0011	1	1	0
0 1100 0111 0101	0	0	0
1 0101 0011 1110	0	0	0
0 1001 1111 1001	1	1	1
...	...	...	...

Abb. 61: Seitentabelle mit zusätzlicher Spalte für R-Bit

CC-BY

Es liegt auf der Hand, dass das R-Bit (genau wie das M-Bit) beim Einlagern einer virtuellen Seite in einen Seitenrahmen im betreffenden Seitentabelleneintrag mit 0 initialisiert wird.

Die MMU nimmt später bei der (erfolgreichen) Umrechnung einer virtuellen in eine physikalische Adresse ein Setzen des R-Bits im betreffenden Eintrag der Seitentabelle vor.

Aufgabe 1

Aufg. 172: Lesen, schreiben, R-Bit?

Es ist klar, dass das R-Bit bei einem **lesenden Zugriff** auf eine Seite von der MMU gesetzt werden muss.

Muss das R-Bit aber auch bei einem **schreibenden Zugriff** gesetzt werden? Zusammen mit dem M-Bit?

Tipp

Wenn du dir nicht sicher bist, dann lies zunächst weiter auf dieser Seite und beantworte diese Frage etwas später.

Arbeitsweise von NRU

Sobald eine Entscheidung bzgl. der zu ersetzenden Seite herbeigeführt werden muss, kategorisiert NRU alle in Seitenrahmen eingelagerte virtuelle Seiten in vier Klassen:

- **Klasse 1:** Eingelagerte Seiten mit R=0 und M=0.
  Von allen in dieser Klasse befindlichen Seiten wird eine zur Ersetzung bestimmt. Wegen M=0 muss diese Seite nicht im Hintergrundspeicher gesichert werden, der aufgetretene Seitenfehler kann also möglichst schnell behoben werden.
  Dies ist der beste Fall, der in Folge eines Page faults eintreten kann.

- **Klasse 2:** Eingelagerte Seiten mit R=0 und M=1.
  Sofern in Klasse 1 keine Seiten vorhanden sind, wird eine Seite aus Klasse 2 zur Auslagerung bestimmt. Wegen M=1 muss diese Seite zwar in den Hintergrundspeicher geschrieben werden (was Zeit kostet), jedoch lässt R=0 darauf hoffen, dass die Seite in Zukunft nicht so schnell wieder benötigt wird.

- **Klasse 3:** Eingelagerte Seiten mit R=1 und M=0.

Sofern in den Klassen 1 und 2 keine Seiten vorhanden sind, wird eine Seite aus Klasse 3 zur Auslagerung bestimmt. Wegen M=0 muss diese Seite nicht im Hintergrundspeicher gesichert werden, jedoch lässt R=1 erwarten, dass die Seite in (naher) Zukunft wieder benötigt wird. Mit hoher Wahrscheinlichkeit tritt deswegen ein Folge-Seitenfehler auf.

- **Klasse 4:** Eingelagerte Seiten mit R=1 und M=1.
  Sofern in den Klassen 1, 2 und 3 keine Seiten vorhanden sind, wird eine Seite aus Klasse 4 zur Auslagerung bestimmt. Wegen M=1 muss diese Seite in den Hintergrundspeicher geschrieben werden. Zusätzlich lässt R=1 erwarten, dass die Seite in (naher) Zukunft wieder benötigt wird.
  Dies ist der schlechteste Fall, der eintreten kann: Viel Zeitbedarf zum Schreiben der Seite in den Hintergrundspeicher und mit hoher Wahrscheinlichkeit in Kürze ein Folge-Seitenfehler.

Aufgabe 2

Aufgabe

Aufg. 173: Vier Klassen ohne Seiten?
Könnte es vorkommen, dass in allen vier Klassen keine Seiten vorhanden sind? Begründe deine Meinung.

Die Arbeitsweise von NRU ist nun weitgehend bekannt. Allerdings fehlt noch die Antwort auf die zweite Frage von oben:

Wie lang ist die mit "in letzter Zeit" gemeinte Zeitspanne?

Tanenbaum 2009 nennt hier "**ungefähr 20 ms**", andere Autoren verzichten ganz auf eine konkrete Angabe. Es darf jedoch davon ausgegangen werden, dass diese Zeitspanne relativ kurz im Verhältnis zum menschlichen Zeitempfinden gewählt werden sollte.

Bevor die Auswirkung dieser Zeitspanne näher erläutert wird, noch folgende Überlegung:

Ein (beliebiger) Seitenersetzungsalgorithmus (und damit speziell auch NRU) lagert eine virtuelle Seite in einen physikalischen Seitenrahmen ein, wenn diese benötigt wird. Es ist eine Reaktion auf einen direkt zuvor aufgetretenen Seitenfehler.

Sobald nun die benötigte Seite eingelagert wurde (und der Scheduler dem betreffenden Prozess die CPU zugeteilt hat), wird der nächste Speicherzugriff sofort auf die gerade eingelagerte Seite erfolgen, das zugehörige R-Bit wird zwangsläufig gesetzt.

Frage

Können deshalb eigentlich eingelagerte Seiten mit R=0 vorkommen?
Sind also Seiten in Klasse 1 und 2 zu erwarten?

Die Wahrscheinlichkeit dafür dürfte sehr gering sein.

Jetzt kommt jedoch die erwähnte Zeitspanne ins Spiel: **Immer nach Ablauf dieser Zeit werden in allen Seitentabellen alle R-Bits gelöscht (also auf 0 gesetzt)!**

Damit dient das R-Bit als Indikator für all diejenigen eingelagerten virtuellen Seiten, die seit dem letzten "Auf-0-setzen" des R-Bits referenziert wurden. Folglich können sehr wohl Seiten in den Klassen 1 und 2 vorkommen.

Aufgabe 3

Aufgabe

Aufg. 174: Auch das M-Bit löschen?
Wenn nach Ablauf der erwähnten Zeitspanne das R-Bit gelöscht wird, sollte dann gleich auch das M-Bit mit gelöscht werden? Begründe deine Meinung!

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.3.2.2.3.3 FIFO - First In First Out Algorithmus

Der **First In First Out Seitenersetzungsalgorithmus**, kurz **FIFO**, ersetzt immer genau die Seite, die bereits am längsten eingelagert ist. </p>
In der Praxis hat dieses Verfahren keine große Bedeutung. Es ist sehr einfach zu implementieren, jedoch ist davon auszugehen, dass die Tatsache, dass eine Seite schon lange eingelagert ist, kein Indiz dafür ist, dass sie nicht in Kürze wieder benötigt werden wird.

Es bleibt zu befürchten, dass der FIFO-Algorithmus Seiten auslagert, die oft benötigt werden. Somit käme es zu einer übermäßig hohen Zahl an Seitenfehlern.

Der Second Chance Algorithmus stellt eine Erweiterung und Verbesserung des FIFO-Algorithmus dar.

Aufgabe 1

Aufgabe

Aufg. 175: FIFO und die Seitentabelle

Angenommen, der FIFO-Algorithmus wird in einem Betriebssystem eingesetzt. Wie sieht dann ein Seitentabelleneintrag aus? Werden insbesondere das R-Bit und das M-Bit benötigt?

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.3.2.2.3.4 Second Chance Algorithmus

Der **Second Chance Seitenersetzungsalgorithmus** ist eine Verbesserung des First In First Out Algorithmus. Er ersetzt nicht einfach stur die am längsten eingelagerte Seite, sondern prüft zuvor, ob diese Seite in der letzten Zeit auch angesprochen wurde. </p> Falls sie nicht angesprochen wurde, so wird sie ersetzt.

Falls sie aber doch angesprochen wurde, so wird mit der am zweitlängsten eingelagerten Seite fortgefahren. Auch hier wird zunächst geprüft, ob die Seite in der letzten Zeit angesprochen wurde. Usw.

Aufgabe 1

Aufg. 176: Angesprochen oder nicht?

Wie kann das Betriebssystem es realisieren, dass für eine eingelagerte Seite festgestellt werden kann, ob sie in letzter Zeit angesprochen wurde? Erläutere das Verfahren!

Aufgabe 2

Aufg. 177: Wenn alle die zweite Chance nutzen

Ein Betriebssystem arbeite mit dem Second Chance Algorithmus.

Was passiert, wenn wirklich **alle eingelagerten Seiten in letzter Zeit angesprochen** wurden? Alle Seiten nutzen damit ihre *zweite Chance*. Aber welche Seite wird dann ersetzt?

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.3.2.2.3.5 Working Set Algorithmus

Der **Working Set Seitenersetzungsalgorithmus** basiert auf interessanten Erkenntnissen: </p>

- Wird ein Prozess auf der CPU ausgeführt, so werden seine Befehle der Reihe nach von Steuerwerk und Rechenwerk verarbeitet. Jeder Befehl stammt aus einer bestimmten virtuellen Seite aus dem Adressraum des Prozesses.
- Der jeweils nächste zu verarbeitende Befehl stammt ebenfalls aus einer virtuellen Seite. In vielen Fällen wird es sogar dieselbe virtuelle Seite wie beim Befehl zuvor sein, denn die Befehle werden überwiegend sequentiell abgearbeitet.
- Zum Beispiel bei Sprüngen kann ein Befehl zur Ausführung kommen, der aus einer anderen virtuellen Seite stammt. Werden danach weitere Befehle sequentiell abgearbeitet, so stammen diese mit hoher Wahrscheinlichkeit wieder alle von derselben virtuellen Seite.

Insgesamt ist damit erkennbar, dass bei vielen nacheinander auf der CPU ausgeführten Befehlen nur relativ wenig verschiedene virtuelle Seiten angesprochen werden. Genau diese Seiten bilden nun das **Working Set** des betrachteten Prozesses.

Der Working Set Algorithmus versucht nun, alle zum Working Set eines Prozesses gehörenden Seiten ständig im Hauptspeicher zu halten.

Optimalerweise geht der Algorithmus dabei sogar noch einen Schritt weiter:
Halte **für alle Prozesse** auch **alle zum jeweiligen Working Set gehörigen virtuellen Seiten** stets in einem Seitenrahmen eingelagert.

Sofern dies gelingt, kann man ein System erwarten, welches nur noch eine verhältnismäßig kleine Zahl an Seitenfehlern produziert.

Lokalitätseffekt

Der Working Set Algorithmus nutzt den sogenannten Lokalitätseffekt von Prozessen, der nach *Glatz 2010* wie folgt umschrieben werden kann:

Der **Lokalitätseffekt** bezeichnet die Idee, dass innerhalb größerer Zeiträume immer nur ein Teil des gesamten Codeumfangs eines Prozesses ausgeführt wird.

Aufgabe 1

Aufgabe

Aufg. 178: Die 80/20-Regel und das Working Set
Du kennst sicher die 80/20-Regel:

> *80 Prozent der von einer Software bereitgestellten Funktionen werden höchstens von 20 Prozent der Nutzer tatsächlich eingesetzt.*
>
> Erläutere (unter der Voraussetzung, dass die Regel zutrifft):
> Wie unterstützt diese Regel den Working Set Algorithmus bei seinem Ziel, möglichst wenig Seitenfehler entstehen zu lassen?

Weiterführende Literatur

Eine detailliertere Auseinandersetzung mit dem Working Set Algorithmus findet sich u.a. bei:

- *Glatz 2010* , Kapitel 7.5.2
- *Mandl 2013* , Kapitel 7.2.3
- *Strelen 2012* , Kapitel 6.6.1
- *Tanenbaum 2009* , Kapitel 3.4.8

und kann dort - je nach Verfügbarkeit der Quellen - nachgelesen werden.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.3.3 Shared Memory

Der englische Begriff **Shared Memory** bedeutet übersetzt **Gemeinsam genutzer Speicher**. Mehrere Prozesse sollen also vom Betriebssystem die Möglichkeit zur Verfügung gestellt bekommen, dass bestimmte Speicherbereiche gemeinsam genutzt werden können. </p>

Hinweis

> Wer sich noch an das Kapitel Limitregister zum Speicherschutz erinnert, dem ist sicher auch noch bewußt, dass die Nutzung **eines** Speicherbereichs durch **mehrere** Prozesse eigentlich gar nicht erwünscht war, und unbedingt unterbunden werden sollte. (Was durch den Einsatz des Limitregisters ja auch realisiert wurde.)

Wenn also Shared Memory jetzt als Betriebssystemkonzept eingeführt wird, dann muss es dafür gute Gründe geben.

Ein Grund ist bereits im Kapitel Interprozesskommunikation angeklungen: Der Austausch von Informationen zwischen verschiedenen Prozessen. Etwas weiter unten in diesem Abschnitt wird dieses nochmals aufgegriffen.

Es gibt jedoch noch einen weiteren sehr guten Grund, der für den Einsatz von gemeinsam genutztem Speicher spricht:

RAM kann man nie genug haben

Der (physikalische) Arbeitsspeicher ist seit jeher eine knappe Ressource. Wenn es also Möglichkeiten gibt, durch geschickte Ausnutzung von Dingen, die sowieso vorhanden sind, "gefühlt mehr RAM" zur Verfügung zu haben, dann macht es Sinn, diese Möglichkeiten auch zu nutzen.

Ein Beispiel aus der Praxis:

Aus der Praxis

Wenn ein Programm gestartet, und dadurch zum Prozess wird, dann muss sein ausführbarer Maschinencode (der Programmtext) im (physikalischen) Speicher zur Verfügung stehen, damit er auf der CPU ausgeführt werden kann. (Durch die virtuelle Speicherverwaltung reicht es bekanntlich, dass nur ein Teil des Programmtextes eingelagert ist.)

Was passiert nun, wenn das gleiche Programm mehrmals gestartet wird?
Dann resultieren daraus auch mehrere Prozesse.

Wenn jetzt für jeden Prozess immer der gleiche Programmtext in unterschiedliche Bereiche des Hauptspeichers eingelagert werden müsste, so ergäbe sich eine Redundanz, die letztlich einer vermeidbaren Speicherverschwendung entspricht.

Windows-User kennen vielleicht **DLL-Dateien** (**Dynamic Link Libraries**). Unter Unix/Linux gibt es etwas Gleichwertiges mit den sogenannten **Shared Libraries** .

In beiden Fällen enthalten diese Dateien ausführbaren Code, der von vielen Prozessen genutzt wird. Wenn dieser Code nur einmal im Hauptspeicher vorgehalten werden muss, dann wird eine signifikante Einsparung von benutztem Hauptspeicher erzielt.

Dabei ist die Umsetzung des Shared Memory Konzepts ganz einfach:

Realisiert durch Dinge, die sowieso vorhanden sind

Man nehme die virtuelle Speicherverwaltung mit ihren Seitentabellen. Das Betriebssystem kann dann für eine beliebige Anzahl (mindestens zwei) an Prozessen das Shared Memory Konzept durch geschickte Einträge in den Seitentabellen der beteiligten Prozesse realisieren.

Der gemeinsam genutzte Speicher entspricht dann einem bestimmten Seitenrahmen des physikalischen Speichers. Hier könnten beispielsweise DLL-Dateien oder Shared Libraries eingelagert sein.

Dieser **eine** Seitenrahmen kann nun sehr einfach in **alle** virtuellen Adressräume der beteiligten Prozesse eingebunden werden. Nötig ist dazu allein ein entsprechender Eintrag in **allen** betreffenden Seitentabellen.

Hinweis

Bedenke: Jeder Prozess besitzt seinen eigenen virtuellen Speicher und damit auch seine eigene Seitentabelle!

Wie einfach die Realisierung von Shared Memory tatsächlich ist, wird durch die folgende Aufgabe klar:

Aufgabe 1

Aufgabe

Aufg. 179: Realisiere Shared Memory

Fertige eine Skizze an.

Anhand der Skizze sollst du für zwei Prozesse mit ihren Seitentabellen erklären, wie ein physikalischer Seitenrahmen in den jeweiligen virtuellen Adressraum der beiden Prozesse eingebunden wird.

Warum eine Skizze? Weil ein Bild immer mehr sagt, als 1.000 Worte.

Erläutere deine Skizze!

Aber denke daran: Du brauchst keine 1.000 Worte mehr...

Aufgabe 2

Aufgabe

Aufg. 180: Weniger Seitenfehler durch Shared Memory?

Was denkst du:

Können durch den Einsatz von Shared Memory Seitenfehler (Page faults) vermieden werden?

Erläutere deine Meinung!

Gemeinsam genutzte Datenbereiche

Bislang wurde in diesem Abschnitt davon ausgegangen, dass Programmtext in den gemeinsam genutzten Speicher eingelagert ist. Da Programmtext für alle Prozesse "read only" ist, sich insbesondere also zur Laufzeit nicht verändert, ist das unkritisch.

Man kann dieses Konzept aber auch auf gemeinsam genutze Datenbereiche anwenden.

Aufgabe 3

Aufgabe

Aufg. 181: Shared Memory mit Daten statt Programmtext

Erläutere was passieren kann, wenn Shared Memory auf einen gemeinsam genutzten Datenbereich angewendet wird. Der Datenbereich darf dabei von allen beteiligten Prozessen ausgelesen und beschrieben (read/write) werden.

Welche bereits bekannten Mechanismen helfen dabei Probleme zu vermeiden?

Tipp

Schau mal in dieses Kapitel mit seinen Unterkapiteln!

Gib mindestens ein Praxisbeispiel an, bei dem mehrere Prozesse auf einen gemeinsamen Datenbereich zugreifen.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.3.4 Speicherverwaltung moderner Betriebssysteme

Auf den folgenden Unterseiten wird die Speicherverwaltung in modernen Betriebssystemen behandelt: </p>

Gliederung

3.3.4 Speicherverwaltung moderner Betriebssysteme
3.3.4.1 Speicherverwaltung unter Linux
3.3.4.2 Speicherverwaltung in Windows
3.3.4.3 Speicherverwaltung unter Android

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.3.4.1 Speicherverwaltung unter Linux

Dieses Kapitel wird in der weiterführenden Literatur behandelt: </p>

Hinweis

Weiterführende Literatur

Mandl 2013 erläutert das Thema in Kapitel 7.3.1. Die Lektüre dieser Quelle sei dem geneigten Leser überlassen.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.3.4.2 Speicherverwaltung in Windows

Dieses Kapitel wird in der weiterführenden Literatur behandelt: </p>

Hinweis

Weiterführende Literatur

Mandl 2013 erläutert das Thema in Kapitel 7.3.2. Die Lektüre dieser Quelle sei dem geneigten Leser überlassen.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.3.4.3 Speicherverwaltung unter Android

Dieses Kapitel wird in der weiterführenden Literatur behandelt: </p>

Hinweis

Weiterführende Literatur

Mandl 2013 erläutert das Thema in Kapitel 7.3.3. Die Lektüre dieser Quelle sei dem geneigten Leser überlassen.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.4 Geräteverwaltung

Im Teil Computerarchitektur wurde bereits im Kapitel Kommunikation mit E/A-Geräten ein grundlegendes Bewusstsein dafür geschaffen, dass das Betriebssystem die Verwaltung der Ein- und Ausgabegeräte übernehmen muss. </p>
Man spricht hierbei allgemein von der Geräteverwaltung.

Definition: Geräteverwaltung

Definition

Unter der **Geräteverwaltung** fasst man alle Aufgaben und Tätigkeiten des Betriebssystems zusammen, welche einer optimierten Zusammenarbeit zwischen dem Betriebssystem (inklusive der darauf ablaufenden Prozesse) und den Peripheriegeräten dienen.

Peripheriegeräte sind beispielsweise Festplatte, Grafikkarte, Maus, Tastatur, Netzwerkkarte und auch viele weitere Geräte, welche zum Teil direkt auf dem Mainboard verbaut sind.

Prizipiell kann auch der Hauptspeicher (RAM) zu den Peripheriegeräten gezählt werden, jedoch macht dies im Kontext der Geräteverwaltung eines Betriebssystems wenig Sinn, denn die Speicherverwaltung des Betriebssystems steuert die Zusammenarbeit mit dem Hauptspeicher. Geräteverwaltung und Speicherverwaltung sind zwei separat betrachtete Teilbereiche von Betriebssystemen.

Aufgabe 1

Aufgabe

Aufg. 182: Überfliege erneut die Kommunikation

Schau dir noch einmal das Kapitel Kommunikation mit E/A-Geräten an.

Für das Verständnis der folgenden Seiten ist es wichtig, dass du u.a. mit diesen Begriffen bzw. Zusammenhängen vertraut bist:

- Controller
- Register auf dem Controller
- Interrupt
- Interrupt-Controller
- Datentransfer über das Bussystem
- Zusammenhang zwischen Datentransfer und Interrupts
  (→ Video: http://youtu.be/nOEW4I_QX2c)

So geht es weiter:

Gliederung

3.4 Geräteverwaltung
3.4.1 Rolle der Geräteverwaltung
3.4.2 Abhängig und gleichzeitig unabhängig
3.4.3 Gerätetreiber
3.4.4 Aufgaben eines Treibers
3.4.5 Geräteklassen
3.4.6 Memory-Mapped-I/O
3.4.7 DMA - Direct Memory Access
3.4.8 Windows-Treiber auf GitHub

Alternative Webquelle zum Thema

Quelle im Web

Operating Systems: I/O Systems

http://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/13_IOSystems.html

Dr. John T. Bell

Department of Computer Science

University of Illinois, Chicago

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.4.1 Rolle der Geräteverwaltung

Wenn Prozesse mit Peripheriegeräten kommunizieren wollen, so können sie dies niemals direkt erledigen. Stattdessen tätigen sie einen Systemaufruf, wodurch das Betriebssystem mit der gewünschten Kommunikation beauftragt wird. </p>
Die Geräteverwaltung ist nun ein Bestandteil des Betriebssystems und realisiert eine Schnittstelle zwischen den Peripheriegeräten und dem Betriebssystem. Die folgende Abbildung zeigt dies an einem Beispiel.

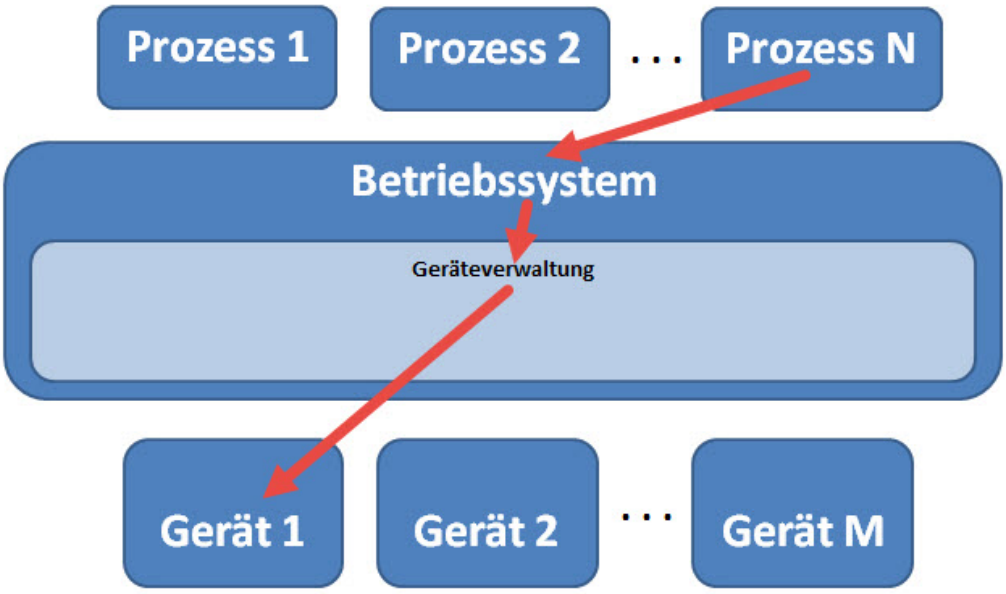


Abb. 62: Geräteverwaltung als Schnittstelle zwischen Peripheriegeräten und Betriebssystem CC-BY

Aufgabe 1

Aufgabe

Aufg. 183: Pfeilrichtungen und Kommunikationswege

In der vorangegangenen Abbildung kommuniziert *Prozess N* über Betriebssystem und Geräteverwaltung mit *Gerät 1*. Die Pfeilrichtungen kennzeichnen diesen Weg.

Müssen (ganz allgemein betrachtet) die Pfeilrichtungen immer so sein?

- Gib' ein Beispiel, in dem die Pfeilrichtungen vom Gerät zum Prozess hin ausgerichtet sind.
- Und gib' ein weiteres Beispiel, bei dem die Pfeile in beide Richtungen zeigen!

Welches konkrete Gerät kommuniziert in deinen Beispielen mit welchem konkreten Prozess?

Aufgabe 2

Aufgabe

Aufg. 184: Systemaufruf zwischen User- und Kernel-Mode

Betrachte noch einmal das Kapitel Kernel-Mode, User-Mode und Systemaufrufe und wiederhole, was es mit eben diesen Fachbegriffen auf sich hat:

- Kernel-Mode
- User-Mode
- Systemaufruf

Aufgabe 3

Aufgabe

Aufg. 185: Pfeil des Systemaufrufs

Welcher Pfeil aus der obigen Abbildung repräsentiert einen Systemaufruf?

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.4.2 Abhängig und gleichzeitig unabhängig

Die Geräteverwaltung hat es schwer: Sie muss mit allen erdenklichen Peripheriegeräten kommunizieren können. </p>

Anmerkung

Die Geräteverwaltung ist ein Bestandteil des Betriebssystems, also eine Software.

Sie muss mit unterschiedlichsten Hardware-Geräten kommunizieren können, u.a. auch mit Hardware, die vielleicht noch gar nicht entwickelt war, als sie (die Geräteverwaltung) programmiert wurde.

Diese Aufgabe wird durch eine klare Trennung bewältigt:

Wichtig

Die Geräteverwaltung besitzt sowohl geräteabhängige, als auch geräteunabhängige Teile.

Die geräteabhängigen Teile sind die sogenannten Gerätetreiber. Die geräteunabhängigen Teile werden durch die Definition unterschiedlicher Geräteklassen realisiert. Beide Themen werden in entsprechenden Unterkapiteln näher erläutert.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.4.3 Gerätetreiber

Damit die Geräteverwaltung ihrer Aufgabe nachkommen kann, besitzt sie gerätespezifische Anteile, die sogenannten Gerätetreiber oder kurz Treiber. Ein Treiber ist demnach auch eine Software. </p>

Definition: Gerätetreiber

Definition

Unter einem **Gerätetreiber** (oder kurz **Treiber**) versteht man eine Softwarekomponente, welche zur Geräteverwaltung des Betriebssystems gehört, und Interaktionen zwischen Betriebssystem und Controller eines bestimmten Peripheriegeräts steuert.

Die folgende Abbildung verdeutlicht diese Definition:

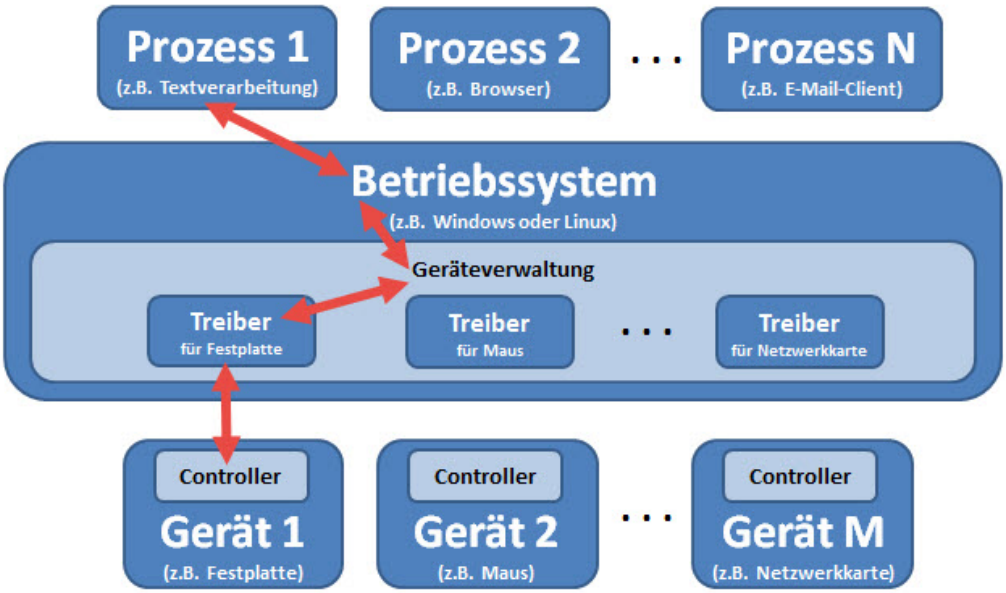


Abb. 63: Treiber als Mittler zwischen Controller und Geräteverwaltung
CC-BY

Wie man sieht, enthält die Geräteverwaltung viele verschiedene Treiber. Warum das so sein muss, klärt die folgende Aufgabe.

Aufgabe 1

Aufgabe

Aufg. 186: Du hast doch schon mal einen Treiber installiert, oder?
Dann weißt du, dass ein Anwender (oder besser: der Administrator des Systems) einen Treiber auch zeitlich nach der Installation des Betriebsystems noch hinzufügen kann. Aber:

- Wer ist üblicherweise für die Programmierung eines Treibers für ein spezielles Gerät zuständig?
- Warum ist es i.d.R. nicht sinnvoll, dass Treiber von jemand anderen programmiert werden? (Wer könnte dieser "jemand anderes" ggf. sein?)

Aufgabe 2

Aufgabe

Aufg. 187: Treiberausführung im User- oder im Kernel-Mode?

Wenn ein Treiber - wie hier beschrieben - eine Software ist, dann werden Bestandteile dieses Treibers (→ Befehle, Maschinencode) irgendwann auch mal auf der CPU ausgeführt.

Läuft diese Ausführung dann im User-Mode oder im Kernel-Mode ab?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.4.4 Aufgaben eines Treibers

Die verschiedenen Aufgaben eines Treibers werden auf den folgenden Seiten erläutert. </p>

So geht es weiter:

Gliederung

3.4.4 Aufgaben eines Treibers
3.4.4.1 Initialisierung des Geräte-Controllers
3.4.4.2 Gerät dem Betriebssystem bekannt machen
3.4.4.3 Bereitstellen einer Schnittstelle zum Geräte-Controller
3.4.4.4 Interruptbehandlung für ein Gerät
3.4.4.5 Bereitstellen einer Schnittstelle zur Geräteverwaltung
3.4.4.6 Pufferung von Daten
3.4.4.7 Koordination nebenläufiger Zugriffe auf ein Gerät

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.4.4.1 Initialisierung des Geräte-Controllers

Ein Treiber initialisiert den Controller seines zugehörigen Geräts beim Systemstart. Damit stehen gültige Werte in den Registern des Controllers und das Gerät selbst wird über seinen Controller bereit für die Entgegennahme von Befehlen. </p>

Abb. 64: Ein initialisierter Controller
CC-BY

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.4.4.2 Gerät dem Betriebssystem bekannt machen

Ein Treiber macht der Geräteverwaltung des Betriebssystems das Gerät bekannt. Erst damit kann das Gerät vom Betriebssystem (und den darauf ablaufenden Prozessen) genutzt werden. </p>

Aus der Praxis

Die folgende Abbildung zeigt den Geräte-Manager eines Windows 7 Betriebssystems mit den derzeit zur Verfügung stehenden Geräten.

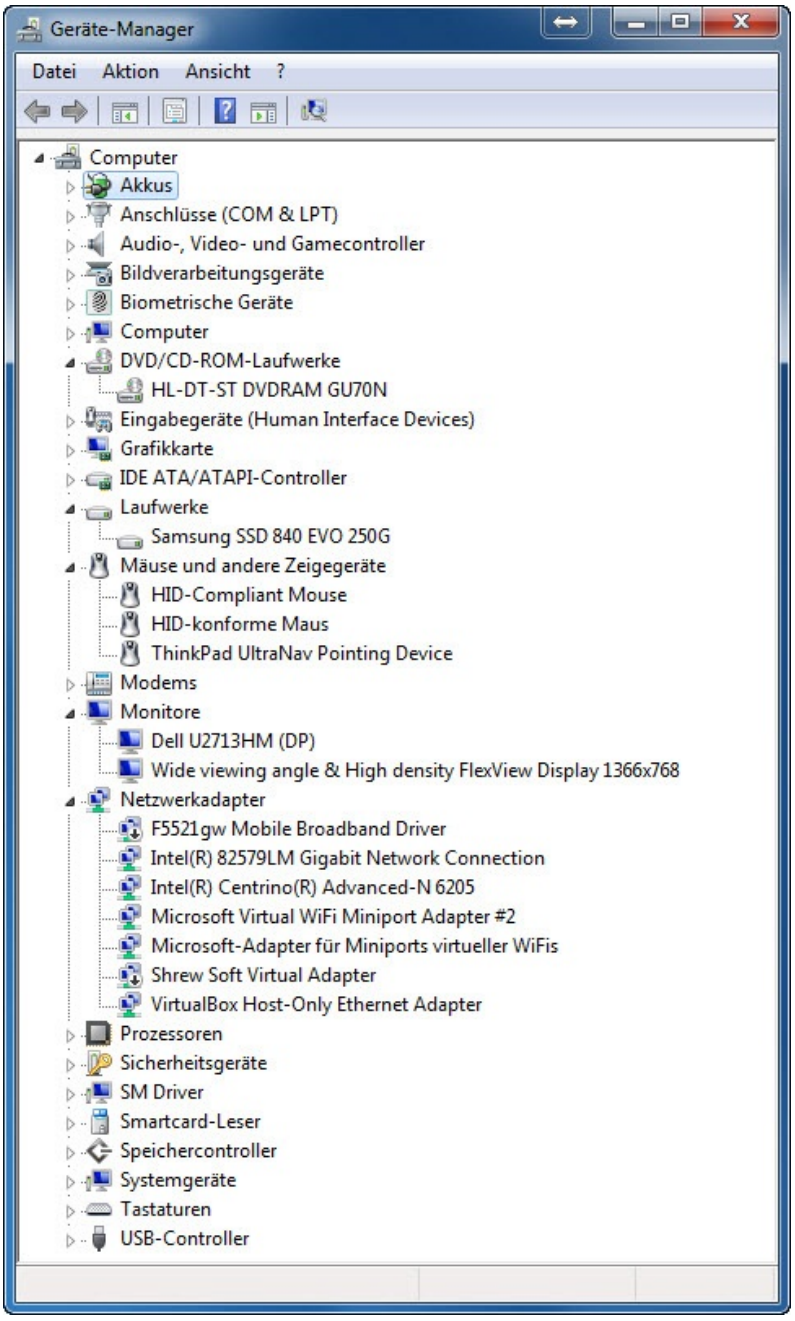


Abb. 65: Geräte-Manager von Windows 7

CC-BY

Aus der Praxis

Auch unter Linux können die bekannten Geräte angezeigt werden. Auf der Kommandozeile steht dafür das Programm *lshw* zur Verfügung, welches bei vielen Distributionen jedoch separat installiert werden muss. Zu seiner Ausführung benötigt es zudem ROOT-Rechte.

Die folgende Abbildung zeigt die kompakte Ausgabe von *lshw -short*. Ohne den short-Parameter ergibt sich eine deutlich längere und detailliertere Ausgabe.

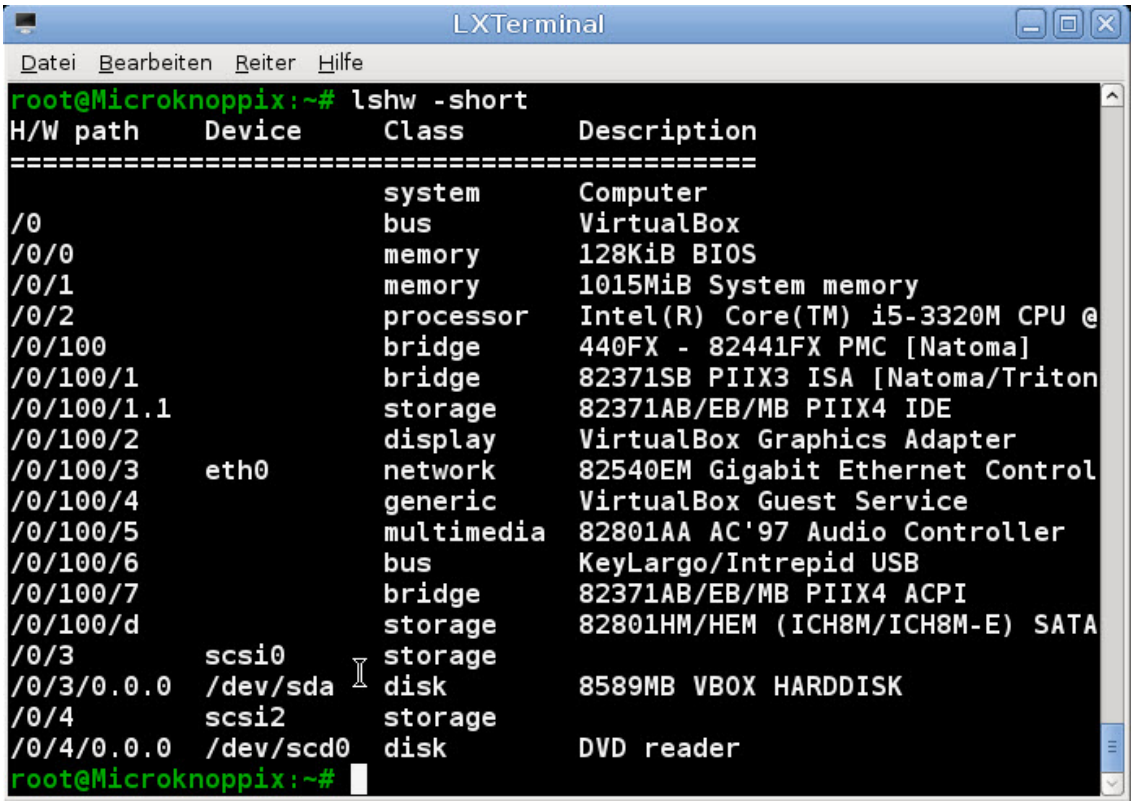


Abb. 66: 'lshw -short' unter Linux

CC-BY

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.4.4.3 Bereitstellen einer Schnittstelle zum Geräte-Controller

Ein Treiber bildet die Schnittstelle zwischen der Geräteverwaltung und dem Controller des betreffenden Geräts. Insbesondere muss der Treiber deshalb wissen, welche Befehle oder Funktionen der Controller zur Verfügung stellt, wie diese anzusprechen sind, und in welcher Weise die Rückgabewerte zu interpretieren sind. </p>

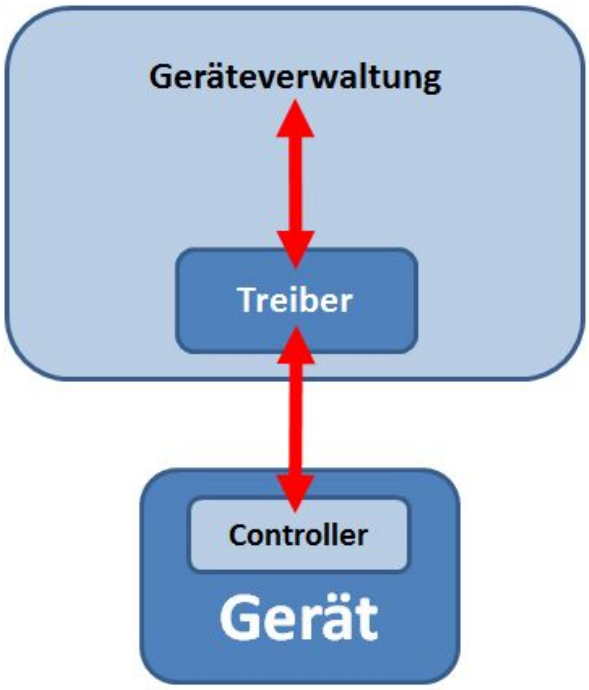


Abb. 67: Der Treiber als Mittler zwischen Geräteverwaltung und Geräte-Controller
CC-BY

Beispiel aus alten Tagen

Tanenbaum 2009 hat in früheren Auflagen seines Buchs gerne ein Beispiel aus *den guten alten Tagen* des Computerzeitalters gegeben: Ein kurzer Blick in die Kommunikation mit einem Floppy Disk Controller (kurz: FDC).

Frage

In Zeiten von USB-Sticks, SSD-Festplatten und Cloud-Speicher weißt du doch noch, was eine Floppy und ein Diskettenlaufwerk war, oder?

Notfalls zeigt Wikipedia dir die Hintergründe:
http://de.wikipedia.org/wiki/Diskette

NEC PD765 Floppy Disk Controller Chip

Die Firma NEC Electronics U.S.A. Inc. hatte seit den 1970er Jahren mit dem NEC PD765 FDC Chip einen oft eingesetzen Controller für Diskettenlaufwerke am Markt. Im Zuge der üblichen Weiterentwicklung existierten verschiedene Versionen dieses Chips.

Quelle im Web

Man findet noch heute die zugehörigen Datenblätter (engl. data sheets) im Internet, z.B. hier:

http://www.bitsavers.org/components/nec/_dataSheets/
uPD765_Data_Sheet_Dec78.pdf

Allgemeine Informationen zum Thema 'Floppy Disk Controller' gibt es auf der englischsprachigen Wikipedia-Seite:

https://en.wikipedia.org/wiki/Floppy-disk_controller

Aufgabe 1

Aufgabe

Aufg. 188: Befehle des NEC PD765 FDC

Finde über dieses NEC PD765 Data Sheet heraus, wieviele und welche Befehle der Controller bereitstellt.

Zitat aus dem Data Sheet

Im NEC PD765 Data Sheet wird erläutert, dass der Controller zwei interne Register bereitstellt: ein Status Register und ein Data Register. Dazu heißt es im Absatz "Internal Registers" auf Seite 6:

Zitat

"The 8-bit Data Register (actually consists of several registers in a stack with only one register presented to the data bus at a time), which stores data, commands, parameters, and FDD status information. Data bytes are read out of, or written into, the Data Register in order to program or obtain the results after a particular command."

Tätigkeiten des FDC-Treibers

Aus dieser Beschreibung lassen sich die Tätigkeiten des zugehörigen Treibers bei einer Kommunikation mit dem PD765 Floppy Disk Controller ableiten:

1. Schreiben des Befehls mit allen nötigen Parametern und Daten in das Data Register des PD765.
2. Abwarten, bis der Befehl ausgeführt wurde.
3. Auslesen der Statusinformationen und Daten zum ausgeführten Befehl aus dem Data Register des PD765.
4. Interpretation der ausgelesenen Statusinformationen.
5. Einleiten weiterer Schritte auf Basis der Interpretation.

Wichtig

Unter den Tätigkeiten 1 und 3 des Treibers versteht man die *Bereitstellung einer Schnittstelle zum Geräte-Controller*.

Aufgabe 2

Aufgabe

Aufg. 189: Several registers in a stack

In dem Zitat oben heißt es:

"The 8-bit Data Register (actually consists of several registers in a stack with only one register presented to the data bus at a time)..."

Angenommen:

- Der Treiber möchte einen Befehl in das Data Register des Controllers schreiben. Dieser Befehl habe eine Gesamtlänge von insgesamt vier Byte.
- Nach der erfolgreichen Befehlsausführung liefere der Controller insgesamt sieben Byte zurück, die vom Treiber über das Data Register ausgelesen werden müssen.

Beschreibe den Ablauf der Kommunikation zwischen Treiber und Data Register vor dem Hintergrund, dass das Data Register nur eine Breite von 8 Bit besitzt und aus *"several registers in a stack"* besteht.

Die Antwort auf diese Aufgabe zeigt, wie genau das Handeln des Treibers auf die Bedürfnisse und Gegebenheiten des Geräte-Controllers zugeschnitten sein muss.

Man kann sich auch vorstellen, dass Controller für andere Geräte, eine völlig unterschiedliche Kommunikation mit dem Treiber erfordern. Dies gilt umso mehr für Controller, die erst sehr viel später als der hier beispielhaft angesprochene NEC PD765 entwickelt wurden.

Die *Schnittstelle* wird mit hoher Wahrscheinlichkeit eine andere sein.

Hinweis

Du verstehst spätestens jetzt, warum es - wie in der folgenden Abbildung angedeutet - für jedes Gerät (bzw. dessen Controller) einen eigenen Treiber geben muss, oder?

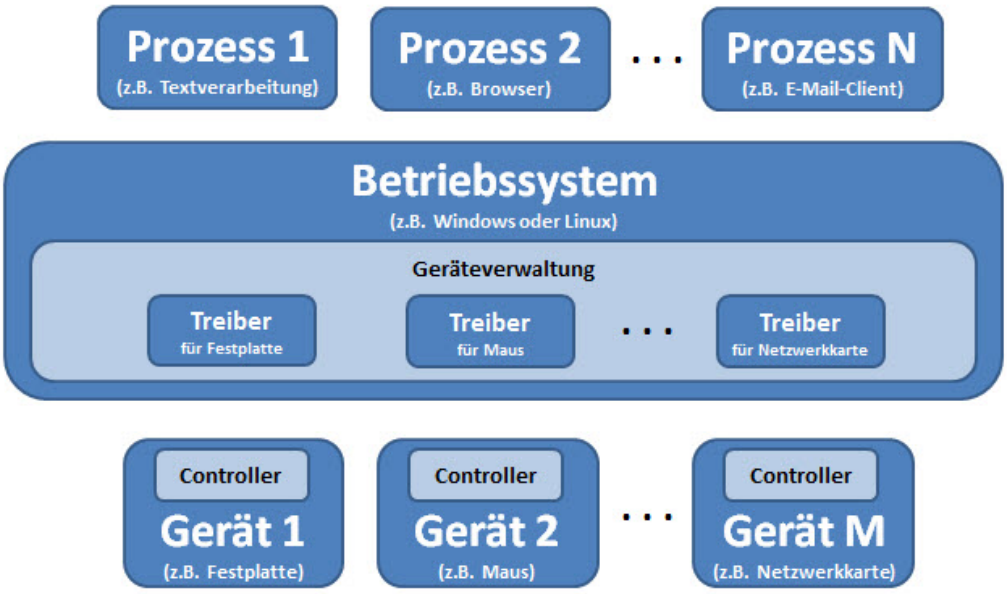


Abb. 68: Für jedes Gerät (bzw. Controller) ein eigener Treiber

CC-BY

Aufgabe 3

Aufgabe

Aufg. 190: Abwarten! - Und Tee trinken?

Unter Punkt 2 der Tätigkeiten des FDC-Treibers ist aufgeführt:
"Abwarten, bis der Befehl ausgeführt wurde."

Woher weiß der Treiber, **wann** der an den Controller gesendete Befehl vom Gerät ausgeführt wurde, und die Ergebniswerte aus den Controller-Registern abgerufen werden können (→ Punkt 3 der Tätigkeiten)?

Tipp

Datentransfer und Interrupts

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.4.4.4 Interruptbehandlung für ein Gerät

Ein Gerätetreiber ist zuständig für die Behandlung von Interrupts, die durch das zugehörige Gerät (bzw. dessen Controller) ausgelöst werden.

Zum tieferen Verständnis dieses einfachen Sachverhalts rufe man sich die bereits bekannte Verfahrensweise rund um Interrupts in Erinnerung:

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/_wVNpUW3kdM

Med. 35: Was passiert bei einem Interrupt? (02:14)
http://youtu.be/_wVNpUW3kdM
CC-BY

In dem vorangegangenen Video wird gezeigt, wie ein Interrupt einen aktuell auf der CPU laufenden Prozess A unterbricht. Prozess A bekommt direkt nach der Abarbeitung der Interruptbehandlungsroutine die CPU zurück. Die folgende Abbildung greift diese Situation auf und zeigt den zeitlichen Verlauf auf dem Prozessor. Weiterhin ist der Unterschied zwischen einem Kontextwechsel und einem Interrupt zu erkennen.

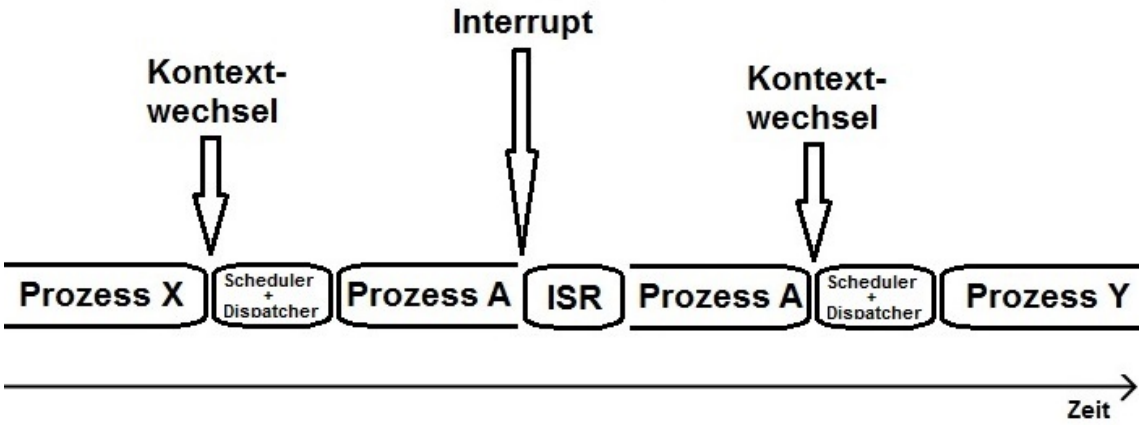


Abb. 69: Ein Interrupt mit seiner zugehörigen Interruptbehandlungsroutine (ISR) unterbricht den Prozess A auf der CPU
CC-BY

Aus dem Abschnitt Datentransfer und Interrupts stammt das folgende Video. Es geht u.a. auf eine Interruptbehandlungsroutine ein, welche im Video ganz allgemein als ein "Teil des Betriebssystems" bezeichnet wird (*ISR #14* ab Minute 1:44).

Video

An dieser Stelle befindet sich online ein YouTube-Video.

https://youtu.be/nOEW4I_QX2c

Med. 36: Ein- und Ausgabe mit Festplatte und Interrupts (04:00)
http://youtu.be/nOEW4I_QX2c
CC-BY

Diese allgemeine Beschreibung (*"Die Interruptbehandlungsroutine ist Teil des Betriebssystems"*) wird nun etwas konkretisiert:

Die Interruptbehandlungsroutine ist Teil des Gerätetreibers!

(Und der Treiber ist Teil der Geräteverwaltung, welche wiederum ein Teil des Betriebssystems ist.)

Aufgabe 1

Aufg. 191: Interruptbehandlung als Tätigkeit

Betrachte noch einmal die beispielhaft erläuterten Tätigkeiten des FDC-Treibers.

Welche der dort aufgelisteten Tätigkeiten beinhaltet die *Interruptbehandlung für ein Gerät*?

(Besteht die Interruptbehandlung vielleicht aus mehr als nur einer Tätigkeit?)

Aufgabe 2

Aufg. 192: Nur der Treiber!

Wenn du in der vorangegangenen Aufgabe die Tätigkeiten identifiziert hast, welche durch die vom Treiber bereitgestellte Interruptbehandlungsroutine durchgeführt werden, dann kannst du sicher auch die folgende Frage beantworten:

Warum kann diese Tätigkeiten **nur der Treiber** vollständig und korrekt durchführen?

Interruptbehandlung bei einzeln übertragenen Datenwörtern

Das oben eingebundene Video verdeutlicht die Interruptbehandlung bei einzeln von der Festplatte übertragenen Datenwörtern. Dabei hatte ein Datenwort gerade einmal die Breite des Datenbusses.

Im Abschnitt Datentransfer und Interrupts findet sich zudem eine Auflistung vieler durchzuführender Schritte, die bei der Übertragung jedes einzelnen Datenwortes anfallen.

Moderne Systeme arbeiten jedoch schon lange nicht mehr nach diesem Verfahren, sondern es kommt ein DMA-Controller zum Einsatz.

Interruptbehandlung bei Einsatz eines DMA-Controllers

Auf der Seite zum DMA-Controller wurde erläutert, wie sich der Einsatz eines DMA-Controllers auf die Anzahl der zu verarbeitenden Interrupts auswirkt.

Aufgabe 3

Aufgabe

Aufg. 193: DMA und Interrupts - Wie war das nochmal?

Erläutere in einem Satz:

Wie wirkt sich der Einsatz eines DMA-Controllers auf die Anzahl der (vom Treiber) zu verarbeitenden Interrupts aus?

(Du kannst für diese Aufgabe vom Standardbeispiel ausgehen: Eine große Datei soll von der Festplatte in den Hauptspeicher kopiert werden.)

Nicht jedes Gerät ist DMA-fähig

Die bisherigen Beispiele haben sich oft auf einen Standardfall bezogen: *Datenübertragung mit der Festplatte*. Jetzt ist eine Festplatte allerdings ein klassisches Gerät, welches Datentransfer per DMA unterstützt.

Wichtig

Außer Festplatten gibt es noch viel weitere Geräte, die von der Geräteverwaltung des Betriebssystems unterstützt werden müssen, und für die ein Treiber eingebunden ist.

Und lange nicht alle Geräte unterstützen den Einsatz eines DMA-Controllers!

Man kann sich an dieser Stelle leicht vorstellen, dass die Interruptverarbeitung im Treiber sehr unterschiedlich ablaufen kann. Mit diesem Umstand beschäftigt sich die folgende Aufgabe vor dem Hintergrund: DMA oder kein DMA.

Aufgabe 4

Aufgabe

Aufg. 194: Geräte und ihre Interrupts

Betrachte die folgenden Geräte:

- Maus
- Tastatur
- Netzwerkkarte

und beantworte die folgenden Fragen:

- Ist das Gerät DMA-fähig?
- Wann tritt bei der Arbeit mit dem Gerät jeweils ein Interrupt auf?
  (Nenne nur die wichtigsten Gründe.)
- Welche Schritte oder Tätigkeiten muss die Interruptbehandlungsroutine im zugehörigen Treiber ausführen?

Im Gegensatz zum bisherigen Standardbeispiel (*Datenübertragung mit der Festplatte*) können die in der vorangegangenen Aufgabe genannten Geräte eine besondere Eigenschaft aufweisen:

Hinweis

Diese Geräte können auch Daten zur weiteren Verarbeitung liefern, obwohl *niemand* danach gefragt hat.

(Unter "*niemand*" kann man hierbei z.B. verstehen: "*der Treiber*", "*das Betriebssystem*" oder "*ein Prozess*".)

Glücklicherweise bietet das Interruptkonzept auch genau für diesen Fall eine einfache Reaktionsmöglichkeit des Treibers auf die ungefragt eintreffenden Daten.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.4.4.5 Bereitstellen einer Schnittstelle zur Geräteverwaltung

Neben der Kommunikation mit dem Geräte-Controller, muss der Treiber auch mit der Geräteverwaltung des Betriebssystems kommunizieren. Er ist damit also ein Mittler zwischen beiden Welten. </p>

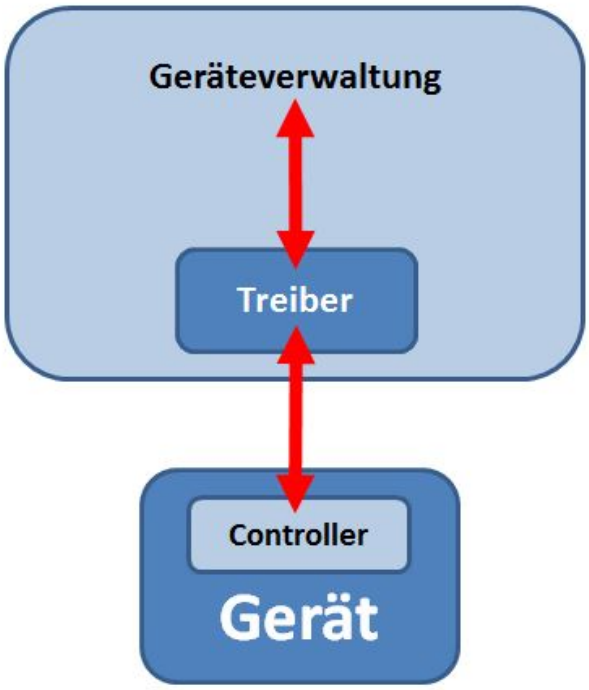


Abb. 70: Der Treiber als Mittler zwischen Geräteverwaltung und Geräte-Controller
CC-BY

Die Bereitstellung einer Schnittstelle zur Geräteverwaltung erfolgt üblicherweise durch die Definition unterschiedlicher Geräteklassen, mit jeweils einer eigenen Schnittstelle.

Wichtig

Eine dieser Geräteklassen (bzw. eine dieser Schnittstellen) muss der Treiber unterstützen.

Der Treiber muss also die von der Geräteverwaltung für die jeweilige Geräteklasse vorgesehenen Funktionen implementieren.

Weitere Informationen liefern die Abschnitte:

- Schnittstelle für blockorientierte Geräte
- Schnittstelle für zeichenorientierte Geräte

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.4.4.6 Pufferung von Daten

-to do -

Dieses Kapitel wird erst in einer späteren Version dieses Skripts zur Verfügung stehen.

Hinweis

3.4.4.7 Koordination nebenläufiger Zugriffe auf ein Gerät

-to do -

Hinweis

Dieses Kapitel wird erst in einer späteren Version dieses Skripts zur Verfügung stehen.

3.4.5 Geräteklassen

Im Zuge seiner geräteunabhängigen Teile der Implementierung definiert die Geräteverwaltung des Betriebssystems unterschiedliche Geräteklassen. Zu den wichtigsten beiden Klassen gehören die blockorientierten Geräte, sowie die zeichenorientierten Geräte. </p>

So geht es weiter:

Gliederung

3.4.5 Geräteklassen
3.4.5.1 Blockorientierte Geräte
3.4.5.2 Zeichenorientierte Geräte
3.4.5.3 Sonstige Geräte

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.4.5.1 Blockorientierte Geräte

Blockorientierte Geräte übertragen Daten jeweils in kompletten Blöcken. Dies gilt sowohl beim Lesen von diesem Gerät, als auch beim Schreiben auf selbiges. Typische Blockgrößen liegen zwischen 512 und 32 768 Byte. Jeder Datenblock ist direkt adressierbar.

Beispiele für blockorientierte Geräte sind:

- Festplatte
- CD- oder DVD-Laufwerk
- Bandlaufwerk

Beispiel

Eine Festplatte arbeite mit einer Blockgröße von 512 Byte. In Datenblock Nr. 723 soll das fünfte Byte geändert werden. Der folgende Ablauf ist dafür nötig:

- Lade Datenblock Nr. 723 von der Festplatte.
- Ändere das fünfte Byte wie gewünscht.
- Schreibe Datenblock Nr. 723 zurück auf die Festplatte.

Es können also immer nur **komplette** Datenblöcke (→ in diesem Beispiel 512 Byte) gelesen oder geschrieben werden.

Der betreffende Datenblock wird direkt adressiert (→ Nr. 723).

Schnittstelle für blockorientierte Geräte

Die Geräteverwaltung definiert üblicherweise eine Standardschnittstelle, welche die Treiber aller blockorientierten Geräte unterstützen müssen. Darin vorgesehen sind beispielsweise Funktionen für

- die Initialisierung des Geräts,
- das Lesen der Daten des adressierten Blocks vom Gerät,
- das Schreiben der Daten des adressierten Blocks zum Gerät, sowie
- die Behandlung eines vom Geräte-Controller ausgelösten Interrupts.

Diese Funktionen werden vom Gerätetreiber implementiert.

Zusammenarbeit von Treiber und Geräteverwaltung

Ist ein blockorientiertes Gerät am System angeschlossen, so wird sein Treiber (z.B. beim Systemstart) in den Hauptspeicher geladen. Der Treiber implementiert (u.a.) die Funktionen

- initDevice(),
- readBlock(),
- writeBlock(), sowie
- handleInterrupt().

Durch das Laden des Treibers in den Hauptspeicher steht ab diesem Moment fest, ab welcher Adresse im Hauptspeicher der ausführbare Code der genannten Funktionen beginnt. Dies gilt für jeden geladenen Treiber, also für jedes unterstützte Gerät.

Beispiel für Festplatte und Brenner

Beispiel

Zum Beispiel für die Festplatte

- ab Adresse 2 048 → initDevice()

- ab Adresse 2 560 → readBlock()
- ab Adresse 3 072 → writeBlock()
- ab Adresse 4 096 → handleInterrupt()

und für den DVD-Brenner

- ab Adresse 6 144 → initDevice()
- ab Adresse 7 168 → readBlock()
- ab Adresse 7 680 → writeBlock()
- ab Adresse 8 704 → handleInterrupt()

und so fort für jeden weiteren Treiber eines blockorientierten Geräts.

Die Geräteverwaltung wird über die Startadressen der implementierten Funktionen informiert und verwaltet diese entsprechend für alle unterstützten blockorientierten Geräte. Ab diesem Moment steht das Gerät für die Nutzung durch die Geräteverwaltung, durch das Betriebssystem und/oder einen Prozess zur Verfügung.

Ein Praxisbeispiel

Aus der Praxis

Eine etwas vereinfachte Darstellung für einen Prozess A, der Daten in eine (bereits geöffnete) Datei auf der Festplatte schreiben möchte:

- Der Prozess A verfügt über die zu schreibenden Daten.
- Der Prozess A informiert das Betriebssystem über seinen Schreibwunsch.
- Das Betriebssystem leitet den Schreibauftrag an die Geräteverwaltung weiter.
- Die Geräteverwaltung identifiziert die Festplatte als gewünschtes Gerät und sorgt dafür, dass Adresse 3 072 (siehe Beispiel oben) in den Programmzähler der CPU geladen wird.
- Die CPU beginnt mit der Ausführung des vom Treiber bereitgestellten Maschinencodes der writeBlock()-Funktion:
  - Der betreffende Block auf der Festplatte wird identifiziert (ggf. auch nacheinander mehrere Blöcke)
  - Die Daten des betreffenden Blocks werden von der Festplatte geladen.
  - Die geladenen Daten des Blocks werden entsprechend des Schreibwunsches des Prozesses A verändert.
  - Die veränderten Daten des Blocks werden zurück auf die Festplatte geschrieben.
  - Die Treiberfunktion ist nun erfolgreich abgearbeitet, es erfolgt ein Rücksprung zur Geräteverwaltung.

- Die Geräteverwaltung informiert das Betriebssystem über die erfolgreiche Abarbeitung des Schreibauftrags.
- Das Betriebssystem informiert den Prozess A über die erfolgreiche Ausführung seines Schreibwunsches.

Aufgabe 1

Aufg. 195: Systemaufruf im Praxisbeispiel

An welcher Stelle oder an welchen Stellen im Praxisbeispiel kommt es zu einem Systemaufruf?

Aufgabe 2

Aufg. 196: Prozesszustände im Praxisbeispiel

An welcher Stelle oder an welchen Stellen im Praxisbeispiel kommt es zu einem Zustandswechsel für den betreffenden Prozess A?

- Von welchem Zustand in welchen anderen Zustand erfolg der jeweilige Welchsel?
- Wer oder was führt den Zustandswechsel jeweils durch?
- Wo verwaltet das Betriebssystem die Information, in welchem Zustand sich der Prozess A gerade befindet?

Aufgabe 3

Aufg. 197: Interrupt im Praxisbeispiel

An welcher Stelle oder an welchen Stellen im Praxisbeispiel kommt es zu einem Interrupt?

Und welcher Prozess wird auf der CPU kurz vor bzw. kurz nach dem Interrupt ausgeführt?

Du kannst für deine Antwort ausgehen von:

- Außer Prozess A existiert noch mindestens ein weiterer Prozess B.
- Die Prozesse A und B sind voneinander unabhängig.

- Das Betriebssystem arbeitet mit Round Robin beim Scheduling.
- Es interessieren nur diejenigen Interrupts, an denen Prozess A "irgendwie" beteiligt ist. Alle sonstigen Interrupts, die für Prozess A keine Bedeutung haben, können im Rahmen dieser Aufgabe ignoriert werden.

Aufgabe 4

Aufgabe

Aufg. 198: Just a simple assembler instruction

Im Praxisbeispiel ist zu lesen: Die Geräteverwaltung "*sorgt dafür, dass Adresse 3 072 (...) in den Programmzähler der CPU geladen wird.*"

Welchen Assemblerbefehl nutzt die Geräteverwaltung für das Überschreiben des Wertes im Programmzähler der CPU?

Orientiere dich an der bereits bekannten *simple machine language* auf dieser Seite:

http://courses.cs.vt.edu/csonline/MachineArchitecture/Lessons/CPU/Lesson.html

Und was passiert dann auf der CPU direkt nachdem dieser Assemblerbefehl ausgeführt wurde? (In der nächsten Fetch-Phase.)

Aufgabe 5

Aufgabe

Aufg. 199: CPU-Registerinhalte sichern im Praxisbeispiel

An welcher Stelle oder an welchen Stellen im Praxisbeispiel müssen Inhalte von CPU-Registern gesichert werden?

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.4.5.2 Zeichenorientierte Geräte

Zeichenorientierte Geräte (engl. character device) erzeugen oder empfangen einen Datenstrom (engl. stream). Eine Folge von Zeichen (evtl. auch nur ein einzelnes Zeichen) wird hierbei zum Gerät oder vom Gerät übertragen. Diese Zeichen sind nicht adressierbar.

Beispiele für zeichenorientierte Geräte sind:

- Maus
- Tastatur
- Netzwerkkarte
- Geräte, die über die parallele Schnittstelle abgeschlossen sind
- Geräte, die über die serielle Schnittstelle abgeschlossen sind
- Geräte, die über eine USB-Schnittstelle abgeschlossen sind

Schnittstelle für zeichenorientierte Geräte

Die Geräteverwaltung definiert üblicherweise eine Standardschnittstelle, welche die Treiber aller zeichenorientierten Geräte unterstützen müssen. Darin vorgesehen sind beispielsweise Funktionen für

- die Initialisierung des Geräts,
- das Lesen eines Zeichenstroms vom Gerät, sowie
- das Schreiben eines Zeichenstroms zum Gerät, sowie
- die Behandlung eines vom Geräte-Controller ausgelösten Interrupts.

Diese Funktionen werden vom Gerätetreiber implementiert.

Zusammenarbeit von Treiber und Geräteverwaltung

Ist ein zeichenorientiertes Gerät am System angeschlossen, so wird sein Treiber (z.B. beim Systemstart) in den Hauptspeicher geladen. Der Treiber implementiert (u.a.) die Funktionen

- initDevice(),
- readChar(),
- writeChar(), sowie
- handleInterrupt().

Durch das Laden des Treibers in den Hauptspeicher steht ab diesem Moment fest, ab welcher Adresse im Hauptspeicher der ausführbare Code der genannten Funktionen beginnt. Dies gilt für jeden geladenen Treiber, also für jedes unterstützte Gerät.

Beispiel für Maus und Netzwerkkarte

Beispiel

Zum Beispiel für die Maus

- ab Adresse 10 240 → initDevice()
- ab Adresse 10 752 → readChar()
- ab Adresse 11 776 → writeChar()
- ab Adresse 12 888 → handleInterrupt()

und für die Netzwerkkarte

- ab Adresse 13 312 → initDevice()
- ab Adresse 14 336 → readChar()
- ab Adresse 15 360 → writeChar()
- ab Adresse 16 384 → handleInterrupt()

und so fort für jeden weiteren Treiber eines zeichenorientierten Geräts.

Die Geräteverwaltung wird über die Startadressen der implementierten Funktionen informiert und verwaltet diese entsprechend für alle unterstützten zeichenorientierten Geräte. Ab diesem Moment steht das Gerät für die Nutzung durch die Geräteverwaltung, durch das Betriebssystem und/oder einen Prozess zur Verfügung.

Aufgabe 1

Aufgabe

Aufg. 200: Ein Mausklick und die Folgen

Angenommen: Auf einem Betriebssystem wird ein Textverarbeitungsprozess ausgeführt. Der Benutzer gibt seinen Text ein, greift zur Maus und klickt einmal mit der linken Maustaste auf die Schaltfläche "Fett" in der Textverarbeitung.

Was passiert im Controller, im Treiber, in der Geräteverwaltung, im Betriebssystem und im Textverarbeitungsprozess als Reaktion auf den Mausklick?

Orientiere dich an dem Praxisbeispiel im Abschnitt Blockorientierte Geräte.

Der Anfang einer Lösung

Deine Lösung zu dieser Aufgabe könnte beginnen mit:

- Der Textverarbeitungsprozess wird auf der CPU ausgeführt.
- Plötzlich klickt der Benutzer einmal mit der linken Maustaste auf die Schaltfläche "Fett".
- . . . (← Was löst der Mausklick aus?)
- . . .

Aufgabe 2

Aufgabe

Aufg. 201: Hier schreibt die Maus

Wie wird die writeChar()-Funktion für die Maus wohl implementiert sein?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.4.5.3 Sonstige Geräte

-to do -

Dieses Kapitel wird erst in einer späteren Version dieses Skripts zur Verfügung stehen.

3.4.6 Memory-Mapped-I/O

-to do -

Dieses Kapitel wird erst in einer späteren Version dieses Skripts zur Verfügung stehen.

3.4.7 DMA - Direct Memory Access

-to do -

Dieses Kapitel wird erst in einer späteren Version dieses Skripts zur Verfügung stehen.

3.4.8 Windows-Treiber auf GitHub

Microsoft stellt das *Windows Driver Framework* auf GitHub bereit:
https://github.com/Microsoft/Windows-Driver-Frameworks </p>
Weiterhin gibt es auf GitHub eine Reihe von Beispielen für programmierte Windows-Treiber:
https://github.com/Microsoft/Windows-driver-samples

Durch diese Veröffentlichung erhofft sich Windows Impulse in der Entwicklergemeinde in Bezug auf die Programmierung besserer Treiber.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.5 Dateiverwaltung

Im Kapitel Speicherwerk wurde bereits darauf hingewiesen, dass der Arbeitsspeicher oder RAM (dies entspricht dem Speicherwerk aus der Von-Neumann-Architektur) zu den flüchtigen Speichern gehört, und Informationen nur solange speichern kann, wie er mit Spannung versorgt wird. Ein Ausschalten des Rechners führt hier also zwangsläufig zu Datenverlust. </p>

Was passiert mit den Daten im RAM, wenn ...

- der zugehörige Prozess terminiert?
- es einen Systemabsturz gibt und der Rechner neu gestartet werden muss?

Diese Fragen sind leicht aus der eigenen Erfahrung heraus zu beantworten.

Um Daten dauerhaft (persistent) zu speichern bedarf es eines sogenannten Dateisystems, dessen zentrale Aufgabe die Verwaltung von Dateien ist.

So geht es weiter:

3.5 Dateiverwaltung
3.5.1 Datei
3.5.2 Dateisystem
3.5.3 Aufgaben eines Dateisystems
3.5.4 Dateisystemkonzepte
3.5.5 Von Windows unterstützte Dateisysteme

Weiterführende Literatur

Weiterführende Literatur

Die hier verlinkte Online-Ausgabe eines Lehrtextes der Otto-Friedrich-Universität Bamberg liefert in Teil III "**Betriebssysteme**" (Kapitel 8 bis 11) detaillierte Informationen zu Betriebssystemen. Die Lektüre dieser Quelle sei unter Beachtung der geltenden Lizenz ausdrücklich empfohlen.

Autoren: Martin Eisenhardt, Andreas Henrich, Stefanie Sieber
'''Rechner- und Betriebssysteme, Kommunikationssysteme, Verteilte Systeme'''

http://www.uni-bamberg.de/fileadmin/uni/fakultaeten/wiai_lehrstuehle/medieninformatik/Dateien/Publikationen/2007/eisenhardt-rbkvs-1.0.pdf

Dieses Werk steht unter der Creative Commons BY-NC-ND-Lizenz
http://creativecommons.org/licenses/by-nc-nd/2.0/de/

Hinweis

Weiterführende Literatur

Mandl 2013 erläutert im kompletten Buch ausführlich das Thema Betriebssysteme. Es sei als Begleitlektüre ausdrücklich empfohlen.

Studierende sind oftmals berechtigt, eine PDF-Version dieses Buches ohne entstehende Kosten über ihre Hochschulen von Springerlink zu beziehen.

Alternative Webquelle zum Thema

Quelle im Web

Operating Systems: Mass-Storage Structure

http://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/10_MassStorage.html

Dr. John T. Bell
Department of Computer Science
University of Illinois, Chicago

Quelle im Web

Operating Systems: File-System Interface

http://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/11_FileSystemInterface.html

Dr. John T. Bell
Department of Computer Science
University of Illinois, Chicago

Quelle im Web

Operating Systems: File-System Implementation

http://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/12_FileSystemImplementation.html

Dr. John T. Bell
Department of Computer Science
University of Illinois, Chicago

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

3.5.1 Datei

Zunächst die Definition: </p>

Definition: Datei

Definition

Unter einer **Datei** versteht man einen Bestand an zusammengehörigen digitalen Daten, die dauerhaft auf einem geeigneten Speichermedium gespeichert sind.

Durch die hier vorgenommene Einschränkung auf *digitale* Daten entsteht eine klare Abgrenzung zu jeglicher Art von nicht-digitalen Daten, die beispielsweise *auf Papier* existieren.

Aufgabe 1

Aufgabe

Aufg. 202: Geeignete Speichermedien

Nenne mindestens fünf Beispiele für ein "geeignetes Speichermedium", welches eine oder mehrere Dateien aufnehmen kann.

CRUD-Operationen

Auf den genannten Speichermedien müssen Dateien angelegt, gelesen, verändert und gelöscht werden können. Man nennt dies auch die CRUD-Operationen, wobei *CRUD* als Abkürzung aus den Anfangsbuchstaben der folgenden englischen Begriffe entstanden ist:

- Create
- Read
- Update
- Delete

Die Organisation von Daten auf Speichermedien und die Bereitstellung der CRUD-Operationen geschieht mit Hilfe eines Dateisystems.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.5.2 Dateisystem

Zunächst die Definition:

Definition: Dateisystem

Definition

Unter einem **Dateisystem** versteht man den Teil eines Betriebssystems, der die geordnete Ablage und das leichte Wiederfinden von Dateien auf geeigneten Speichermedien ermöglicht, sowie die erforderlichen Zugriffsmöglichkeiten auf die verwalteten Dateien bereitstellt.

Zugriffsmöglichkeiten auf *einzelne Dateien* wurden oben bereits mit den CRUD-Operationen angesprochen. Ein Dateisystem stellt darüber hinaus noch Operationen zur Verfügung, welche sich üblicherweise auf *mehrere Dateien* auswirken:

- Verwalten von Verzeichnissen
- Verwalten von Datei- oder Verzeichnisattributen
- Kopieren von Dateien oder Verzeichnissen
- Verschieben von Dateien oder Verzeichnissen

Aufgabe 1

Aufgabe

Aufg. 203: CRUD für Verzeichnisse?

Lassen sich im Hinblick auf die *Verwaltung von Verzeichnissen* auch CRUD-Operationen angeben? Erläutere!

Viele verschiedene Dateisysteme

Im Laufe der Jahrzehnte wurden bemerkenswert viele verschiedene Dateisysteme entwickelt.

Quelle im Web

Eine umfassende Auflistung vieler in der Vergangenheit entwickelter Dateisysteme gibt ein Wikipedia-Artikel:

http://de.wikipedia.org/wiki/Liste_von_Dateisystemen

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.5.3 Aufgaben eines Dateisystems

- Verwaltung des freien/belegten Speicherplatzes

- Verwalten von Verzeichnissen
- strukturiertes Ablegen von Dateien
- Zugriffsrechte verwalten
- Attribute verwalten

Hinweis

Dieses Kapitel wird erst in einer späteren Version dieses Skripts zur Verfügung stehen.

3.5.4 Dateisystemkonzepte

-to do -

Hinweis

Dieses Kapitel wird erst in einer späteren Version dieses Skripts zur Verfügung stehen.

3.5.5 Von Windows unterstützte Dateisysteme

Russinovich et.al. 2012b listen die von Windows standardmäßig unterstützten Dateisysteme auf:

- CDFS - CD-ROM File System
- UDF - Universal Disk Format
- FAT12 - File Allocation Table 12
- FAT16 - File Allocation Table 16
- FAT32 - File Allocation Table 32
- exFAT - Extended File Allocation Table
- NTFS - New Technology File System

Eine Unterstützung weiterer Dateisysteme ist nur durch die Installation zusätzlicher Treiber möglich.

Definition: Datenträger

Definition

Unter einem **Datenträger** versteht man ein physisches Speichermedium, welches in der Lage ist, digitale Daten dauerhaft zu speichern.

Unter einen *Datenträger* fallen somit Geräte oder Medien wie zum Beispiel:

- Festplatte (herkömmlich oder auch SSD)
- CD
- DVD
- USB-Stick

Definition: Disk

Definition

Unter einer **„Disk"** *(im Sinne des Betriebssystems Microsoft Windows)* versteht man einen Datenträger.

Definition: Sektor

Definition

Unter einem **Sektor** versteht man einen Hardware-adressierbaren Block auf einer Disk.

Beispielsweise ist das Speichermedium *Festplatte* aufgeteilt in eine herstellungsbedingte Anzahl an Sektoren. Diese Sektoren haben üblicherweise eine Größe von 512 Byte, bei neueren Festplattenmodellen auch 4 096 Byte. Multipliziert mit der *Anzahl der Sektoren* ergibt sich so die Gesamt-Speicherkapazität einer Festplatte.

Das Speichermedium *CD (Compact Disk)* hat eine übliche Sektorgröße von 2 048 Byte.

Definition: Partition

Definition

Unter einer **Partition** versteht man eine Folge von zusammenhängenden Sektoren.

Definition: Volume

Definition

Unter einem **Volume** *(im Sinne des Betriebssystems Microsoft Windows)* versteht man ein dem Nutzer des Betriebssystems bereitgestellten Speicherbereich für Dateien und Verzeichnisse.

Damit ist ein *Volume* nicht automatisch gleichzusetzen mit einer *Partition*.

Üblicherweise weist das Betriebssystem Windows einem Volume einen Laufwerksbuchstaben (z.B. C: oder D:) zu.

Beispiel

Ein Volume kann identisch mit einer Partition sein.

In diesem Fall wird aus einer Partition auf einer Festplatte einfach ein Volume, welches dann für den Nutzer einen Laufwerksbuchstaben zugewiesen bekommt.

Siehe *Volume C:* auf *Datenträger 0* in der folgenden Abbildung.

Beispiel

Ein Volume kann aber auch aus mehreren Partitionen bestehen.

Beispiel: Zwei Festplatten mit jeweils einer (gleichgroßen) Partition können unter Windows zu einem *gespiegelten Volume* zusammengefasst werden. Man spricht in diesm Fall von einem RAID , genauer einem Software-RAID mit Level 1 (Mirroring).

Siehe *Volume D:* auf *Datenträger 1* und *Datenträger 2* in der folgenden Abbildung.

Kurz zur Erläuterung:

Bei einem gespiegelten Volume dieser Art (RAID 1) werden Dateien und Verzeichnisse gleichzeitig auf zwei unterschiedlichen Datenträgern abgespeichert. Sollte einer der beiden Datenträger einen technischen Defekt erleiden, so sind alle Daten noch auf dem zweiten Datenträger vorhanden. Die Datensicherheit wird also erhöht.

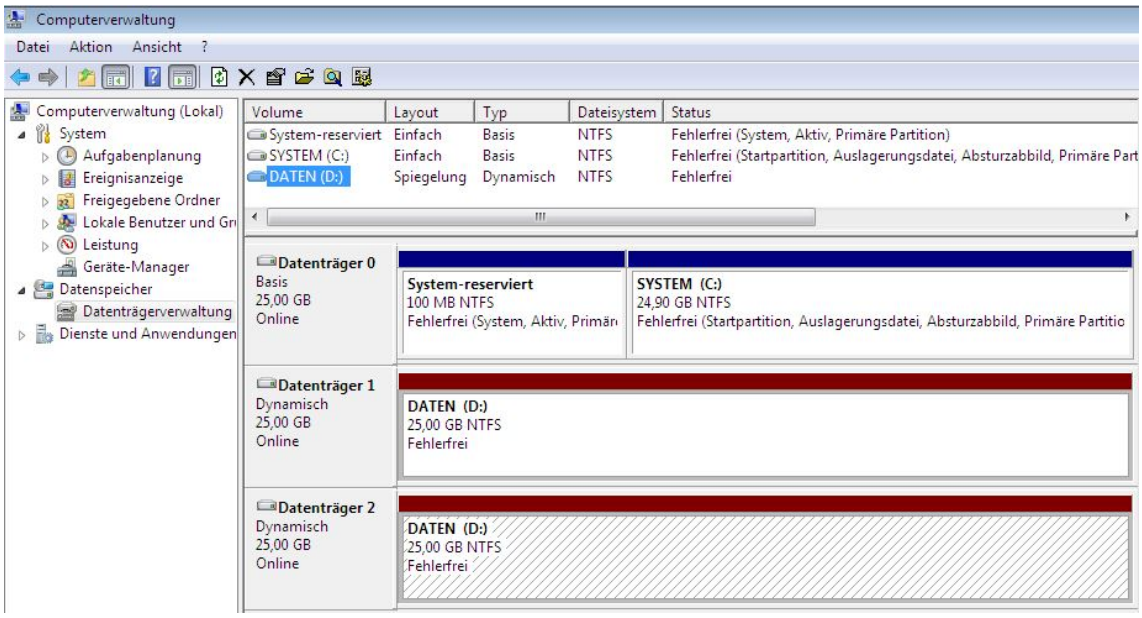


Abb. 71: Zwei Volumes auf drei Partitionen unter Windows 7
CC-BY

Definition: Cluster

Definition

Unter einem **Cluster** versteht man einen adressierbaren Block in einem Dateisystem, der gleichzeitig eine logische Zusammenfassung von Sektoren einer Disk darstellt.

Ein *Cluster* besteht demnach aus einem oder mehreren Sektoren. Seine Größe ist stets ein ganzzahliges Vielfaches der Sektorgröße.

Typische Clustergrößen liegen zwischen 512 Byte und 32 768 Byte.

Da *Cluster* vom Dateisystem verwaltet werden, liegen alle Sektoren eines Clusters stets auf demselben Volume.

Hinweis

Erkenne den Unterschied: Sektor vs. Cluster!

Erkennbar an diesen Definitionen ist eine Unterscheidung zwischen den Begriffen Sektor und Cluster. Beide Varianten sind eindeutig adressierbar, jedoch hängt die Adresse eines Sektors von der Hardware (z.B. der Festplatte) ab, während die Adresse eines Clusters von der Software (Betriebssystem) abhängt.

Definition: Datei

Definition

Unter einer **Datei** *(im Sinne des Betriebssystems Microsoft Windows)* versteht man eine Zusammenfassung von Clustern.

Diese Definition impliziert, dass eine Datei stets auf genau einem Volume gespeichert ist.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.5.5.1 FAT - File Allocation Table

Unter der **File Allocation Table** (kurz: **FAT**, auf deutsch: Dateizuordnungstabelle) versteht man eine Tabelle, über die einerseits eine Zuordnung von Clustern zu Dateien verwaltet wird, andererseits gibt die FAT Aufschluss über freie und belegte Cluster eines Dateisystems.

Im Zuge der technologischen Weiterentwicklung wurden seit Mitte der 1970er-Jahre verschiedene FAT-Versionen entwickelt. Unter anderem waren dies:

- FAT12 - File Allocation Table 12
- FAT16 - File Allocation Table 16
- FAT32 - File Allocation Table 32
- exFAT - Extended File Allocation Table

Diese Versionen unterscheiden sich beispielsweise in der maximal unterstützten Dateigröße und Dateisystemgröße.

Eine einfache FAT

Die folgende Abbildung zeigt ein Beispiel für eine einfache FAT, bestehend nur aus einer einzelnen Spalte. Die einzelnen Zeilen dieser Spalte sind durch die laufende Cluster-Nummer nummeriert.

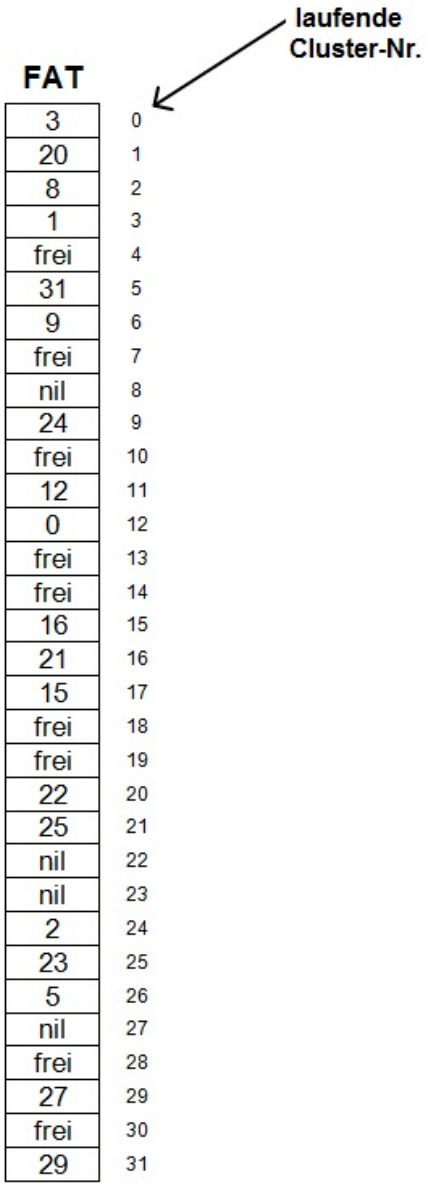


Abb. 72: Beispiel einer FAT
CC-BY

Freie Cluster

Aus dem FAT-Beispiel der vorangegangenen Abbildung ist erkennbar, dass die Cluster mit den folgenden Nummern noch frei sind:

4, 7, 10, 13, 14, 18, 19, 28 und 30.

Sobald eine weitere Datei auf dem durch diese FAT verwaltete Dateisystem gespeichert werden soll, kann die Datei - entsprechend ihrer Größe - auf diese freien Cluster verteilt werden.

Datei zusammensetzen

Beim Zugriff auf eine Datei, welche auf einem Dateisystem gespeichert ist, das durch die FAT aus der vorangegangenen Abbildung verwaltet wird, muss das Betriebssystem die zu dieser Datei gehörigen Cluster ermitteln. Dies ist beispielsweise wie folgt möglich:

- Die Datei *computer.txt* beginnt in Cluster 6. (Diese Information bezieht das Betriebssystem aus dem zu dieser Datei gehörigen Verzeichniseintrag. Der Verzeichniseintrag ist nicht Bestandteil der FAT!)
- Das Betriebssystem sucht in der FAT die Zeile 6 (→ laufende Cluster-Nr.).
- Hier gibt es einen Verweis auf Cluster-Nr. 9
- Zeile 9 verweist auf Cluster-Nr. 24.
- Zeile 24 verweist auf Cluster-Nr. 2.
- Zeile 2 verweist auf Cluster-Nr. 8.
- Zeile 8 verweist auf "nil", was bedeutet, dass kein weiterer Cluster folgt.

Somit kann der Inhalt der Datei *computer.txt* dieses Beispiels durch die Inhalte der folgenden Cluster zusammengesetzt werden:
6, 9, 24, 2, 8

Beim Zusammensetzen des Dateiinhalts ist die Reihenfolge der Cluster unbedingt zu beachten.

Die folgende Abbildung verdeutlicht die "Clusterspur" der Datei *computer.txt* in der Beispiel-FAT:

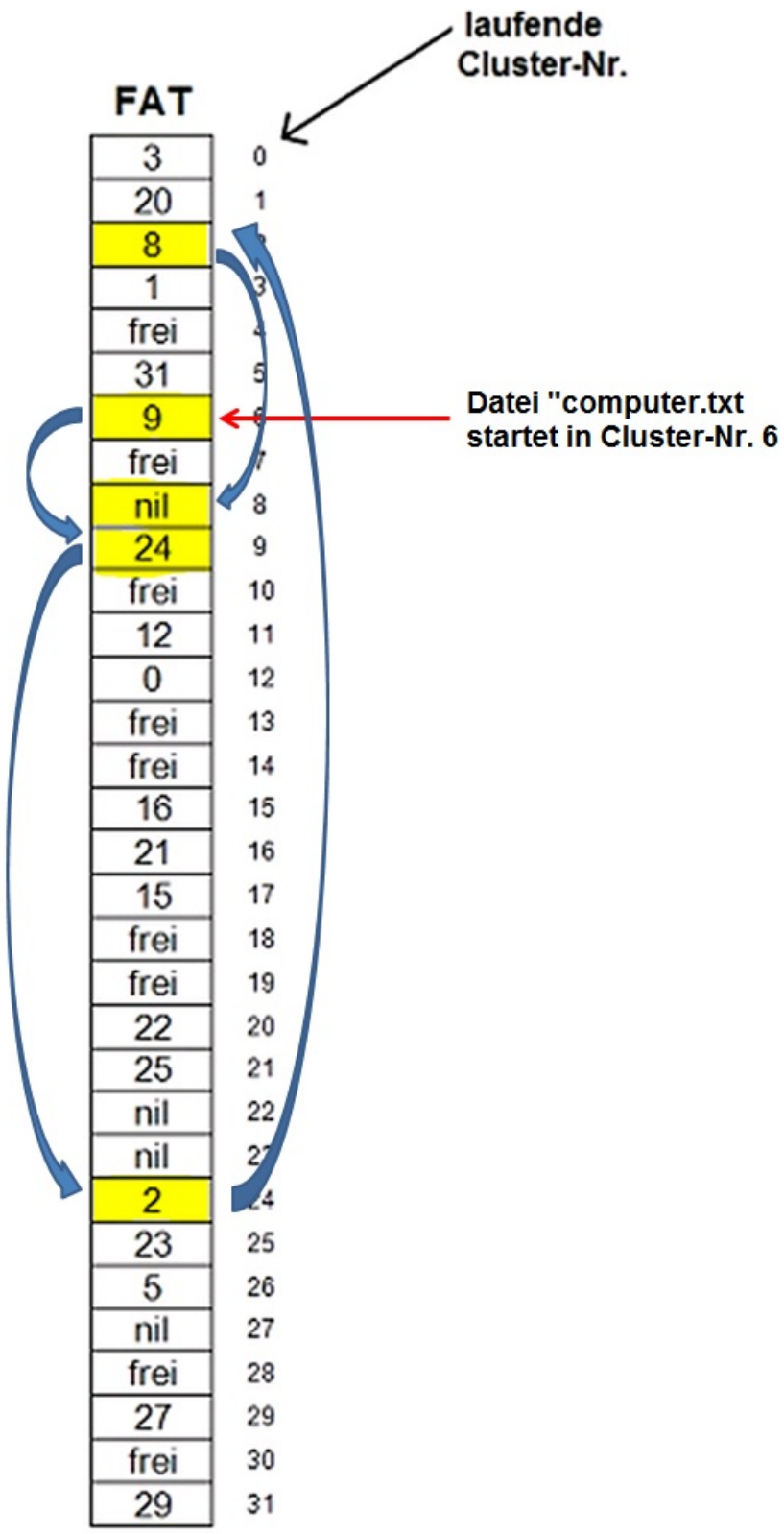


Abb. 73: Clusterspur einer Datei in der FAT
CC-BY

Aufgabe 1

Aufgabe

Aufg. 204: Setze die Datei zusammen!

Betrachte die oben gegebene Beispiel-FAT. Die Datei *xyz.jpg* beginne in Cluster-Nr. 11.

Aus den Inhalten welcher Cluster setzt sich der Dateiinhalt insgesamt zusammen? Achte auf die richtige Reihenfolge der Cluster-Nummern!

Aufgabe 2

Aufgabe

Aufg. 205: Lösche die Datei!

Betrachte die oben gegebene Beispiel-FAT. Die Datei *xyz.jpg* beginne in Cluster-Nr. 11 und soll gelöscht werden.

Wie ändern sich die Einträge in der FAT durch das Löschen der Datei?

Aufgabe 3

Aufgabe

Aufg. 206: Speichere eine neue Datei!

Betrachte die oben gegebene Beispiel-FAT. Eine neue Datei *abc.mp3* soll gespeichert werden. Aufgrund der Dateigröße werden insgesamt fünf Cluster zum Speichern benötigt.

- In welchem Cluster beginnt die Datei? (Wähle eine geeignete Cluster-Nr.!)
- In welchen weiteren Clustern legst du die Datei ab?
- Wie ändern sich also die Einträge in der FAT?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

3.5.5.2 NTFS - New Technology File System

NTFSInfo

Das kleine Tool ntfsinfo.exe aus der Sysinternals Suite liefert eine Reihe von Informationen zu den verwendeten Größen für Volume, Sektor, Cluster, etc. bei Verwendung des NTFS-Dateisystems unter Windows.

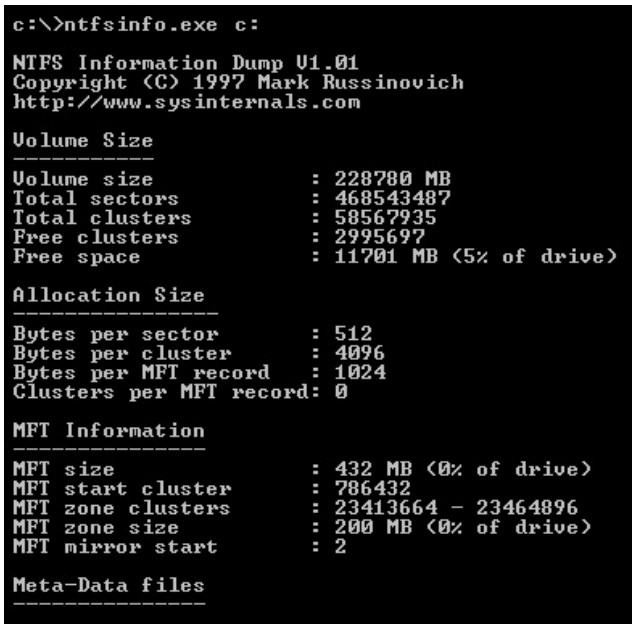


Abb. 74: Ein Aufruf von 'ntfsinfo.exe c:' unter Windows 7
CC-BY

MFT - Master File Table

Im **Master File Table** (kurz MFT) speichert das NTFS-Dateisystem Informationen zu allen Dateien und Verzeichnissen eines mit NTFS formatierten Volumes.

4 Aufgaben zur Prüfungsvorbereitung

In den folgenden Abschnitten finden sich Aufgaben mit Bezug zu allen Kapiteln dieses Moduls. Sie sollen zur Wiederholung des Lernstoffs dienen und somit zu einer optimalen Prüfungsvorbereitung beitragen. </p>

So geht es weiter:

Gliederung

4 Aufgaben zur Prüfungsvorbereitung
4.1 Aufgaben zum Kapitel Vom Anwender zur digitalen Schaltung
4.2 Aufgaben zum Kapitel Prozessoren und ihre Befehle
4.3 Aufgaben zum Kapitel Weitere Komponenten der Computerarchitektur
4.4 Aufgaben zum Kapitel Einführung Betriebssysteme
4.5 Aufgaben zum Kapitel Prozessverwaltung
4.6 Aufgaben zum Kapitel Speicherverwaltung
4.7 Aufgaben zum Kapitel Geräteverwaltung
4.8 Aufgaben zum Kapitel Dateiverwaltung

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

4.1 Aufgaben zum Kapitel Vom Anwender zur digitalen Schaltung

Aufgabe 1

Aufgabe

Aufg. 207: Von-Neumann-Architektur

Beschreibe die einzelnen Bestandteile eines von-Neumann-Rechners und ihre Aufgaben.

Fertige zusätzlich eine Skizze an, die den Aufbau der Von-Neumann-Architektur beschreibt.

Aufgabe 2

Aufgabe

Aufg. 208: Von-Neumann-Flaschenhals

Erläutere: Wodurch kommt es zum Von-Neumann-Flaschenhals?

Aufgabe 3

Aufgabe

Aufg. 209: Von-Neumann-Zyklus

Zähle (ohne nähere Erläuterung) die fünf Phasen des Von-Neumann-Zyklus in der korrekten zeitlichen Reihenfolge auf!

Aufgabe 4

Aufgabe

Aufg. 210: 8-Bit-Register

Skizziere ein 8-Bit-Register mit seinen Eingangs- sowie Ausgangsleitungen. Benenne die jeweiligen Leitungen!

(Auf das Zeichnen einzelner Gatter oder Flip-Flops kann verzichtet werden.)

Aufgabe 5

Aufgabe

Aufg. 211: Bitfolge in ein Register schreiben

In ein 8-Bit-Register soll die Bitfolge „0 1 0 1 0 1 0 1“ geschrieben werden. Welche Werte müssen dazu an den jeweiligen Eingangsleitungen des Registers anliegen?

Aufgabe 6

Aufgabe

Aufg. 212: Rund um diese Komponente

Beschreibe die Funktionsweise der folgenden Komponente. Wozu dient sie? Benenne die Bestandteile der Komponente und gehe auch auf die Signale an den Eingangsleitungen ein.

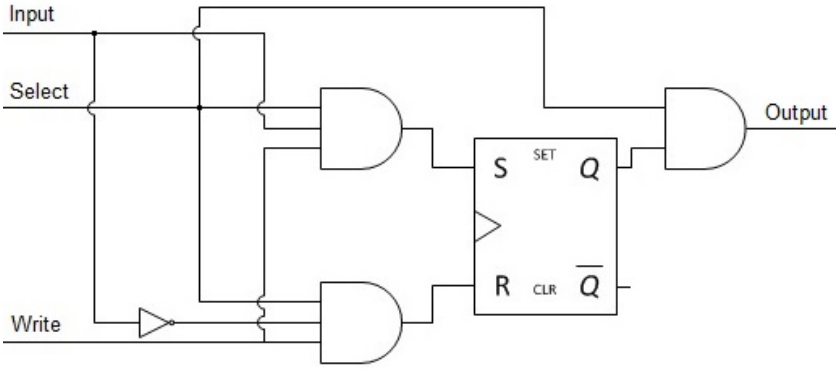


Abb. 75: Um diese Komponente geht es in dieser Aufgabe
CC-BY

Aufgabe 7

Aufgabe

Aufg. 213: Flip-Flop

Die folgende Abbildung zeigt ein Flip-Flop, welches genau ein Bit speichern kann.

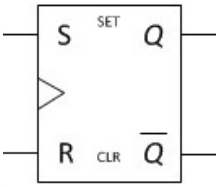


Abb. 76: Flip-Flop
CC-BY

- Wofür steht die Abkürzung „S“ im gezeigten Flip-Flop?
- Wofür steht die Abkürzung „R“ im gezeigten Flip-Flop?
- Was bedeutet der Querstrich über dem Q, also dem unteren Ausgang des gezeigten Flip-Flops?
- Was passiert bei dem gezeigten Flip-Flop, wenn an beiden Eingangsleitungen (S und R) jeweils der Wert "1" anliegt?

Aufgabe 8

Aufgabe

Aufg. 214: Ermittle die Werte

Betrachte diese elektrische Schaltung:

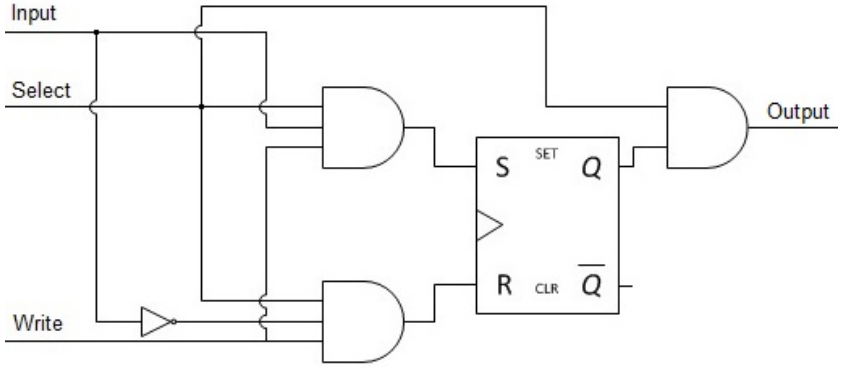


Abb. 77: Um diese Komponente geht es in dieser Aufgabe
CC-BY

Vervollständige die folgende Tabelle, indem du ausgehend von den Werten an den Eingangsleitungen und dem aktuellen Inhalt des Flip-Flops jeweils den zu erwartenden neue Wert im Flip-Flop sowie den zu erwartenden Wert an der Output-Leitung ermittelst und einträgst:

Input	Select	Write	Flip-Flop	Flip-Flop (neu)	Output
1	1	1	0		
0	1	1	0		
0	1	0	1		
0	0	0	1		
0	1	0	0		

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

4.2 Aufgaben zum Kapitel Prozessoren und ihre Befehle

Aufgabe 1

Aufgabe

Aufg. 215: ADD im Einadressformat

Der Assembler-Befehl ADD benötigt für seine Ausführung zwei Werte, welche addiert werden sollen. Wird dieser Befehl im Einadressformat angegeben, so wird nur ein Operand spezifiziert. Über diesen einen Operanden bekommt der ADD-Befehl also einen der beiden zu addierenden Werte.

Woher bekommt der ADD-Befehl (im Einadressformat) den zweiten zu addierenden Wert?

Wohin schreibt der ADD-Befehl (im Einadressformat) sein berechnetes Ergebnis?

Aufgabe 2

Aufgabe

Aufg. 216: Dreiadressformat

Werden Assembler-Befehle im Dreiadressformat angegeben, so definiert die Reihenfolge der anzugebenden Operanden ihre jeweilige Bedeutung.

Erläutere diese Bedeutung anhand eines selbstgewählten Beispiels!

Aufgabe 3

Aufgabe

Aufg. 217: Entwicklung eines Befehlssatzes

Ein kompletter Befehl (Opcode und Operand) vom Typ NOOP, LOAD, STORE, ADD, SUB, EQUAL, JUMP und HALT besteht derzeit (wie im Video erläutert) aus insgesamt 16 Bit, wobei die ersten sechs Bit als Reserve nicht genutzt werden.

Entwickle einen Befehlssatz,

- der maximal 256 Befehle umfassen kann, aber zunächst nur die bereits bekannten Befehle NOOP, LOAD, STORE, ADD, SUB, EQUAL, JUMP und HALT definiert,
- der maximal 64 Steueroptionen unterscheiden kann,
  - bislang waren es nur zwei Steueroptionen, siehe <Num> im Video ,
  - die hier angegebene Anzahl ist "*insgesamt für den Befehl*" gemeint, und **nicht** "*pro Operand*",
- der je sechs Bit für die Adressierung von <Ergebnis>, <Operand1> und <Operand2> vorsieht (Dreiadressformat),
- dessen Befehlslänge ein Vielfaches von acht Bit sein soll.

Aus wie vielen Bit besteht damit ein kompletter Befehl (Opcode und Operand) dieses neuen Befehlssatzes mindestens?

Gebe das SUM-Program aus dem Video damit an! Herauskommen sollte eine Übersicht wie diese, angepasst an den neuen Befehlssatz:

Reserve	Befehl	Num	Operand	Assembler
000000	001	1	000010	LOAD #2
000000	010	0	001101	STORE 13
000000	001	1	000101	LOAD #5
000000	010	0	001110	STORE 14
000000	001	0	001101	LOAD 13
000000	011	0	001110	ADD 14
000000	010	0	001111	STORE 15
000000	111	0	000000	HALT

Befehlscodierung

000	→	NOOP
001	→	LOAD
010	→	STORE
011	→	ADD
100	→	SUB
101	→	EQUAL
110	→	JUMP
111	→	HALT

Num (Nummer-Bit)

0	→	Operand ist Speicheradresse
1	→	Operand ist Zahl (#)

Abb. 78: Abbildung zu Aufgabe 1
CC-BY

Erwartet wird also, dass das SUM-Programm sowohl in Maschinensprache (Einsen und Nullen!), als auch in Assembler angegeben wird. Der Additionsbefehl mit drei Operanden könnte in Assembler so aussehen:

ADD ACC, ACC, 14

"ACC" bezeichnet das Register Akkumulator, die Bedeutung des Befehls ist damit: *Addiere den im Akkumulutar gespeicherten Wert und den in Speicherzelle 14 gespeicherten Wert, und lege das Ergebnis im Akkumulator ab*.

Wie löst du das Problem, dass in der binären Codierung (Maschinensprache) unterschieden werden muss, ob ein Operand ein Register, eine Speicherzelle oder gar einen konkreten Wert bezeichnet? (Stichwort: # im Video . Falls du hier Hilfe benötigst, so diskutiere Lösungsansätze in deiner Lerngruppe!)

Aufgabe 4

Aufgabe

Aufg. 218: Adressierungen des Hauptspeichers

Erläutere für den Zugriff auf den Hauptspeicher (Speicherwerk) die Unterschiede zwischen der effektiven Adresse und der indizierten Adressierung mit Verschiebung.

Aufgabe 5

Aufgabe

Aufg. 219: Adressierungsarten

Welche Bedeutungen haben die folgenden Befehle:

- **ADD** R1, ACC, 8
- **SUB** ACC, 10, (R2)
- **JUMP** (R3)
- **ADD** ACC, (ACC), #5

Aufgabe 6

Aufgabe

Aufg. 220: Prozess

Definiere den Begriff "Prozess".

Aufgabe 7

Aufgabe

Aufg. 221: Von-Neumann vs. Harvard

Skizziere die Von-Neumann-Architektur sowie die Harvard-Architektur und erläutere anhand einiger Worte den grundlegenden Unterschied beider Architekturen!

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

4.3 Aufgaben zum Kapitel Weitere Komponenten der Computerarchitektur

Aufgabe 1

Aufgabe

Aufg. 222: Basis- und Limit-Register

Erläutere die Bedeutung von Basis- und Limitregister.

Wie ist der im Basis- bzw. Limit-Register gespeicherte Wert zu interpretieren?

Welche Aufgabe/Aufgaben erfüllt das jeweilige Register?

Aufgabe 2

Aufgabe

Aufg. 223: Interrupt-Gründe

Nenne drei Gründe für die Auslösung eines Interrupts.

Aufgabe 3

Aufgabe

Aufg. 224: DMA

Wofür steht die Abkürzung DMA (im Kontext der Computerarchitektur)?

Aufgabe 4

Aufgabe

Aufg. 225: Infos für den DMA-Controller

Mit welchen Informationen muss ein DMA-Controller versorgt werden, damit er seine Tätigkeit aufnehmen, und die CPU entlasten kann?

Aufgabe 5

Aufgabe

Aufg. 226: DMA und Interrupts

Erläutere den grundlegenden Unterschied in Bezug auf Interrupts, der bei einem Datentransfer mit oder ohne Beteiligung eines DMA-Controllers gegeben ist.

Aufgabe 6

Aufgabe

Aufg. 227: MMU

Wofür steht die Abkürzung MMU (im Kontext der Computerarchitektur)?

Aufgabe 7

Aufgabe

Aufg. 228: Aufgabe der MMU

Erläutere die grundlegende Aufgabe der MMU.

Aufgabe 8

Aufgabe

Aufg. 229: Virtuelle Größen

Betrachte einen Computer mt 2 GiB physikalischen Speicher (RAM). Jeder gestartete Prozess besitze 4 GiB virtuellen Speicher. Die typische Größe einer einzelnen virtuellen Seite bei Verwendung der virtuellen Speicherverwaltung betrage 4 KiB.

- Wie groß ist dann ein einzelner Seitenrahmen des physikalischen Speichers?
- In wieviele Seiten ist der gesamte virtuelle Speicher eines Prozesses unterteilt?
- Wieviele Seitenrahmen gibt es insgesamt?
- Wieviele Speicherzellen zu je 8 Bit (= 1 Byte) besitzt ein einzelner Seitenrahmen?
- Wieviele Speicherzellen zu je 8 Bit (= 1 Byte) besitzt eine einzelne Seite?

Gib jeweils kurz den Rechenweg mit an!

Aufgabe 9

Aufgabe

Aufg. 230: North- und Southbridge

Welcher Unterschied besteht bei Northbridge und Southbridge im Hinblick auf Geschwindigkeit bei der Verbindung mit Rechnerkomponenten?

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

4.4 Aufgaben zum Kapitel Einführung Betriebssysteme

Aufgabe 1

Aufg. 231: Betriebsmittel

Nenne (ohne nähere Erläuterung) fünf Betriebsmittel!

Aufgabe 2

Aufg. 232: Kernel- und User-Mode

Erläutere den/die Unterschied(e) zwischen Kernel-Mode und User-Mode!

Aufgabe 3

Aufg. 233: Sinn von Kernel- und User-Mode

Warum macht eine Unterscheidung von Kernel- und User-Mode aus Sicht des Betriebssystems Sinn? Nenne zwei Gründe.

Aufgabe 4

Aufg. 234: Systemaufruf

Erläutere was passiert, wenn ein momentan auf der CPU ausgeführter Anwendungs-Prozess (z.B. eine Textverarbeitung) einen Systemaufruf tätigt.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

4.5 Aufgaben zum Kapitel Prozessverwaltung

Das Kapitel zur Prozessverwaltung ist sehr umfangreich, woraus auch viele Übungsaufgaben resultieren. Diese werden in thematisch unterteilten Unterkapiteln bereitgestellt.

So geht es weiter:

Gliederung

4.5 Aufgaben zum Kapitel Prozessverwaltung
4.5.1 Aufgaben zu Prozesse und Threads
4.5.2 Aufgaben zum Scheduling
4.5.3 Aufgaben zur Synchronisation
4.5.4 Aufgaben zu Deadlocks

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

4.5.1 Aufgaben zu Prozesse und Threads

Aufgabe 1

Aufgabe

Aufg. 235: Prozesszustände

Skizziere in einem Schaubild die drei grundlegenden Prozesszustände und die jeweils möglichen Zustandsübergänge. Gebe für alle Zustandsübergänge jeweils den Grund des Zustandswechsels an.

Aufgabe 2

Aufgabe

Aufg. 236: Von 'rechnend' nach 'bereit'

Bei welcher Art von Schedulingstrategien ist auch ein direkter Übergang vom Prozesszustand "rechnend" zum Prozesszustand "bereit" möglich?

Aufgabe 3

Aufgabe

Aufg. 237: Rund um die Prozessbegriffe

Grenze die Begriffe Programm, Prozess, Prozesskontext und Prozesszustand gegeneinander ab und erkläre diese.

Aufgabe 4

Aufg. 238: Kontextwechsel

Nenne zwei Dinge, die bei einem Kontextwechsel auf der CPU erledigt werden müssen.

Aufgabe 5

Aufg. 239: Prozesserzeugung unter Unix/Linux

Welcher grundlegende Befehl steht unter Unix/Linux für die Erzeugung eines neuen Prozesses zur Verfügung? (Nenne nur den Befehl.)

Durch die Ausführung dieses Befehls entsteht aus dem Eltern-Prozess ein Kind-Prozess. An welcher Stelle beginnt anschließend die Abarbeitung des Kind-Prozesses (sobald er die CPU zugeteilt bekommt)?

Aufgabe 6

Aufg. 240: Prozesskontrollblock

Nenne fünf Informationen, die vom Betriebssystem für einen Prozess in seinem zugehörigen Prozesskontrollblock verwaltet werden.

Aufgabe 7

Aufg. 241: Alle Prozesskontrollblöcke

Jeder Prozess besitzt seinen eigenen Prozesskontrollblock. Wie nennt man die Datenstruktur, in der alle Prozesskontrollblöcke zusammengefasst werden?

Aufgabe 8

Aufg. 242: Prozess und Thread

Erläutere Unterschiede und Gemeinsamkeiten zwischen Prozessen und Threads.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

4.5.2 Aufgaben zum Scheduling

Aufgabe 1

Aufg. 243: Scheduler und Dispatcher

Erläutere den Unterschied zwischen einem Scheduler und einem Dispatcher.

Aufgabe 2

Aufg. 244: Scheduling-Ziele

Nenne drei Ziele, die ein Scheduling-Algorithmus verfolgen sollte.

Erläutere, inwieweit ein gleichzeitiges Erreichen der genannten Ziele realistisch ist.

Aufgabe 3

Aufg. 245: Scheduling

Wofür stehen die Abkürzungen FCFS, SJF, SRTN, RR, PS im Kontext des Scheduling?

Aufgabe 4

Aufg. 246: SJF und FCFS

Diskutieren Sie die Vor- und Nachteile der Scheduling-Strategien SJF und FCFS.

Aufgabe 5

Aufg. 247: RR und FCFS

Erläutern Sie in Bezug auf Scheduling, warum das Round-Robin-Verfahren nicht die Nachteile des FCFS-Verfahrens hat.

Aufgabe 6

Aufg. 248: (Non-) preemptive

Erläutere den Unterschied zwischen "preemptive" und "non-preemptive" in Bezug auf Scheduling-Strategien.

Aufgabe 7

Aufg. 249: Preemptive oder non-preemptive

Betrachte die Scheduling-Verfahren FCFS, SJF, RR und PS.
Welche dieser Verfahren sind preemptive, welche sind non-preemptive?

Aufgabe 8

Aufg. 250: Scheduling für P1 bis P5

Gegeben seien fünf Prozesse P1, P2, P3, P4 und P5 mit den Gesamtrechenzeiten 2, 3, 6, 10 und 19 Zeiteinheiten, die quasi gleichzeitig zum Zeitpunkt 0 am System eintreffen. Gehen Sie davon aus, dass die Reihenfolge des Eintreffens (bei kaum wahrnehmbaren zeitlichen Unterschieden) P4, P1, P5, P3, P2 lautet. Es gibt keine Unterscheidung unterschiedlicher Prioritäten.

Skizzieren Sie für diese Situation wann welcher Prozess bearbeitet wird für die Schedulingstrategien FCFS, SJF, SRTN und RR. Gehen Sie für die preemptive(n) Strategie(n) von einer Zeitscheibe der Länge 2 Zeiteinheiten aus.

(Hinweis: Diese Aufgabe ist so gemeint, dass zum Zeitpunkt 0 bereits alle Prozesse im Zustand "bereit" auf eine Zuteilung der CPU warten.)

Aufgabe 9

Aufg. 251: Scheduling mit Priorität

Betrachte die Prozesse A bis E, welche mit unterschiedlichen Prioritäten versehen sind. Die Rechenzeit ist schon im Voraus bekannt, siehe folgende Tabelle.

Welche Ausführungsreihenfolge ergibt sich beim Priority-Scheduling, wenn zusätzlich mit einem Zeitquantum von einer Zeiteinheit (ZE) gearbeitet wird?

Prozess	**A**	**B**	**C**	**D**	**E**
Rechenzeit (ZE)	12	7	4	8	10
Priorität	Mittel	Niedrig	Hoch	Hoch	Mittel

Hinweis: Alle Prozesse sind zum Startzeitpunkt bereit und können sofort die CPU übernehmen. Die Priorität der Prozesse ändert sich nicht, bei gleicher Priorität ist die Round-Robin-Strategie anzuwenden.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

4.5.3 Aufgaben zur Synchronisation

Aufgabe 1

Aufg. 252: Kritischer Abschnitt

Was versteht man unter einem "kritischen Abschnitt"?

Aufgabe 2

Aufg. 253: Aktives Warten mit while

Um bei mehreren Prozessen mit gemeinsam genutzten Betriebsmitteln kritische Abschnitte so zu schützen, dass immer nur maximal ein Prozess zur Zeit diesen kritischen Abschnitt ausführen darf, soll das aktive Warten eingesetzt werden. Zur Realisierung des aktiven Wartens kann eine while-Schleife genutzt werden.

Gebe den für das aktive Warten nötigen Programmcode an, der zu Beginn bzw. am Ende eines kritischen Abschnitts ausgeführt werden muss.

Aufgabe 3

Aufg. 254: Nachteil des aktiven Wartens

Erläutere warum das aktive Warten einen zeitlichen Nachteil mit sich bringt, und unter welcher Bedingung dieser Nachteil in Kauf genommen werden kann.

Aufgabe 4

Aufg. 255: TSL

Wofür steht die Abkürzung TSL (im Kontext der Prozess-Synchronisation)?

Aufgabe 5

Aufg. 256: Aufgabe des TSL-Befehls

Erläutere welche Aufgabe der TSL-Befehl bei der Ausführung auf einer CPU erfüllt.

Aufgabe 6

Aufg. 257: Problemlöser TSL

Welches Problem löst der TSL-Befehl in Zusammenhang mit aktivem Warten?

Aufgabe 7

Aufg. 258: Semaphor

Ein Semaphor ist eine Datenstruktur, welche aus zwei "Dingen" sowie zwei darauf definierten Funktionen besteht.

- Welche beiden "Dinge" sind gemeint?
- Wie heißen die beiden Funktionen üblicherweise, und welche Aufgabe erfüllen sie?
- Warum müssen die beiden Funktionen atomar ausgeführt werden?

Aufgabe 8

Aufgabe

Aufg. 259: Mutex

Was versteht man unter einem Mutex (im Kontext der Prozess-Synchronisation)?

Aufgabe 9

Aufgabe

Aufg. 260: Arbeitsweise eines Mutex

Erläutere anhand eines selbstgewählten Beispiels die Arbeitsweise eines Mutex. Welche Funktionsaufrufe werden rund um die kritischen Abschnitte getätigt?

Aufgabe 10

Aufgabe

Aufg. 261: Arbeitsweise eines Zählsemaphors

Erläutere anhand eines selbstgewählten Beispiels die Arbeitsweise eines Zählsemaphors. Welche Funktionsaufrufe werden rund um die kritischen Abschnitte getätigt?

Aufgabe 11

Aufgabe

Aufg. 262: Aktiv wartender Semaphor?

Wenn Semaphore zur Prozess-Synchronisation eingesetzt werden, ist das dann auch "aktives Warten"? Oder was ist anders? Erläutere!

Aufgabe 12

Aufgabe

Aufg. 263: Ein Abend an der Bar

In einer gemütlichen Feierabend-Kneipe ist nicht viel los. Dem Barkeeper sitzt lediglich ein Gast gegenüber. Dieser Gast jedoch hat gewaltigen Durst. Er trinkt ein Glas Schnaps nach dem anderen auf ex. Der Barkeeper füllt das leere Glas immer wieder bis oben hin auf.

Überführe diese Situation in den Kontext der Prozess-Synchronisation mit Semaphoren. Orientiere dich dabei an folgenden Fragen:

- Was repräsentiert in der Kneipe den/die Prozess(e)? (Wieviele gibt es?)

- Was repräsentiert in der Kneipe das/die Betriebsmittel? (Wieviele gibt es?)
- Was muss synchronisiert werden?
- Ist dabei eine Reihenfolge wichtig? Wenn ja: welche?
- Kann der Gast zweimal nacheinander das Glas (auf ex!) austrinken, ohne dass der Barkeeper es zwischendurch wieder füllt?
- Oder umgekehrt: Kann der Barkeeper das Glas zweimal nacheinander (bis oben hin!) füllen, ohne dass der Gast es zwischendurch austrinkt?
- Wieviele Semaphore brauchst du?
- Wann wird jeweils die P()- bzw. die V()-Operation ausgeführt? Von welchem Prozess?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

4.5.4 Aufgaben zu Deadlocks

Aufgabe 1

Aufgabe

Aufg. 264: Deadlock

Erläutere: Was versteht man unter einem Deadlock?

Aufgabe 2

Aufgabe

Aufg. 265: Deadlock-Beispiel

Erläutere anhand eines selbstgewählten Beispiels, wie ein Deadlock zwischen zwei (oder mehr) Prozessen entstehen kann.

Aufgabe 3

Aufgabe

Aufg. 266: Betriebsmittelgraph

Gegeben seien die Prozesse P1 bis P5 und die Ressourcen A bis G. Jede Ressource sei genau einmal vorhanden, sie kann auch nur exklusiv genutzt werden, d.h. von maximal einem Prozess zur Zeit.

Aktuell besteht folgende Ressourcenzuteilung und -anforderung:

- P1 belegt D und fordert C und G an.
- P2 belegt E und fordert B an.
- P3 belegt B und fordert D an.
- P4 belegt G und fordert A an.
- P5 belegt C und fordert E an.

Zeichne den Betriebsmittelgraphen!

Woran kannst du am Graphen allgemein erkennen, ob ein Deadlock vorliegt?

Liegt bei der o.a. Ressourcenzuteilung und -anforderung ein Deadlock vor? Falls ja: Welche Prozesse sind daran beteiligt?

Aufgabe 4

Aufgabe

Aufg. 267: Bedingungen nach Coffman

In Zusammenhang mit Deadlocks wurden von Coffman et al. vier Bedingungen identifiziert, die allesamt eingetreten sein müssen, damit ein Deadlock entsteht.

Nenne und erläutere diese vier Bedingungen in jeweils einem Satz!

Aufgabe 5

Aufgabe

Aufg. 268: Auszahlung mit Limit

In einer Bank gibt es verschiedenen Konten, die jeweils einen positiven oder negativen Kontostand aufweisen können. Das Limit zu einem Konto gibt an, um wieviel ein Kontostand überzogen sein darf. (Ein Limit von "-500,00" bedeutet also, der Kontostand darf -500,00 nicht unterschreiten.)

Gegeben sei das folgende "Programmstück", das eine Auszahlung von einem Konto prüft und bucht:

(1) Lese Kontostand des zu belastenden Kontos.
(2) Prüfe ob (Kontostand – Auszahlungsbetrag) (2a) Falls ja: Gebe eine Fehlermeldung aus.
(2b) Falls nein: Überschreibe den Kontostand mit (Kontostand – Auszahlungsbetrag).

Erläutere warum mehrere Auszahlungen nicht beliebig parallel durchgeführt werden dürfen. Was könnte passieren?

Erläutere ferner wie das Programmstück abgesichert werden kann.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

4.6 Aufgaben zum Kapitel Speicherverwaltung

Aufgabe 1

Aufg. 269: Swapping und Paging

Erläutere den grundlegenden Unterschied zwischen Swapping und Paging!

Aufgabe 2

Aufg. 270: Optimale Seitenersetzung

Erläutere die Funktionsweise des optimalen Seitenersetzungsalgorithmus und warum er nicht in der Praxis implementiert werden kann.

Aufgabe 3

Aufg. 271: LRU und NRU

Was bedeuten die Abkürzungen LRU und NRU im Hinblick auf Seitenersetzungsalgorithmen?

Aufgabe 4

Aufg. 272: Arbeitsweise LRU

Erläutere die Arbeitsweise des LRU-Algorithmus.

Aufgabe 5

Aufgabe

Aufg. 273: Arbeitsweise NRU

Erläutere die Aufgabe und Funktionsweise der NRU-Seitenersetzungsstrategie.

Aufgabe 6

Aufgabe

Aufg. 274: Spalten der Seitentabelle bei NRU

Bei dem NRU-Seitenersetzungsalgorithmus besteht die Seitentabelle aus (mindestens) 4 Spalten. Benenne die vier Spalten und erläutere deren jeweilige Aufgabe(n).

Aufgabe 7

Aufgabe

Aufg. 275: Umstände beim Seitenfehler

Wann tritt ein Seitenfehler (Page fault) auf? Erläutere die Umstände.

Aufgabe 8

Aufgabe

Aufg. 276: Auslagern bei NRU

Die Speicherverwaltung eines Betriebssystems arbeite nach der NRU-Seitenersetzungsstrategie.

Welchen Spalten der Seitentabelle kommt bei einem Seitenfehler eine besondere Bedeutung zu?

In welcher Weise beeinflussen die möglichen Werte in diesen Spalten die Entscheidung bzgl. der Auswahl einer zum Auslagern geeigneten Seite?

Aufgabe 9

Aufgabe

Aufg. 277: Virtuelle Speicherverwaltung

31	000	0
30	000	0
29	000	0

28	000	0
27	000	0
26	000	0
25	000	0
24	000	0
23	000	0
22	000	0
21	000	0
20	000	0
19	000	0
18	000	0
17	000	0
16	000	0
15	000	0
14	000	0
13	000	0
12	000	0
11	000	0
10	000	0
9	000	0
8	000	0
7	100	1
6	000	0
5	000	0
4	000	1

111	1	
2	110	1
1	001	1
0	010	1

Gegeben sei die nebenstehende komplette Seitentabelle mit zwei Spalten (ganz links ist lediglich eine Nummerierung der Zeilen).

Eine einzelner Seitenrahmen im physikalischen Adressraum habe eine Größe von 1 KiB (= 1024 Byte).

(a)
Wie groß ist eine einzelne Seite im virtuellen Adressraum?

(b)
Wie viele Seitenrahmen im physikalischen Adressraum gibt es laut Seitentabelle? Woran ist das erkennbar?

(c)
Wie viele Seiten im virtuellen Adressraum gibt es laut Seitentabelle? Woran ist das erkennbar?

(d)
Bezogen auf die Gegebenheiten dieser Aufgabe:
Warum besteht eine virtuelle Adresse aus genau 15 Bit? Erläutere!

(e)
Gegeben sei folgende virtuelle Adresse: 0 0 0 1 1 1 0 0 1 1 0 0 0 0 1
Ermittle daraus die zugehörige physikalische Adresse!

(f)
Gegeben sei folgende virtuelle Adresse: 0 1 0 0 0 1 0 1 0 1 0 0 1 1 0
Diese kann mit Hilfe der nebenstehenden Seitentabelle nicht in die zugehörige physikalische Adresse umgerechnet werden.

Warum nicht?

Erläutere **stichpunktartig** den weiteren Ablauf auf dem System bis zu dem Zeitpunkt, an dem die Umrechnung endlich klappt!

Aufgabe 10

Aufgabe

Aufg. 278: Speicher und Seitentabellen

Ein Computersystem besitze 2 GiB RAM (physikalischer Speicher). Das Betriebssystem billigt jedem Prozess 4 GiB virtuellen Speicher zu. Derzeit existieren insgesamt 10 Prozesse, die vom Betriebssystem verwaltet werden. Ein Seitenrahmen habe eine Größe von 16 KiB.

- Wieviele Seitentabellen werden in der beschriebenen Situation aktuell vom Betriebssystem verwaltet?
- Um welchen Faktor größer ist der in der beschriebenen Situation insgesamt vom Betriebssystem zugeteilte virtuelle Speicher gegenüber dem physisch im Computersystem eingebauten RAM?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

4.7 Aufgaben zum Kapitel Geräteverwaltung

Aufgabe 1

Aufgabe

Aufg. 279: Geräte

Nenne beispielhaft fünf Geräte, welche durch die Geräteverwaltung eines Betriebssystems verwaltet werden!

Aufgabe 2

Aufgabe

Aufg. 280: Zusammenspiel

Erläutere anhand eines selbstgewählten Beispiels das Zusammenspiel zwischen Prozess, Systemaufruf, Kernel-Mode, User-Mode, Interrupt, Interrupt-Controller, Geräte-Treiber und Geräte-Controller.
Inwieweit spielen Prozesszustände dabei eine Rolle?

Aufgabe 3

Aufgabe

Aufg. 281: Aufgaben eines Treibers

Nenne fünf Aufgaben eines Geräte-Treibers!

Aufgabe 4

Aufgabe

Aufg. 282: Treiber-Hersteller

Erläutere warum es i.d.R. für jedes Gerät einen eigenen Treiber gibt, und warum es Sinn macht, dass der jeweilige Gerätehersteller diesen Treiber zur Verfügung stellt.

Aufgabe 5

Aufgabe

Aufg. 283: Gefährlicher Treiber

Erläutere: Warum kann es gefährlich sein, wenn ein Treiber im Kernel-Mode ausgeführt wird?

Für wen wird es hier gegebenenfalls gefährlich?

Aufgabe 6

Aufgabe

Aufg. 284: ISR und Registerinhalte

Eine Interruptbehandlungsroutine (ISR) besteht aus einer Reihe von Befehlen, welche auf der CPU ausgeführt werden. Dabei unterbricht die ISR einen auf der CPU laufenden Prozess A.

Wenn davon ausgegangen werden kann, dass die ISR nur Auswirkungen auf einen anderen Prozess B hat, so darf Prozess A direkt nach Abarbeitung der ISR seine Arbeit auf der CPU fortsetzen.

Was passiert mit den für Prozess A wichtigen Registerinhalten auf der CPU, während die ISR abgearbeitet wird?

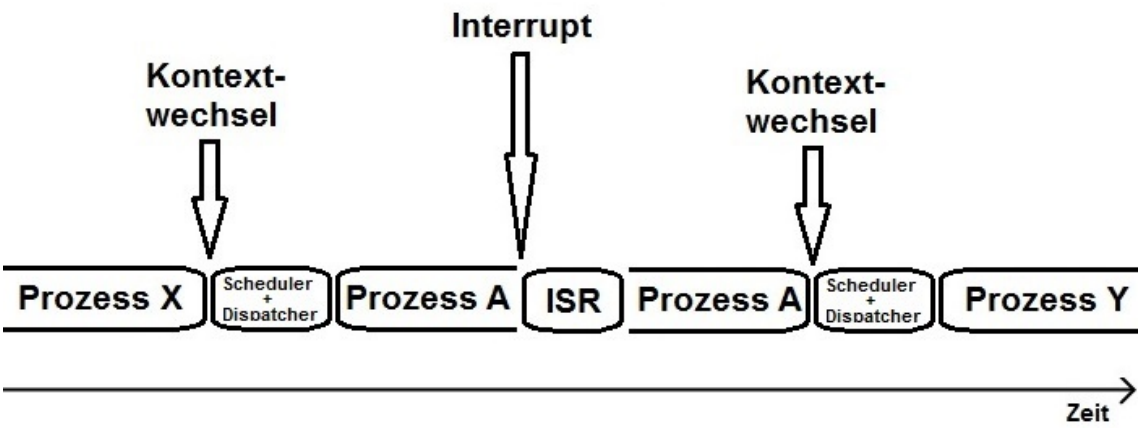


Abb. 79: Ein Interrupt mit seiner zugehörigen Interruptbehandlungsroutine (ISR) unterbricht den Prozess A auf der CPU
CC-BY

Aufgabe 7

Aufgabe

Aufg. 285: Block- vs. zeichenorientiert

Erläutere die Unterschiede zwischen "blockorientierten Geräten" und "zeichenorientierten Geräten".

Aufgabe 8

Aufgabe

Aufg. 286: Beispiele für block- und zeichenorientiert

Nenne zwei Beispiele für "blockorientierte Geräte" und zwei weitere Beispiele für "zeichenorientierte Geräte".

Aufgabe 9

Aufgabe

Aufg. 287: Zusammenspiel

Ein Prozess A möchte nur ein Datenwort in eine (bereits erfolgreich geöffnete) Datei auf der Festplatte schreiben. Ein DMA-Controller steht nicht zur Verfügung.

Erläutere das Zusammenspiel zwischen CPU, Prozess(en), Prozesszuständen, User-Mode, Kernel-Mode, Systemaufruf, Interrupt, Interrupt-Controller, Geräte-Treiber und Geräte-Controller.

Diese Seite steht unter der Creative Commons Namensnennung 3.0 Unported Lizenz http://i.creativecommons.org/l/by/3.0/80x15.png

4.8 Aufgaben zum Kapitel Dateiverwaltung

Aufgabe 1

Aufg. 288: CRUD

In Zusammenhang mit dem Umgang mit Dateien und Verzeichnissen spricht man von den sogenannten CRUD-Operationen. Wofür steht die Abkürzung "CRUD"?

Aufgabe 2

Aufg. 289: Dateisysteme

Nenne zwei unterschiedliche Dateisysteme!

Aufgabe 3

Aufg. 290:

Erläutere den Unterschied zwischen "Sektor" und "Cluster" im Kontext von Festplatten und Dateisystemen.

Aufgabe 4

Aufg. 291: FAT

Wofür steht die Abkürzung FAT (im Kontext von Dateisystemen)?

Aufgabe 5

Aufg. 292: NTFS

Wofür steht die Abkürzung NTFS (im Kontext von Dateisystemen)?

Aufgabe 6

Aufgabe

Aufg. 293: Arbeiten mit der FAT

0	3
1	20
2	8
3	1
4	frei
5	31
6	9
7	frei
8	nil
9	24
10	frei
11	12
12	0
13	frei
14	frei
15	16
16	21
17	15
18	frei
19	frei
20	22
21	25
22	nil

23	nil
24	2
25	23
26	5
27	nil
28	frei
29	27
30	frei
31	29

Die nebenstehende FAT besteht nur aus einer Spalte (ganz links ist die Nummerierung der Cluster, gleichbedeutend mit der Nummerierung der Zeilen der Tabelle).

(a)
Die Datei xyz.jpg beginne in Cluster-Nr. 11. Aus den Inhalten welcher Cluster setzt sich der Dateiinhalt insgesamt zusammen? (Achte auf die richtige Reihenfolge der Cluster-Nummern!)

(b)
Die Datei xyz.jpg beginne in Cluster-Nr. 11 und soll gelöscht werden. Wie ändern sich die Einträge in der FAT durch das Löschen der Datei?

(c)
Welche Nummer hat der Startcluster der Datei, deren letzter Cluster die Nummer 27 hat?

(d)
Eine neue Datei abc.mp3 soll gespeichert werden. Aufgrund der Dateigröße werden insgesamt fünf Cluster zum Speichern benötigt. In welchen Clustern legst du die Datei abc.mp3 ab?

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

5 Logbuch

Gliederung

5 Logbuch
5.1 Jahr 2013
5.2 Jahr 2014
5.3 Jahr 2015
5.4 Jahr 2016

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

5.1 Jahr 2013

Hinweis

Die Einträge im Logbuch starten ganz unten auf dieser Seite.
Hier oben stehen die jüngsten Einträge.

05.12.2013

Erstaunlich ruhig im Moment, d.h. es meckert keiner wegen der noch unfertigen Kapitel?!? Nun ja, die Arbeiten am Modul sind in den letzten Tagen wieder ordentlich in Fahrt gekommen. In Kürze dürfte das Kapitel Speicherverwaltung fertig werden. Ein aktueller PDF-Export besitzt 293 Seiten. Das ist doch schon mal was, oder?

20.11.2013

Es wird langsam eng! Der zu Semesterbeginn veröffentlichte Zeitplan sieht vor, dass die Studierenden sich in dieser Woche mit dem Kapitel Speicherverwaltung beschäftigen sollen. Aber dieses Kapitel ist aktuell erst in einer Übersicht fertig. Momentan bleibt nur der Verweis auf die Literatur und das Vertrösten auf die nächste Woche, in der voraussichtlich etwas mehr Zeit zur weiteren Ausarbeitung besteht. Mal abwarten, wieviel deswegen gemeckert wird...

05.11.2013

Bislang hatten Leserinnen und Leser nur Rechtschreibfehler im Text dieses Lernmoduls verbessert. Doch heute kam es dazu, dass ein aufmerksamer Leser einen inhaltlichen Fehler verbessern wollte. Er tat es auch, nahm seine Änderung aber bereits fünf Minuten später wieder zurück, als er merkte, dass der Fehler nicht im Modul, sondern zwischen seinen Ohren saß ;-)

01.11.2013

Die Arbeiten stecken irgendwo im Kapitel über die Scheduling-Verfahren. Es geht voran, aber es bleibt im Moment leider keine Zeit, weitere erklärende Videos zu produzieren.

31.10.2013

Ein aufmerksamer Leser (oder war es eine Leserin?) hat gleich vier Änderungsvorschläge für eine Seite eingereicht. Allesamt berechtigt und sinnvoll, weshalb sie gerne angenommen wurden. Danke für die Mitarbeit!

29. und 30.10.2013

Es wurden einige Fehler gemeldet (Danke Sebastian!), die mit dem bereits bekannten Problem der index-Tags beim PDF-Export zusammenhängen. An mehr als 10 Stellen im Quelltext verschiedener Seiten werden Veränderungen vorgenommen. Ein erneuter Abruf des PDFs ist somit für alle empfehlenswert, die mit dieser Offline-Version arbeiten.

11.10.2013

Anwender berichten, dass die erstellbare ePub-Version dieses Lernmoduls feherhaft sei, und verschiedene ePub-Reader beim Öffnen der Datei abstürzen. Oncampus wurde informiert.
Ein Workaround zu Problembehebung: die heruntergeladene ePub-Version einmal in Sigil (Multi-platform: Windows, Linux and Mac) öffnen und ohne Änderungen wieder speichern. Anschließend stürzt der ePub-Reader nicht mehr ab.

08.10.2013

Anwender berichten, dass im Dokument des PDF-Exports einige Textabsätze nicht enthalten seien, die in der Online-Version zu sehen sind. Die Ursache dieses Problems liegt wiederum in den index-Tags im Quelltext der Seiten, insbesondere wenn diese innerhalb bestimmter loop-area-Tags verwendet werden.
Im Quelltext sind die index-Tags an eine andere Position verschoben worden, das Problem ist damit umgangen.

04.10.2013

Die Arbeiten am Kapitel Betriebssysteme starten.

29.09.2013

Das Kapitel Computerarchitektur ist endlich fertig! Es fehlen zwar noch einige Index-Einträge, und der eine oder andere Rechtschreibfehler wird auch noch enthalten sein, aber Inhalte und Aufgaben stehen soweit für das aktuelle Wintersemester 2013/14.

Eingebunden wurden 23 Videos, 41 Abbildungen und 48 Aufgaben.

24.09.2013

Es ist ein umfangreicher Index hinzu gekommen. Allerdings mit einem kleinen Nachteil: Die (eigentlich unsichtbaren) Index-Tags im Quellcode der Seiten werden fälschlicherweise in der PDF-Version des Moduls angezeigt. Oncampus wurde informiert.

17.09.2013

Die ersten drei Nutzer dieses Moduls haben nur zwei Tage nach Semesterstart Korrekturen eingereicht und damit aktiv zur Weiterentwicklung beigetragen. Danke dafür!

15.09.2013

Im Wintersemester 2013/14 werden die hier veröffentlichten Inhalte im gleichnamigen Modul "Computerarchitektur und Betriebssysteme" des Studiengangs "Online-Medieninformatik" an verschiedenen Hochschulen des Verbunds der Virtuellen Fachhochschulen Deutschland als Lernmaterialien verwendet. Heute ist offizieller Semesterbeginn, die Studierenden werden über den eingerichteten Moodle-Kurs an ihren Hochschulen über die Existenz dieser Website informiert.

05.09.2013

Heute beschleicht den Autoren zum ersten Mal das Gefühl, dass die Gliederung des Kapitels Computerarchitektur jetzt "rund" ist und für einen ersten Semesterdurchlauf mit Studierenden in der Hochschulausbildung so eingesetzt werden kann. Es fehlen aber auf einer Reihe von Seiten noch Inhalte und insbesondere erklärende Videos.

03.09.2013

Die Inhalte im Kapitel Vom Anwender zur digitalen Schaltung sind soweit erstellt, es fehlen lediglich noch Aufgaben.

02.09.2013

Da die Inhalte dieses Moduls an verschiedenen Hochschulen des VFH-Verbundes im WS 13/14 eingesetzt werden sollen, wurden die lehrenden Kollegen an den anderen Standorten heute vom aktuellen Stand bei der Ausarbeitung des informiert. Semesterbeginn ist bereits in zwei Wochen.

31.08.2013

Mittlerweile wurden eine Reihe von Videos mit Animationen erstellt, auf YouTube veröffentlicht und in verschiedenen Kapiteln eingebunden. Die Gliederung wurde immer wieder angepasst und verfeinert, damit der "rote Faden" (hoffentlich) klar erkennbar bleibt.

23.08.2013

Die ersten Videos sind erstellt und eingebunden. Es geht voran.

19.08.2013

Die Arbeit an einzelnen Unterkapiteln sind bereits weit fortgeschritten. Es fehlen aber noch einige Abbildungen und Videos.

Anfang August 2013

Nach einem erholsamen Familienurlaub beginnt die Arbeit am Kapitel Computerarchitektur. Der "rote Faden" und eine Gliederung werden erstellt. Parallel erfolgt die Sichtung verschiedener Quellen.

16.07.2013

Die Inhalte sind nun zum Abruf im Internet für Jedermann und Jederfrau verfügbar. Eine Anmeldung ist nur erforderlich, wenn Änderungen oder Erweiterungen eingepflegt werden sollen.

01.07.2013

Hinweise zum Copyright werden eingefügt.

25.06.2013

Hallo Welt!
Die ersten Kapitel werden zum Inhaltsverzeichnis hinzugefügt.
Dieses Logbuch entsteht.

14.06.2013

Nach der Klärung einiger Detailfragen ist das LOOP nun eingerichtet und einsatzbereit. Danke Stefanie und Guido.
An diesem Tag ist dieses LOOP praktisch geboren!

11.06.2013

Die Einrichtung dieses LOOPs wurde offiziell beantragt. Natürlich einfach, schnell und unkompliziert per E-Mail.

07.06.2013

Während des VFH-Symposiums in Lübeck wurden erste Absprachen mit der oncampus GmbH bzgl. der Einrichtung dieses LOOPs getroffen.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

5.2 Jahr 2014

Hinweis

Die Einträge im Logbuch starten ganz unten auf dieser Seite.
Hier oben stehen die jüngsten Einträge.

17.12.2014

Dieses Modul ist in den letzten Tagen Opfer mehrerer SPAM-Angriffe geworden. Ein User hat verschiedene Seiten mit unsinnigen Inhalten neu angelegt. Da die Seiten vor ihrer Veröffentlichung jedoch immer erst noch einmal überprüft werden, flog der Schwindel natürlich sofort auf. Die Seiten wurden entfernt, der betreffende User gesperrt.

20.11.2014

Es gibt eine neue Möglichkeit, eine Druck-Version der Kursinhalte zu erstellen. Es ist eine "Bastel-Lösung", welche jedoch zu einem Ausdruck führt, der optisch sehr nah an das Erscheinungsbild im Browser heranreicht. Siehe: Inhalt dieses LOOPs ausdrucken.

11.11.2014

Die Arbeiten am Kapitel Geräteverwaltung haben begonnen.

13.10.2014

Eine Reihe von kosmetischen Änderungen und Anpassungen sind heute im Kapitel Betriebssysteme erfolgt. Es gab jedoch keine inhaltlichen Änderungen, d.h. der Lernstoff an sich ist unverändert geblieben.

12.10.2014

Eine Reihe von kosmetischen Änderungen und Anpassungen sind heute im Kapitel Computerarchitektur erfolgt. Es gab jedoch keine inhaltlichen Änderungen, d.h. der Lernstoff an sich ist unverändert geblieben.

21.06.2014

Als Autor will man ja immer informiert werden, wenn ein Nutzer Änderungsvorschläge einpflegt. Bislang ging das über den RSS-Feed. Aber bei zu vielen abonnierten Feeds übersieht man schnell mal einen Vorschlag. Besser ist eine Benachrichtigung per E-Mail, was durch "If This Than That" sehr leicht zu realisieren ist (https://www.ifttt.com).
Danke für den Tipp, Marc!

14.06.2014

Happy birthday!

Der erste Geburtstag dieser Lehrmaterialien. Bislang sind insgesamt 224 Benutzer registriert, d.h. 224 User haben die Möglichkeit, diesen Kurs zu verbessern oder zu erweitern. Danke an alle, die bislang geholfen haben!

31.01.2014

Heute wird im Verbund der Virtuellen Fachhochschulen Deutschlands die Klausur zu den Inhalten dieses Moduls geschrieben. Damit ist das erste Semester praktisch zu Ende. Zumindest aus Sicht der Studierenden, denn die Lehrenden müssen die Klausuren ja noch durchsehen und bewerten.

Fazit für die Inhalte in diesem LOOP: Es ist nicht alles fertig geworden, einige Kapitel existieren bislang nur mit ihrer Überschrift. Aber nach dem Semester ist vor dem Semester, und die Entwicklung wird weitergehen.

17.01.2014

Es gab im LOOP-Bugtracker einen Hinweis, wie man die Probleme beim PDF-Export umgehen kann. Der Quelltext aller derzeit existenten Seiten wurde daraufhin angepasst. Anschließend konnten keine Probleme beim Export mehr festgestellt werden. Es scheint tatsächlich zu funktionieren. Da geht das Jahr doch gut los! ;-)

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

5.3 Jahr 2015

Hinweis

Die Einträge im Logbuch starten ganz unten auf dieser Seite.
Hier oben stehen die jüngsten Einträge.

07.09.2015

Eine weitere Überarbeitungsrunde dieses Kurses startet. In den kommenden Wochen sollen kleine und große Baustellen angegangen werden.

06.02.2015

Das Kapitel zur Prozess-Synchronisation wurde neu strukturiert und überarbeitet. Eine neu erstellte Zusammenfassung vedeutlicht den Aufbau und die Zusammenhänge dieses Kapitels.

30.01.2015

Viele Aufgaben zur Prüfungsvorbereitung wurden als eigenständiges Kapitel hinzugefügt.

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

5.4 Jahr 2016

Hinweis

Die Einträge im Logbuch starten ganz unten auf dieser Seite.
Hier oben stehen die jüngsten Einträge.

15.01.2016

Aufgabe zur CPU-Simulation ergänzt.
https://vfhcab.eduloop.de/loop/Animation_der_Zusammenarbeit#Aufgabe_5

6 Anhang

Gliederung

6 Anhang
6.1 Zweier-Potenzen
6.2 Bits und Bytes
6.3 GiB, MiB, KiB im Vergleich zu GB, MB, KB
6.4 Java-Applets
6.5 Inhalt dieses LOOPs ausdrucken
6.6 Impressum

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

6.1 Zweier-Potenzen

$2^0 = 1$

$2^1 = 2$

$2^2 = 4$

$2^3 = 8$

$2^4 = 16$

$2^5 = 32$

$2^6 = 64$

$2^7 = 128$

$2^8 = 256$

$2^9 = 512$

$2^{10} = 1.024$

$2^{11} = 2.048$

$2^{12} = 4.096$

$2^{13} = 8.192$

$2^{14} = 16.384$

$2^{15} = 32.768$

$2^{16} = 65.536$

$2^{17} = 131.072$

$2^{18} = 262.144$

$2^{19} = 524.288$

$2^{20} = 1.048.576$

$2^{21} = 2.097.152$

$2^{22} = 4.194.304$

$2^{23} = 8.388.608$

$2^{24} = 16.777.216$

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

6.2 Bits und Bytes

Ein Bit ist die kleinstmögliche digitale Informationseinheit. Es kann den Wert 0 oder 1 annehmen, alternativ "wahr" oder "falsch, bzw. englisch "true" oder "false". </p>

8	Bit	=	1	Byte
8	Bit	=	1	Byte
8	Bit	=	1	Oktett

1.024	Byte	=	2^{10}	Byte	=	1 Kibibyte
1.024	Kibibyte	=	2^{20}	Byte	=	1 Mebibyte
1.024	Mebibyte	=	2^{30}	Byte	=	1 Gibibyte
1.024	Gibibyte	=	2^{40}	Byte	=	1 Tebibyte

Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

6.3 GiB, MiB, KiB im Vergleich zu GB, MB, KB

Nicht nur in der Schreibweise gibt es einen Unterschied zwischen GiB, MiB sowie KiB im Vergleich zu GB, MB und KB. Während die erstgenannten Einheiten auf 2er-Potenzen basieren, begründen sich die letztgenannten auf 10er-Potenzen. </p>

Basierend auf 2er-Potenzen

Die Physikalisch-Technische Bundesanstalt hat durch Mitteilung in Heft 2 aus Juni 2007 (Seite 164) die internationale Norm IEC 60027-2 (2005, 3. Auflage) umgesetzt, nach der mit den folgenden Abkürzungen versehene Angaben jeweils auf 2er-Potenzen basieren:

4 **GiB (GibiByte)**

= 4 * 2^{10} **MiB**

= 4 * 1.024 **MiB**
= 4.096 **MiB (MebiByte)**

= 4 * 2^{10} * 2^{10} **KiB**
= 4 * 1.024 * 1.024 **KiB**
= 4.194.304 **KiB (KibiByte)**

= 4 * 2^{10} * 2^{10} * 2^{10} **Byte**
= 4 * 1.024 * 1.024 * 1.024 **Byte**
= 4.294.967.296 **Byte**

= 4 * 2^{10} * 2^{10} * 2^{10} * 8 **Bit**
= 4 * 1.024 * 1.024 * 1.024 * 8 **Bit**
= 34.359.738.368 **Bit**

Basierend auf 10er-Potenzen

Im Vergleich die Bedeutung der Schreibweisen *GB*, *MB* und *KB*. Diese basieren auf 10er-Potenzen:

Hinweis

4 **GB (GigaByte)**

= 4 * 10^3 **MB**
= 4 * 1.000 **MB**
= 4.000 **MB (MegaByte)**

= 4 * 10^3 * 10^3 **KB**
= 4 * 1.000 * 1.000 **KB**
= 4.000.000 **KB (KiloByte)**

= 4 * 10^3 * 10^3 * 10^3 **Byte**
= 4 * 1.000 * 1.000 * 1.000 **Byte**
= 4.000.000.000 **Byte**

= 4 * 10^3 * 10^3 * 10^3 * 8 **Bit**
= 4 * 1.000 * 1.000 * 1.000 * 8 **Bit**
= 32.000.000.000 **Bit**

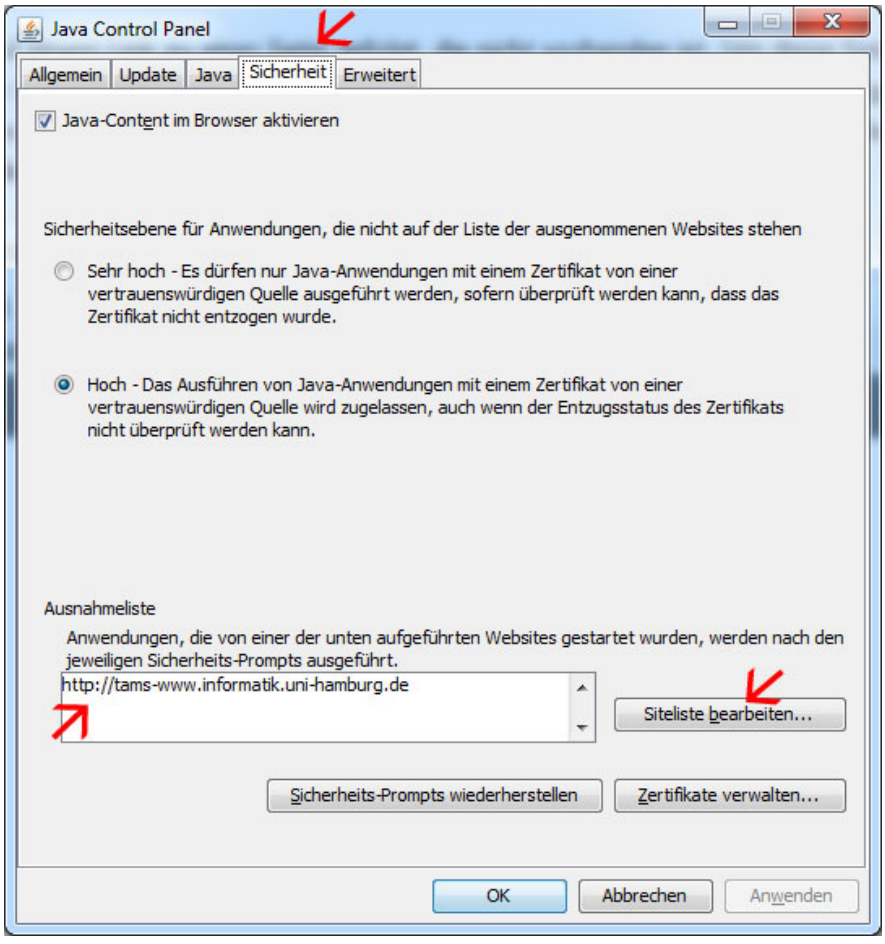
Diese Seite steht unter der ~~Creative Commons Namensnennung 3.0 Unported Lizenz~~ ~~http://i.creativecommons.org/l/by/3.0/80x15.png~~

6.4 Java-Applets

Um Java-Applets ausführen zu können, ist der Eintrag der Ursprungsdomain in die Liste der aktzeptierten Domains notwendig.

Nach dem Download und der Installation des JRE (Java Runtime Environment) befindet sich unter Windows in der Systemsteuerung das Icon *Java*.

Durch einen Doppelklick auf dieses Icon öffnet sich das *Java Control Panel*. Hier muss - wie im Bild dargestellt - auf der Registerkarte *Sicherheit* die Ursprungsdomain eingetragen werden.



Beispiel

Beispiel:

Der Zugriff auf ein Java-Applet auf der Seite

http://tams-www.informatik.uni-hamburg.de/applets/hades/webdemos/16-flipflops/10-srff/srff.html

erfordert den Eintrag der Ursprungsdomain

http://tams-www.informatik.uni-hamburg.de

im *Java Control Panel*, Registerkarte *Sicherheit*.

6.5 Inhalt dieses LOOPs ausdrucken

Die erste Wahl beim Drucken des Inhalts dieses LOOPs geht in jedem Fall über die PDF-Version dieses Skripts. </p>
Wer etwas mehr Wert darauf legt, dass der Ausdruck möglichst exakt so aussieht, wie die Inhalte hier im Browser, der kann zu dieser Bastel-Lösung greifen.

Bastel-Lösung

Die Bastel-Lösung ist aus der Not heraus entstanden, weil die PDF-Export-Funktion in einem anderen LOOP aus unerklärlicher Weise nicht funktionierte, aber dennoch kurzfristig ein PDF erstellt werden musste.

Das folgende AutoIt-Skript ruft nachheinander alle Seiten des LOOP-Inhaltsverzeichnisses im Internet Explorer auf, macht einen Screenshot von den verschiedenen Teilen der Seite, und fügt diese in ein Word-Dokument ein.

So geht es halt auch, dauert nur etwas länger.

DOM-Manipulation im Browser

Und wer vielleicht aus dem Bereich der Webprogrammierung schon immer mal ein praktisches Beispiel für DOM-Manipulation im Browser haben wollte: Das Skript macht ausgibig Gebrauch davon!

Download

Verwendete Software

Neben Microsoft Windows, Internet Explorer und Microsoft Word wurden noch verwendet:

- AutoIt - https://www.autoitscript.com/site/autoit/
- FireShot - https://getfireshot.com/

Quellcode des Skripts

</>
Code

```
1 #include <INet.au3>
2 #include <_HTML.au3>
3
4 #include <IE.au3>
5 #include <Word.au3>
6
7 #include <MsgBoxConstants.au3>
8
```

```
 9 ; ===============================================================
10 ; AutoIt-Skript
11 ; zum Extrahieren des Content eines LOOP (http://loop.oncampus.de).
12 ; ---------------------------------------------------------------
13 ; Andreas Wilkens im November 2014
14 ; aw@dozaw.de
15 ; ---------------------------------------------------------------
16 ; Die einzelnen Webseiten werden im Internet Explorer (IE) dargestellt
17 ; und mit Hilfe von FireShot wird ein Screenshot von bestimmten
18 ; Seitenteilen angefertigt. Dabei werden alle Nicht-Content-Inhalte
19 ; zuvor per DOM-Manipulation im Browser ausgeblendet (Navigationselemente,
20 ; Header, Footer, Sidebar links und rechts)
21 ; Der Screenshot wird über die Windows-Zwischenablage in ein
22 ; Word-Dokument eingefügt.
23 ; Der User kann das Word-Dokument am Ende nach belieben ändern (z.B.
24 ; formatieren) und abspeichern, oder auch als PDF exportieren.
25 ; ===============================================================
26 ;
27 ; Diese Software ist noch im Teststadium.
28 ; Der Autor übernimmt keine Garantie für die Funktionsfähigkeit.
29 ; Der Autor haftet nicht für Schäden, die durch die Ausführung dieser
30 ; Software entstehen.
31 ; Jegliche Verwendung geschieht auf eigene Gefahr.
32 ;
33 ; ===============================================================
34 ; Diese Software wurde entwickelt und getestet unter
35 ; Microsoft Windows 7 Professional, 64 Bit
36 ; AutoIt Version 3.3.12.0
37 ; Internet Explorer Version 11.0.9600.17420
38 ; FireShot for Internet Explorer v.0.98.63
39 ; Microsoft Word Version 14.0.7128.5000, 64 Bit, aus dem
40 ; Microsoft Office Professional Plus 2010 Paket
41 ; ===============================================================
42 ;
43 ; Diese Software ist freie Software.
44 ; Sie darf beliebig verwendet, verändert, veröffentlicht und
45 ; weitergegeben werden.
46 ;
47 ; ===============================================================
48
49 Opt("SendKeyDelay", 0)
50
51 ; Mit ESC wird die Ausführung des Skripts sofort beendet.
52 ; Es erfolgt keine weitere Rückfrage.
53 HotKeySet("{ESC}", "Terminate")
54
55 Func Terminate()
56  MsgBox( 0, "", "Skriptausführung beendet." )
57  Exit
58 EndFunc ;==>Terminate
59
60 ; ------------------------------------------------------------------------
61 ; Wenn $VERBOSE = True gesetzt ist, dann erfolgen beim Start des Skripts
62 ; eine Reihe von Rückfragen und Auswahlmöglichkeiten für den User. Zum
63 ; Testen sei dies unbedingt empfohlen!
64 ; Bei $VERBOSE = False fängt das Skript sofort an zu arbeiten. Es werden
65 ; dann die vorgegebenen Belegungen der Variablen unverändert übernommen.
```

```
 66
 67 Local $VERBOSE = True
 68
 69 ; ------------------------------------------------------------------------------
 70 ; Noch einige Variablen, die später benutzt werden.
 71
 72 Local $oIE            ; Für den Internet Explorer
 73 Local $hIE_Window_Handle
 74
 75 Local $oWord        ; Für Word
 76 Local $hWord_Window_Handle
 77
 78 Local $TIMEOUT_SECONDS = 3
 79 Local $result                    ; Für Rückgabewerte aus MsgBox oder InputBox
 80
 81 ; ------------------------------------------------------------------------------
 82
 83 If ($VERBOSE) Then
 84   $result = MsgBox( $MB_OKCANCEL, "Word schließen","Es darf keine Microsoft
Word Anwendung geöffnet sein." & @CRLF & @CRLF & "Gegebenenfalls bitte jetzt alle
aktiven Word-Instazen schließen." )
 85  If ( ($result <> $IDOK) And ($result <> $IDTIMEOUT) ) Then
 86      ; User hat den Cancel-Button gedrückt.
 87      Exit
 88  EndIf
 89 EndIf
 90
 91 ; ------------------------------------------------------------------------------
 92 ; Um alle Links auf dieser Seite geht es.
 93 ; Üblicherweise ist dies eine LOOP-Seite mit dem TOC (Table of Contents)
 94
 95 Local $sURL_TOC = "http://vfhcab.oncampus.de/loop/MediaWiki:Loop_toc"
 96
 97 If ($VERBOSE) Then
 98  $sURL_TOC = InputBox("", "Welche Webseite ist der Ausgangspunkt?" & @CRLF &
@CRLF  &  "Üblicherweise  ist  dies  eine  LOOP-Seite  mit  dem  'Table  of  Con
tents'",$sURL_TOC, "", 500)
 99  If (@error <> 0) Then
100      ; User hat den Cancel-Button gedrückt.
101      ; Skript wird beendet.
102      Exit
103  EndIf
104 EndIf
105
106 ; ------------------------------------------------------------------------------
107
108 If ($VERBOSE) Then
109  $result = MsgBox( $MB_OKCANCEL, "IE vorbereiten","Es muss jetzt bereits ein
Fenster des Internet Explorers geöffnet sein. Du als User musst dieses Fenster öff
nen." & @CRLF & @CRLF & "FireShot muss als AddOn im Internet Explorer installiert
sein." & @CRLF & "FireShot muss eingestellt sein auf" & @CRLF & "--> Capture visi
ble Area and" & @CRLF & "--> Copy to Clipboard" & @CRLF & @CRLF & "Bitte FireShot
unbedingt testen, da es direkt nach der Installation viele Rückfragen hat. Bei al
len Rückfragen mit 'Remember my decision' dafür sorgen, dass sie nicht wieder er
scheinen." & @CRLF & @CRLF & "Die LOOP-Seite " & $sURL_TOC & @CRLF & "muss einmal
im Internet Explorer geöffnet sein." & @CRLF & "Mit Benutzernamen und Passwort an
melden, falls erforderlich." & @CRLF & "Linke und rechte Sidebar ausblenden.")
```

```
110  If ( ($result <> $IDOK) And ($result <> $IDTIMEOUT) ) Then
111      ; User hat den Cancel-Button gedrückt.
112      Exit
113  EndIf
114 EndIf
115
116 ; ------------------------------------------------------------------------
117
118 If ($VERBOSE) Then
119  $result = MsgBox( $MB_OKCANCEL, "","Durch drücken der ESC-Taste wird die Aus
führung dieses Skripts sofort und ohne Rückfragen beendet.")
120  If ( ($result <> $IDOK) And ($result <> $IDTIMEOUT) ) Then
121      ; User hat den Cancel-Button gedrückt.
122      Exit
123  EndIf
124 EndIf
125
126 ; ------------------------------------------------------------------------
127 ; Alle Links dieser Seite ermitteln.
128
129 Local $HTML_SOURCE = _INetGetSource($sURL_TOC)
130
131 Dim $ALL_LINKS = _HTML_GetAllLinks($HTML_SOURCE)
132
133 If Not IsArray($ALL_LINKS) Then
134  MsgBox( 0, "Fehler", "Es ist ein Fehler beim Ermitteln der Links aufgetreten"
& @CRLF & @CRLF & sURL_TOC )
135  ; Das Skript wird beendet
136  Exit
137 EndIf
138
139 ; ====================================================================
140 ; Jetzt stehen alle ermittelten Links in $ALL_LINKS.
141 ; Aber es sind noch Links für die Navigation etc. dabei.
142 ; ====================================================================
143
144 ; --------------------------------------------------------------------
145 ; Ein wenig Information ausgeben.
146
147 If ($VERBOSE) Then
148   $result = MsgBox( $MB_OKCANCEL, "Anzahl Links","Auf der Webseite" & @LF &
$sURL_TOC & @LF & "gibt es insgesamt" & @LF & UBound($ALL_LINKS) & " Links" )
149  If ( ($result <> $IDOK) And ($result <> $IDTIMEOUT) ) Then
150      ; User hat den Cancel-Button gedrückt.
151      Exit
152  EndIf
153 EndIf
154
155 ; --------------------------------------------------------------------
156 ; $START_NO enthält den Index des ersten Links im Array $ALL_LINKS.
157 ; Der Wert dieser Variablen definiert, bei welchem Link später
158 ; mit dem Extrahieren des Content begonnen wird.
159
160 Local $START_NO = 4
161
162 If ($VERBOSE) Then
163  Local $sTEXT = "Dies sind die ersten Links auf der Seite '" & $sURL_TOC & "'"
```

```
& @LF
164  For $i = 0 To 32
165      If ($i < UBound($ALL_LINKS)) Then
166          $sTEXT = $sTEXT & @LF & $i & " : " & $ALL_LINKS[$i]
167      EndIf
168  Next
169   $sTEXT = $sTEXT & @LF & @LF & "Bei welcher Link-Nr. soll gestartet werden?
(Die Navigations-Links zu Beginn sollten übersprungen werden.)"
170
171  $START_NO = InputBox("Start-Nr.",$sTEXT,$START_NO,"",800,600)
172
173  If (@error <> 0) Then
174      ; User hat den Cancel-Button gedrückt.
175      ; Skript wird beendet.
176      Exit
177  EndIf
178 EndIf
179
180 ; ---------------------------------------------------------------------
181 ; $END_NO enthält den Index des letzten Links im Array $ALL_LINKS.
182 ; Der Wert dieser Variablen definiert, bei welchem Link später
183 ; das Extrahieren des Content beendet wird.
184
185 ; Local $END_NO = UBound($ALL_LINKS) - 1
186 Local $END_NO = $START_NO + 2
187
188 If ($VERBOSE) Then
189  Local $sTEXT = ""
190  Local $x
191  If (UBound($ALL_LINKS) >= 33) Then
192      $x = UBound($ALL_LINKS) - 33
193  Else
194      $x = 0
195  EndIf
196  For $i = (UBound($ALL_LINKS) - 1) To $x Step -1
197      $sTEXT = $i & " : " & $ALL_LINKS[$i] & @LF & $sTEXT
198  Next
199  $sTEXT = $sTEXT & @LF & @LF & "Bei welcher Link-Nr. soll beendet werden? (Die
Navigations-Links am Ende sollten nicht mit verarbeitet werden.)"
200
201  $END_NO = InputBox("End-Nr.",$sTEXT,$END_NO,"",800,600)
202
203  If (@error <> 0) Then
204      ; User hat den Cancel-Button gedrückt.
205      ; Skript wird beendet.
206      Exit
207  EndIf
208 EndIf
209
210 ; ---------------------------------------------------------------------------
211
212 If ($VERBOSE) Then
213  $result = MsgBox( $MB_OKCANCEL, "","Die Verarbeitung startet jetzt." & @CRLF
& @CRLF & "Dabei klickt das Skript auf den FireShot-Button im Internet Explorer."
& @CRLF & "Die Bildschirmkoordinaten für den MouseClick sind im Skript hinterlegt
und müssen ggf. angepasst werden. Siehe:" & @CRLF & @CRLF & "Local $FireShot_X =
80" & @CRLF & "Local $FireShot_Y = 66" & @CRLF )
```

```
214  If ( ($result <> $IDOK) And ($result <> $IDTIMEOUT) ) Then
215      ; User hat den Cancel-Button gedrückt.
216      Exit
217  EndIf
218 EndIf
219
220 ; ===============================================================
221 ; Jetzt haben wir alle Links in dem Array $ALL_LINKS.
222 ; $START_NO gibt an, bei welchem Array-Index in $ALL_LINKS wir starten.
223 ; $END_NO gibt an, bei welchem Array-Index in $ALL_LINKS wir enden.
224 ; Dazwischen werden alle Links verarbeitet.
225 ; ===============================================================
226
227 ; -------------------------------------------------------------------------
228 ; Word starten
229
230 $oWord = _Word_Create()
231 $hWord_Window_Handle = WinGetHandle("Microsoft Word", "")
232
233 ; Für Word-Properties siehe:
234 ; http://msdn.microsoft.com/en-us/library/ff841712%28v=office.14%29.aspx
235
236 ; Word auf dem Bildschirm positionieren
237 WinMove( $hWord_Window_Handle, "", 900, 0 )
238
239 ; Einige Vorbereitungen in Word treffen
240 Send( "^n" )        ; STRG n --> Neues Dokument in Word
241 Send( "^e" )        ; STRG e --> Absatz zentrieren
242
243 ; ===================================================================
244 ; Jetzt beginnt die Verarbeitung der gewünschten Links.
245 ; ===================================================================
246
247 ; Schleife über alle zu verarbeitenden Links
248 For $URL_NO = $START_NO To $END_NO
249
250  WinActivate( $hWord_Window_Handle, "" )         ; Dem Word-Fenster den Fokus
geben.
251
252  Send( "^{Enter}" )        ; STRG Enter --> Neue Seite in Word
253
254  ; Internet Explorer aufrufen mit URL dieses FOR-Durchlaufs
255  $oIE = _IECreate( $ALL_LINKS[ $URL_NO ], 0 )
256  $hIE_Window_Handle = _IEPropertyGet( $oIE, "hwnd" )
257
258  ; Das Fenster des Internet Explorers in der linken oberen Ecke des
259  ; Bildschirms positionieren, damit der Mausklick auf den FireShot-Button
260  ; weiter unten auch mit den korrekten Koordinaten ausgeführt wird.
261  ; (FireShot hat im IE aktuell keinen Shortcut, es muss also zwingend ein
262  ; Mausklick erfolgen)
263
264  _IEPropertySet( $oIE, "left", 0 )
265  _IEPropertySet( $oIE, "top", 0 )
266
267  ; Die Breite des Fensters entspricht der Breite des Content.
268   ; Das ist wichtig, da FireShot genau in der Breite des Fensterinhalts auf
nimmt.
```

```
269
270  _IEPropertySet( $oIE, "width", 873 )
271
272   ; Die Höhe des Fensters ist bewusst klein gewählt. FireShot scrollt bei der
Aufnahme
273   ; automatisch und baut das Bild korrekt zusammen.
274   ; Wäre die Höhe hier großer eingestellt, als der Content lang ist, so würde
unnötig
275   ; viel leerer Bildschirn aufgenommen werden.
276   ; Vorsicht: weniger als 150 macht hier keinen Sinn.
277    ; Wenn viele Symbolleisten etc. eingeblendet sind, kann auch ein deutlich
größerer
278   ; Wert nötig sein.
279
280  _IEPropertySet( $oIE, "height", 450 )
281
282   ; Über diese Koordinaten klickt die Maus auf den FireShot-Button im Internet
Explorer.
283   ; Die Koordinaten sind relativ zur linken oberen Ecke des Bildschirms!
284   Local $FireShot_X = 80
285   Local $FireShot_Y = 66
286
287   ; Kurz warten, bis das Bild im Browser korrekt aufgebaut ist.
288   Sleep( 500 )
289
290   ; ---------------------------------------------------------------------
291   ; ---------------------------------------------------------------------
292   ; Ab hier erfolgt die DOM-Manipulation im Browser.
293   ; ---------------------------------------------------------------------
294   ; ---------------------------------------------------------------------
295
296   ; Alle div's entfernen, die nicht mit in den Screenshot sollen.
297    Local $divToBeDeleted = ["header", "navbar1", "navbar2", "pageaudio", "side
bar_left", "sidebar_right", "content_bar", "content_footer", "footer"]
298   For $j = 0 To (UBound($divToBeDeleted) - 1)
299    $oDiv = _IEGetObjById($oIE, $divToBeDeleted[$j])     ; Dies ist das betref
fende div.
300   $oDiv.outerhtml = ""                                 ; So wird sein Inhalt aus
der Browseranzeige entfernt:
301   Next
302
303   ; Einige weitere Variablen, die zum Ermitteln des Contents nötig sind.
304    Local $sTitle = ""          ; Nimmt die ermittelte Seitenüberschrift der URL
dieses FOR-Durchlaufs auf.
305   Local $oDiv;                 ; Nimmt div-Container auf, um daraus den Content
zu ermitteln.
306   Local $oChildNodes;         ; Nimmt alle ChildNodes eines div-Containers auf.
307   Local $oNode;              ; Nimmt genau einen Node auf.
308
309    ; Die folgenden drei Zeilen dienen dazu, dem Array $HTMLParts die richtige
Größe zu geben.
310   $oDiv = _IEGetObjById($oIE, "mw-content-text")
311   $oChildNodes = $oDiv.childNodes                 ; Aus dem div alle childNodes
ermitteln
312   Local $HTMLParts[ $oChildNodes.length + 1 ]    ; Für Überschriften und Nodes,
die später als Screenshot in Word landen
313
```

```
Local $part = 0;

; ---------------------------------------------------------------
; Jetzt den Seitentitel ermitteln

; Bei "alten" LOOPs steht die Überschrift der Seite in diesem div
$oDiv = _IEGetObjById($oIE, "content_body")

 $oChildNodes = $oDiv.childNodes         ; Aus dem div alle childNodes ermit
teln

For $oNode in $oChildNodes

          If  (StringInStr(  $oNode.outerhtml,  '<h1>'  )  =  1)  or  (String
InStr( $oNode.outerhtml, '<h2>' ) = 1) Or (StringInStr( $oNode.outerhtml, '<h3>' )
= 1) Or (StringInStr( $oNode.outerhtml, '<h4>' ) = 1) Or (StringInStr( $oNode.out
erhtml, '<h5>' ) = 1) Or (StringInStr( $oNode.outerhtml, '<h6>' ) = 1) Then

         $sTitle = $oNode.innertext
         $HTMLParts[$part] = $sTitle
         $part += 1

         ; Den Inhalt dieses Nodes löschen, da er nicht ein zweites Mal
         ; verarbeitet werden darf.
         $oNode.outerhtml = ""

         ExitLoop        ; Die For-Schleife sofort verlassen,
                        ; weil hier nicht mehr sinnvoller Inhalt stehen kann.

     EndIf

Next        ; For $oNode in $oChildNodes

; ENDE: Sonderfall "alte" LOOPs.

; Bei "neuen" LOOPs steht die Überschrift der Seite in diesem div
; Und alle weiteren Nodes dieses div enthalten den Content der Seite.
$oDiv = _IEGetObjById($oIE, "mw-content-text")

 $oChildNodes = $oDiv.childNodes         ; Aus dem div alle childNodes ermit
teln

For $oNode in $oChildNodes

     If ( $sTitle = "" ) Then
         ; Es wurde bislang noch keine Überschrift der Seite gefunden.
         ; Entweder ist dieser $oNode eine Überschrift,
         ; oder sein Inhalt kann ignoriert werden,
         ; weil über der Seitenüberschrift nichts Sinnvolles stehen kann.

              If (StringInStr( $oNode.outerhtml, '<h1>' ) = 1) or (String
InStr( $oNode.outerhtml, '<h2>' ) = 1) Or (StringInStr( $oNode.outerhtml, '<h3>' )
= 1) Or (StringInStr( $oNode.outerhtml, '<h4>' ) = 1) Or (StringInStr( $oNode.out
erhtml, '<h5>' ) = 1) Or (StringInStr( $oNode.outerhtml, '<h6>' ) = 1) Then

            ; Ja, hier ist die Seitenüberschrift.
            ; Jetzt die Überschrift extrahieren:
```

```
363             $sTitle = $oNode.innertext
364
365             $HTMLParts[$part] = $sTitle
366             $part += 1
367
368             ; Den Inhalt dieses Nodes löschen, da er nicht ein zweites Mal
369             ; verarbeitet werden darf.
370             $oNode.outerhtml = ""
371
372         EndIf
373     Else
374         ; Es wurde bereits die Seitenüberschrift gefunden.
375         ; Jetzt weiter mit allen anderen Inhalten
376
377         If ($oNode.innertext = '') Then
378            $oNode.outerhtml = ''
379         Else
380
381            ; Alle HTML-Tags, die Überschriften sind:
382                If (StringInStr( $oNode.outerhtml, '<h1>' ) = 1) or (String
InStr( $oNode.outerhtml, '<h2>' ) = 1) Or (StringInStr( $oNode.outerhtml, '<h3>' )
= 1) Or (StringInStr( $oNode.outerhtml, '<h4>' ) = 1) Or (StringInStr( $oNode.out
erhtml, '<h5>' ) = 1) Or (StringInStr( $oNode.outerhtml, '<h6>' ) = 1) Then
383              ; Überschrift
384              $HTMLParts[$part] = $oNode.innertext
385              $part += 1
386             Else
387              ; keine Überschrift
388
389                  If Not (StringInStr( $oNode.outerhtml, '<div class="auto
it_do_not_print">' ) = 1) Then
390                  $HTMLParts[$part] = $oNode.outerhtml
391                  $part += 1
392              EndIf
393
394             EndIf
395
396         EndIf
397
398     EndIf
399
400  Next        ; For $oNode in $oChildNodes
401
402  ; ------------------------------------------------------------------
403
404  ; In dem Array $HTMLParts stehen jetzt die aufgetrennten Seitenteile.
405  ; Die Variable $part enthält aktuell die Anzahl der ermittelten Seitenteile.
406  ;
407  ; Wenn ein Feld in $HTMLParts mit einem < beginnt, so handelt es sich un ei
nen HTML-Tag.
408  ; Wenn ein Feld in $HTMLParts nicht mit einem < beginnt, so handelt es sich
um eine
409  ; Überschrift. Von Überschriften wird nur der reine Text gespeichert.
410  ;
411  ; $HTMLParts[0] enthält den Titel der in diesem Durchlauf behandelten Seite
$ALL_LINKS[ $URL_NO ]
412
```

```
413  ; ------------------------------------------------------------------
414
415  ; Jetzt jeweils nur einen Teil im Browser darstellen,
416  ; (also ein Feld aus $HTMLParts)
417  ; dann einen Screenshot machen,
418  ; und den Screenshot in Word einfügen.
419
420  ; ----------------------------------------------------------
421  ; Einfügen der Seitenüberschrift in Word
422
423  WinActivate( $hWord_Window_Handle, "" )         ; Dem Word-Fenster den Fokus
geben.
424
425  Send( $sTitle )
426  Send( "^l" )                ; STRG l --> Den Absatz linksbündig formatieren.
"l" wie Ludwig!
427  Send( "!1" )                 ; ALT 1 --> Dem Absatz "Überschift 1" zuweisen.
Ziffer "1"!
428  Send( "{ENTER}" )
429  Send( "{ENTER}" )
430  Send( "^e" )               ; STRG e --> Absatz zentrieren
431
432  ; Einen Moment abwarten, bis Word mit dem Einfügen fertig ist.
433  Sleep( 100 )
434
435  ; ------------------------------------------------------------------------
436  ; Jetzt die Screenshots machen
437
438  For $partnr = 1 To ($part - 1)
439
440      If Not (StringInStr( $HTMLParts[ $partnr ], '<' ) = 1) Then
441          ; Der aktuelle Teil beginnt NICHT mit einem '<', deshalb muss es eine
Überschrift sein.
442
443          ; Einfügen der Seitenüberschrift in Word
444
445          ; Dem Word-Fenster den Fokus geben.
446          WinActivate( $hWord_Window_Handle, "" )
447
448          Send( "{ENTER}" )
449          Send( $HTMLParts[ $partnr ] )
450            Send( "^l" )                   ; STRG l --> Den Absatz linksbündig
formatieren. "l" wie Ludwig!
451           Send( "!2" )                 ; ALT 2 --> Dem Absatz "Überschift 2"
zuweisen.
452          Send( "{ENTER}" )
453          Send( "^e" )               ; STRG e --> Absatz zentrieren
454
455          ; Einen Moment abwarten, bis Word mit dem Einfügen fertig ist.
456          Sleep( 200 )
457
458      Else
459
460          $oDiv.innerhtml = $HTMLParts[ $partnr ]
461
462          ; Einen Moment abwarten, damit die Inhalte im Browser auch
463          ; korrekt aufgebaut werden können.
```

```
464          Sleep( 500 )
465
466          ; ------------------------------------------------------------
467          ; Anpassen der Höhe des Browserfensters, damit FireShot
468          ; bei der Aufnahme möglichst nicht scrollen braucht
469
470          $oDoc = _IEDocGetObj( $oIE )
471              ; siehe document object http://msdn.microsoft.com/en-us/library/
ms535862%28v=vs.85%29.aspx#properties
472
473          $oBody = $oDoc.body
474                 ; siehe body object http://msdn.microsoft.com/en-us/library/
ms535205%28v=vs.85%29.aspx#properties
475
476          ; Dies ist die Höhe des Content-Teils im Browser:
477          $bodyHeight = _IEPropertyGet( $oBody, "height" )
478
479          ; Dies ist die Höhe des Browsers, ohne die angezeigte Webseite.
480           ; Also Titelleiste im Fenster, Menüzeile, Linksleiste, Statusleiste,
etc.
481          ; Dieser Wert muss ggf. an den eigenen Browser angepasst werden.
482          Local $individualBrowserHeight = 107
483
484          ; Neue Fensterhöhe setzen:
485             _IEPropertySet( $oIE, "height", $individualBrowserHeight + $body
Height )
486
487          ; ------------------------------------------------------------
488
489          ; Dem IE-Fenster den Fokus geben
490          WinActivate( $hIE_Window_Handle, "" )
491
492          ; Auf den FireShot-Button im Browser klicken.
493          ; Die Koordinaten müssen stimmen!
494          MouseClick( "left", $FireShot_X, $FireShot_Y )
495
496          ; Einen Moment abwarten, damit FireShot seine Arbeit tun kann.
497          Sleep( 1000 )
498
499          ; ------------------------------------------------------------
500          ; Einfügen des Screenshot in Word
501
502          ; Dem Word-Fenster den Fokus geben.
503          WinActivate( $hWord_Window_Handle, "" )
504
505          ; Auf die Titelleiste des Word-Fensters klicken.
506          ; MouseClick( "left", $WORD_X, $WORD_Y)
507
508          Send( "^v" )        ; STRG v --> Inhalt der Zwischenablage einfügen
509          Send( "{Enter}" )
510
511          ; Einen Moment abwarten, bis Word mit dem Einfügen fertig ist.
512          Sleep( 200 )
513
514      EndIf
515
516  Next        ; For $partnr = 1 To ($part - 1)
```

```
517
518  ; ------------------------------------------------------------------------
519
520  ; Jetzt ist eine einzelne URL aus dem Array $ALL_LINKS abgearbeitet,
521  ; also eine LOOP-Seite ist fertig.
522
523  ; ------------------------------------------------------------------------
524
525  ; Das Fenster des IE wieder schließen
526  _IEQuit( $oIE )
527
528 Next    ; For $URL_NO = $START_NO To $END_NO
529
530 ; ------------------------------------------------------------
531 ; Einfügen der Seitenüberschrift in Word
532
533 ; Dem Word-Fenster den Fokus geben.
534 WinActivate( $hWord_Window_Handle, "" )
535
536 Send( "^{HOME}" )            ; STRG POS1 --> An den Dokumentenanfang springen.
537 Send( "^l" )                 ; STRG l --> Den Absatz linksbündig formatieren.
"l" wie Ludwig!
538 Send( "Quelle: {TAB}" )
539 Send( $sURL_TOC )
540 Send( "{ENTER}" )
541 Send( "{ENTER}" )
542 Send( "Stand: {TAB}" )
543 Send( "!D" )                ; ALT SHIFT d --> Das aktuelle Datum einfügen.
544 Send( "{ENTER}" )
545 Send( "{ENTER}" )
546 Send( "Bei Bedarf:" )
547 Send( "{ENTER}" )
548 Send( "{TAB}Manuell ein Inhaltsverzeichnis einfügen." )
549 Send( "{ENTER}" )
550 Send( "{TAB}Seitenzahlen in Kopf- oder Fußzeile einfügen." )
551 Send( "{ENTER}" )
552 Send( "{ENTER}" )
553
554 ; Einen Moment abwarten, bis Word mit dem Einfügen fertig ist.
555 Sleep( 200 )
556
557 Exit
```

6.6 Impressum

Andreas Wilkens
Dr.-Kelp-Str. 5c
26160 Ofen
Tel. (04 41) 88 53 078
E-Mail: aw(at)dozaw.de
http://www.dozaw.de/

Steuer-Nr. 2369 06914805354
Die Seite http://vfhcab.eduloop.de verfolgt keine kommerziellen Ziele, sondern sieht sich selbst als kleinen Baustein der OER-Bewegung (OER = Open Educational Resources).
Bitte beachten Sie bzgl. des Haftungsausschlusses den Disclaimer dieser Webseite.
Bitte beachten Sie die Hinweise zum Copyright unter
http://vfhcab.eduloop.de/loop/Copyright
<hr>

Disclaimer

Haftung für Inhalte

Die Inhalte unserer Seiten wurden mit größter Sorgfalt erstellt. Für die Richtigkeit, Vollständigkeit und Aktualität der Inhalte können wir jedoch keine Gewähr übernehmen. Als Diensteanbieter sind wir gemäß § 7 Abs.1 TMG für eigene Inhalte auf diesen Seiten nach den allgemeinen Gesetzen verantwortlich. Nach §§ 8 bis 10 TMG sind wir als Diensteanbieter jedoch nicht verpflichtet, übermittelte oder gespeicherte fremde Informationen zu überwachen oder nach Umständen zu forschen, die auf eine rechtswidrige Tätigkeit hinweisen. Verpflichtungen zur Entfernung oder Sperrung der Nutzung von Informationen nach den allgemeinen Gesetzen bleiben hiervon unberührt. Eine diesbezügliche Haftung ist jedoch erst ab dem Zeitpunkt der Kenntnis einer konkreten Rechtsverletzung möglich. Bei Bekanntwerden von entsprechenden Rechtsverletzungen werden wir diese Inhalte umgehend entfernen.

Haftung für Links

Unser Angebot enthält Links zu externen Webseiten Dritter, auf deren Inhalte wir keinen Einfluss haben. Deshalb können wir für diese fremden Inhalte auch keine Gewähr übernehmen. Für die Inhalte der verlinkten Seiten ist stets der jeweilige Anbieter oder Betreiber der Seiten verantwortlich. Die verlinkten Seiten wurden zum Zeitpunkt der Verlinkung auf mögliche Rechtsverstöße überprüft. Rechtswidrige Inhalte waren zum Zeitpunkt der Verlinkung nicht erkennbar. Eine permanente inhaltliche Kontrolle der verlinkten Seiten ist jedoch ohne konkrete Anhaltspunkte einer Rechtsverletzung nicht zumutbar. Bei Bekanntwerden von Rechtsverletzungen werden wir derartige Links umgehend entfernen.

Urheberrecht

Die durch die Seitenbetreiber erstellten Inhalte und Werke auf diesen Seiten unterliegen dem deutschen Urheberrecht. Die Vervielfältigung, Bearbeitung, Verbreitung und jede Art der Verwertung außerhalb der Grenzen des Urheberrechtes bedürfen der

schriftlichen Zustimmung des jeweiligen Autors bzw. Erstellers. Downloads und Kopien dieser Seite sind nur für den privaten, nicht kommerziellen Gebrauch gestattet. Soweit die Inhalte auf dieser Seite nicht vom Betreiber erstellt wurden, werden die Urheberrechte Dritter beachtet. Insbesondere werden Inhalte Dritter als solche gekennzeichnet. Sollten Sie trotzdem auf eine Urheberrechtsverletzung aufmerksam werden, bitten wir um einen entsprechenden Hinweis. Bei Bekanntwerden von Rechtsverletzungen werden wir derartige Inhalte umgehend entfernen.

Datenschutz

Die Nutzung unserer Webseite ist in der Regel ohne Angabe personenbezogener Daten möglich. Soweit auf unseren Seiten personenbezogene Daten (beispielsweise Name, Anschrift oder eMail-Adressen) erhoben werden, erfolgt dies, soweit möglich, stets auf freiwilliger Basis. Diese Daten werden ohne Ihre ausdrückliche Zustimmung nicht an Dritte weitergegeben.

Wir weisen darauf hin, dass die Datenübertragung im Internet (z.B. bei der Kommunikation per E-Mail) Sicherheitslücken aufweisen kann. Ein lückenloser Schutz der Daten vor dem Zugriff durch Dritte ist nicht möglich.

Der Nutzung von im Rahmen der Impressumspflicht veröffentlichten Kontaktdaten durch Dritte zur Übersendung von nicht ausdrücklich angeforderter Werbung und Informationsmaterialien wird hiermit ausdrücklich widersprochen. Die Betreiber der Seiten behalten sich ausdrücklich rechtliche Schritte im Falle der unverlangten Zusendung von Werbeinformationen, etwa durch Spam-Mails, vor.

Quelle:
Disclaimer von eRecht24, dem Portal zum Internetrecht von Rechtsanwalt Sören Siebert
http://www.e-recht24.de/muster-disclaimer.htm

I Literaturverzeichnis

Akademischer Verein Hütte e.V.(2012). Das Ingenieurwissen; Kapitel J: Technische Informatik. (34). Springer Vieweg. Abgerufen von http://link.springer.com/chapter/10.1007/978-3-642-22850-6_10.

Achilles, Albrecht (2006). Betriebssysteme - Eine kompakte Einführung mit Linux. Springer-Verlag. Abgerufen von http://link.springer.com/book/10.1007/978-3-540-29376-7.

Brinkschulte, Uwe und Ungerer, Theo(2010). Mikrocontroller und Mikropozessoren. (3). Springer-Verlag. Abgerufen von http://link.springer.com/book/10.1007/978-3-642-05398-6.

Böttcher, Axel(2006). Rechneraufbau und Rechnerarchitektur. Springer-Verlag. Abgerufen von http://link.springer.com/book/10.1007/3-540-44731-8.

Eisenhardt, Martin; Henrich, Andreas; Sieber, Stefanie(2007). Rechner- und Betriebssysteme, Kommunikationssysteme, Verteilte Systeme. (Online-Ausgabe eines Lehrtextes der Universität Bamberg (CC BY-NC-ND)). Abgerufen von http://www.uni-bamberg.de/minf/RBKVS-Buch.

Glatz, Eduard(2010). Betriebssysteme. dpunkt.verlag GmbH.

Mandl, Peter(2013). Grundkurs Betriebssysteme: Architekturen, Betriebsmittelverwaltung, Synchronisation, Prozesskommunikation. Springer Vieweg. Abgerufen von http://link.springer.com/book/10.1007/978-3-8348-2301-4.

Plate, Jürgen(2018). Einführung Datenverarbeitungssysteme. (Online-Skript, Fachhochschule München; Fachbereich Elektrotechnik und Informationstechnik). Abgerufen von http://www.netz-mafia.de/skripten/dvs/index.html.

Russinovich, Mark; Solomon, David A.; Ionescu, Alex(2012). Windows Internals, Sixth Edition, Part 1. Microsoft Press.

Russinovich, Mark; Solomon, David A.; Ionescu, Alex(2012). Windows Internals, Sixth Edition, Part 2. Microsoft Press.

Solomon, David A.; Russinovich, Mark E.(2006). Windows Internals. (4). Microsoft Press.

Strelen, Johann Christoph(2012). Betriebssysteme und parallele Programme. bookboon.com (Ventus Publishing ApS). Abgerufen von http://bookboon.com/de/betriebssysteme-und-parallele-programme-ebook.

Tanenbaum, Andrew S.(2005). Computerarchitektur: Strukturen - Konzepte - Grundlagen. (5). Pearson Studium.

Tanenbaum, Andrew S.(2009). Moderne Betriebssysteme. (3). Pearson Studium.

Wüst, Klaus(2011). Mikroprozessortechnik: Grundlagen, Architekturen, Schaltungstechnik und Betrieb von Mikroprozessoren und Mikrocontrollern. (4). Vieweg+Teubner Verlag. Abgerufen von http://link.springer.com/book/10.1007/978-3-8348-9881-4.

Weitere Literatur

Coy, Wolfgang(1992). Aufbau und Arbeitsweise von Rechenanlagen. Verlag Vieweg.

Ehses, Erich; Köhler, Lutz; Riemer, Petra; Stenzel, Horst; Victor, Frank(2012). Systemprogrammierung in Unix/Linux: Grundlegende Betriebssystemkonzepte und praxisorientierte Anwendungen. Vieweg+Teubner Verlag. Abgerufen von http://link.springer.com/book/10.1007/978-3-8348-8277-6.

Flik, Thomas(2005). Mikroprozessortechnik und Rechnerstrukturen. Springer-Verlag. Abgerufen von http://link.springer.com/book/10.1007/b137981.

Plate, Jürgen(2014). Einführung in Betriebssyteme. (Online-Skript, Fachhochschule München; Fachbereich Elektrotechnik und Informationstechnik). Abgerufen von http://www.netzmafia.de/skripten/bs/index.html.

Tanenbaum, Andrew S.(2008). Modern Operating Systems. (3). Prentice Hall.

Tanenbaum, Andrew S.; Austin, Todd(2012). Structured Computer Organization. Prentice Hall.

Witt, Kurt-Ulrich(1995). Elemente des Rechneraufbaus. Carl Hanser Verlag.

II Abbildungsverzeichnis

Abb. 1: Das Mainboard und die ausgebauten Komponenten 17

Abb. 2: John von Neumann (um 1940) 17

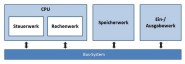


Abb. 3: Von-Neumann-Architektur 19

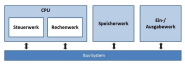


Abb. 4: Von-Neumann-Architektur 20

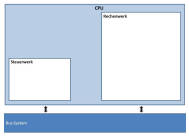


Abb. 5: CPU mit Steuerwerk und Rechenwerk 22

Abb. 6: Eine einfache Sicht auf die Arbeitsweise des Steuerwerks 23

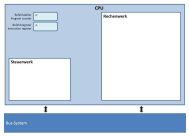


Abb. 7: Steuerwerk mit Befehlszähler und Befehlsregister 24

Abb. 8: Eine detailliertere Sicht auf die Arbeitsweise des Steuerwerks 26

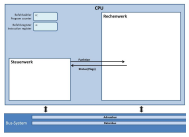


Abb. 9: Steuerwerk mit Adress- und Datenbus 27

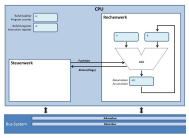


Abb. 10: Rechenwerk mit ALU 29

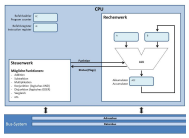


Abb. 11: Funktionen des Rechenwerks, die vom Steuerwerk in Auftrag gegeben werden können 30

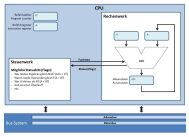


Abb. 12: Statusbits des Rechenwerks, die vom Steuerwerk ausgewertet werden können 30

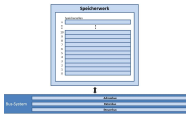


Abb. 13: Speicherwerk mit Bus-System 33

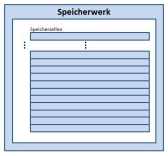


Abb. 14: Abbildung zu Aufgabe 2 34

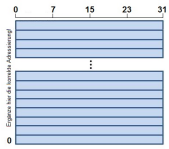


Abb. 15: Abbildung zu Aufgabe 4 35

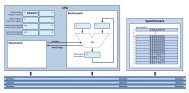


Abb. 16: Abbildung zu Aufgabe 4 36

Abb. 17: Ein-/Ausgabewerk 37

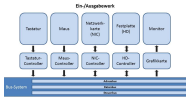


Abb. 18: Gesamtbild eines Von-Neumann-Rechners 38

Abb. 19: Arbeitsweise des Steuerwerks II 39

Abb. 20: Von-Neumann-Zyklus in der Arbeitsweise des Steuerwerks 40

Abb. 21: Von-Neumann-Zyklus 40

Abb. 22: Gesamtbild eines Von-Neumann-Rechners 44

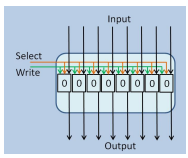


Abb. 23: Register mit 8 Speicherzellen 45

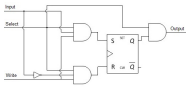


Abb. 24: Aufbau einer einzelnen Speicherzelle 46

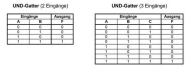


Abb. 25: Wahrheitstafeln von UND-Gattern mit zwei bzw. drei Eingängen 46

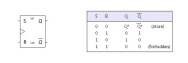


Abb. 26: RS-Flip-Flop und die zugehörige Wahrheitstafel 47

Abb. 27: Unterschiedliche Gatter 50

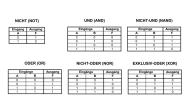



Abb. 28: Wahrheitstafel für Gatter 51

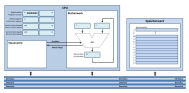


Abb. 29: Abbildung zu Aufgabe 6 59

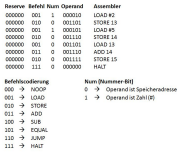
000 → NOOP
001 → LOAD
010 → STORE
011 → ADD
100 → SUB
101 → EQUAL
110 → JUMP
111 → HALT

Abb. 30: Einfacher Befehlssatz....59

Abb. 31: Von-Neumann-Zyklus....63

000 → NOOP
001 → LOAD
010 → STORE
011 → ADD
100 → SUB
101 → EQUAL
110 → JUMP
111 → HALT

Abb. 32: Einfacher Befehlssatz....64



Reserve Befehl Num Operand Assembler
000000 001 1 000010 LOAD #2
000000 010 0 001101 STORE 13
000000 001 1 000101 LOAD #5
000000 010 0 001110 STORE 14
000000 001 0 001101 LOAD 13
000000 011 0 001110 ADD 14
000000 010 0 001111 STORE 15
000000 111 0 000000 HALT

Befehlscodierung
000 → NOOP
001 → LOAD
010 → STORE
011 → ADD
100 → SUB
101 → EQUAL
110 → JUMP
111 → HALT

Num (Nummer-Bit)
0 → Operand ist Speicheradresse
1 → Operand ist Zahl (#)

Abb. 33: Abbildung zu Aufgabe 1....65

Abb. 34: Der aktuelle Betrachtungsstand: Nicht mehr als ein einfacher Taschenrechner....74

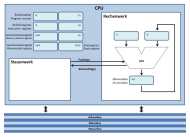


Abb. 35: Stackregister auf der CPU....75

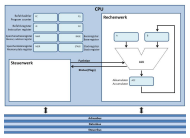


Abb. 36: CPU mit Basisregister....77

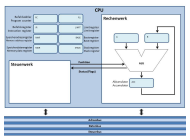


Abb. 37: CPU mit Limitregister....83

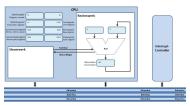


Abb. 38: CPU mit Interrupt-Controller....86

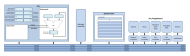


Abb. 39: Interrupt-Controller im Gesamtbild....86

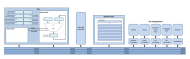


Abb. 40: Abbildung zu Aufgabe 1....93

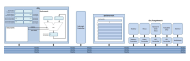


Abb. 41: CPU, Interrupt-Controller, Speicherwerk und weitere Controller mit ihren Komponenten....94

Abb. 42: Allgemeiner Aufbau eines Controllers....95

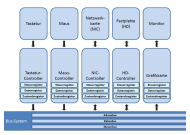


Abb. 43: Verschiedene Controller mit eigenen Registersätzen....96

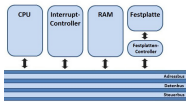


Abb. 44: Vereinfachter Rechner, noch ohne DMA-Controller 102

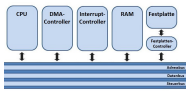


Abb. 45: Vereinfachter Rechner, mit DMA-Controller 104

Abb. 46: Aufbau eines DMA-Controllers 106

Abb. 47: DMA-Datentransfer als Schleife 108

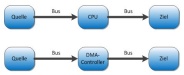


Abb. 48: Daten müssen den Umweg über die CPU nehmen 110

Abb. 49: Umweg über DMA-Controller 111

Abb. 50: Aufbau eines DMA-Controllers 111

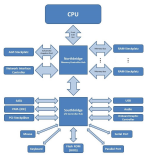


Abb. 51: North- und Southbridge als zentrale Verbindungspunkte 117

Abb. 52: Betriebssystem und Anwendungprogramm(e) vermitteln zwischen Benutzer und Hardware 124

Abb. 53: 138

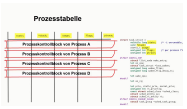


Abb. 54: 156

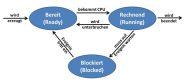


Abb. 55: Prozesszustände mit Übergängen 158

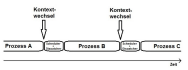


Abb. 56: Scheduler und Dispatcher werden bei einem Kontextwechsel aktiv 169

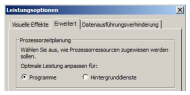


Abb. 57: Windows 7 Prozessorzeitplanung 181

Abb. 58: Windows Server 2008 R2 Prozessorzeitplanung 181

Abb. 59: Die im Video gezeigte Seitentabelle 243

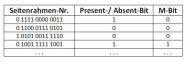


Abb. 60: Seitentabelle mit zusätzlicher Spalte für M-Bit 255

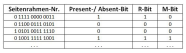


Abb. 61: Seitentabelle mit zusätzlicher Spalte für R-Bit 259

Abb. 62: Geräteverwaltung als Schnittstelle zwischen Peripheriegeräten und Betriebssystem 271

Abb. 63: Treiber als Mittler zwischen Controller und Geräteverwaltung 274

Abb. 64: Ein initialisierter Controller 275

Abb. 65: Geräte-Manager von Windows 7 276

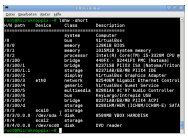


Abb. 66: 'lshw -short' unter Linux 277

Abb. 67: Der Treiber als Mittler zwischen Geräteverwaltung und Geräte-Controller 278

Abb. 68: Für jedes Gerät (bzw. Controller) ein eigener Treiber 281

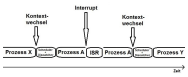


Abb. 69: Ein Interrupt mit seiner zugehörigen Interruptbehandlungsroutine (ISR) unterbricht den Prozess A auf der CPU 283

Abb. 70: Der Treiber als Mittler zwischen Geräteverwaltung und Geräte-Controller 286

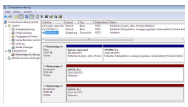


Abb. 71: Zwei Volumes auf drei Partitionen unter Windows 7 302

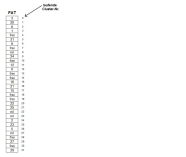


Abb. 72: Beispiel einer FAT 304

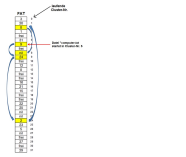


Abb. 73: Clusterspur einer Datei in der FAT 305

Abb. 74: Ein Aufruf von 'ntfsinfo.exe c:' unter Windows 7 307

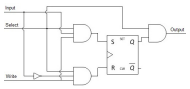


Abb. 75: Um diese Komponente geht es in dieser Aufgabe.....310

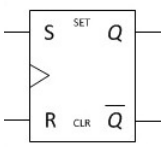


Abb. 76: Flip-Flop.....311

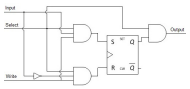
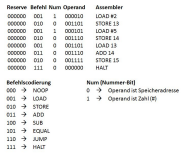


Abb. 77: Um diese Komponente geht es in dieser Aufgabe.....312



Reserve	Befehl	Num	Operand	Assembler
000000	001	1	000010	LOAD #2
000000	010	0	001101	STORE 13
000000	001	1	000101	LOAD #5
000000	010	0	001110	STORE 14
000000	001	0	001101	LOAD 13
000000	011	0	001110	ADD 14
000000	010	0	001111	STORE 15
000000	111	0	000000	HALT

Befehlscodierung

000	→	NOOP
001	→	LOAD
010	→	STORE
011	→	ADD
100	→	SUB
101	→	EQUAL
110	→	JUMP
111	→	HALT

Num (Nummer-Bit)

0	→	Operand ist Speicheradresse
1	→	Operand ist Zahl (#)

Abb. 78: Abbildung zu Aufgabe 1..... 314

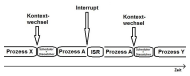


Abb. 79: Ein Interrupt mit seiner zugehörigen Interruptbehandlungsroutine (ISR) unterbricht den Prozess A auf der CPU.....333

III Medienverzeichnis

Med. 1: Überblick: Vom Anwender zur digitalen Schaltung (00:54) ... 15
Med. 2: Erstmal aufschrauben! (08:31) ... 16
Med. 3: Von-Neumann-Architektur (01:35) ... 18
Med. 4: Von-Neumann-Flaschenhals (01:53) ... 20
Med. 5: Befehlszähler und Befehlsregister im Zusammenspiel mit dem Bus-System (03:07) ... 24
Med. 6: Detailliertere Arbeitsweise des Steuerwerks (06:01) ... 26
Med. 7: Arbeitsweise des Rechenwerks (02:26) ... 29
Med. 8: Arbeitsweise des Speicherwerks (02:06) ... 32
Med. 9: Von-Neumann-Zyklus und die Arbeitsweise des Steuerwerks (02:22) ... 39
Med. 10: Animation der Zusammenarbeit (01:02) ... 41
Med. 11: Aufbau und Arbeitsweise eines Registers (04:40) ... 45
Med. 12: Arbeitsweise einer Speicherzelle (04:49) ... 46
Med. 13: Gatter und ihre Wahrheitstafeln (02:56) ... 51
Med. 14: Überblick: Prozessoren und ihre Befehle (01:15) ... 52
Med. 15: Vom Quellcode zum Prozessor (14:09) ... 55
Med. 16: Stackregister und Arbeitsweise des Stacks (04:12) ... 75
Med. 17: Basisregister als Voraussetzung für die Unterbringung mehrerer Programme im Hauptspeicher (03:10) ... 79
Med. 18: Speicheraufteilung bei mehreren Programmen im RAM (03:08) ... 80
Med. 19: Swapping (02:05) ... 82
Med. 20: Limitregister zum Speicherschutz (02:33) ... 84
Med. 21: Interrupt-Controller (01:30) ... 85
Med. 22: Was passiert bei einem Interrupt? (02:14) ... 88
Med. 23: Ein- und Ausgabe mit Festplatte und Interrupts (04:00) ... 98
Med. 24: Grundlagen virtueller Speicherverwaltung mit MMU (04:47) ... 113
Med. 25: Kontextwechsel unter Windows 7 ... 141
Med. 26: UNIX/Linux: Prozesserzeugung mit fork() (12:19) ... 144
Med. 27: Prozesskontrollblock im Linux-Quellcode ... 149
Med. 28: Prozessverwaltung unter Windows ... 160
Med. 30: Race Conditions (04:50) ... 188
Med. 31: Deadlocks erkennen mit Hilfe eines Betriebsmittelgraphen (09:46) ... 223
Med. 32: Grundlagen virtueller Speicherverwaltung mit MMU (04:47) ... 237
Med. 33: MMU: Adressumrechnung mit einstufiger Seitentabelle (06:32) ... 242
Med. 34: MMU: Page fault bei der Adressumrechnung (06:32 Gesamtlänge, Page fault ab 05:30) ... 248
Med. 35: Was passiert bei einem Interrupt? (02:14) ... 283
Med. 36: Ein- und Ausgabe mit Festplatte und Interrupts (04:00) ... 283

IV Listingverzeichnis

List. 1: Beispiel: Java-Threads ... 166
List. 2: Listing 1: Beispiel für Race Conditions ... 188
List. 3: Listing 1: Beispiele für kritische und unkritische Abschnitte ... 191
List. 4: Beispiel: Aktives Warten ... 196
List. 5: Listing 1: Beispiele für kritische und unkritische Abschnitte ... 207
List. 6: Listing 2: Ergänzt um einen Mutex ... 208
List. 7: Beispiel: Aktives Warten ... 211

V Aufgabenverzeichnis

Aufg. 1: Größer als....31
Aufg. 2: Speicherzelle 3....32
Aufg. 3: Von dezimal zu binär....33
Aufg. 4: Die 4 GiB-Grenze....34
Aufg. 5: Der Weg der Daten....36
Aufg. 6: Pfeilrichtungen....37
Aufg. 7: Wettbewerb in deiner Lerngruppe....38
Aufg. 8: Animation....42
Aufg. 9: lda und sta....42
Aufg. 10: Hellseher....42
Aufg. 11: See How The CPU Works In One Lesson....42
Aufg. 12: Visual 6502....43
Aufg. 13: Flip-Flop I....47
Aufg. 14: Flip-Flop II....47
Aufg. 15: Flip-Flop III....48
Aufg. 16: Simulationsframework Hades....48
Aufg. 17: Forbidden (RS-Flip-Flop)....48
Aufg. 18: Sum Program....55
Aufg. 19: Count Program....57
Aufg. 20: EQUAL-Befehl....58
Aufg. 21: JUMP-Befehl....58
Aufg. 22: Assembler-2-Hochsprache....58
Aufg. 23: Speicherzellen für die Befehle....58
Aufg. 24: Klassifizierungen....61
Aufg. 25: Operand oder nicht?....62
Aufg. 26: Wenn der Operand keine Adresse ist....62
Aufg. 27: Entwicklung eines Befehlssatzes....64
Aufg. 28: Nur sechs Bit....66
Aufg. 29: Adressierungsarten....67
Aufg. 30: Finde es heraus....68
Aufg. 31: Adressierungsarten....69
Aufg. 32: Programm-2-Prozess....71
Aufg. 33: VNA vs. Harvard....72
Aufg. 34: RISC vs. CISC....73
Aufg. 35: Welche Situation gilt?....79
Aufg. 36: Zusammenhang mit Basisregister....82
Aufg. 37: Fragmentierter Hauptspeicher....82
Aufg. 38: Zugriff auf freien Speicherbereich....84
Aufg. 39: Speicherbereich eines Prozesses vergrößern....84
Aufg. 40: Prüfe die Bedingungen!....88
Aufg. 41: Schätze die Folgen ab!....88
Aufg. 42: Wer wenn nicht das Steuerwerk?....89
Aufg. 43: Vier Bit....89
Aufg. 44: Weiter laufen lassen?....90
Aufg. 45: Ausnahme und Interrupt....90
Aufg. 46: Integration des Taktgebers....92
Aufg. 47: Betriebssystem und Prozesse im Wechsel....93
Aufg. 48: Lesen und/oder schreiben?....96
Aufg. 49: Alternative zum Warten?....100
Aufg. 50: Interrupt und die Register....100
Aufg. 51: Zwei Bit?....100
Aufg. 52: Mehrere Register je Typ....101
Aufg. 53: Mal wieder das Betriebssystem....101
Aufg. 54: Datentransfer (noch) ohne DMA....102
Aufg. 55: Datentransfer mit DMA....104

Aufg. 56: DMA und CPU ... 105
Aufg. 57: Arten des DMA-Transfers ... 106
Aufg. 58: Start- und Zieladresse ändern sich ... 108
Aufg. 59: DMA mit RAM und RAM? ... 109
Aufg. 60: Datentransfer mit DMA ... 109
Aufg. 61: Umweg über DMA-Controller ... 111
Aufg. 62: Eindeutig oder mehrdeutig? ... 114
Aufg. 63: Umrechnung und was noch? ... 115
Aufg. 64: Virtuelle Größen ... 115
Aufg. 65: Swapping bei virtueller Speicherverwaltung? ... 115
Aufg. 66: Was wäre wenn... (I) ... 125
Aufg. 67: Was wäre wenn... (II) ... 125
Aufg. 68: Treiber ... 126
Aufg. 69: Datei nur exklusiv oder gemeinsam nutzbar? ... 129
Aufg. 70: Welche weiteren Betriebsmittel? ... 130
Aufg. 71: Liste von Systemaufrufen ... 137
Aufg. 72: Wo ist der Systemaufruf? ... 138
Aufg. 73: Sind Kontextwechsel positiv oder negativ? ... 141
Aufg. 74: Perfmon unter Windows ... 142
Aufg. 75: Vater- und Sohn-Variable ... 145
Aufg. 76: fork gleich mehrmals ... 145
Aufg. 77: Fork und DMA? ... 146
Aufg. 78: Ein anderes Programm mittels fork starten ... 146
Aufg. 79: Die Spalten der Prozesstabelle ... 157
Aufg. 80: Merke dir den Zustand! ... 159
Aufg. 81: Process Explorer unter Windows ... 160
Aufg. 82: Geschwindigkeitsvorteil ... 165
Aufg. 83: Vor- und Nachteil des Betriebsmittelzugriffs ... 165
Aufg. 84: Threadzustände ... 165
Aufg. 85: Unerwünschte Nebeneffekte bei mehreren Threads ... 166
Aufg. 86: Starte die Threads! ... 167
Aufg. 87: Ein Dispatcher muss tun, was ein Dispatcher tun muss ... 169
Aufg. 88: Früher als spätestens ... 171
Aufg. 89: Der Nächste ist der Vorherige? ... 171
Aufg. 90: Wenn ein Prozess blockiert ... 171
Aufg. 91: Scheduling-Ziele ... 173
Aufg. 92: Ein Hauch von Texas ... 174
Aufg. 93: FCFS animiert ... 175
Aufg. 94: Interessante FCFS-Situation ... 175
Aufg. 95: FCFS auf deinem Rechner ... 176
Aufg. 96: SPN animiert ... 176
Aufg. 97: Kurz oder lang ... 177
Aufg. 98: Blockieren und Interrupts ... 177
Aufg. 99: Erzeugt oder was? ... 177
Aufg. 100: SJF auf deinem Rechner ... 178
Aufg. 101: SRTN animiert ... 178
Aufg. 102: SJF vs. SRTN ... 179
Aufg. 103: Wie kurz bist du? ... 179
Aufg. 104: RR animiert ... 180
Aufg. 105: Zu kurz oder zu lang ... 180
Aufg. 106: RR und E/A-lastige Prozesse ... 182
Aufg. 107: Verschiedene Prozesse mit gleicher Priorität ... 182
Aufg. 108: Was passiert bei niedriger Priorität? ... 183
Aufg. 109: Erweitere PS ... 183
Aufg. 110: Herauf statt herunter! ... 183
Aufg. 111: Scheduling-Kriterien ... 185
Aufg. 112: Wann genau finden Kontextwechsel statt? ... 187

Aufg. 113: Gemeinsam genutzes Betriebsmittel ... 189
Aufg. 114: Kleiner Fehler bei den Erklärungen im Video ... 190
Aufg. 115: Abschnitte in Thread_B ... 192
Aufg. 116: Viele andere Betriebsmittel ... 193
Aufg. 117: Warum Systemaufrufe? ... 193
Aufg. 118: Zwei Prozesse und kritische Abschnitte ... 193
Aufg. 119: Die Bedeutung des Semikolons! ... 196
Aufg. 120: Aktives Warten implementieren ... 196
Aufg. 121: Warte aktiv! ... 198
Aufg. 122: Der Nachteil des aktiven Wartens ... 198
Aufg. 123: Kontextwechsel im ungünstigsten Moment ... 200
Aufg. 124: Mehrere ungünstigste Momente ... 200
Aufg. 125: Speicherzelle 13 ... 203
Aufg. 126: XCHG bei Intel ... 203
Aufg. 127: Gilt der Nachteil noch? ... 203
Aufg. 128: Warum up() und down()? ... 205
Aufg. 129: P() und V() sind atomar ... 205
Aufg. 130: P() und V() und die Zustände ... 206
Aufg. 131: V()-Operation und die Warteschlange ... 206
Aufg. 132: Mutex in Action ... 210
Aufg. 133: Aktives Warten vs. Mutex ... 211
Aufg. 134: Mutex und Prozesse bei Wikipedia ... 213
Aufg. 135: Applet zum wechselseitigen Ausschluss ... 214
Aufg. 136: Applet zur Reihenfolgedurchsetzung ... 214
Aufg. 137: Reihenfolgedurchsetzung vs. Wechselseitiger Ausschluss ... 215
Aufg. 138: Applet zum Erzeuger- / Verbraucherproblem ... 216
Aufg. 139: Applet zum Philisophenproblem ... 216
Aufg. 140: Deadlock-Philosophen ... 220
Aufg. 141: Applet zum Philisophenproblem ... 221
Aufg. 142: Deadlocks erkennen mit Hilfe eines Betriebsmittelgraphen ... 223
Aufg. 143: Zeichne den Betriebsmittelgraph und erkenne den Deadlock! ... 223
Aufg. 144: Was tun bei erkanntem Deadlock? ... 225
Aufg. 145: Wie oft nach einem Deadlock suchen? ... 225
Aufg. 146: Implementiere! ... 226
Aufg. 147: Spooling ... 227
Aufg. 148: Überweis' mal was ... 227
Aufg. 149: Überweisen ohne Deadlock? ... 228
Aufg. 150: IPC-Grundbegriffe ... 230
Aufg. 151: Lock ... 232
Aufg. 152: Shared Memory realisieren ... 232
Aufg. 153: Deine tägliche Socket-Dosis ... 234
Aufg. 154: Rahmen vs. Seite ... 238
Aufg. 155: Rechne nach! ... 238
Aufg. 156: Besonders schnell ist besonders wichtig! ... 240
Aufg. 157: Wenn du es umrechnen kannst, dann kannst du es auch verstehen! ... 241
Aufg. 158: Eine Seitentabelle hat es in sich ... 243
Aufg. 159: Nur 64 KiB RAM ... 244
Aufg. 160: Du bist jetzt die MMU ... 245
Aufg. 161: Swapping vs. Paging I ... 247
Aufg. 162: Swapping vs. Paging II ... 247
Aufg. 163: Paging und die Seitentabelle ... 248
Aufg. 164: Page Fault or No Page Fault? ... 248
Aufg. 165: Was passiert bei einem Page fault? ... 249
Aufg. 166: Wenn eine Seite Programmtext enthält ... 253
Aufg. 167: Wenn eine Seite Daten enthält ... 254
Aufg. 168: M-Bit beim Einlagern ... 255
Aufg. 169: Die MMU und Seitenersetzungsverfahren ... 257

Aufg. 170: Implementiere den optimalen Seitenersetzungsalgorithmus 258
Aufg. 171: Optimales M-Bit 258
Aufg. 172: Lesen, schreiben, R-Bit? 260
Aufg. 173: Vier Klassen ohne Seiten? 261
Aufg. 174: Auch das M-Bit löschen? 262
Aufg. 175: FIFO und die Seitentabelle 262
Aufg. 176: Angesprochen oder nicht? 263
Aufg. 177: Wenn alle die zweite Chance nutzen 263
Aufg. 178: Die 80/20-Regel und das Working Set 264
Aufg. 179: Realisiere Shared Memory 267
Aufg. 180: Weniger Seitenfehler durch Shared Memory? 267
Aufg. 181: Shared Memory mit Daten statt Programmtext 268
Aufg. 182: Überfliege erneut die Kommunikation 270
Aufg. 183: Pfeilrichtungen und Kommunikationswege 272
Aufg. 184: Systemaufruf zwischen User- und Kernel-Mode 272
Aufg. 185: Pfeil des Systemaufrufs 273
Aufg. 186: Du hast doch schon mal einen Treiber installiert, oder? 274
Aufg. 187: Treiberausführung im User- oder im Kernel-Mode? 275
Aufg. 188: Befehle des NEC PD765 FDC 280
Aufg. 189: Several registers in a stack 281
Aufg. 190: Abwarten! - Und Tee trinken? 282
Aufg. 191: Interruptbehandlung als Tätigkeit 284
Aufg. 192: Nur der Treiber! 284
Aufg. 193: DMA und Interrupts - Wie war das nochmal? 285
Aufg. 194: Geräte und ihre Interrupts 285
Aufg. 195: Systemaufruf im Praxisbeispiel 291
Aufg. 196: Prozesszustände im Praxisbeispiel 291
Aufg. 197: Interrupt im Praxisbeispiel 291
Aufg. 198: Just a simple assembler instruction 292
Aufg. 199: CPU-Registerinhalte sichern im Praxisbeispiel 292
Aufg. 200: Ein Mausklick und die Folgen 294
Aufg. 201: Hier schreibt die Maus 294
Aufg. 202: Geeignete Speichermedien 298
Aufg. 203: CRUD für Verzeichnisse? 299
Aufg. 204: Setze die Datei zusammen! 306
Aufg. 205: Lösche die Datei! 307
Aufg. 206: Speichere eine neue Datei! 307
Aufg. 207: Von-Neumann-Architektur 309
Aufg. 208: Von-Neumann-Flaschenhals 309
Aufg. 209: Von-Neumann-Zyklus 310
Aufg. 210: 8-Bit-Register 310
Aufg. 211: Bitfolge in ein Register schreiben 310
Aufg. 212: Rund um diese Komponente 310
Aufg. 213: Flip-Flop 311
Aufg. 214: Ermittle die Werte 311
Aufg. 215: ADD im Einadressformat 312
Aufg. 216: Dreiadressformat 313
Aufg. 217: Entwicklung eines Befehlssatzes 313
Aufg. 218: Adressierungen des Hauptspeichers 315
Aufg. 219: Adressierungsarten 315
Aufg. 220: Prozess 315
Aufg. 221: Von-Neumann vs. Harvard 315
Aufg. 222: Basis- und Limit-Register 316
Aufg. 223: Interrupt-Gründe 316
Aufg. 224: DMA 316
Aufg. 225: Infos für den DMA-Controller 316
Aufg. 226: DMA und Interrupts 316

Aufg. 227: MMU ... 317
Aufg. 228: Aufgabe der MMU ... 317
Aufg. 229: Virtuelle Größen ... 317
Aufg. 230: North- und Southbridge ... 317
Aufg. 231: Betriebsmittel ... 318
Aufg. 232: Kernel- und User-Mode ... 318
Aufg. 233: Sinn von Kernel- und User-Mode ... 318
Aufg. 234: Systemaufruf ... 318
Aufg. 235: Prozesszustände ... 319
Aufg. 236: Von 'rechnend' nach 'bereit' ... 319
Aufg. 237: Rund um die Prozessbegriffe ... 319
Aufg. 238: Kontextwechsel ... 320
Aufg. 239: Prozesserzeugung unter Unix/Linux ... 320
Aufg. 240: Prozesskontrollblock ... 320
Aufg. 241: Alle Prozesskontrollblöcke ... 320
Aufg. 242: Prozess und Thread ... 320
Aufg. 243: Scheduler und Dispatcher ... 321
Aufg. 244: Scheduling-Ziele ... 321
Aufg. 245: Scheduling ... 321
Aufg. 246: SJF und FCFS ... 321
Aufg. 247: RR und FCFS ... 321
Aufg. 248: (Non-) preemptive ... 322
Aufg. 249: Preemptive oder non-preemptive ... 322
Aufg. 250: Scheduling für P1 bis P5 ... 322
Aufg. 251: Scheduling mit Priorität ... 322
Aufg. 252: Kritischer Abschnitt ... 323
Aufg. 253: Aktives Warten mit while ... 323
Aufg. 254: Nachteil des aktiven Wartens ... 323
Aufg. 255: TSL ... 324
Aufg. 256: Aufgabe des TSL-Befehls ... 324
Aufg. 257: Problemlöser TSL ... 324
Aufg. 258: Semaphor ... 324
Aufg. 259: Mutex ... 324
Aufg. 260: Arbeitsweise eines Mutex ... 325
Aufg. 261: Arbeitsweise eines Zählsemaphors ... 325
Aufg. 262: Aktiv wartender Semaphor? ... 325
Aufg. 263: Ein Abend an der Bar ... 325
Aufg. 264: Deadlock ... 326
Aufg. 265: Deadlock-Beispiel ... 326
Aufg. 266: Betriebsmittelgraph ... 326
Aufg. 267: Bedingungen nach Coffman ... 327
Aufg. 268: Auszahlung mit Limit ... 327
Aufg. 269: Swapping und Paging ... 328
Aufg. 270: Optimale Seitenersetzung ... 328
Aufg. 271: LRU und NRU ... 328
Aufg. 272: Arbeitsweise LRU ... 328
Aufg. 273: Arbeitsweise NRU ... 329
Aufg. 274: Spalten der Seitentabelle bei NRU ... 329
Aufg. 275: Umstände beim Seitenfehler ... 329
Aufg. 276: Auslagern bei NRU ... 329
Aufg. 277: Virtuelle Speicherverwaltung ... 329
Aufg. 278: Speicher und Seitentabellen ... 332
Aufg. 279: Geräte ... 332
Aufg. 280: Zusammenspiel ... 332
Aufg. 281: Aufgaben eines Treibers ... 332
Aufg. 282: Treiber-Hersteller ... 333
Aufg. 283: Gefährlicher Treiber ... 333

Aufg. 284: ISR und Registerinhalte ... 333
Aufg. 285: Block- vs. zeichenorientiert ... 334
Aufg. 286: Beispiele für block- und zeichenorientiert ... 334
Aufg. 287: Zusammenspiel ... 334
Aufg. 288: CRUD ... 335
Aufg. 289: Dateisysteme ... 335
Aufg. 290: ... 335
Aufg. 291: FAT ... 335
Aufg. 292: NTFS ... 335
Aufg. 293: Arbeiten mit der FAT ... 336

VI Index

#
8-Bit-Register|Register, 8-Bit ... 45
Stack, POP ... 75

A
Adressierungsart|Operand ... 66
Adressierung, indiziert mit Verschiebung ... 66
Aktives Warten ... 194
Aufbau, Controller|Controller, Aufbau ... 94
Arbeitsweise, CPU|CPU, Arbeitsweise ... 41
Arbeitsweise, MMU ... 240, ... 112
ALU ... 49, ... 28
Arithmetisch-logische Einheit ... 49
Aufbau, ALU ... 49
Arbeitsweise, ALU ... 49
Arithmetic-logical Unit ... 49
Arbeitsweise, Speicherwerk ... 49
Aufbau, Speicherwerk ... 49
Aufbau, Bus ... 50
Arbeitsweise, Bus ... 50
Aufbau, Speicherzelle|Speicherzelle, Aufbau ... 45
Arbeitsweise, Speicherzelle|Speicherzelle, Arbeitsweise ... 45
Ausgangsleitung|Ausgangsleitung, RS-FlipFlop|RS-Flip-Flop, Ausgangsleitung|Flip-Flop, Ausgangsleitung ... 45
Arbeitsweise, RS-Flip-Flop ... 45
Aufgaben Treiber ... 275
aktiver Prozess|Prozess, aktiv ... 76
Adressierung, unmittelbar ... 76
Arithmetisch-logischer Befehl|Befehl, arithmetisch-logisch ... 59
Ablauf, DMA ... 102
atomare Aktion ... 199
Aktion, atomar ... 199
atomare Operation ... 199
Auflistung, Dateisysteme ... 298
absolute Adressierung ... 68, ... 69
Adressierung, effektiv ... 68, ... 69, ... 68
Anwedungsprogramm ... 121
Auflistung, Betriebssysteme ... 121
Adressumrechnung mit einstufiger Seitentabelle ... 241, ... 248
Akkumulator ... 69, ... 67, ... 68
Asynchrone Kommunikation ... 230
Anycast ... 230
Ausführen von Java-Applets ... 348
Abschnitt, kritisch ... 190
Abschnitt, unkritisch ... 190
Adressbus ... 116, ... 22
AGP-Grafikkarte ... 116
Ablauf, kritisch ... 188
Arbeitsweise, Rechenwerk ... 28
Adressierung, unsichtbar ... 67
Ausnahme ... 90
Arbeitsweise, Speicherwerk|Speicherwerk, Arbeitsweise ... 31
Arbeitsweise, Stack ... 75
Anweisung ... 70
ausführbare Datei ... 53
Assembler ... 53
Aufgabe, Betriebssystem ... 130
Adressierung, registerindirekt ... 66, ... 68
Adressierung, versteckt ... 76
Arbeitsweise, DMA ... 102
Adressierung, absolut ... 68, ... 69
Adressierung, direkt ... 69
Adresse, virtuell ... 112
arithmetic-logical Unit ... 28
arithmetisch-logische Einheit ... 28

B
Busy, Controller ... 94
Bus, Arbeitsweise ... 50
Bus, Aufbau ... 50
Basisregister ... 76
base register ... 76
binäre Codierung, Befehl ... 60
Befehlsformat ... 60
Befehlssatz, CPU|CPU, Befehlssatz ... 59
Befehlsart ... 59
Befehl, Systemsteuerbefehl ... 59
Betriebsmittel|Ressource ... 126
Betriebsmittel ... 127
Betriebsmittel, gemeinsam nutzbar ... 127
Betriebssystemarchitektur ... 131
Betriebssystemarchitektur, monolithisch ... 131
Betriebssystemarchitektur, schichtenorientiert ... 131
Betriebssystemarchitektur, Mikrokern ... 131
Betriebssystemarchitektur, Unix ... 131
Betriebssystemarchitektur, Linux ... 131
Betriebssystemarchitektur, Android ... 131
Betriebssystemarchitektur, Windows ... 131
Blockorientiertes Gerät ... 288
block device ... 288
Betriebsmittelgraph ... 222
Belegungs-Anforderungs-Graph ... 222
Betriebssystem ... 121
Betriebssystem, Liste ... 121
Betriebssystem, Geschichte ... 123
Broadcast ... 230
Beispiel Systemaufruf ... 134
Befehl, CPU ... 52
Bereit, Prozesszustand ... 158
Blockiert, Prozesszustand ... 158
Breite, Steuerbus|Steuerbus, Breite ... 31
Breite, Adressbus|Adressbus, Breite ... 31
Breite, Datenbus ... 31
Befehlszähler|PC, Befehlszähler|Program counter|PC, Program Counter|Befehlsregister|IR, Befehlsregister|Instruction Register|IR, Instruction Register ... 22
Bus, Paralleler Bus|Binary Unit System|Paralleler Bus ... 22

Busbreite|Breite, Bus ... 22
Bus, Datenbus ... 22
Batch-Job ... 133
Bytecode ... 53
Beispiel, Maschinencode ... 53
Betriebssystem, zentrale Aufgabe ... 130
Betriebsmittelverwaltung ... 130
Busy Waiting ... 194
Befehl, binäre Codierung ... 60
Befehl, Programmsteuerbefehl ... 59
Betriebsmittel, exklusiv nutzbar ... 127
Byte ... 347
Bit ... 347
Bedingungen präzise Unterbrechung ... 85
Bus, Adressbus ... 22

C
Controller, Ready ... 94
Controller, Busy ... 94
Controller, Read Error ... 94
Controller, Write Error ... 94
Controller, Out of paper ... 94
CISC ... 72
Complex Instruction Set Computer ... 72
Client ... 131
Computerarchitektur ... 14
CreateProcess ... 146
CRUD-Operationen ... 297
Circular wait condition, Deadlock ... 222, ... 221
Controller ... 36, ... 93
CPU, Kontextwechsel ... 140, ... 162
Context switch ... 140
CD/DVD-Laufwerk ... 116
CPU mit integrierter Northbridge ... 116
C.A.R. Hoare ... 217
CPU, Multicore ... 123
CPU, Hyper-Threading ... 123
CreateProcess, Windows ... 143
CPU, Befehl ... 52
CPU-Scheduling ... 168
Clock Page Seitenersetzungsalgorithmus ... 256
Coffman, 4 Bedingungen, Deadlock|Deadlock, Coffman ... 221
Computerprogramm ... 70
C, Hochsprache ... 53
Compiler ... 53
Count program ... 53
CDFS ... 300
Cluster, Dateisystem ... 300
character device ... 292
CPU ... 21, ... 52
Central Processing Unit ... 21
Controller, Zustandsregister ... 94
Cache ... 72
Computerarchitektur, Fazit ... 119
CD-ROM File System ... 300

D
direkte Adressierung|Adressierung, direkt|absolute Adressierung|Adressierung, absolut ... 66
Datenregister, Controller|Controller, Datenregister ... 94
Dreiadressformat ... 60, ... 62, ... 62
Datenbewegungsbefehl|Befehl, Datenbewegungsbefehl ... 59
dialogorientiert ... 131
Datenblock ... 288
DMA-Controller|Controller, DMA|Direct Memory Access|DMA ... 102
Dirty-Bit ... 254
Datei ... 297
Dateisystem ... 298
Dateisystem, Liste ... 298
Deadlock ... 219
Deadlock-Zustand ... 219
Digitale Schaltung ... 44
direkte Adressierung|Adressierung, direkt ... 68
Dateizuordnungstabelle (FAT) ... 303
direkte Adressierung ... 69
Datenbus ... 116
Dispatcher ... 168
DLL-Dateien ... 265
Dynamic Link Libraries ... 265
Datenstruktur, Stack|Stack, Datenstruktur ... 75
Datenbus ... 22
Deadlock, Circular wait condition ... 221, ... 222
Deklaration ... 70
Datei, ausführbar ... 53
Datenträger ... 300
Disk, Windows ... 300
Datei, Windows ... 300
Diskettenlaufwerk ... 278
DMA, Ablauf ... 102
DMA, Arbeitsweise ... 102
Datenbus, Breite ... 31
Deadlock, No preemption condition ... 221

E
effektive Adresse|Adresse, effektive ... 66
Eingangsleitung|Eingangsleitung, RS-FlipFlop|RS-Flip-Flop, Eingangsleitung|Flip-Flop, Eingangsleitung ... 45
eingelagern, Prozess|Einlagern von Festplatte ... 76
Einadressformat ... 60, ... 61
Entziehbares Betriebsmittel|Betriebsmittel, entziehbar ... 127
Exklusiv nutzbares Betriebsmittel ... 127
erzeugen, Prozess, CreateProcess ... 146
enormer Aufwand ohne DMA ... 102
Einsparung von Interrupts ... 102
effektive Adressierung ... 68, ... 69, ... 68
Eigenschaften eines VNR ... 43
Ein-/Ausgabewerk ... 36
erzeugen, Prozess, fork ... 144
Elternprozess ... 144
EXKLUSIV-ODER-Gatter ... 50
Ethernet-Controller ... 116

Ethernet-Port ... 116
EPROCESS, Windows ... 148
executive process structure, Windows ... 148
Exception ... 90
E-Mail-Server ... 131
einstufige Seitentabelle ... 241
exFAT ... 300
Extended File Allocation Table ... 300

F
forbidden, RS-Flip-Flop ... 45
Festplatte, auslagern|Auslagern auf Festplatte ... 76
Floppy Disk Controller|FDC, Floppy Disk Controller ... 278
Floppy ... 278
FAT ... 303, ... 300
File Allocation Table ... 303, ... 300
FIFO, Seitenersetzungsalgorithmus ... 262, ... 256
First In First Out Algorithmus Seitenersetzungsalgorithmus ... 262
Fazit Computerarchitektur ... 119
First Come First Serve ... 175
FCFS ... 175
fork ... 144, ... 143
Fehlermeldung bei Java-Applet ... 348
Front Side Bus|Chipsatz|Northbridge|Southbridge ... 116
Festplatte|Laufwerk, Festplatte ... 116
Flags ... 28
First In First Out Seitenersetzungsalgorithmus ... 256
flüchtiger Speicher|Speicher, flüchtiger ... 31
Flaschenhals ... 19
Flip-Flop, Arbeitsweise ... 45
File-Server ... 131
Frame ... 236
FAT12 ... 300
File Allocation Table 12 ... 300
FAT16 ... 300
File Allocation Table 16 ... 300
FAT32 ... 300
File Allocation Table 32 ... 300

G
Gerätetreiber, Aufgaben ... 275
Gemeinsam nutzbares Betriebsmittel ... 127
Großrechner|Server|Laptop|Notebook|Personal Computer ... 131
Gnome ... 131
Gerät, blockorientiert ... 288
Gatter, EXCLUSIV-ODER ... 50
Gate, XOR ... 50
Geräteverwaltung ... 269
Geschichte, Betriebssystem ... 123
GiB ... 347
GB ... 347
GigaByte ... 347
Gründe, Interrupt ... 89
Großrechner ... 78
Geschwindigkeit, Bus ... 116
Gigabit-Ethernet-Karte ... 116
gegenseitiger Ausschluss ... 206
Gerät, zeichenorientiert ... 292
gemeinsamer Speicher ... 232
Geschäftiges Warten ... 194
Gatter, NICHT-ODER ... 50
Gate, NOR ... 50
GibiByte ... 347

H
Hades|Hades Framework|Hades Simulationsframework ... 45
Hardware-Ressource ... 126
Harvard-Architektur ... 43, ... 72
Halbduplex-Betrieb ... 230
Hinweis zu Java-Applets ... 348
Hauptspeicher ... 116, ... 53
HyperTransport ... 116
Hoare ... 217
Hansen ... 217
Hyper-Threading ... 123
Hardware-Taktgeber ... 91
Hauptspeicherverwaltung ... 235
Hochsprache ... 53

I
Interpretation, Operand|Operand, Interpretation ... 66
indizierte Adressierung mit Verschiebung ... 66
Input-Leitung, Speicherzelle|Speicherzelle, Input-Leitung ... 45
Indizierte Adressierung mit Verschiebung|Adressierung, indiziert mit Verschiebung ... 76
Interrupt, Einsparung ... 102
Interrupt, Gründe ... 89
indizierte Adressierung mit Verschiebung|Adressierung, indiziert mit Verschiebung ... 69
Interprozesskommunikation ... 230
Interrupt|Interrupt-Controller|Controller, Interrupt ... 85
Interruptbehandlungsroutine ... 85, ... 91
ISR, Interrupt Service Routine|Interrupt Service Routine ... 85
Imprecise interrupt ... 85
Interrupt, imprecise ... 85
Interrupt ... 282, ... 90
Interruptbehandlung im Treiber ... 282
init-Prozess, Linux ... 143
Instruktion ... 70
Interprocess Communication ... 230
IPC ... 230
Interrupt, precise ... 85

J
Java-Applet ... 348
Java-Applet lässt sich nicht ausführen ... 348
JRE ... 348
Java Runtime Environment ... 348

Java ... 348
Java, Threads ... 166
Java, Scheduling ... 184
Java, Hochsprache ... 53
John von Neumann ... 17

K
Konstante ... 66, ... 76, ... 67
Klassifizierung, Befehlsformat ... 60
Kernel, monolithisch ... 131
Kernel, schichtenorientiert ... 131
KDE ... 131
Konstruktion Betriebsmittelgraph ... 222
Kindprozess ... 144
KB ... 347
KiloByte ... 347
Kommunikation CPU-E/A-Gerät ... 89
Kommunikation, verbindungsorientiert ... 230
Kommunikation, verbindungslos ... 230
Kommunikations, speicherbasiert ... 230
Kommunikation, nachrichtenbasiert ... 230
Kommunikation, synchron ... 230
Kommunikation, asynchron ... 230
Kernel-Mode|privilegierter Modus ... 134
Kontextwechsel ... 140, ... 162
Kontextwechsel, Prozess ... 140
kritischer Abschnitt ... 190
Kontext, Prozess ... 140
kritischer Ablauf ... 188
Kollision, Bus ... 22
Kriterien, Scheduling ... 185
Kollision ... 19
Kosten ... 96
Kontextwechsel, ungünstigster Moment ... 199
KiB ... 347
KibiByte ... 347
Kontextwechsel, Thread ... 162

L
Laptop ... 131
Linux ... 131
Liste, Dateisysteme ... 298
Liste, Betriebssysteme ... 121
Länge, virtuelle Adresse ... 241
Länge, physische Adresse ... 241
Limitregister ... 82
Lochkarte ... 78
Linux, Prozesse ... 168
Linux, Threads ... 168
Linux, Scheduling ... 184
LRU, Seitenersetzungsalgorithmus ... 256
Least Recently Used Seitenersetzungsalgorithmus ... 256
Lesender Zugriff, Bus|empfangen, Bus|Bus, lesender Zugriff|Bus, empfangen ... 22
Lokalitätseffekt ... 263
Lock-Mechanismus ... 231
Laufwerk, CD ... 116
Laufwerk, DVD ... 116
Leerlaufprozess, Windows ... 143

M
MMU, Arbeitsweise ... 240, ... 112
mehrere Prozesse gleichzeitig im Hauptspeicher ... 76
monolithischer Kernel ... 131
monolithische Betriebssystemarchitektur ... 131
Mikrokern ... 131
Mainframe|Stapelbetrieb|Batch-Job|Transaktion|transaktionsorientiert|z/OS ... 131
MacOS ... 131
Modifiziert-Bit ... 254
M-Bit ... 254
MB ... 347
MegaByte ... 347
Multicast ... 230
MMU|Memory Management Unit ... 112
mehrere Programme im Hauptspeicher ... 78
mehrstufige Seitentabelle ... 245
Mainboard ... 116
Monitor (Synchronisation) ... 217
Mutex ... 206
mutal exclusion ... 206
MFT ... 307
Master File Table ... 307
Multicore ... 123
Malware Hunting ... 160
Mark Russinovich ... 160
MMU, Seitentabelle ... 240
Memory Address Register ... 31
MAR ... 31
Memory Data Register ... 31
MDR ... 31
Mutual exclusion condition, Deadlock|Deadlock, Mutual exclusion condition ... 221
Multitasking ... 133
Mehrprogrammbetrieb ... 133
Maschinencode ... 70, ... 53
Maschinensprache ... 53
Mirroring, RAID 1 ... 300
Memory, Shared ... 232
MiB ... 347
MebiByte ... 347

N
NICHT-Gatter|Gatter, NICHT|NOT-Gate|Gate, NOT ... 45, ... 50
NOOP-Befehl|Befehl, NOOP|LOAD-Befehl|Befehl, LOAD|STORE-Befehl|Befehl, STORE|ADD-Befehl|Befehl, ADD|SUB-Befehl|Befehl, SUB|EQUAL-Befehl|Befehl, EQUAL|JUMP-Befehl|Befehl, JUMP|HALT-Befehl|Befehl, HALT ... 59
NEC PD765 Floppy Disk Controller Chip ... 278
Nicht entziehbares Betriebsmittel|Betriebsmittel, nicht entziehbar ... 127
Notebook ... 131
Notwendigkeit Betriebssystem ... 98, ... 119, ... 78, ... 91, ... 81

NICHT-UND-Gatter|Gatter, NICHT-UND|NAND-Gate|Gate, NAND ... 50
NICHT-ODER-Gatter ... 50
Nachrichtenbasierte Kommunikation ... 230
Northbridge in CPU integriert ... 116
NRU, Seitenersetzungsalgorithmus ... 259, ... 256
NTFS|New Technology File System ... 307
NTFSInfo.exe, Sysinternals ... 307
Nebenläufigkeit ... 187
Non-preemptive Scheduling ... 168
Not Recently Used Algorithmus ... 256
NFU, Seitenersetzungsalgorithmus ... 256
Not Frequently Used Seitenersetzungsalgorithmus ... 256
No preemption condition, Deadlock ... 221
NOR-Gate ... 50
Not Recently Used Seitenersetzungsalgorithmus ... 259
nicht-unterbrechendes Scheduling ... 168
nicht-verdrängendes Scheduling ... 168
Neumann, John von ... 17
NTFS ... 300
New Technology File System ... 300

O
Out of paper, Controller ... 94
Output-Leitung, Speicherzelle|Speicherzelle, Output-Leitung ... 45
Operand ... 60, ... 62, ... 61, ... 69, ... 67, ... 62
Opcode ... 60, ... 62, ... 61, ... 62
Operation, atomar ... 199
ODER-Gatter|Gatter, ODER|OR-Gate|Gate, OR ... 50
Optimaler Seitenersetzungsalgorithmus ... 258, ... 256
Operation, Rechenwerk ... 28
oberes Ende des Stacks|Stack, oberes Ende ... 75

P
Physikalische Adresse|Adresse, physikalisch ... 66
Polling ... 194
Prozess ausgelagern|auslagern, Prozess ... 76
Programmsteuerbefehl ... 59
PD765 Floppy Disk Controller Chip ... 278
Personal Computer ... 131
Prozess erzeugen, CreateProcess ... 146
Persistente Datenspeicherung ... 296
Persistenz ... 296
physische Adresse, Länge ... 241
Prozess erzeugen, fork ... 144
Peripheriegerät ... 269, ... 93
präzise Unterbrechung ... 85
Prozess-Kontextwechsel ... 140
physischer Speicher|Speicher, physischer ... 112
Physische Speicheradresse|Physische Adresse|Speicheradresse, physisch|Adresse, physisch ... 112
Peripheral-Component-Interconnect ... 116
PCI ... 116
Per Brinch Hansen ... 217
Priority Scheduling ... 182
PS ... 182
Prozess ... 139, ... 70
Programm in Ausführung ... 139, ... 70
Prozess|Prozess-ID|PID ... 143
Prozess-ID|Process-ID|process identifier|PID ... 143
Prozess erzeugen|erzeugen, Prozess ... 143
Prozesskontext ... 140
Prozesskontrollblock ... 147
PCB ... 147
Process control block ... 147
Prozessor ... 52
Prozessorbefehl ... 52
Prozesstabelle ... 158
process table ... 158
Process Explorer, Sysinternals ... 160
Prozesszustand ... 158
Prozesszustand: Bereit ... 158
Prozesszustand: Rechnend ... 158
Prozesszustand: Blockiert ... 158
Prozess-Scheduling ... 168
Preemptive Scheduling ... 168
PUSH, Stack|Stack, PUSH ... 75
POP, Stack ... 75
Paging ... 246
Prozess-Synchronisation ... 185
Page ... 236
Programm ... 70, ... 53
Pascal, Hochsprache ... 53
Programmtext ... 53
Partition ... 300
Pipe ... 233
Pseudo-Harvard-Architektur ... 72
Print-Server ... 131
Protokoll ... 230
präzise Unterbrechung, Bedingungen ... 85
Precise interrupt ... 85
procexp.exe, Sysinternals ... 160
P()-Operation Semaphor ... 203
Pageframe ... 236

Q
Quasi-gleichzeitige Ausführung mehrerer Prozesse ... 89
QuickPath Interconnect ... 116
quasi-gleichzeitige Ausführung mehrerer Prozesse ... 91
Quantum ... 168
Quellcode ... 53

R
Register ... 66, ... 69, ... 67, ... 21
Registeradressierung|Adressierung, Register ... 66
registerindirekte Adressierung ... 66, ... 68
Ready, Controller ... 94
Registersatz ... 94
Read Error, Controller ... 94
Register|Aufbau, Register|Arbeitsweise, Register|Register, Aufbau|Register, Arbeitsweise ... 45
RS-Flip-Flop|Flip-Flop, RS-Flip-Flop ... 45
RS-Flip-Flop, forbidden ... 45

RISC ... 72
Reduced Instruction Set Computer ... 72
Ressource, Software ... 126
Rechnerarchitektur ... 14
RAM ... 112
Referenziert-Bit ... 259
R-Bit ... 259
Rechnend, Prozesszustand ... 158
Race Conditions ... 188
Rechenwerk ... 28
Rechenwerk, Arbeitsweise ... 28
Registeradressierung ... 67
Reihenfolgedruchsetzung ... 214
Round Robin ... 179
RR ... 179
RAM|Random Access Memory ... 31
RAID ... 300
RAID 1, Mirroring ... 300
RAID-Level ... 300
Ressourcenverwaltung ... 130
Registerbreite ... 21
RS-Flip-Flop, Arbeitsweise ... 45
Ressource, Hardware ... 126
Rahmen ... 236

S

Sperrkennzeichen ... 194
Sperrvariable ... 194
ständiges Abfragen (Polling) ... 194
Steuerregister, Controller|Controller, Steuerregister ... 94
See How The CPU Works In One Lesson, Video ... 41
Speichereinheit, Register ... 45
Speicherzelle ... 45
Select-Leitung, Speicherzelle|Speicherzelle, Select-Leitung ... 45
Simulationsframework Hades|Framework Hades ... 45
Steuerinformation ... 60
Systemsteuerbefehl ... 59
Software-Ressource ... 126
schichtenorientierter Kernel ... 131
schichtenorientierte Betriebssystemarchitektur ... 131
Seitentabelleneintrag, Modifiziert-Bit ... 254
Seitentabelleneintrag, M-Bit ... 254
Schaltung, digital ... 44
Systemprogramm ... 121
Seitentabelle, einstufig ... 241
Seitentabelleneintrag ... 241
Scheduling, FCFS ... 175
See How Computers Add Numbers In One Lesson, Video ... 50
Speicherschutzverletzung ... 89, ... 90
Springerlink ... 11
Speicherbasierte Kommunikation ... 230
Synchrone Kommunikation ... 230
Systembus ... 85
Sicherheitsproblem Java-Applet ... 348
Systemaufruf ... 134, ... 143
Systemaufruf, Beispiel ... 134
Speicherschutz ... 82
Speicheraufteilung ... 78
Seitentabelle, mehrstufig ... 245
Steuerbus ... 116
Seitentabelleneintrag, Referenziert-Bit ... 259
Seitentabelleneintrag, R-Bit ... 259
Seitenersetzungsalgorithmus, optimaler ... 258
Scheduling, PS ... 182
System-Idle-Prozess, Windows ... 143
Sysinternals Suite ... 160, ... 307
Statusbits ... 28
Scheduling, RR ... 179
Scheduling ... 168
Scheduler|Prozess-Scheduler|CPU-Scheduler ... 168
Scheduling, preemptive ... 168
Scheduling, unterbrechend ... 168
Scheduling, verdrängend ... 168
Scheduling-Ziele ... 172
Scheduling, Unix ... 184
Scheduling, Linux ... 184
Scheduling, Windows ... 184
Scheduling, Java ... 184
Second Chance Seitenersetzungsalgorithmus ... 263, ... 256
Seitenersetzung ... 251
Seitentabelle ... 240
Semaphore|Dijkstra|Edsger Wybe Dijkstra ... 203
Semaphor|P()-Operation|up()-Operation|V()-Operation|down()-Operation ... 203
Semaphor, binär|Binärer Semaphor ... 203
Semaphor, P()-Operation ... 203
Semaphor, V()-Operation ... 203
Shared Libraries ... 265
Shortest Job First ... 176
Shortest Process Next ... 176
SPN ... 176
Scheduling, SPN ... 176
Shortest Remaining Time Next ... 178
SRTN ... 178
Scheduling, SRTN ... 178
Speicherverwaltung ... 235
Speicherwerk ... 31
Steuerbus|Bus, Steuerbus ... 31
Speicheradressregister ... 31
Speicherdatenregister ... 31
Stackregister|Stack ... 75
Stack, Arbeitsweise ... 75
Steuerwerk|Leitwerk ... 22
Schreibender Zugriff, Bus|senden, Bus|Bus, schreibender Zugriff|Bus, senden ... 22
Swapping ... 81, ... 246
Synchronisation ... 185
Scheduling, Vergleichskriterien ... 185
Scheduling, Kriterien ... 185
Seitenrahmen ... 236
Seite ... 236
Sum program ... 53

Sektor, Speichermedium ... 300
Software-RAID ... 300
Stream, character device ... 292
Shared Memory ... 232
Speicher, gemeinsam genutzt ... 232
Server ... 131
Systemcall ... 134
Syscall ... 134
Speicheradresse, virtuell ... 112
Speicherbereich, zusammenhängend ... 78
Scheduling, non-preemptive ... 168
Scheduling, nicht-unterbrechend ... 168
Scheduling, nicht-verdrängend ... 168
SJF ... 176
Scheduling, SJF ... 176
Stapelbetrieb ... 133

T
Treiber, Aufgaben ... 275
Trennung zwischen Programm und Daten ... 43
Treiber@Gerätetreiber ... 273
Threads, Beispiel ... 166
Threads, Java-Beispiel ... 166
Trap ... 90
Top of Stack|Stack, Top of Stack ... 75
Thread-Synchronisation ... 185
Thread|Leichtgewichtiger Prozess ... 162
Threadkontrollblock|Thread Control Block|TCB|
Thread, Kernel-Ebene ... 162
Threadbibliothek|Thread, User-Ebene|Java-Laufzeitumgebung|Java Runtime Environment|JRE ... 162
Thread, Mischform ... 162
Threadzustand ... 162
Taschenrechner ... 73
Terminal-Server ... 131
Thread-Kontextwechsel ... 162

U
unmittelbare Adressierung|Adressierung, unmittelbar ... 66
UND-Gatter|Gatter, UND|AND-Gate|Gate, AND ... 45, ... 50
Unmittelbare Adressierung ... 76
ungünstigster Moment ... 199
Universalmaschine ... 43
Unicast ... 230
Unterbrechung, Interrupt ... 85
Unpräzise Unterbrechung ... 85
Unterbrechung, unpräzise ... 85
User-Mode ... 134
unkritischer Abschnitt ... 190
Universal-Serial-Bus ... 116
USB ... 116
Unix, Prozesse ... 168
Unix, Threads ... 168
unsichtbare Adressierung ... 67
unterbrechendes Scheduling ... 168
Unix, Scheduling ... 184
unmittelbare Adressierung ... 67
Unix ... 131
Unterbrechung, präzise ... 85
UDF ... 300
Universal Disk Format ... 300

V
Von-Neumann-Zyklus ... 41, ... 38
Visual Transistor-level Simulation of the 6502 CPU ... 41
Visual 6502 ... 41
Verdrahtung, Gatter|Gatter, Verdrahtung ... 45
Von-Neumann-Architektur ... 72, ... 18
versteckte Adressierung ... 76
Verzeichnis, Dateisystem ... 298
Verklemmung ... 219
virtuelle Adresse, Länge ... 241
Verbindungsorientierte Kommunikation ... 230
Verbindungslose Kommunikation ... 230
Vollduplex-Betrieb ... 230
Verwaltung Hauptspeicher|Hauptspeicher, Verwaltung mit MMU ... 112
Virtuelle Speicherverwaltung|Speicherverwaltung, virtueller Speicher ... 112
Virtueller Speicher|Speicher, virtuell ... 112
Virtuelle Speicheradresse ... 112
verdrängendes Scheduling ... 168
Verdrängungsstrategie ... 251
V()-Operation Semaphor ... 203
Vergleichskriterien, Scheduling ... 185
Vier Bedingungen, Deadlock|Deadlock, Vier Bedingungen|4 Bedingungen, Deadlock|Deadlock, 4 Bedingungen ... 221
virtuelle Speicherverwaltung|Speicherverwaltung, virtuell ... 236
virtuelle Speicherverwaltung, Grundgedanken|Grundgedanken, virtuelle Speicherverwaltung ... 236
virtuelle Seite ... 236
VNA ... 18
Von-Neumann-Flaschenhals ... 19
Von-Neumann-Rechner ... 17
VNR ... 17
Volume, Windows ... 300
Virtuelle Adresse ... 112

W
Write Error, Controller ... 94
Write-Leitung, Speicherzelle|Speicherzelle, Write-Leitung ... 45
Wahrheitstafel ... 45, ... 50
Windows ... 131
Wir brauchen ein Betriebssystem ... 98, ... 119, ... 78, ... 91, ... 81
Wir brauchen ein Betriebssystem|Notwendigkeit Betriebssystem ... 112
wechselseitiger Ausschluss ... 206
Windows, Prozesse ... 167
Windows, Threads ... 167
Windows, Scheduling ... 184

Working Set Seitenersetzungsalgorithmus ... 256, ... 263
Wait for condition, Deadlock|Deadlock, Wait for condition ... 221
Warten, aktiv ... 194
Windows Server ... 131

X
XOR-Gate ... 50

Z
Zustandsregister, Controller ... 94
Zweiadressformat ... 60, ... 69, ... 62
Zustand, Deadlock ... 219
Zyklus im Betriebsmittelgraph ... 222
zusammenhängender Speicherbereich ... 78
Zustand, Prozess ... 158
Zusammenarbeit Rechenwerk-Steuerwerk ... 28
Zeitscheibe ... 168
Zeit-Quantum ... 168
Ziele, Scheduling ... 172
Zustand, Thread ... 162
Zeichenorientiertes Gerät ... 292
Zentrale Aufgabe Betriebssystem ... 130
Zentraleinheit ... 21
Zählsemaphor ... 215
Zyklus, Von-Neumann-Zyklus ... 41