Prof. Dr. Rolf Klein, Prof. Dr. Jörg Haake, Prof. Dr. Anja Haake

Modul 63012

Softwaresysteme

Lehrveranstaltung
Betriebssysteme und Rechnernetze

LESEPROBE

Fakultät für
Mathematik und Informatik

FernUniversität in Hagen

Das Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, insbesondere das Recht der Vervielfältigung und Verbreitung sowie der Übersetzung und des Nachdrucks bleiben, auch bei nur auszugsweiser Verwertung, vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder ein anderes Verfahren) ohne schriftliche Genehmigung der FernUniversität reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Inhalt

1 Aufgaben eines Betriebssystems **5**
- 1.1 Einführung . . . 5
- 1.2 Die Hardwarekomponenten eines Computers . . . 9
  - 1.2.1 Prozessor und Hauptspeicher . . . 10
  - 1.2.2 Cache . . . 13
  - 1.2.3 Sekundärspeicher . . . 14
  - 1.2.4 Tertiärspeicher . . . 17
- 1.3 Die Struktur von Computersystemen . . . 20
  - 1.3.1 Gerätesteuerung . . . 20
  - 1.3.2 Exkurs: Abstraktion, Kapselung und Schichtenmodell . . . 23
  - 1.3.3 Unterbrechungen . . . 25
  - 1.3.4 Direkter Speicherzugriff (DMA) . . . 28
  - 1.3.5 System- und Benutzermodus; Speicherschutz . . . 29
- 1.4 Die Struktur von Betriebssystemen . . . 31
  - 1.4.1 Prozesszustände . . . 32
  - 1.4.2 Rekapitulation: ein Gerätezugriff . . . 35
  - 1.4.3 Programmierschnittstelle . . . 37
  - 1.4.4 Benutzerschnittstelle . . . 40
- 1.5 Andere Systeme . . . 42
  - 1.5.1 Parallelrechner . . . 42
  - 1.5.2 Verteilte Systeme . . . 43
  - 1.5.3 Realzeitsysteme . . . 43
- Literatur . . . 45
- Lösungen der Übungsaufgaben . . . 47

2 Prozesse und Dateisysteme **55**

3 Architektur und Anwendungen von Rechnernetzen **95**

4 Vermittlung in Rechnernetzen **177**

Kurseinheit 1

Aufgaben eines Betriebssystems

Der Autor

Univ.-Prof. Dr. rer. nat. Rolf Klein

Geboren 1953 in Münster. Studium der Mathematik und mathematischen Logik an der Universität Münster (1973 bis 1979). Wissenschaftlicher Mitarbeiter an der Universität Erlangen-Nürnberg (1980 bis 1983), 1982 Promotion bei Professor Dr. Wulf-Dieter Geyer. Wissenschaftlicher Mitarbeiter an der Universität Karlsruhe (1983–1987), dazwischen Gastprofessor an der University of Waterloo in Kanada. Hochschulassistent an der Universität Freiburg (1987–1989), 1989 Habilitation in Informatik bei Professor Dr. Thomas Ottmann. Von 1989 bis 1991 Professor für Praktische Informatik an der Universität Essen, von 1991 bis 2000 Professor für Praktische Informatik an der FernUniversität Hagen, seit 2000 Professor für Theoretische Informatik an der Universität Bonn.

1.1 Einführung

Diese Einführung soll Ihnen einen Überblick darüber geben, worum es im vorliegenden Kurs geht und an welchem Platz die Kursinhalte in der Informatik als Wissenschaft angesiedelt sind.

Drei Ursachen für Computerverbreitung:

Seit den 1980er Jahren hat der Computer nicht nur die Berufswelt von Grund auf verändert, er hat auch in den Bereichen Bildung und Freizeit Einzug gehalten. Diese Entwicklung wurde durch verschiedene Faktoren ermöglicht und begünstigt:

1. Hardware preiswert und schnell

Zum einen ist die *Hardware* der Computer – dazu gehören unter anderem der Prozessor, die Hauptspeicherbausteine und die Plattenlaufwerke – immer leistungsfähiger und preiswerter geworden. Erst dadurch wurden Computer für kleine Betriebe und für private Benutzer überhaupt erschwinglich.

Zum anderen ist der Umgang mit dem Computer immer einfacher geworden: Wer gelegentlich ein Textverarbeitungssystem oder ein Spielprogramm

benutzen möchte, kommt mit den komplexen technischen Details der Hardware überhaupt nicht in Berührung und braucht sie deshalb nicht zu kennen.

Hier hat sich eine ähnliche Entwicklung vollzogen wie bei den Kraftfahrzeugen: Um einen Oldtimer zu starten, musste man Zündzeitpunkt und Gemischanreicherung sorgfältig einstellen und die technischen Zusammenhänge kennen – heute wird dem Fahrer diese Aufgabe von elektronischen Motormanagementsystemen abgenommen.

2. Betriebssystem verwaltet Hardware

Beim modernen Computer übernimmt das *Betriebssystem* diese Funktion. Es bildet eine Schicht zwischen der Rechnerhardware und den Benutzern und ihren Anwendungsprogrammen;[1] siehe Abbildung 1.1. Hierdurch werden Benutzer und Programme von der Hardware unabhängig.

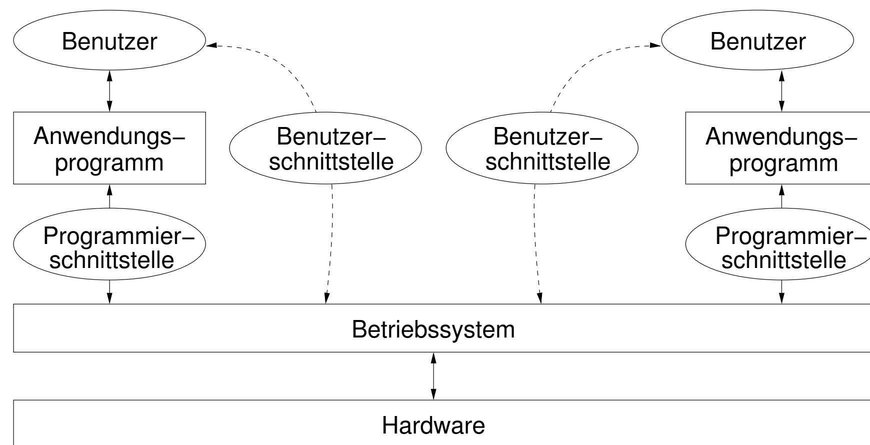


Abbildung 1.1: Das Betriebssystem als Vermittler zwischen der Rechnerhardware und den Anwendungsprogrammen.

Schnittstellen

Zwei *Schnittstellen* werden hierfür vom Betriebssystem zur Verfügung gestellt:

- eine Benutzerschnittstelle[2] für die Kommunikation der Benutzer mit dem Betriebssystem und
- eine Programmierschnittstelle für die Kommunikation der Programme mit dem Betriebssystem.

Dieses Konzept bildete eine wesentliche Voraussetzung für die Entstehung des *Personalcomputers (PC)*[3], der von Jedermann bedient werden kann, und für die Entwicklung kostengünstiger Anwendungsprogramme.

[1]Anwendungsprogramme werden manchmal auch *Anwender*programme oder *Applikationen* genannt.

[2]Statt von Benutzerschnittstellen sprechen wir manchmal auch von *Benutzeroberflächen*, wenn wir ihre graphische Gestaltung hervorheben wollen.

[3]Darunter verstehen wir einen gängigen, preiswerten Rechner, der einem einzelnen Benutzer zum persönlichen Gebrauch zur Verfügung steht, z. B. Intel-basiert oder -kompatibel und Macintosh.

Ein dritter Grund für die rasante Verbreitung von Computern liegt darin, dass man sie zu *Netzwerken* zusammenschalten kann, die ganz neue Möglichkeiten eröffnen. So kann etwa ein Netz aus einfachen Personalcomputern den Mitarbeitern einer Abteilung den Zugriff auf gemeinsame Datenbestände erlauben und dabei viel preisgünstiger sein als ein Abteilungsrechner mit gleicher Gesamtleistung und einer entsprechenden Anzahl von Terminals. Ein weiteres Beispiel ist das *World Wide Web*, das den für den Einzelnen zugänglichen Informationsraum in einer Weise erweitert, die bis zu den 1980er Jahren noch nicht vorstellbar war.

3. Vernetzbarkeit

In diesem Kurs beschäftigen wir uns in den ersten beiden Kurseinheiten mit *Betriebssystemen*; die Kurseinheiten drei und vier befassen sich mit *Rechnernetzen.* Natürlich kann keines der beiden Themen in diesem Rahmen umfassend behandelt werden; unser Ziel ist es, Ihnen einige wesentliche Prinzipien und Grundtatsachen nahezubringen, die in der Praxis immer wieder auftreten und deswegen von allgemeinem Interesse sind. Wer diesen Stoff im späteren Studium vertiefen möchte, sei z. B. auf die Kurse *Betriebssysteme*, *Verteilte Systeme*, *Kommunikations- und Rechnernetze* und *Sicherheit im Internet* hingewiesen.

Kursinhalt

Vertiefung

An welchem Platz steht nun der Inhalt dieses Kurses im Gesamtkontext der Informatik? Stark vereinfacht kann man sagen, die Informatik beschäftigt sich damit, wie man von einem Problem der realen Welt zu einer Computerlösung kommt.

vom Problem zur Computerlösung

Am Anfang wird im Dialog mit dem Anwender durch Abstraktion eine zunächst noch unscharfe Beschreibung des realen Problems gewonnen; sie muss dann weiter präzisiert werden, bis eine formale *Problemspezifikation* vorliegt, auf der dann das Pflichtenheft aufbauen kann. Dieser schwierige Prozess ist in der Informatik Gegenstand des *Software Engineering.*

Problemspezifikation

Anschließend wird ein *Algorithmus* zur Lösung des Problems entwickelt und auf Korrektheit und Effizienz geprüft. Ein wenig mathematisches Rüstzeug ist dabei unentbehrlich.

Algorithmus

Es gibt wichtige Problembereiche, für die im Laufe der Zeit eigene algorithmische Techniken entstanden sind; Beispiele sind die Bereiche *Datenbanken und Informationssysteme*, *Künstliche Intelligenz*, *Computergraphik* und *kombinatorische Algorithmen.* Zu allen diesen Bereichen können Sie Wahlkurse der Informatik in Ihrem Studiengang belegen.

Aus dem Algorithmus entsteht dann ein *Programm*, zum Beispiel in Pascal, wie es im Kurs *Einführung in die imperative Programmierung* gelehrt wird, oder in Java, wie es der Kurs *Einführung in die Objektorientierte Programmierung* vermittelt.

Programm

Programmierung

Wie aus einem für Menschen lesbaren Programm ein Programm in Maschinensprache wird, zeigt der Kurs *Übersetzerbau.* In den Kursen der *Technischen Informatik* wird erklärt, wie die Maschine, also die Hardware-Plattform, aufgebaut ist.

Maschine

Zu guter Letzt kommt dieser Kurs zum Tragen: Wir untersuchen hier, was geschieht, während das Maschinenprogramm von einer oder mehreren Maschinen ausgeführt wird.

Programmausführung

Zielgruppen

Wer hat mit Betriebssystemen und Rechnernetzen zu tun? Zwei Gruppen hatten wir bereits identifiziert:

Benutzer

- Wer als *reiner Benutzer* auf seinem Computer ausschließlich fertige Anwendungsprogramme laufen lässt, arbeitet gelegentlich mit der Benutzerschnittstelle des Betriebssystems, etwa um ein Programm zu starten, ein neues Verzeichnis anzulegen oder um nach einer Datei zu suchen.

Anwendungs-programmierer

- Wer als *Anwendungsprogrammierer* selbst Software entwickelt, muss die Programmierschnittstelle des Betriebssystems kennen. Bei allen netzbasierten Anwendungen sind außerdem solide Kenntnisse über Rechnernetze vonnöten, zum Beispiel über Protokolle und den Client-Server-Betrieb.

Es lassen sich zumindest zwei weitere Berufsgruppen ausmachen, für die der Inhalt dieses Kurses ebenfalls interessant ist:

Systemadministrator

- Wer als *Systemadministrator* einzelne Computer einrichtet und an ein vorhandenes Netzwerk anschließt oder selbst ein kleines oder größeres Netzwerk einrichtet, betreibt und wartet, muss neben den Benutzer- und Programmierschnittstellen der Betriebssysteme der beteiligten Computer auch deren spezielle Kommandos für den Systemadministrator (Super-User) kennen, mit denen die Konfiguration des Netzwerks festgelegt wird. Benötigt werden auch Kenntnisse der internen Abläufe, um Performanzprobleme beheben zu können. Schließlich sind für die Wartung auch grundlegende Hardwarekenntnisse erforderlich.

Systementwickler

- Wer schließlich selbst an der *Weiterentwicklung von Betriebssystemen* oder anderer Systemsoftware[4] mitarbeitet, muss detaillierte Kenntnisse über die interne Struktur des Betriebssystems und der Rechnerhardware besitzen.

Natürlich lassen sich diese Tätigkeiten nicht scharf gegeneinander abgrenzen; wer etwa privat intensiv mit seinem PC arbeitet, ist oft Benutzer, Programmierer und Administrator in einer Person.

Bevor wir nun mit den Betriebssystemen beginnen, noch ein paar Vorbemerkungen. Ein Kurs über Betriebssysteme will in allgemeiner Form beschreiben, welche Aufgaben moderne Betriebssysteme erfüllen müssen, und Prinzipien zur Lösung dieser Aufgaben vorstellen. Er will *nicht* die Dokumentation eines konkreten Betriebssystems ersetzen[5]. Trotzdem wird in diesem Kurs gelegentlich ein reales Betriebssystem als Beispiel auftreten, und die Beispiele sollten für Sie zu Hause am Rechner nachvollziehbar sein. Aus diesem Grund haben wir uns für das System Linux entschieden. Informationen zu Linux und Installationshinweise stehen zum Beispiel in [4, 5, 3, 6, 1, 2].

Linux als Beispiel

[4] Zur Systemsoftware zählen z. B. Debugger, Datenbankmanagementsysteme (DBMS), graphische Benutzeroberflächen, Compiler und Binder, Netzsoftware, aber auch das Betriebssystem selbst.

[5] Denn Dokumentationen veralten rasch, während allgemeine Prinzipien ihre Gültigkeit behalten.

Die Fachsprache der Informatik ist Englisch. Hilfetexte und Handbücher sind oft nur auf Englisch verfügbar. Wer Informatik studiert, sollte deshalb nach besten Kräften Englisch lernen.[6] Wir ergänzen ihn hier durch einschlägige Fachvokabeln aus den Bereichen Rechnernetze und Betriebssysteme, die meist in Klammern hinter den deutschen Begriffen aufgeführt werden.

In den nächsten Kapiteln finden Sie eine Reihe von *Übungsaufgaben* und, jeweils am Kapitelende, ihre Lösungen. Sie dienen zur Selbstkontrolle beim Lesen, zur Einübung des Stoffs und zu einem geringen Teil auch zur Ergänzung. Alle Leser werden nachdrücklich ermutigt, sich mit diesen Übungsaufgaben zu beschäftigen. Sie sind mit nebenstehendem Zeichen gekennzeichnet, das auch den Schwierigkeitsgrad angibt (1 = leicht, 5 = schwierig).

1.2 Die Hardwarekomponenten eines Computers

In der Einführung hatten wir festgestellt, dass das Betriebssystem eine Zwischenschicht ist, die die Benutzer und ihre Programme von der „nackten“ Hardware des Rechners trennt. Um die Funktionen des Betriebssystems richtig verstehen zu können, sollten wir uns deshalb eine ungefähre Vorstellung davon machen, wie die Hardware aufgebaut ist. Dazu dient dieser Abschnitt.

von-Neumann-Architektur

Wir beschränken uns dabei auf die klassische Architektur des von-Neumann-Rechners,[7] wie sie auch im Kurs *Konzepte imperativer Programmierung* beschrieben wird.

Um diese Architektur richtig würdigen zu können, muss man wissen, dass in der Anfangszeit die Rechner durch *physische Veränderung der Hardware* programmiert wurden, der legendäre ENIAC[8] etwa durch das Umstöpseln von Steckverbindungen. Erst 1946 hatte von Neumann eine bahnbrechende Idee: Das auszuführende Programm ist nicht mehr ein fester Bestandteil des Rechners; es wird vielmehr – genau wie die benötigten Daten – vor dem Programmlauf in den Speicher des Rechners geladen und hinterher wieder entfernt. Fest in den Rechner einbauen muss man also nur noch die Fähigkeit, ein beliebiges im Speicher befindliches Programm ausführen zu können – und hieran braucht man dann nie wieder etwas zu verändern! Diese Vorstellung vom Computer als einer *universellen Maschine zur Ausführung von Programmen* wird Ihnen in der Theoretischen Informatik wiederbegegnen.

von Neumanns Idee

[6] Solide Englischkenntnisse bilden eine Schlüsselqualifikation, die in zahlreichen Stellenausschreibungen verlangt wird. Sprachvermischungen, wie „Die CPU macht busy-waiting, bis ein interrupt gehandlet werden muss“ sind aber nicht so gern gesehen.

[7] John von Neumann war einer der vielseitigsten Mathematiker des 20. Jahrhunderts. Er hat nicht nur das Bild des modernen Rechners geprägt, er war zum Beispiel auch ein Begründer der Spieltheorie.

[8] ENIAC war ein Elektronenrechner, der mit 18.000 Röhren bestückt war, 30 Tonnen wog und einen derart hohen Stromverbrauch hatte, dass beim Einschalten das Licht in ganz Philadelphia dunkler geworden sein soll.

Bestandteile der Hardware

Die wesentlichen Bestandteile der Hardware eines modernen von-Neumann-Rechners sind daher

- der Prozessor (CPU = central processing unit),
- der Hauptspeicher (main memory) und
- die Ein-/Ausgabegeräte (I/O devices); dazu zählen heute typischerweise
  - Magnetplattenlaufwerk (hard disk drive),
  - Diskettenlaufwerk (floppy disk drive)
  - Bandlaufwerk (tape drive),
  - CD-ROM-Laufwerk (compact disk - read only memory),
  - Drucker (printer) und
  - Bildschirm (monitor), Maus (mouse) und Tastatur (keyboard), zusammen auch Terminal genannt,
  - Kommunikationsgeräte zum Anschluss an Rechnernetze, zum Beispiel Modem, ISDN-Controller oder Ethernet-Controller.

Dazu kommen oft Audio- und Videogeräte und Joysticks, wie sie für den den Betrieb von Computerspielen benötigt werden.

In den nächsten beiden Abschnitten gehen wir auf einige dieser Hardwarebestandteile etwas näher ein.

1.2.1 Prozessor und Hauptspeicher

Alle eigentlichen Berechnungen eines Computers finden in seinem *Prozessor* statt, genauer gesagt im Rechenwerk, das manchmal auch als ALU (arithmetic-logic unit) bezeichnet wird.

Aufbau des Hauptspeichers

Der Prozessor greift auf die Programme und die Daten zu, die sich zur Laufzeit im *Hauptspeicher* befinden. Dieser ist als eine lange Folge von gleich großen Speicherzellen organisiert, die einzeln adressiert werden können.

Wegen dieses *wahlfreien Zugriffs* hat sich für den Hauptspeicher auch die Bezeichnung RAM (random access memory) eingebürgert.[9]

In Pascal-Notation ist also der Hauptspeicher ein sehr langes

array[0..n-1] of word,

Bytes und Bits

wobei ein Wort gerade der Inhalt einer Speicherzelle ist. Es besteht aus einem oder mehreren Bytes. Ein Byte enthält acht Bit, und ein Bit hat den Wert null oder eins. Die Speicherkapazität des Hauptspeichers beträgt daher n mal die Wortlänge. Sie wird in Kilobyte (KByte), Megabyte (MByte) oder Gigabyte (GByte) angegeben, wobei mit Zweierpotenzen gerechnet wird: 1 MB = 1024 KB = 1024^2 Byte[10].

[9] Dem Zufall bleibt beim Speicherzugriff nichts überlassen; „random access“ meint, dass die Zugriffsreihenfolge nicht im voraus bekannt ist.

[10] Wir verwenden im Kurs keine Schreibweise nach *IEC-Präfixe-Notation*: 1 MiByte = 2^{10} KiByte = 1024^2 Byte. Wenn z. B. heute von Festplattenherstellern für Speicherkapazitätsangaben die Dezimaleinheiten genutzt werden, dann sind sie auch dezimal gemeint, mit dem vielleicht erwünschten Effekt, dass damit größere Zahlen herauskommen

Bei der von-Neumann-Architektur enthält der Hauptspeicher Daten und Programme. Entsprechend kann ein Wort die Binärdarstellung einer Zahl sein, der Code für einen Prozessorbefehl oder die Adresse einer anderen Speicherzelle. Einem Wort selbst kann man nicht ansehen, was es darstellt. Trotzdem sind Verwechselungen ausgeschlossen, denn der Prozessor „weiß“ bei jedem Zugriff auf eine Speicherzelle, ob darin eine Zahl oder ein Befehl steht. Schauen wir dem Prozessor einmal bei der Arbeit zu! Zur Illustration dient Abbildung 1.2. Sie zeigt einen Hauptspeicher der Kapazität 64 KByte, der aus 2^{16} Speicherzellen der Wortlänge 1 Byte besteht, und einige typische Bestandteile einer CPU, wie sie zum Beispiel beim Intel 8080 anzutreffen waren.

Zahlen und Befehle

Das *Befehlszählregister* in der CPU enthält die Adresse – also die Binärdarstellung[11] des Index – von derjenigen Speicherzelle, in der der Code des nächsten auszuführenden Befehls steht. In Abbildung 1.2 ist das die Zelle mit dem Index 4; die Adresse als 16-Bit-Zahl lautet also

ein Befehlszyklus des Prozessors

$$0000000000000100.$$

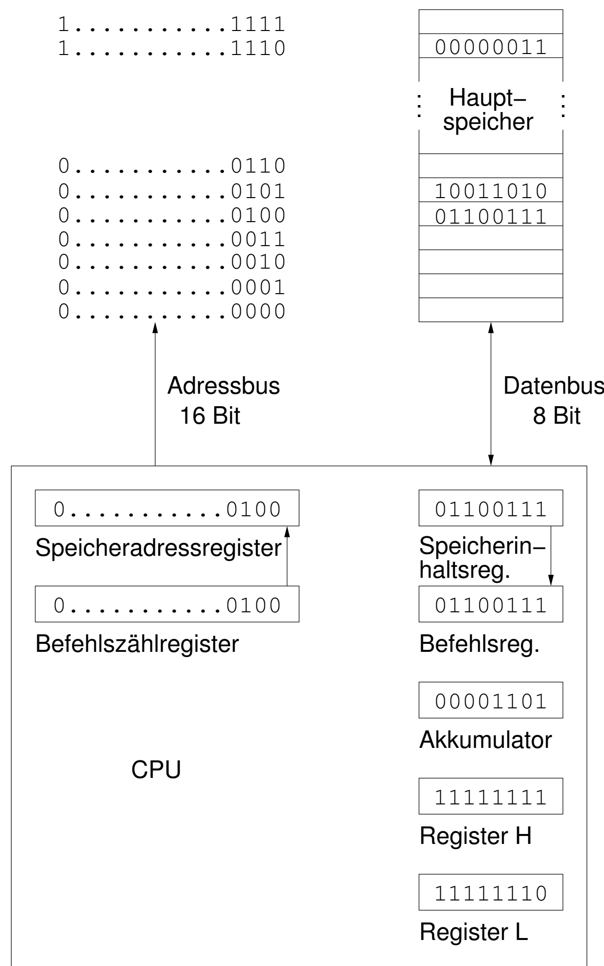


Abbildung 1.2: Der Prozessor greift über Daten- und Adressbus auf den Hauptspeicher zu.

[11] Die Binärdarstellung einer Zahl ist Gegenstand von Übungsaufgabe 1.1.

Um auf diese Speicherzelle zugreifen zu können, wird die Adresse in das *Speicheradressregister* übertragen und gelangt auf den *Adressbus*. Hierdurch wird die entsprechende Zelle des Hauptspeichers angesprochen, und ihr Inhalt wird über den *Datenbus* in das *Speicherinhaltsregister* des Prozessors übertragen. Weil zu diesem Zeitpunkt feststeht, dass das Wort 01100111 einen Befehl darstellt, wird es in das *Befehlsregister* kopiert; dort wird der Befehl decodiert und interpretiert.
Angenommen, der auszuführende Befehl lautet

$$A := A + (HL).$$

Das bedeutet: Der Inhalt des Akkumulators – eines speziellen 8-Bit-Registers der CPU – soll um die Zahl erhöht werden, die in der Speicherzelle steht, deren Adresse sich ergibt, wenn man die Worte in den Registern H (high) und L (low) hintereinanderschreibt; man spricht hierbei von *indirekter Adressierung*. Im Beispiel ergibt sich hierdurch die Adresse

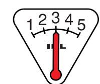
Indirekte Adressierung

$$1111111111111110;$$

sie wird in das Speicheradressregister übertragen, und ein weiterer Speicherzugriff wird ausgelöst, der das Wort 00000011 in das Speicherinhaltsregister bringt. Aus dem Zusammenhang ist klar, dass es sich hierbei um die Binärdarstellung einer Zahl handeln muss, denn das Wort tritt ja als Operand einer Addition auf. Diese Addition kann nun ausgeführt werden; im Akkumulator steht danach die Binärdarstellung der Zahl $13 + 3 = 16$. Außerdem wird der Inhalt des Befehlszählregister um eins erhöht und damit die Ausführung des nächsten Befehls vorbereitet, dessen Code im Normalfall in der folgenden Speicherzelle steht. Zum Beispiel könnte jetzt das Ergebnis der Addition in eine Hauptspeicherzelle zurückgeschrieben werden.

Durch dieses Hochzählen um eins erklärt sich auch der Name „Befehlszählregister“; man findet auch die Bezeichnung *Programmzähler* (program counter). Wäre der letzte Befehl keine Addition, sondern ein Sprungbefehl gewesen, z. B. „verzweige zu einer bestimmten Adresse“, so stünde jetzt diese Adresse im Befehlszählregister.

Programmzähler



Übungsaufgabe 1.1 Die Binärdarstellung einer natürlichen Zahl $n \geq 1$ ist die Folge $a_k a_{k-1} \dots a_1 a_0$ von Nullen und Einsen a_i, für welche

$$n = a_k \cdot 2^k + a_{k-1} \cdot 2^{k-1} + \dots + a_1 \cdot 2^1 + a_0 \cdot 2^0$$

gilt und $a_k = 1$; führende Nullen lässt man also fort. Welche Zahl n wird durch 10011101 dargestellt? Bestimmen Sie die Binärdarstellung von 160.

Übungsaufgabe 1.2 Beim Prozessor 68000 von Motorola beträgt die Wortlänge 2 Byte. Wie breit muss der Adressbus sein – das heißt: aus wievielen Bits bestehen die Adressen – bei einer Hauptspeicherkapazität von 16 MByte?

Die CPU arbeitet *getaktet*. Pro Arbeitstakt können gewisse Elementarbefehle ausgeführt werden wie etwa das Kopieren einer Adresse vom Befehlszählregister ins Speicheradressregister. Als *Taktfrequenz* bezeichnet man die Anzahl der Takte pro Sekunde; mehrere hundert Megahertz (MHz) sind heute üblich.

Takt

Auch das Heraufzählen des Befehlszählregisters lässt sich in einem einzigen Arbeitstakt erledigen. Dagegen nimmt ein Hauptspeicherzugriff viele Arbeitstakte in Anspruch.[12] Denn einerseits sind mehrere Elementarbefehle auszuführen, wie wir am Beispiel des Befehls A := A + (HL) oben gesehen haben. Andererseits vergeht eine gewisse Zeit zwischen dem Aufbringen einer Adresse auf den Adressbus und der Ankunft des zugehörigen Speicherzelleninhalts über den Datenbus. Diese Zeitspanne hängt von der physikalischen Beschaffenheit des Hauptspeichers ab; sie wird als *Zugriffszeit* bezeichnet.

Zugriffszeit

1.2.2 Cache

Schnelle Hauptspeicherchips kosten pro MByte Speicherkapazität viel mehr als langsame Chips. Aus Kostengründen wird deshalb oft neben dem normalen Hauptspeicher ein kleinerer, aber schnellerer Zwischenspeicher verwendet, der meist als *Cache* bezeichnet wird.[13] Häufig benutzte Daten werden vorübergehend vom Hauptspeicher in den Cache kopiert. Wenn Daten benötigt werden, schaut man zunächst im Cache nach. Findet man dort eine Kopie der gesuchten Daten, so verwendet man sie und braucht keinen Hauptspeicherzugriff auszuführen. Wird man aber im Cache nicht fündig, ist ein Zugriff auf den Hauptspeicher unvermeidlich; in diesem Fall legt man eine Kopie der Daten im Cache ab, weil man davon ausgeht, dass diese Daten bald noch einmal benötigt werden. Dieser einfache *Cache-Algorithmus* spielt eine wichtige Rolle; er wird uns in Abschnitt 1.2.3 wiederbegegnen.

Cache

Cache-Algorithmus

Für das Betriebssystem stellen sich hier zwei wichtige Aufgaben:

1. **Cache-Management**
   Wenn Daten im Cache abgelegt werden sollen und dort kein freier Platz mehr vorhanden ist, müssen alte Daten überschrieben werden. Welche Daten soll man dafür opfern?[14]

Cache-Management

2. **Cache-Konsistenz**
   Angenommen, der Wert einer Variablen soll verändert werden. Wenn diese Änderung nur an der Kopie im Cache vollzogen wird, so besteht anschließend ein Unterschied zwischen dem Original im Hauptspeicher und der Kopie im Cache. Das Betriebssystem muss dafür sorgen, dass sich hieraus keine Fehler ergeben. Dieses Problem ist besonders gravierend bei

Cache-Konsistenz

[12] Es gibt auch andere Prozessorbefehle, die mehr als einen Arbeitstakt erfordern. Manche Hersteller geben deshalb neben der Taktfrequenz auch die mittlere Anzahl von Befehlen an, die pro Sekunde abgearbeitet werden kann. Gemessen wird diese Zahl in MIPS (million instructions per second).

[13] Das Wort Cache klingt im Englischen ähnlich wie *cash* = Bargeld. Auch wenn ein schneller Cache viel Geld kostet, sollte man die beiden Wörter nicht verwechseln.

[14] Erst durch diese Festlegung wird ein Cache-Algorithmus vollständig bestimmt. Man kann zum Beispiel die Strategie FIFO (first-in, first-out) anwenden und stets diejenigen Daten überschreiben, die schon am längsten im Cache stehen.

Multiprozessorsystemen, bei denen jeder Prozessor über einen eigenen Cache verfügt.

Welchen Effizienzgewinn die Verwendung eines schnellen Zwischenspeichers bringt, hängt zum einen von der Schnelligkeit der Hardware und vom Cache-Management ab, im Einzelfall aber auch davon, wie ein konkretes Programm auf seine Daten zugreift.

Übungsaufgabe 1.3 Angenommen, ein Schreib-/Lesezugriff auf den Cache ist neun mal so schnell wie ein Zugriff auf den Hauptspeicher. Welchen Zeitvorteil ergibt die Verwendung eines Cache nach dem oben beschriebenen Algorithmus, wenn bei 80 % aller Zugriffe die gesuchten Daten im Cache gefunden werden?

Hauptspeicher: knapp und flüchtig

Mit Hauptspeicher (und Cache) allein kann ein Rechner nicht auskommen, denn

1. selbst wenn der Hauptspeicher pro MByte preiswerter als der Cache ist, kann man seine Kapazität aus Kostengründen nicht so groß auslegen, dass alle benötigten Programme und Daten darin Platz finden;

2. beim Abschalten des Rechners oder bei einem Stromausfall geht der Inhalt des Hauptspeichers verloren.

Außerdem braucht man Geräte, mit deren Hilfe der Rechner Programme und größere Datenmengen mit seiner Umwelt austauschen kann. Aus diesem Grund besitzen die meisten Computer außer dem Hauptspeicher, der auch als *Primärspeicher* bezeichnet wird, noch *Sekundärspeicher* und *Tertiärspeicher*. Mehr hierzu finden Sie in den nächsten Abschnitten.

1.2.3 Sekundärspeicher

Magnetplatte

Der gebräuchlichste Typ von Sekundärspeicher (secondary memory) ist zur Zeit die Magnetplatte. Sie ist auf beiden Seiten mit einer magnetisierbaren Oberfläche beschichtet und dreht sich mit rund einhundert Umdrehungen pro Sekunde. Früher gab es Wechselplatten, heute sind die Magnetplatten meist fest in ihr Laufwerk eingebaut. Bei manchen Plattenlaufwerken sind mehrere Platten übereinander angebracht, die von einer gemeinsamen Spindel gedreht werden; siehe Abbildung 1.3. Auf jeder Seite einer Platte gleitet auf einem dünnen Luftkissen ein Schreib-/Lesekopf. Er wird von einem Arm geführt, der an einer Welle befestigt ist. Durch Verdrehen der Welle werden alle Köpfe simultan positioniert.

Festplatte

Diese Anordnung mag an altmodische Plattenspieler erinnern, es bestehen aber zwei wesentliche Unterschiede zur Vinyl-Schallplatte: Bei der Festplatte wird die Information magnetisch gespeichert, und die Daten sind nicht spiralförmig auf der Festplatte angeordnet; vielmehr ist jede Oberfläche einer Platte in einige tausend kreisförmige *Spuren* gleicher Breite unterteilt, die wiederum aus einigen hundert *Sektoren* bestehen. Die jeweils übereinanderliegenden Spuren bilden einen *Zylinder*. Die weiter außen liegenden Spuren können mehr Sektoren enthalten als die inneren.

Spuren, Sektoren, Zylinder

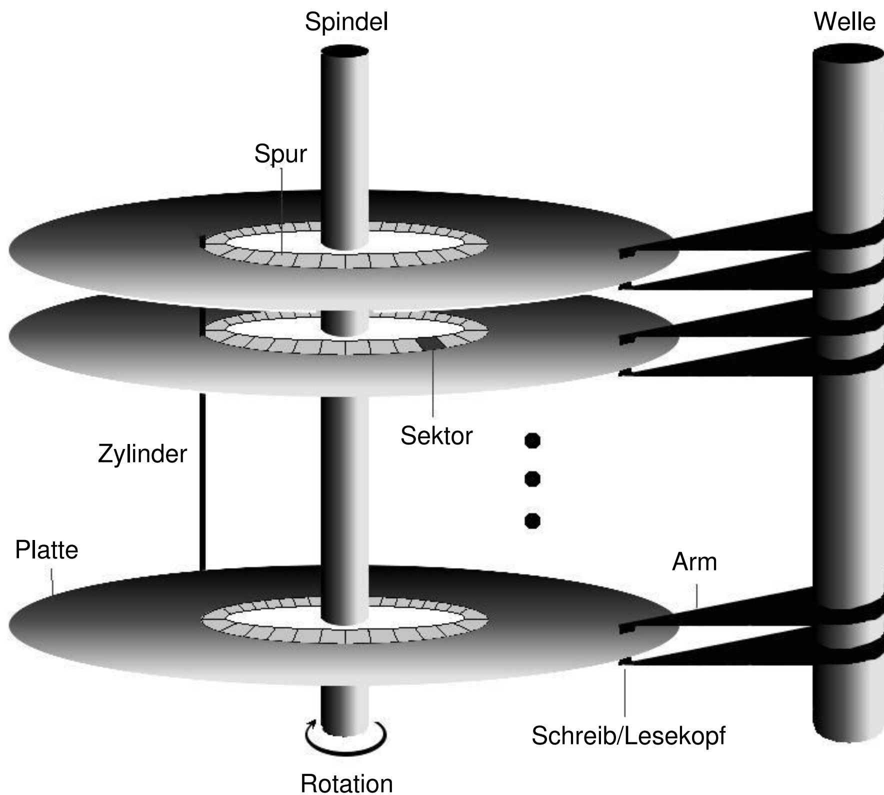


Abbildung 1.3: Ein Plattenstapel mit Schreib-/Leseköpfen.

Bei starken Erschütterungen kann es vorkommen, dass ein Schreib-/Lesekopf die Plattenoberfläche im Betrieb berührt und sie beschädigt. Bei einem solchen *head crash* können Daten verlorengehen. Um dieses Risiko zu verringern, sind Magnetplatten meist in staubdicht verschlossenen Gehäusen untergebracht, und der Arm wird in einer sicheren Ruhestellung geparkt, wenn keine Aufträge vorliegen.

Die Magnetplatte ermöglicht *wahlfreien Zugriff* auf einzelne Sektoren. Um den Inhalt eines bestimmten Sektors zu lesen, muss zunächst der Kopf durch Verdrehen der Welle auf den richtigen Zylinder gebracht werden. Die Zeit für die Positionierung des Kopfes nennt man *Positionierungszeit* . Dann wird abgewartet, bis der gesuchte Sektor am Kopf vorbeiläuft (*Latenzzeit*). Schließlich kann der Sektorinhalt gelesen werden. Die gelesenen Daten kommen zunächst in einen Pufferspeicher (buffer). Die Zeit, die hierfür benötigt wird, nennt man *Übertragungszeit* . Schreibzugriffe funktionieren entsprechend. Die *Zugriffszeit* auf einen Sektor ist definiert als die Summe von Positionierungs-, Latenz- und Übertragungszeit.

wahlfreier Zugriff

Positionoierungszeit

Latenzzeit

Puffer

Übertragungszeit

Am längsten dauert gegenwärtig noch die Positionierung des Schreib-/Lesekopfes auf den richtigen Zylinder, Diese Summe aus Positionierungs- und Latenzzeit nennt man *Suchzeit* (seek time). Die Zugriffszeit auf eine Magnetplatte ist deshalb sehr viel länger – etwa tausend mal – als die Zeit, die beim Zugriff auf eine Zelle im Hauptspeicher vergeht.

Suchzeit

Zugriffszeit

Die Suchzeit für die Kopfpositionierung entspricht in etwa der Distanz, die der Kopf zurücklegt. Diese Distanz wird durch die Anzahl der Spuren gemes-

sen, über die der Kopf bewegt werden muss. Folgende Maßnahmen helfen, die Suchzeit zu verringern:

Suchzeit verringern

- zusammengehörende Information sollte möglichst in benachbarten Sektoren und Zylindern gespeichert werden;
- wenn bei einem Plattenlaufwerk mehrere Schreib-/Leseaufträge gleichzeitig vorliegen, sollte eine günstige Bearbeitungsreihenfolge gewählt werden (disk scheduling).

Auch diese Aufgaben werden vom Betriebssystem erledigt; näheres hierzu folgt in Abschnitt 1.3.1.

Übungsaufgabe 1.4 Wenn für ein Plattenlaufwerk[15] mehrere Aufträge vorliegen, kann man sie nach folgenden Strategien abarbeiten:

- FCFS (first-come, first-served) bearbeitet die Aufträge in der Reihenfolge ihres Eingangs.
- SSTF (shortest-seek-time-first) bearbeitet jeweils denjenigen Auftrag als nächsten, dessen Spur der momentanen Position des Schreib-/Lesekopfs am nächsten liegt.
- SCAN bewegt den Kopf abwechselnd von außen nach innen und von innen nach außen über die gesamte Platte und führt dabei die Aufträge aus, deren Spuren gerade überquert werden.

Angenommen, es sind Zugriffsaufträge für Sektoren in den Spuren

32, 185, 80, 126, 19, 107

in dieser Reihenfolge eingegangen. Der Kopf steht anfangs auf Spur 98 und bewegt sich im Falle der SCAN-Strategie gerade nach außen. Welche Distanz muss der Kopf bei Anwendung der drei Strategien jeweils insgesamt zurücklegen, bis alle Aufträge erledigt sind?

Zwar dauert ein Zugriff auf die Magnetplatte sehr viel länger als ein Hauptspeicherzugriff, aber dafür liefert er ein sehr viel größeres Datenvolumen: Ein Sektor enthält zwischen 0,5 und 2 KByte Nutzinformation, zusätzlich sind Sektor- und Zylindernummer und Hilfsdaten für die Fehlerkorrektur gespeichert. Zu diesem Zweck kann man sogenannte *parity bits* einführen, deren Wert die Summe[16] der Werte bestimmter Datenbits sind. Beim Schreiben bzw. Lesen eines Sektors werden diese Summen berechnet und in die Prüfbits eingetragen bzw. mit den dort gespeicherten Werten verglichen. Ergeben sich Abweichungen, so ist ein Fehler aufgetreten. Bei Verwendung geeigneter Codes kann man feststellen, welches Bit einen falschen Wert hat und den Fehler automatisch korrigieren. Die Nutzinformation eines Sektors wird als ein *Block* bezeichnet.

parity bit

Block

[15] Der Einfachheit halber nehmen wir an, dass das Laufwerk nur eine einzelne Platte enthält. Die Spuren sind von außen nach innen durchnummeriert, mit 0 beginnend.

[16] Beim Addieren von Bits wird modulo 2 gerechnet. Anders ausgedrückt: Eine Summe vor Nullen und Einsen hat den Wert eins, wenn die Anzahl der Einsen ungerade ist, und sonst den Wert null.

Um zeitaufwendige Plattenzugriffe nach Möglichkeit einzusparen, kann man einen *Cache* im Hauptspeicher einrichten, in dem einzelne Blöcke oder die Inhalte ganzer Spuren gespeichert werden, in der Annahme, dass man sie in Kürze noch einmal benötigt. Das Prinzip ist dasselbe wie in Abschnitt 1.2.2.

Cache für Platten

Manche Computer verfügen über eine sogenannte *RAM-disk*. Hierunter versteht man einen fest reservierten Teil des Hauptspeichers, der von Anwenderprogrammen und Anwendern wie eine Magnetplatte benutzt werden kann. Zum Beispiel werden Befehle zur Dateiverarbeitung, wie wir sie in Abschnitt 1.4.3 kennenlernen werden, in entsprechende Speicherzugriffe übersetzt. Das geht beinahe so schnell wie bei „reinen" Hauptspeicherzugriffen. Ein Nachteil: Auch bei der RAM-disk sind die Daten nach Abschalten des Stroms verschwunden.

RAM-disk

Auf der Magnetplatte gespeicherte Programme und Daten sind zwar gegen Stromausfall geschützt; trotzdem sind Platten für die *Langzeitarchivierung* aus folgenden Gründen nicht gut geeignet:

- Magnetplatten sind für wahlfreien Zugriff konzipiert und deshalb pro MByte Kapazität zu teuer, um darauf Informationen abzulegen, auf die nur selten zugegriffen wird.
- Sie sind oft im selben Gehäuse untergebracht wie Prozessor und Hauptspeicher und erlauben es deshalb nicht, Informationen an getrenntem Ort zu verwahren oder auf andere Rechner zu transferieren.

Aus diesen Gründen gibt es neben dem Sekundärspeicher auch *Tertiärspeicher*; damit beschäftigt sich der folgende Abschnitt.

1.2.4 Tertiärspeicher

Das wesentliche Merkmal von Tertiärspeichern ist, dass sich der Datenträger preiswert herstellen und leicht vom Rechner entfernen lässt. Sie eignen sich gut für den Transfer von Daten zwischen nicht vernetzten Rechnern.

Bis ca. 2000 waren *Disketten* weit verbreitet, besonders bei PCs. Danach wurden Diskettenlaufwerke aber immer seltener in neue Computer eingebaut, heute sind sie praktisch verschwunden, ihre Rolle als bequem transportierbares Speichermedium nahmen zunächst die CDs und DVDs ein, die aber wegen der eingeschränkten Wiederbeschreibbarkeit nicht wirklich funktionsgleich sind, und dann die USB-Sticks; siehe dazu weiter unten in diesem Abschnitt.

Diskette

Diskettenlaufwerke sind im Prinzip ähnlich den Festplattenlaufwerken, sie sind aber einfacher aufgebaut. Auch die Diskette bietet wahlfreien Zugriff; dem im Verhältnis zur Magnetplatte deutlich geringeren Preis stehen aber eine erheblich längere Zugriffszeit und ein viel kleineres Datenvolumen (nur ca. 1 MByte) gegenüber.

Für die *Langzeitarchivierung* von Daten und Programmen werden gern *Magnetbänder* verwendet. Ihre Kapazität ist sehr viel größer als die einer Magnetplatte, da auch die magnetisierbare Oberfläche größer ist. Wie im Audiobereich gibt es Bandspulen und Kassetten, die leichter zu wechseln sind. Auch Videokassetten finden Verwendung. Die Daten werden blockweise aufgezeichnet. Im Gegensatz zu den bis jetzt besprochenen Speicherarten bieten

Magnetband

sequentieller Zugriff

Magnetbänder aber nur *sequentiellen Zugriff*: Man kann nur auf die beiden Blöcke direkt zugreifen, die sich gerade vor oder hinter dem Schreib-/Lesekopf befinden. Will man andere Blöcke lesen, muss das Band erst vor- oder zurückgespult werden, was zu extrem langen Zugriffszeiten führt. Im Unterschied zur Magnetplatte brauchen beim Magnetband die Blöcke nicht gleich lang zu sein. Weil auf Bändern oft keine Dateisysteme angelegt werden, kann es Mühe bereiten, ein fremdes Band richtig zu lesen.

In den Rechenzentren früherer Jahre war die Bedienmannschaft (Operator) dafür verantwortlich, bei Bedarf die benötigten Bänder oder Wechselplatten in die Laufwerke einzulegen. Diese Aufgabe kann heute von Robotern übernommen werden. Oft fasst man einige hundert Kassetten zu einer sogenannten *juke box*[17] zusammen, die mit einer Wechselmechanik versehen ist.

juke box

CD-ROM
DVD

Als Ersatz für die Disketten kamen zunächst die *CD-ROMs* (compact disks, kurz CDs) und dann die *DVDs* (digital versatile disks) zum Einsatz. Es handelt sich um 1,2 mm dicke Scheiben aus transparentem Polycarbonat mit 12 cm Durchmesser, die Speicherkapazität beträgt bei CDs mindestens 650 MByte und bei DVDs mindestens 4,38 und maximal 8,5 GByte. Entsprechende Laufwerke werden seit ca. 1995 in praktisch alle Computer eingebaut. Kommerzielle Software wird heute meist auf CDs oder DVDs ausgeliefert, die bei hoher Kapazität extrem preisgünstig herzustellen und leicht zu transportieren sind. Die Daten werden optisch gespeichert und von einem Laser abgetastet.

Zu den nur lesbaren CD-ROMs und DVD-ROMs kamen dann auch schnell die einmal beschreibbaren CD-Rs und DVD-Rs sowie die löschbaren und deshalb mehrfach wiederbeschreibbaren Varianten CD-RW (compact disk – read/write) und DVD-RW, durch die die Anwendungsmöglichkeiten dieser Speichertechnik stark vergrößert wurden. Trotzdem sind diese Medien kein vollwertiger Ersatz für die Disketten, weil nicht beliebige Sektoren schnell gelöscht und überschrieben werden können. Die Zugriffszeiten für CDs und DVDs variieren je nach Umdrehungsgeschwindigkeit und Zugriffsart, sind aber jedenfalls wesentlich länger als die von Magnetplatten und kürzer als die von Disketten.

Blu-ray Disk
HD DVD

Die neuen optischen Speichermedien *Blu-ray Disk* (kurz BD) und *HD DVD* wurden als Nachfolger der DVD entwickelt. Die beiden Formate sind technisch sehr ähnlich, den *Formatkrieg* hat schließlich die BD gewonnen. Die Speicherkapazität der Blu-ray Disk beträgt mindestens 25 GByte, die Datenübertragungsrate ist viermal so hoch wie die einer DVD bei gleicher Umdrehungszahl, beträgt also mindestens 4,5 MByte pro Sekunde bei sogenannter „einfacher“ und 36 MByte pro Sekunde bei „achtfacher“ Geschwindigkeit.

Übungsaufgabe 1.5 Sortieren Sie alle bis jetzt behandelten Speicherformen nach Zugriffszeit.

Externe Festplatten

Externe Festplatten sind handelsübliche Festplatten, die statt direkt in den Computer in ein besonderes, meist recht kleines Gehäuse eingebaut und über eine Standardschnittstelle wie z. B. USB, Firewire oder eSATA angeschlossen

[17] Der Name *juke box* erinnert an die Musik-Boxen, die Vinyl-Schallplatten wechseln können.

werden. Oft erfolgt sogar die Stromversorgung auch über diese Schnittstelle, so dass die Festplatte sehr leicht angeschlossen, beschrieben, abgezogen, transportiert und woanders wieder angeschlossen werden kann. Die Speicherkapazität ist im Vergleich zu CDs und DVDs sehr groß, und die Zugriffszeiten sind je nach Schnittstelle fast so kurz wie von internen Festplatten.

Flash

Noch preisgünstiger als externe Festplatten sind *Flash*-basierte Speicher. Hier handelt es sich um Speicherchips, die Daten permanent, also auch ohne Stromversorgung, speichern, und die auch schnell überschrieben werden können. Solche Speicher existieren einerseits in der Form von sogenannten *USB-Sticks*, d. h. direkt an einen USB-Stecker angeschlossene Speicherchips in einem sehr kleinen Gehäuse, und andererseits als noch kleinere *Speicherkarten*, die in Digitalkameras und Mobiltelefonen eingesetzt werden. Kartenschächte zur Aufnahme dieser Speicherkarten werden schon in viele neuere Rechner eingebaut.

USB-Sticks
Speicherkarten

Daten sichern!

Zum Abschluss unserer Besprechung der verschiedenen Speicherarten möchte ich einen dringenden Appell an Sie richten: Sichern Sie Ihre Programme und Daten regelmäßig, und nutzen Sie dabei die Möglichkeiten, die Ihnen die verschiedenen Speichertypen bieten!

Konkret bedeutet das: Wer interaktiv an einem längeren Text oder an einem Progammierprojekt arbeitet, sollte regelmäßig den aktuellen Stand auf der Festplatte sichern.[18] Ansonsten kann es passieren, dass bei einem Stromausfall oder bei einem Rechnerabsturz die Arbeit einiger Stunden verloren geht – vielleicht keine Katastrophe, aber doch höchst ärgerlich.

Speicherung auf einer Festplatte allein genügt aber nicht, denn bei einem Laufwerksdefekt kann der Platteninhalt verloren sein. Verschwindet dabei zum Beispiel kurz vor dem Abgabetermin Ihre Abschlussarbeit, so kann das durchaus katastrophale Folgen haben. Deshalb muss man regelmäßig von wichtiger Information Sicherheitskopien (backups) anlegen, sei es nun auf Diskette, Magnetband, CD, externer Festplatte oder Flash-Speicher.

Wichtig ist, dass diese Sicherungskopien an einem anderen Ort verwahrt werden als der Rechner. Denn sonst bleibt die Gefahr, dass bei Einbruch, Feuer- oder Wasserschaden doch alle Daten verloren gehen; im Geschäftsleben könnte das den Ruin bedeuten.

Bezeichnungen

Zum Schluss ein paar Bemerkungen zu den Bezeichnungen. Die Namen Haupt- bzw. Primärspeicher, Sekundärspeicher und Tertiärspeicher weisen auf eine hierarchische Ordnung hin: Was im Primärspeicher keinen Platz findet, kommt in den Sekundärspeicher; was dort nicht hineinpasst, steht im Tertiärspeicher. Es gibt noch eine zweite Beziehung: Der Sekundärspeicher enthält eine (möglicherweise aktuellere) Kopie von Teilen der Information des Tertiärspeichers. Ebenso enthält der Hauptspeicher aktuellere Kopien von Sekundärspeicherninformation. Jede Schicht in Abbildung 1.4 fungiert also als Cache für die Schicht unter ihr.

Cache-Modell

[18] Manche Anwenderprogramme führen solche Sicherungen im Abstand einiger Minuten automatisch durch; es kann aber nicht schaden, selbst regelmäßig die Funktion *Sichern* (save) aufzurufen, die von den meisten Anwendungen angeboten wird.

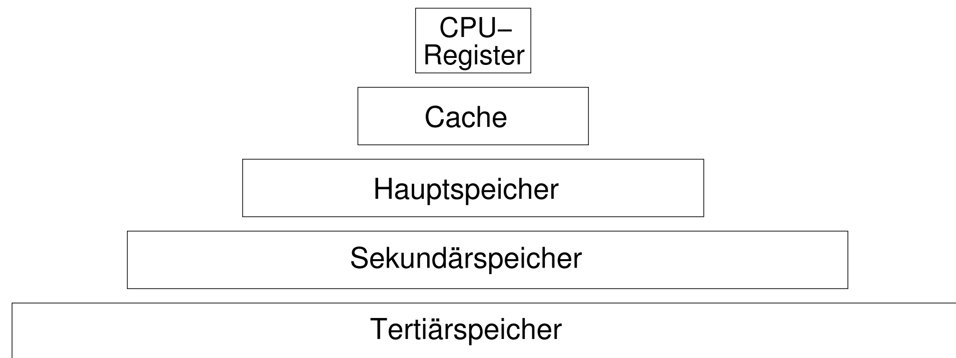


Abbildung 1.4: Jede Schicht arbeitet als Cache für die nächsttiefere Schicht. Die Zugriffszeiten nehmen nach oben hin ab, die Kosten nehmen zu.

Externspeicher

Ein-/Ausgabegeräte

Sekundär- und Tertiärspeicher werden manchmal auch *Externspeicher* genannt, weil sie in eigenen Gehäusen untergebracht sein können. Man bezeichnet diese Komponeten auch als *Ein-/Ausgabegeräte*, zusammen mit Drucker, Monitor, Tastatur und Maus, denn sie ermöglichen den Fluss von Information in das Rechnerinnere herein und hinaus.

1.3 Die Struktur von Computersystemen

In den vorangegangenen Abschnitten haben wir uns mit wichtigen Komponenten vertraut gemacht, aus denen die Hardware eines Computers besteht: Prozessor, Cache, Hauptspeicher, Sekundär- und Tertiärspeicher. In diesem Kapitel wollen wir untersuchen, wie die Komponenten eines Computers zusammenwirken, welche Hilfe die Rechnerhardware dabei leistet und welche Aufgaben das Betriebssystem übernimmt.

1.3.1 Gerätesteuerung

Bus

Protokoll

In Abschnitt 1.2.1 wurde beschrieben, wie CPU und Hauptspeicher über Adress- und Datenbus miteinander kommunizieren. Auch die Ein-/Ausgabegeräte sind mit Prozessor und Hauptspeicher über einen *Bus* verbunden. Ein Bus ist – grob gesagt – ein Bündel von Leitungen zusammen mit einem *Protokoll*, das genau festlegt, welche Nachrichten über den Bus geschickt werden können und wie diese Nachrichten durch Signale auf den Leitungen dargestellt werden. Ein Bus verbindet – anders als ein Kabel – nicht nur zwei Geräte miteinander, sondern viele Geräte, die alle über den Bus miteinander kommunizieren können. Zu jedem Zeitpunkt kann aber nur *eine* Nachricht über den Bus verschickt werden; welches Gerät den Bus zuerst „ergreift", ist an der Reihe. Abbildung 1.5 zeigt als Beispiel einen Teil eines Computersystems.[19]

Controller

Es fällt auf, dass die Geräte nicht direkt an den Bus angeschlossen sind,

[19]Darstellungen wie diese sind schematisch; die Architektur realer Rechner kann erheblich komplizierter sein. Zum Beispiel kann es einen speziellen Bus geben, der CPU, Cache und Hauptspeicher miteinander verbindet.

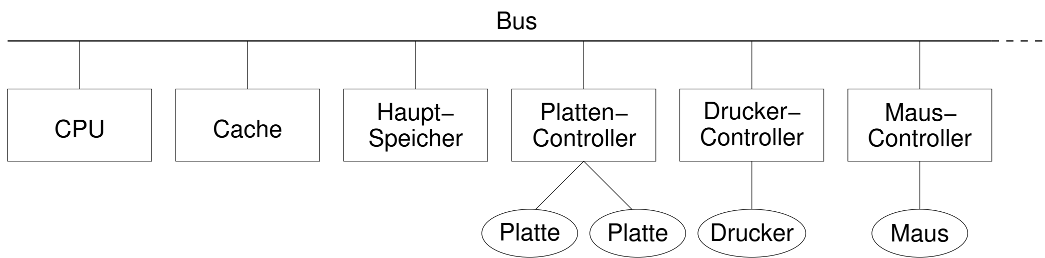


Abbildung 1.5: Ein Computersystem.

sondern über einen sogenannten *Controller*. Das hat folgenden Grund: Wenn die CPU alle Geräte selbst steuern müsste – man denke nur an das ständige Berechnen von Prüfsummen beim Lesen einer Platte –, wäre sie damit so belastet, dass für die eigentliche Programmausführung zu wenig Zeit bliebe. Aus diesem Grund gibt es für jeden Gerätetyp einen Controller, der die Steuerung des „nackten" Geräts übernimmt. Oft können mehrere Geräte gleichen Typs an einen Controller angeschlossen werden. Die Controller und die CPU arbeiten parallel, aber nur einer von ihnen kann zu einem gegebenen Zeitpunkt den Bus benutzen.

Platten-Controller

Ein Controller ist ein Stück elektronische Hardware. Wie kompliziert er ist, hängt vom jeweiligen Gerätetyp ab. So kommt zum Beispiel eine Maus mit einem sehr einfachen Controller aus, während der Controller einer Magnetplatte schon recht kompliziert ist. Meist wird hierfür ein vollwertiger Prozessor verwendet, der auf einer Platine im Gehäuse des Plattenlaufwerks untergebracht ist. Der Plattencontroller steuert zum Beispiel die Armbewegung, führt Fehlererkennung durch und kann einen schadhaften Sektor der Platte dem Betriebssystem melden. Wenn keine Fehler vorliegen, extrahiert der Controller die Nutzinformation aus dem gelesenen Sektor und schreibt den Block in einen controllereigenen Pufferspeicher.[20]

Gerätetreiber

Das Betriebssystem hat die Aufgabe, die Geräte zu steuern; dazu kommuniziert es mit den Controllern. Weil sich die Controller sehr voneinander unterscheiden, ist für jeden Controller ein eigener Teil des Betriebssystems zuständig: ein *Gerätetreiber*; siehe Abbildung 1.6. Als Teil des Betriebssystems ist der Gerätetreiber ein Stück Software.

Register des Controllers

Der Gerätetreiber kommuniziert nur mit dem Controller, nicht direkt mit dem Gerät. Zu diesem Zweck besitzt der Controller mehrere Register, darunter die folgenden:

- *Datenausgangsregister* (data-out); hierhin schreibt der Treiber Daten, die für den Controller bestimmt sind;
- *Dateneingangsregister* (data-in), in das der Controller Daten schreibt, die für den Gerätetreiber bestimmt sind;
- *Statusregister* (status); hier kann der Treiber den Zustand des Geräts

[20]Das geschieht deshalb, weil der für die Übertragung in den Hauptspeicher benötigte Bus gerade besetzt sein könnte; Näheres hierzu finden Sie in Abschnitt 1.3.4.

abfragen, ob es zum Beispiel noch beschäftigt ist oder ob Daten aus dem Dateneingangsregister abgeholt werden können;

- *Kontrollregister* (control); hier hinterlegt der Treiber Befehle an den Controller, zum Beispiel einen Lesebefehl.

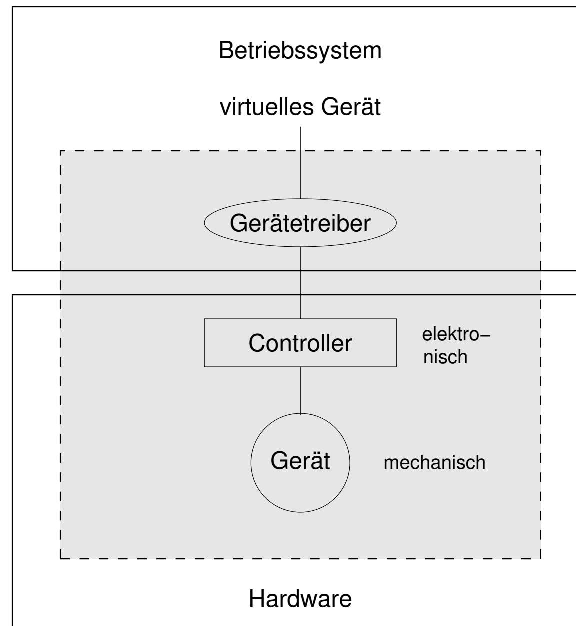


Abbildung 1.6: Ein virtuelles Gerät.

Für die Kommunikation mit dem Controller kann der Gerätetreiber entweder spezielle Ein-/Ausgabebefehle verwenden, die über besondere Busadressen auf die Controllerregister zugreifen. Oder man kann die Register des Controllers als Teil des Hauptspeichers realisieren. Der auf der CPU laufende Gerätetreiber kann dann die sehr viel schnelleren Prozessorbefehle für den Datenaustausch mit den Controllerregistern verwenden; man vergleiche das Beispiel in Abschnitt 1.2.1. Diese Technik wird *speicherabgebildete Ein-/Ausgabe* (memory-mapped I/O) genannt.

memory-mapped I/O

Die Anwendungsprogramme und andere Teile des Betriebssystems können nur über den Gerätetreiber auf das Gerät zugreifen.

Wie sich Controller und Gerätetreiber die Arbeit teilen, ist von Fall zu Fall verschieden. Generell ist die Implementierung von Funktionen in Hardware effizienter als eine Softwarelösung, und sie dient der Abstraktion, weil die Details in der Hardware verschwinden. Eine Implementierung in Software macht dagegen oft weniger Arbeit und lässt sich leichter ändern.

Hardware vs. Software

Gerätetreiber, Controller und das Gerät bilden konzeptuell eine Einheit, die als *virtuelles Gerät*[21] bezeichnet wird; siehe Abbildung 1.6.

virtuelles Gerät

[21] *Virtuell* kommt von dem lateinischen Wort *virtus* für Kraft oder Stärke und bedeutet *der Kraft oder Möglichkeit nach vorhanden.*

1.3.2 Exkurs: Abstraktion, Kapselung und Schichtenmodell

Im Modell des virtuellen Geräts stecken drei wichtige Konzepte:

- Abstraktion,
- Kapselung, auch Geheimnisprinzip genannt, und
- Schichtenmodell.

Diese Konzepte spielen in der Informatik eine große Rolle, nicht nur bei den Betriebssystemen, sondern auch beim Software Engineering und beim Algorithmenentwurf. Grund genug also, sich etwas näher mit ihnen zu beschäftigen. Als Beispiel wählen wir ein Magnetplattenlaufwerk, dessen Funktionsweise in Abschnitt 1.2.3 diskutiert worden ist.

Abstraktion

Abstraktion bedeutet, von technischen Details abzusehen und sich auf das Wesentliche, Prinzipielle zu konzentrieren. Wesentlich an der Festplatte ist, dass wir sie als

array[0..n-1] of block

auffassen können und dass man auf die Blöcke wahlfrei zugreifen kann. Unwesentlich ist dagegen, wie ein Block auf der Platte durch Sektoren realisiert wird und welche Verwaltungsinformation darin gespeichert wird; denn diese zusätzliche Information wird nur intern vom Controller benötigt, aber nicht außerhalb des virtuellen Geräts.

Unwesentlich ist auch, wie weit der Arm des Laufwerks bewegt werden muss, um einen bestimmten Block zu lesen; auf der Abstraktionsebene des virtuellen Geräts – das heißt an der Schnittstelle des Gerätetreibers zum Rest des Betriebssystems – gibt es nur Befehle wie

read(b), write(b), seek(i),

wobei b einen Block bezeichnet und i eine Blocknummer.

Kapselung

Rekapitulieren wir: Das Konzept der Abstraktion besagt, dass man auf höherer Ebene gewisse Details nicht zu kennen *braucht*. Das Prinzip der *Kapselung* geht noch einen Schritt weiter: man *darf* die Details nicht einmal kennen! Aus dieser Forderung ist die alternative Bezeichnung *Geheimnisprinzip* entstanden.

Im Beispiel mit der Magnetplatte bedeutet das: Die Befehle für die Bewegung des Arms existieren nur im Innern des virtuellen Geräts; außerhalb kann man diese Befehle dagegen nicht aufrufen.

Gründe für Kapselung

Warum ist solch eine strenge Kapselung sinnvoll? Zwei gute Gründe lassen sich anführen. Erstens: Wenn Sie den Auftrag bekommen, einen Gerätetreiber zu schreiben, so haben Sie volle und alleinige Kontrolle darüber, dass Befehle zur Armsteuerung nur auf sinnvolle Weise verwendet werden, also zum Beispiel nicht bei stehendem Motor. Könnte dagegen jeder Benutzer diese Befehle aufrufen, so ergäbe sich hier ein Problem.Und zweitens: Nehmen wir

Missbrauch verhindern

Änderungen erleichtern

an, das Plattenlaufwerk wird irgendwann durch ein leistungsfähigeres ersetzt. Dann ändern sich möglicherweise auch die Befehle zur Armsteuerung. Würden nun diese Befehle im gesamten Betriebssystem und sogar von Anwenderprogrammen benutzt, so müsste die gesamte Software erneuert werden – ein enorm hoher Aufwand! Bei guter Kapselung braucht man dagegen nur den Gerätetreiber zu ersetzen.[22]

schadhafte Sektoren

Bei Magnetplatten sind noch weitere Funktionen im virtuellen Gerät gekapselt. Ein Beispiel ist die Erkennung schadhafter Sektoren. Manche Controller unterhalten schon ab Werk ein Verzeichnis aller Sektoren auf den Plattenoberflächen, die nicht einwandfrei funktionieren. Die vom Betriebssystem an den Treiber übergebenen Blöcke werden natürlich nur auf intakte Sektoren geschrieben. Folglich kann es Unterschiede zwischen Block- und Sektornummern geben, die intern vom virtuellen Gerät verwaltet werden.

disk scheduling

Auch die Optimierung der Armbewegungen, wie wir sie in Übungsaufgabe 1.4 diskutiert hatten, lässt sich im virtuellen Gerät kapseln.

Objekte und Methoden

Soviel zu den Begriffen Abstraktion und Kapselung; sie werden Ihnen im weiteren Studium wiederbegegnen, insbesondere im Zusammenhang mit *abstrakten Datentypen*, siehe z. B. die Kurse über *Datenstrukturen* und *objektorientierte Programmierung*. In beiden Fällen geht es darum, Objekte und die Methoden, mit denen auf die Objekte zugegriffen werden kann, zu Einheiten zusammenzufassen und den Zugriff auf die Objekte nur über diese Methoden zu gestatten.

Schichtenmodell

Als drittes wesentliches Konzept erkennen wir am virtuellen Gerät den *Aufbau in Schichten;* siehe Abbildung 1.6. In unserem Beispiel ist die oberste Schicht in Software implementiert, die mittlere in elektronischer Hardware und die unterste in mechanischer Hardware.

Die Kommunikation mit der Außenwelt erfolgt nur in der obersten Schicht. In jeder Schicht werden von oben ankommende Aufträge bearbeitet und zur weiteren Bearbeitung an die nächsttiefere Schicht weitergeleitet. Wenn in der untersten Schicht der Auftrag vollständig erledigt ist, wird dort eine Antwort generiert und durch alle Schichten nach oben geleitet. Die oberste Schicht schickt dann die Antwort an den externen Auftraggeber.

Vor- und Nachteile

Ein solcher Ansatz ist natürlich nur dann sinnvoll, wenn jede Schicht einen wesentlichen Beitrag zur Bearbeitung der Aufträge und Antworten leisten kann.[23] Das Schichtenmodell bietet einen großen Vorteil: Die Implementierung einer einzelnen Schicht ist verhältnismäßig einfach, weil nur die beiden Schnittstellen zur nächsthöheren und zur nächsttieferen Schicht realisiert werden müssen und weil im Innern der Schicht nur eine klar umrissenen Teilaufgabe gelöst werden muss.

Beim Entwurf eines Systems nach dem Schichtenmodell muss man eine Abwägung (trade-off) vornehmen: Je „dünner“, also je einfacher man die Schichten macht, desto mehr Schichten werden benötigt und desto höher wird der Verwaltungsaufwand (overhead) für die Kommunikation zwischen den Schichten.

[22] Treiber für die wichtigsten Betriebssysteme werden oft vom Gerätehersteller mitgeliefert.

[23] „Stempeln und Weiterleiten“ kommt auch bei Behörden allmählich aus der Mode.

1.3.3 Unterbrechungen

Für Menschen ist es ziemlich störend, ständig bei der Arbeit unterbrochen zu werden. Für den Computer sind Unterbrechungen (interrupts) eine natürliche Form der Kommunikation. Warum das so ist, wollen wir jetzt diskutieren.

Beispiel: Lesezugriff

Nehmen wir an, der Gerätetreiber einer Magnetplatte bekommt den Auftrag, einen bestimmten Block zu lesen. Wie in Abschnitt 1.3.1 besprochen, hinterlegt der Treiber einen entsprechenden Lesebefehl im Kontrollregister des Controllers.

Der Controller kann jetzt mit seiner Arbeit beginnen. Aber wie erfährt die CPU davon, wenn der Leseauftrag ausgeführt ist und der gewünschte Block im internen Puffer des Controllers bereitliegt? Auf keinen Fall soll die schnelle CPU auf das langsame Gerät warten müssen. Zwei bessere Möglichkeiten bieten sich an:

polling

- Die CPU kann – neben ihrer anderen Arbeit – immer wieder das Statusregister des Controllers abfragen, um festzustellen, ob der Auftrag schon erledigt ist; diesen *Abfragebetrieb* nennt man im Englischen *polling*. Liegen die Register im Hauptspeicherbereich – wie im Fall von speicherabgebildeter Ein-/Ausgabe – so lässt sich eine solche Abfrage zwar recht schnell erledigen, aber wenn sie immer wieder erfolglos bleibt, wird insgesamt viel CPU-Zeit damit verbraucht.

interrupt

- Der Controller benachrichtigt die CPU, sobald er den Auftrag ausgeführt hat; hierzu unterbricht er die CPU bei ihrer augenblicklichen Arbeit. Dieser *Unterbrechungsbetrieb* bildet die Grundlage für die Arbeitsweise moderner Computersysteme.

Wir werden uns deshalb in diesem Abschnitt damit befassen, wie der Unterbrechungsbetrieb organisiert ist.

Hardware-Unterbrechungen

Die CPU besitzt einen besonderen Eingang für Unterbrechungen. Jede Hardwarekomponente kann über den Bus ein binäres Signal an diesen Eingang legen. Entdeckt die CPU ein Unterbrechungssignal, rettet sie den Inhalt des Befehlszählregisters – also die Adresse des nächsten auszuführenden Befehls – in einen systemeigenen Bereich des Hauptspeichers und lädt stattdessen die Anfangsadresse einer allgemeinen Prozedur zur Unterbrechungsbehandlung (interrupt handler). Diese stellt zunächst fest, welches Gerät die Unterbrechung auslösen will. Hierzu kann sie alle vorhandenen Geräte abfragen; in manchen Rechnern gibt es aber einen speziellen *Unterbrechungs-Controller*, der die Unterbrechungsanforderungen der Geräte verwaltet und ihre Herkunft kennt.

Unterbrechungsvektor

Die Nummer i des Geräts, das die Unterbrechung wünscht, wird nun als Index für den *Unterbrechungsvektor* verwendet. Dieser steht als

array[0..MaxDeviceNumber] of address

im systemeigenen Speicher und enthält an Position i die Startadresse der speziellen Unterbrechungsroutine für Gerät Nummer i; siehe Abbildung 1.7. Diese

Adresse wird in das Befehlszählregister geladen, und die spezielle Unterbrechungsroutine (interrupt service routine) wird gestartet.

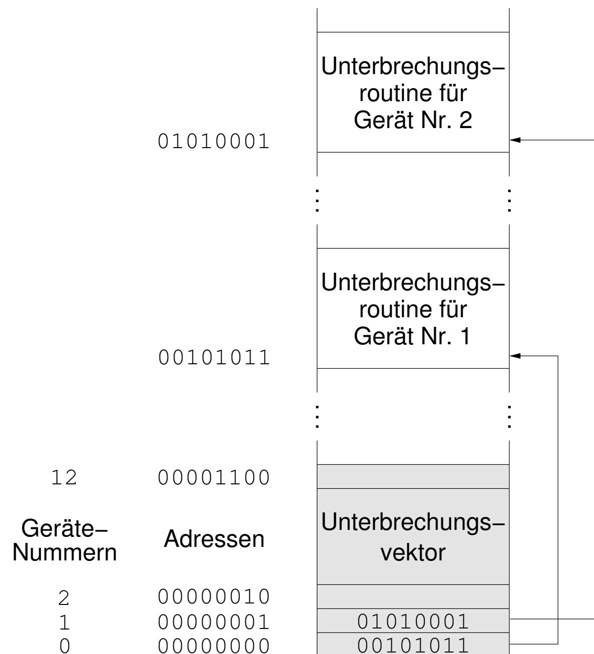


Abbildung 1.7: Ein Unterbrechungsvektor.

Unterbrechungs-routine

Wenn sie für ihre Arbeit längere Berechnungen anstellen muss, so werden zuvor auch die Registerinhalte der CPU in den systemeigenen Speicher gerettet, der meist als *Stapel*[24] (stack) organisiert ist. Die Unterbrechungsroutine hat die Aufgabe, den Unterbrechungswunsch des Geräts zu bearbeiten: Sie kann zum Beispiel den Gerätetreiber darüber informieren, dass der Controller seinen Lesebefehl ausgeführt hat und die Daten nunmehr bereitstehen.

Sobald die Unterbrechungsroutine ihre Arbeit beendet hat, wird zunächst ein spezieller Befehl ausgeführt, der die Rückkehr von der Unterbrechung (return from interrupt) bewirkt; vorher werden die alten Registerinhalte und der alte Wert des Befehlszählregisters wiederhergestellt. Dann kann die CPU mit ihrer unterbrochenen Arbeit fortfahren.

Mehrere Unterbrechungen

Es kann vorkommen, dass während der Bearbeitung einer Unterbrechung ein weiterer Unterbrechungswunsch eintrifft. Zur Vermeidung eines Durcheinanders sind verschiedene Maßnahmen möglich.

Unterbrechungen sperren

Zum einen lässt sich bei vielen Prozessoren der Unterbrechungseingang vorübergehend außer Betrieb setzen (interrupt disabled), so dass weitere Unterbrechungssignale wirkungslos bleiben.[25] So kann die CPU die zuerst eingetroffene Unterbrechung ungestört bearbeiten. Dieses Verfahren bietet sich auch in anderen Situationen an, wenn die CPU bei der Ausführung kritischer Abschnitte eines Programms nicht unterbrochen werden darf; hierauf werden wir in Abschnitt 2.1.2 zurückkommen.

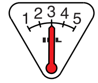
[24] Ein Stapel ist eine Datenstruktur, bei der – wie bei einem Tellerstapel – nur das jeweils oberste Datenelement entfernt werden kann oder ein neues Element oben hinzugefügt werden kann; hierzu mehr im Kurs *Datenstrukturen I*.

[25] Man spricht in diesem Zusammenhang von *maskierten* Unterbrechungen.

Zum anderen kann man jedem Unterbrechungswunsch eine Priorität zuordnen; Unterbrechungen niederer Priorität können dann von solchen mit höherer Priorität unterbrochen werden.

Prioritäten vergeben

Übungsaufgabe 1.6 Nach jeder Ausführung eines Befehls überprüft die CPU, ob an ihrem Unterbrechungseingang ein Signal angekommen ist. Schleicht sich hier durch die Hintertür der ineffiziente Abfragebetrieb ein, den wir durch Einführung des Unterbrechungsbetriebs eigentlich hatten vermeiden wollen?



Wir haben oben gesehen, dass Geräte (genauer: ihre Controller) Unterbrechungen der CPU auslösen, wenn sie ihren Auftrag ausgeführt haben oder wenn ein Fehler aufgetreten ist.

Software-Unterbrechungen

Aber auch die Software kann Unterbrechungen (sogenannte *traps*) auslösen. Hierfür kann es ganz verschiedene Ursachen geben, zum Beispiel Division durch null oder versuchter Zugriff auf nichtexistierende oder geschützte Hauptspeicheradressen; für solche Unterbrechungen wird auch die Bezeichnung *Ausnahme* (exception) verwendet.

Ausnahmen

Systemaufrufe

Eine andere Art von Software-Unterbrechung stellt dagegen keine Ausnahme dar; sie ist vielmehr ein übliches Verfahren zur Kontrollübergabe: Wann immer ein Programm Dienste des Betriebssystems in Anspruch nehmen will, führt es einen *Systemaufruf* (system call) durch. Wir werden in Abschnitt 1.4.3 ausführlicher darauf eingehen, welche Systemaufrufe ein Betriebssystem üblicherweise anbietet. Die verfügbaren Systemaufrufe bilden zusammen die *Programmierschnittstelle* zwischen den Anwenderprogrammen und dem Betriebssystem; siehe auch Abschnitt 1.1.

Parameterübergabe

Solch ein Systemaufruf hat die Form eines Funktionsaufrufs. Insbesondere können dabei auch Parameter übergeben werden, entweder direkt in einem Register der CPU oder indirekt durch Angabe der Anfangsadresse des zu übergebenden Datenbereichs im Hauptspeicher; dieses Vorgehen empfiehlt sich bei großen Datenmengen wie zum Beispiel Bildschirminhalten. Man kann zur Parameterübergabe auch den Stapel des Systems benutzen.

Wenn der Systemaufruf seine Parameter für die Übergabe vorbereitet hat, führt er eine besondere Instruktion[26] aus, die die eigentliche Unterbrechung auslöst. Diese Instruktion ist bei allen Systemaufrufen dieselbe; der gewünschte Systemdienst wird beim Aufruf durch einen Parameter bezeichnet.

Jetzt geschieht in etwa dasselbe wie bei einer Hardware-Unterbrechung: Befehlszähler- und Registerinhalte werden gerettet, und in Abhängigkeit vom übergebenen Parameter für den gewünschten Dienst wird eine entsprechende Routine des Betriebssystems gestartet, die nun die Rolle der Unterbrechungsroutine spielt. Bei Systemaufrufen werden meistens Daten an das aufrufende Programm zurückgegeben.

Wer in einer Hochsprache programmiert, wird möglicherweise nicht immer bemerken, dass sein Programm Systemaufrufe auslöst: Zum Beispiel gibt es in Pascal einen Standardbefehl namens *read*, mit dem ein Datensatz einer Datei

[26] Die Begriffe *Befehl, Anweisung* und *Instruktion* werden in diesem Kurs synonym verwendet.

gelesen werden kann. Ein solcher Befehl wird vom Compiler automatisch in einen entsprechenden Systemaufruf übersetzt, ohne dass der Programmierer die Details kennen muss.

1.3.4 Direkter Speicherzugriff (DMA)

Wir hatten in Abschnitt 1.3.3 besprochen, was geschieht, wenn ein Plattenlaufwerk einen Leseauftrag ausgeführt hat: Der gewünschte Datenblock steht im Pufferspeicher des Controllers bereit, und der Controller schickt ein Unterbrechungssignal an die CPU, um sie hiervon zu informieren.[27] Wie gelangt der Block nun in den Hauptspeicher? Hier gibt es zwei unterschiedliche Verfahren.

PIO

Bei der *programmierten Ein-/Ausgabe* (programmed I/O = PIO) schreibt der Controller jeweils ein Wort in sein Dateneingangsregister und löst eine Unterbrechung aus; siehe auch Abschnitt 1.3.1. Dadurch wird der Gerätetreiber gestartet; er veranlasst, dass das Wort von der CPU in Empfang genommen und in den Hauptspeicher geschrieben wird. Wenn ein Block 512 Bytes enthält und ein Wort zwei Bytes umfasst, wird die CPU bei diesem Verfahren 256 mal unterbrochen, bevor der Datenblock endlich im Hauptspeicher steht!

DMA

Eigentlich ist die CPU für solche niederen Übertragungsdienste zu schade. Deshalb wird für schnelle Peripheriegeräte häufig ein anderes Verfahren gewählt: der *direkte Speicherzugriff* (direct memory access = DMA). Hierbei wird im Computer ein spezieller DMA-Controller eingesetzt, der selbständig über den Bus Daten in den Hauptspeicher übertragen kann, ohne die CPU zu bemühen. Auch der Gerätecontroller muss für das DMA-Verfahren ausgelegt sein.

Arbeitsweise von DMA

Der Gerätetreiber teilt dem Gerätecontroller über dessen Register die Nummer des zu lesenden Blocks mit und dem DMA-Controller die Anfangsadresse des Hauptspeicherbereichs, in den die Daten übertragen werden sollen. Danach kann die CPU sich anderen Aufgaben widmen. Der Gerätecontroller setzt sich nun über spezielle Leitungen mit dem DMA-Controller in Verbindung. Wann immer das nächste Wort im Register des Gerätecontrollers bereitsteht, sendet er über eine *Anforderungsleitung* ein Signal an den DMA-Controller; dieser schreibt die entsprechende Hauptspeicheradresse auf den Adressbus und signalisiert über die *Bestätigungsleitung* dem Gerätecontroller, die Übertragung durchzuführen. Dann zählt der DMA-Controller die Adresse für das nächste Wort hoch. Erst wenn alle Wörter des gesamten Blocks übertragen sind, löst er eine Unterbrechung der CPU aus. Ein Beispiel für den Aufbau eines DMA-fähigen Systems zeigt Abbildung 1.8.

Übungsaufgabe 1.7 Das Ziel des DMA-Verfahrens liegt in einer Entlastung der CPU. Wieso kann es trotzdem vorkommen, dass die Arbeit der CPU verzögert wird, während DMA-Vorgänge ablaufen?

handshaking

Eine Kommunikation wie die zwischen DMA- und Gerätecontroller wird im Englischen als *handshaking* bezeichnet; sie stellt einen einfachen Fall eines Protokolls dar.

[27] Wir nehmen in diesem Abschnitt an, dass im Unterbrechungsbetrieb gearbeitet wird.

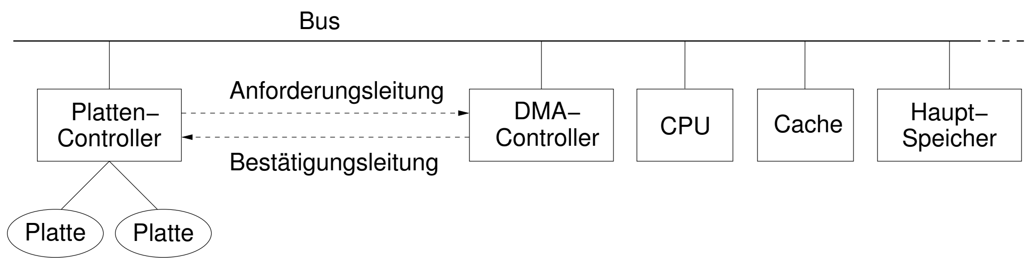


Abbildung 1.8: Eine mögliche Rechnerarchitektur für direkten Speicherzugriff (DMA).

1.3.5 System- und Benutzermodus; Speicherschutz

Start des Rechners

Auch das Betriebssystem selbst ist ein Programm. Wenn der Rechner eingeschaltet wird, steht ein kleiner Teil des Systems – der sogenannte *Ur-Lader* – in einem speziellen Teil des Hauptspeichers, der seinen Inhalt auch nach Abschalten des Stroms behält. Meist ist der Ur-Lader in einem ROM (read only memory) fest eingebrannt.

Ur-Lader

Der Ur-Lader lädt zunächst den eigentlichen Lader von der Magnetplatte.[28] Der Lader wird nun gestartet und lädt das Betriebssystem – zumindest seine wesentlichen Teile – in den Hauptspeicher. Dieser Vorgang wird als *hochfahren* (booting) bezeichnet.

booting

Wenn jetzt ein Anwenderprogramm ausgeführt wird, könnte es im Prinzip versuchen, Teile des Hauptspeicherbereichs zu überschreiben, in dem das Betriebssystem steht; dazu braucht das Programm ja nur die entsprechenden Speicherzellen zu adressieren, wie wir in Abschnitt 1.2.1 besprochen haben. Ebenso könnte ein Benutzerprogramm sich selbst oder andere Programme, die sich gerade im Hauptspeicher befinden, verändern.

Schutz von Speicher und Geräten

Ob eine solche Veränderung nun versehentlich oder mit Absicht geschieht: die Folgen können schlimm sein.[29] Aus diesem Grund kann es einem Benutzerprogramm nicht gestattet werden, auf beliebige Teile des Hauptspeichers zuzugreifen. Auch der Sekundärspeicher muss vor unbefugtem Zugriff geschützt werden, denn von dort wird ja das Betriebssystem beim nächsten Einschalten geladen, und es befinden sich dort auch andere Programme. Schließlich bedarf auch der Zugriff auf die anderen Geräte der Kontrolle; wenn zum Beispiel mehrere Programme gleichzeitig auf den Drucker zugreifen, kann sonst ein Durcheinander entstehen.

System- und Benutzermodus

Um diesen Schutz zu gewährleisten, können moderne Prozessoren im *Systemmodus*[30] (system mode) oder im *Benutzermodus* (user mode) arbeiten;[31] zur Unterscheidung wird ein besonderes Bit im Prozessor verwendet. Alle An-

[28]Hierdurch wird erreicht, dass der Lader aktualisiert werden kann, wenn später einmal eine erweiterte Betriebssystemversion installiert wird.

[29]Der Autor erinnert sich an eigene Erfahrungen mit einem kleinen Rechner, der in Maschinensprache programmiert wurde. Wenn man sich bei den Sprungadressen im Hexadezimalcode verrechnet hatte, traten die merkwürdigsten Fehler auf.

[30]Der Systemmodus wird auch Supervisor-Modus genannt.

[31]Beim Intel 8088 war diese Möglichkeit nicht vorhanden. Deshalb kannte MS-DOS keine Schutzmechanismen, was unter anderem die Anfälligkeit gegen Würmer und Viren erklärt.

privilegierte Befehle

wendungsprogramme laufen im Benutzermodus. Bestimmte *privilegierte Maschinenbefehle* können aber nur im Systemmodus ausgeführt werden. Damit solch ein Schutzmechanismus wirksam ist, muss natürlich der Befehl zum Umschalten vom Benutzer- in den Systemmodus selbst auch privilegiert sein. Muss man also schon privilegiert sein, um Privilegien zu bekommen?

Dank des Betriebssystems nicht! Dieses Problem wird so gelöst: Wenn eine Unterbrechung auftritt – insbesondere wenn ein Benutzerprogramm einen Systemaufruf auslöst – schaltet die privilegierte Prozedur zur Unterbrechungsbehandlung den Prozessor in den Systemmodus und startet dann die entsprechende Unterbrechungsroutine; vergleiche Abschnitt 1.3.3. Sie ist Teil des Betriebssystems und deshalb vertrauenswürdig. Bevor die Unterbrechungsroutine terminiert, schaltet sie in den Benutzermodus zurück.

Kein Benutzer kann also direkt auf ein Gerät zugreifen; man muss zu diesem Zweck das mit höheren Rechten ausgestattete Betriebssystem um Hilfe bitten.

Mit diesen Hilfsmitteln lässt sich auch ein wirksamer Speicherschutz realisieren; und zwar auf folgende Weise:

Adressraum

Bevor ein Programm gestartet wird, weist ihm das Betriebssystem einen bestimmten Bereich im Hauptspeicher zu, seinen sogenannten *Adressraum*. Das Programm darf nur auf solche Adressen zugreifen, die in seinem eigenen Adressraum liegen. Das bedeutet: Alle möglicherweise benötigten Daten, aber auch das Programm selbst müssen in diesem Adressraum enthalten sein.

Basis- und Grenzregister

Ein zusammenhängender Adressraum lässt sich durch die Inhalte von zwei speziellen Registern der CPU festlegen. Im *Basisregister* (base register) steht die niedrigste Adresse des Adressraums; das *Grenzregister* (limit register) enthält die Länge des Adressraums, also die Differenz aus der höchsten und der niedrigsten erlaubten Adresse. Die Inhalte von Basis- und Grenzregister können nur mit speziellen Maschinenbefehlen verändert werden. Diese Befehle sind privilegiert.

Wann immer das Programm zur Laufzeit versucht, auf eine Speicherzelle zuzugreifen, wird zunächst überprüft, ob die angegebene Adresse im erlaubten Bereich liegt; siehe Abbildung 1.9. Wenn das der Fall ist, wird der Speicherzugriff durchgeführt, andernfalls wird eine Software-Unterbrechung ausgelöst, wie in Abschnitt 1.3.3 beschrieben.

Mit einer leichten Modifikation dieser Technik lässt sich gleich noch ein zweites Problem erledigen. Wenn ein Programm vom *Compiler* in Maschinensprache übersetzt wird, steht noch nicht fest, wo im Hauptspeicher der Adressraum des Programms später liegen wird – das hängt ja auch davon ab, welche anderen Programme dann gerade im Hauptspeicher stehen. Der Compiler kann deshalb an die Befehle und die Daten zunächst nur *relative Adressen* vergeben wie etwa „17 Wörter hinter dem Anfang dieses Programmstücks“.[32]

relative Adressen

Wird das Programm nun in seinen Adressraum geladen, stimmen die im Programm verwendeten relativen Adressen nicht mit den absoluten Adressen überein, an denen die Befehle und Daten tatsächlich stehen. Man könnte deshalb vor dem Laden alle relativen Adressen im Programm um die An-

absolute Adressen

[32] Auch die relativen Adressen werden in Binärdarstellung angegeben. Würde der Adressraum bei 0 beginnen, könnte man sie als absolute Adressen verwenden. An der Adresse 0 beginnt aber oft der Unterbrechungsvektor.

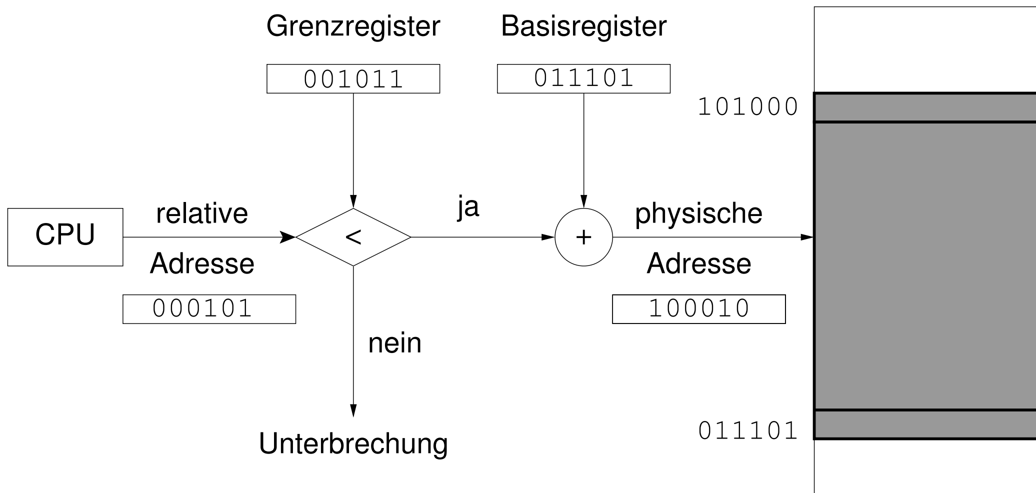


Abbildung 1.9: Speicherschutz mit Basis- und Grenzregister.

fangsadresse des Adressraums erhöhen. Diese Lösung ist nicht so elegant; sie benötigt nämlich zusätzliche Information darüber, wo im Programm die Adressen stehen.[33] Diese Hilfsinformation müsste eigens vom Computer generiert werden. Viel günstiger ist es, die Neuberechnung der Adressen nach der Formel

$$\text{absolute Adresse} = (\text{Basisregister}) + \text{relative Adresse}$$

erst zur Laufzeit durchzuführen! Das heißt: Wenn das Programm eine Speicherzelle adressieren will, wird die im Programm enthaltene relative Adresse zur niedrigsten Adresse des Adressraums addiert; diese steht im Basisregister.

Damit wird auch sichergestellt, dass die absolute Adresse nicht unterhalb des Adressraums liegt. Vor der Addition ist noch der erste Test aus Abbildung 1.9 nötig, damit der Zugriff nicht über die obere Grenze des Adressraums ausgeht.

Weil die Programme zusammen mit ihren Adressräumen beliebig im Hauptspeicher verschoben werden können, spricht man bei diesem Verfahren von *relokierbaren Programmen* (relocatable code).

relokierbare Programme

Übungsaufgabe 1.8 Warum darf der Unterbrechungsvektor nicht im Adressraum eines Anwendungsprogramms liegen?

Übungsaufgabe 1.9 Muss der Befehl, mit dem der Unterbrechungseingang der CPU außer Betrieb gesetzt wird, privilegiert sein?

1.4 Die Struktur von Betriebssystemen

In Abschnitt 1.3 haben wir untersucht, wie Computersysteme aufgebaut sind und welche Dienste die Hardware dem Betriebssystem zur Erfüllung seiner

[33] Erinnern wir uns an Abschnitt 1.2.1: Den Worten allein sieht man nicht an, ob sie Befehle, Daten oder Adresse darstellen.

Aufgaben anbietet. Beispiele dafür sind Unterbrechungen, DMA und die Umschaltung zwischen System- und Benutzermodus.

Wir wollen uns in diesem Abschnitt näher mit dem Aufbau des Betriebssystems beschäftigen und weiter untersuchen, welche Dienste es den Benutzern und ihren Programmen zur Verfügung stellt.

1.4.1 Prozesszustände

In Abschnitt 1.3.3 haben wir beobachtet, was geschieht, wenn die CPU bei der Ausführung eines Programms unterbrochen wird: Die Adresse des nächsten auszuführenden Befehls und die Registerinhalte werden in Systemstapel gespeichert; damit wird die CPU für andere Aufgaben frei. Später können die Registerinhalte wieder geladen werden, und die CPU kann ihre Arbeit an der richtigen Stelle fortsetzen, so als hätte es die Unterbrechung nicht gegeben.

Prozess

Prozesskontext

Ein Programm, das sich gerade in Ausführung befindet, heißt *Prozess*. Zum Prozess gehört aber nicht nur ein Programmstück, sondern auch der *Prozesskontext* bestehend aus

- den Registerinhalten, insbesondere
- Befehlszähler und
- Grenzen des Adressraums, sowie der
- Prozessnummer

PCB

und anderen Informationen. Diese Angaben werden im *Prozesskontrollblock* (process control block = PCB) zusammengefasst. Dort steht auch vermerkt, ob der Prozess im System- oder im Benutzermodus arbeitet.

Prozesszustände

Ein Prozess wird zu irgendeinem Zeitpunkt *erzeugt* und zu einem späteren Zeitpunkt *beendet*. Dazwischen kann er mehrfach zwischen drei Zuständen wechseln, die in Abbildung 1.10 dargestellt sind.[34]

rechnend

bereit

blockiert

Bei den Einprozessorsystemen, wie wir sie in diesem Kurs betrachten, ist zu jedem Zeitpunkt genau ein Prozess im Zustand *rechnend*; nämlich der Prozess, dessen Programm gerade von der CPU ausgeführt wird. Jeder andere existierende Prozess ist entweder *bereit* und bewirbt sich mit den übrigen bereiten Prozessen um die Zuteilung der CPU, oder er ist *blockiert* und wartet darauf, dass seine Ein-/Ausgabeanforderung erledigt wird oder dass ein anderes Ereignis eintritt.

Beispiel: Ein-/Ausgabe

Wenn zum Beispiel ein laufendes Benutzerprogramm auf eine Eingabe von der Tastatur wartet, führt es einen Systemaufruf durch, wie in Abschnitt 1.3.3 beschrieben.

Ohne die Eingabedaten kann der Benutzerprozess nicht weiterarbeiten; er wird deshalb vom Betriebssystem vom Zustand rechnend in den Zustand blockiert versetzt.

[34]Solche Diagramme werden gern zur Beschreibung von Systemen verwendet, die endlich viele innere Zustände annehmen können. Die Beschriftung an einem Pfeil gibt an, durch welchen äußeren Reiz der betreffende Zustandswechsel ausgelöst wird. Dort könnte auch vermerkt sein, wie das System dabei nach außen reagiert. Diese Systeme heißen *endliche Automaten*. Sie werden in Kurs *Grundlagen der Theoretischen Informatik* behandelt.

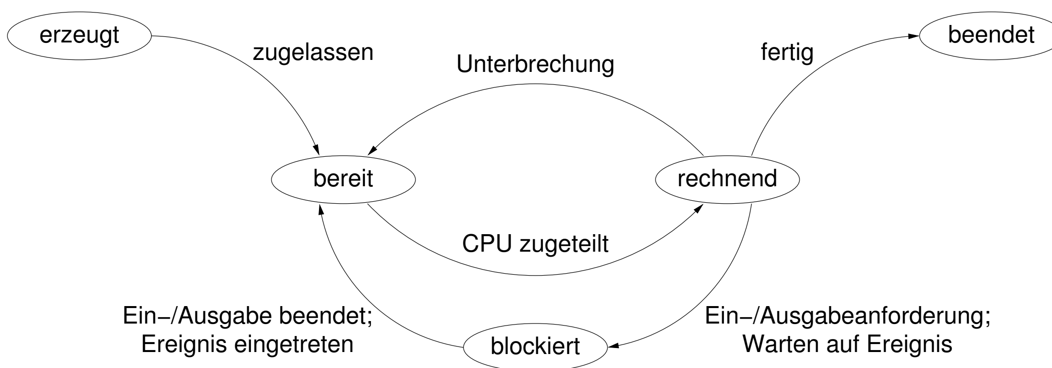


Abbildung 1.10: Mögliche Prozesszustände

Danach ist die CPU frei, um andere Prozesse zu bearbeiten. Wenn dann die Daten bereitstehen und der Tastaturcontroller eine Unterbrechung auslöst, wird der Benutzerprozess in den Zustand bereit gebracht, denn seine Arbeit kann nun weitergehen. Das heißt aber nicht, dass er sofort die CPU zugeteilt bekommt: Welcher der bereiten Prozesse als nächster rechnen darf, entscheidet der *CPU-Scheduler*.[35]

CPU-Scheduler

Der Scheduler kann seine Entscheidungen nach unterschiedlichen Strategien treffen: Bei FCFS (first-come, first-served) darf zuerst rechnen, wer zuerst kommt; vergleiche Abschnitt 1.2.3. Dieses Verfahren ist leicht zu implementieren; es genügt, eine *Warteschlange* (queue) einzurichten, bei der immer der erste Prozess als nächster an die Reihe kommt und neu eintreffende bereite Prozesse sich hinten anstellen. Der Nachteil: Ein früh eintreffender Prozess mit hohem Rechenzeitbedarf oder einer Endlosschleife hält alle späteren Prozesse auf.

Scheduling-Strategien

FCFS

Diesen Nachteil kann man durch das Verfahren *SJF* (shortest job first) vermeiden; hier werden die bereiten Prozesse in der Reihenfolge aufsteigenden Rechenzeitbedarfs bearbeitet: die kurzen zuerst.[36] Voraussetzung für eine sinnvolle Anwendung von SJF ist, dass sich die benötigte Rechenzeit (bis zur nächsten Unterbrechung) aus Erfahrungswerten gut vorhersagen lässt.[37] Unter dieser Voraussetzung ist SJF sehr effizient, wie die folgende Aufgabe zeigt.

SJF

Übungsaufgabe 1.10 Gegeben sei eine feste Menge von endlich vielen Prozessen mit bekannten Rechenzeiten. Beweisen Sie, dass SJF die Gesamtwartezeit minimiert, also die Summe aller Wartezeiten der einzelnen Prozesse. Dabei ist die Wartezeit eines Prozesses diejenige Zeit bis zur Beendigung, in der sich der Prozess im Zustand bereit befindet.

[35] Das entsprechende deutsche Wort *Planer* ist unüblich.

[36] Man beachte die Ähnlichkeit zwischen SJF und dem Verfahren SSTF in Abschnitt 1.2.3. Beides sind *gierige* (greedy) Strategien, die stets den im Moment maximal möglichen Vorteil suchen.

[37] Allgemeine Vorhersagen über die Rechenzeit eines beliebigen Programms zu treffen, ist nicht möglich. Selbst die Frage, ob ein Programm überhaupt jemals anhält, ist *unentscheidbar*, wie in der Theoretischen Informatik bewiesen wird.

Übungsaufgabe 1.11 Zu welchem Problem kann es bei einem System mit SJF-Scheduling kommen, wenn einige längere Prozesse laufen und immer wieder kurze Prozesse gestartet werden? Wie kann man die SJF-Strategie modifizieren, so dass dieses Problem vermieden wird?

Batch-Betrieb

Das Verfahren SJF eignet sich besonders gut für den *Stapel-* oder *Batch-Betrieb*, bei dem der Rechner gleich einen ganzen Schub von Aufträgen (jobs) erhält, die keine Interaktion mit dem Benutzer erfordern und regelmäßig auszuführen sind, so dass man ihre Laufzeiten in etwa kennt.

Time-Sharing-Betrieb

Moderne Rechner werden aber meist im *Time-Sharing-Betrieb* verwendet: Mehrere Benutzer können zur selben Zeit an einem Rechner arbeiten, und jeder kann gleichzeitig mehrere Programme laufen lassen, zum Beispiel in einem Fenster einen Compiler, in einem anderen ein Werkzeug für elektronische Post und in einem dritten Fenster einen WWW-Browser.[38]

Diese „Gleichzeitigkeit" ist natürlich eine Illusion, denn bei einem Computer mit nur einer CPU kann nur ein einziger Prozess rechnend sein. Der Eindruck von Gleichzeitigkeit entsteht dadurch, dass der Scheduler in schnellem Wechsel jedem bereiten Prozess ein gewisses Quantum an Rechenzeit zukommen lässt; die Umschaltung zwischen den Prozessen erfolgt so schnell, dass die Unterbrechungen nicht spürbar werden. Im einzelnen geschieht das folgendermaßen:

Zeitscheibe

Wenn ein bereiter Prozess rechnend gemacht wird, weist der Scheduler ihm eine *Zeitscheibe* (time slice) an Rechenzeit zu; ihre Dicke kann von der *Priorität* des Prozesses abhängen oder davon, wieviel Rechenzeit der Prozess insgesamt schon verbraucht hat.[39] Ein *Zeitgeber* (timer) wacht darüber, dass der Prozess sein Quantum nicht überschreitet; oft ist dafür ein eigener Chip im Rechner vorhanden, der an den Takt der CPU angeschlossen ist. Die Dicke der zugewiesenen Zeitscheibe wird in einem Register des Zeitgebers gespeichert. Nach jeder verstrichenen Zeiteinheit wird der Inhalt dieses Registers um eins verringert. Ist der Wert bei null angekommen, so ist die zugewiesene Zeitscheibe abgelaufen, und der Zeitgeber löst eine Unterbrechung der CPU aus. Der Prozess wird unterbrochen und wieder in die Menge der bereiten Prozesse eingereiht; vergleiche Abbildung 1.10. Nun kommt ein anderer Prozess an die Reihe.

Übungsaufgabe 1.12 Muss der Zugriff auf das Register des Zeitgebers privilegiert sein?

round robin

Bleibt noch zu klären, in welcher Reihenfolge die Prozesse ihre Zeitscheiben erhalten. Hier ist ein einfaches Verfahren namens *round robin* weit verbreitet, bei dem die bereiten Prozesse *reihum* bedient werden. Dazu kann man sie in der Reihenfolge ihres Eintreffens in einer kreisförmigen Warteschlange speichern.

Wie effizient dieses Verfahren arbeitet, hängt von der Dicke der Zeitscheiben ab. Die nächste Aufgabe zeigt, dass extreme Werte unzweckmäßig sind.

Dispatcher

Das Umschalten zwischen Prozessen bezeichnet man auch als *Kontextwechsel* (context switch). Während der Scheduler die Strategie für die Rechenzeitvergabe festlegt, führt der *Dispatcher* die eigentlichen Kontextwechsel durch.

[38] Statt von Time-Sharing spricht man manchmal auch von *Multitasking*.
[39] Diese Angabe steht ebenfalls im Prozesskontrollblock.

Da solche Kontextwechsel häufig vorkommen, ist es wichtig, dass der Dispatcher möglichst schnell arbeitet.

Übungsaufgabe 1.13 Welche Gefahren drohen, wenn beim Verfahren round robin die Zeitscheiben zu dick oder zu dünn gewählt werden?

Prozesse sind ein wichtiger Bestandteil der Struktur moderner Betriebssysteme; wir werden uns in Abschnitt 2.1 noch ausführlicher damit beschäftigen, wie Prozesse erzeugt werden und wie sie miteinander kommunizieren. Zum vollständigen Verständnis von Abbildung 1.10 müssen wir hier noch eine Tatsache erwähnen: Ein rechnender Prozess kann durch einen Systemaufruf mitteilen, dass er auf ein bestimmtes *Ereignis* warten will; er wird dann blockiert und kommt in eine spezielle Warteschlange, in der möglicherweise noch andere blockierte Prozesse stehen, die auf dasselbe Ereignis warten. Wenn dann dieses Ereignis eintritt, kann der Prozess, welcher es auslöst, ein *Signal* abschicken. Dieses Signal bewirkt, dass alle Prozesse aus der Warteschlange wieder in den Zustand bereit versetzt werden.

Warten auf Ereignisse

1.4.2 Rekapitulation: ein Gerätezugriff

In diesem Abschnitt wollen wir im Zusammenhang darstellen, was das Betriebssystem leistet, wenn es bei der Ausführung eines Programms auf eine Ein-/Ausgabeanforderung stößt, etwa auf einen Lesebefehl

read(f,b),

bei dem der Datensatz b von der Datei f gelesen werden soll. Die meisten Teilschritte haben wir in den vorangegangenen Abschnitten bereits diskutiert.

So wurde am Ende von Abschnitt 1.3.3 festgestellt, dass schon bei der Übersetzung des Programms der Hochsprachen-Befehl *read* durch einen entsprechenden Systemaufruf ersetzt wird, der den Ein-/Ausgabeteil des Betriebssystems startet. Außerdem muss die *logische* Beschreibung der gewünschten Daten – „Datensatz b in Datei f“ – in eine *physische* Beschreibung abgebildet werden, wie etwa „Block Nr. i auf der Magnetplatte M“. Dies geschieht zur Laufzeit des Programms mit Hilfe des *Dateisystems* (file system), auf das wir in Abschnitt 2.2.2 ausführlicher eingehen werden.

Dateisystem

Übungsaufgabe 1.14 Warum wird nicht schon beim Compilieren die logische Datenbeschreibung durch die physische Beschreibung ersetzt?

Im Time-Sharing-Betrieb kann es vorkommen, dass viele Prozesse kurz nacheinander auf ein Gerät zugreifen wollen. Deshalb wird zu jedem Speichergerät eine eigene *Geräte-Warteschlange* (device queue) eingerichtet, an die der Ein-/Ausgabeteil des Betriebssystems Aufträge anhängen kann; jeder Auftrag wird mit der Nummer des Prozesses versehen, der ihn erteilt hat. Der Gerätetreiber holt die Aufträge einzeln aus der Warteschlange ab und führt sie dem Controller zu. Dabei ist der Treiber nicht an die Reihenfolge in der Warteschlange gebunden; er kann vielmehr versuchen, durch geschickte Wahl des

Geräte-Warteschlange

nächsten Auftrags die Zugriffszeit des Geräts zu minimieren, wie in Übungsaufgabe 1.4 besprochen.[40]

Nun verfolgen wir, was bei Ausführung des zu *read* gehörigen Systemaufrufs in einem Prozess P geschieht:

1. Eine Software-Unterbrechung wird ausgelöst; der Kontext von P wird zunächst gerettet,[41] dann wird der (geräteunabhängige) Ein-/Ausgabeteil des Betriebssystems aufgerufen.

2. Dieser prüft zunächst, ob die benötigten Daten nicht schon im Cache im Hauptspeicher stehen; in dem Fall werden sie in den Adressraum von P kopiert, und nach der Rückkehr von der Software-Unterbrechung wird der Prozess P sofort wieder rechnend.

3. Stehen die Daten nicht im Cache, ist ein Plattenzugriff erforderlich. Der Prozess P wird solange blockiert. Der Leseauftrag wird an die Geräte-Warteschlange des Plattenlaufwerks angehängt. Dann gibt es dem Gerätetreiber Bescheid.

4. Sobald der Gerätetreiber frei ist und rechnend wird, entnimmt er den Auftrag der Geräte-Warteschlange.[42] Der Gerätetreiber reserviert im betriebssystemeigenen Bereich Speicherplatz für die zu lesenden Daten und schickt einen Lesebefehl an den Gerätecontroller. Dann führt der Treiber den Systemaufruf Warten aus und blockiert.

5. Der Gerätecontroller bestimmt zuerst die physische Adresse, dann führt er den Leseauftrag aus und überträgt zusammen mit dem DMA-Controller die Daten in den reservierten Bereich im systemeigenen Speicher; währenddessen können auf der CPU beliebige andere Prozesse rechnen. Danach wird eine Unterbrechung ausgelöst.

6. Mittels des Unterbrechungsvektors wird die Unterbrechungsroutine des Plattenlaufwerks gestartet. Sie schickt das Signal, auf das der Gerätetreiber wartet, und kehrt dann von der Unterbrechung zurück. Der Gerätetreiber wird dadurch bereit.

7. Sobald der Gerätetreiber rechnend wird, sieht er in der Geräte-Warteschlange nach, von welchem Prozess der Leseauftrag stammte, gibt dem Ein-/Ausgabeteil des Betriebssystems Bescheid und geht in den Zustand bereit.[43]

[40] Beim Drucker muss man anders vorgehen. Wenn nämlich mehrere Prozesse ihre Ausgabe zeilen- oder absatzweise zum Drucker schicken und sich dabei gegenseitig unterbrechen, entsteht auf dem Papier ein Durcheinander. Deshalb werden die Ausgaben zunächst nach Prozessen getrennt in einer *Spooler-Datei* gesammelt, die erst dann gedruckt wird, wenn der Prozess seine Druckausgabe abgeschlossen hat.

[41] Man kann sagen, dass P jetzt im Zustand bereit ist, auch wenn der Prozesskontrollblock von P momentan nicht in der entsprechenden Warteschlange steht, sondern auf dem Systemstapel.

[42] Nehmen wir einmal an, dass dieser Auftrag sofort an die Reihe kommt.

[43] Zuvor schaut der Treiber noch in der Warteschlange nach, ob weitere Aufträge für das Gerät vorliegen.

8. Der Ein-/Ausgabeteil des Betriebssystems kopiert die Daten aus dem systemeigenen Speicher in den Adressraum des Prozesses P und versetzt P aus dem Zustand blockiert in den Zustand bereit.

9. Sobald der Prozess P wieder rechnen darf, wird der Systemaufruf beendet, und P setzt seine Arbeit hinter dem *read* fort.

An manchen Stellen mag man sich fragen, warum der Ablauf gerade so und nicht anders organisiert ist. In der Regel sprechen gute – wenn auch nicht immer zwingende – Gründe für die hier beschriebene Organisation. Ein Beispiel zeigt die folgende Aufgabe.

Übungsaufgabe 1.15 Warum werden die gelesenen Daten erst im systemeigenen Speicherbereich abgelegt und nicht gleich im Adressraum des anfordernden Prozesses P?

1.4.3 Programmierschnittstelle

In den vorangegangenen Abschnitten haben wir bei der Rechnerhardware begonnen und uns von unten nach oben (bottom-up) in die Aufgaben und Strukturen eines Betriebssystems eingearbeitet. Nun können wir einen Blick auf ein ganzes System werfen.

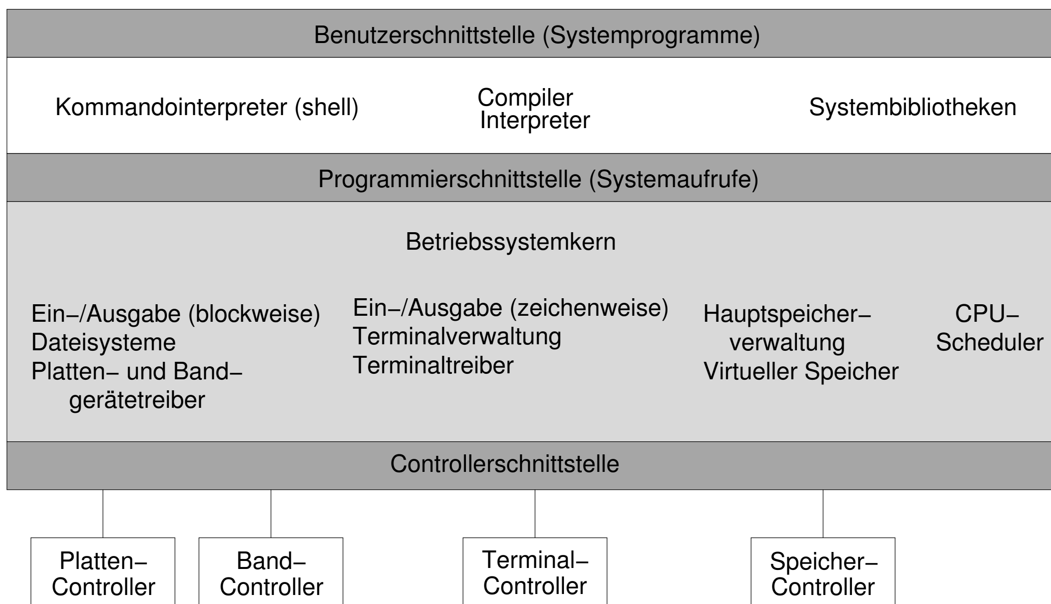


Abbildung 1.11: Die Struktur des Betriebssystems UNIX.

In Abbildung 1.11 ist die Struktur von UNIX skizziert. Der sogenannte *Kern* (kernel) des Betriebssystems ist hellgrau dargestellt. Er enthält geräteunabhängige Teile für blockweise Ein- und Ausgabe, wie sie bei Platten- und Bandgeräten gebräuchlich ist, und für zeichenweise Ein- und Ausgabe wie bei Modems und Terminals. Ein weiterer Teil des Kerns ist für die Hauptspeicherverwaltung zuständig. Der CPU-Scheduler gehört zum Kern, und in UNIX und

Kern

Linux auch die Gerätetreiber. Daneben gibt es weitere Teile für die Erledigung anderer Aufgaben.

Schnittstellen

Wir sehen in Abbildung 1.11, dass der Kern zwei Schnittstellen aufweist: Die Verbindung nach unten zur Hardware erfolgt über die *Controllerschnittstelle*. Die *Programmierschnittstelle* stellt die Dienste des Betriebssystems nach oben, also für die Programme zur Verfügung; dies geschieht durch die Gesamtheit der *Systemaufrufe* (system calls), wie wir sie in Abschnitt 1.3.3 besprochen haben.

Oberhalb der Programmierschnittstelle sind in Abbildung 1.11 die *Systemprogramme* eingezeichnet. Sie bilden zusammen die *Benutzerschnittstelle*, mit der wir uns in Abschnitt 1.4.4 befassen werden.

Systemaufrufe

In diesem Abschnitt wollen wir eine Übersicht über einige wichtige Dienste geben, die als Systemaufrufe angeboten werden.[44] Diese Systemaufrufe lassen sich grob folgenden Aufgabenbereichen zuordnen:

- Prozesse,
- Dateien,
- Information und
- Kommunikation.

Prozesse

Einige Beispiele für Systemaufrufe im Zusammenhang mit *Prozessen* haben wir bereits kennengelernt: Ein Prozess kann *auf ein Ereignis warten* (wait event) und wird blockiert, bis ein anderer Prozess durch ein *Signal* mitteilt, dass das erwartete Ereignis eingetreten ist (signal event); vergleiche das Ende von Abschnitt 1.4.1.

anhalten und abbrechen

Ein Prozess muss freiwillig *anhalten* können (halt), wenn er mit seiner Arbeit fertig ist; er wird dadurch in den Zustand beendet versetzt. Wenn ein Fehler auftritt, muss der Prozess *abgebrochen* werden (abort); dabei sollte es möglich sein, eine Fehlermeldung auszugeben oder den Inhalt von Registern und Adressraum für die Fehlersuche zu speichern (dump).

Ein Prozess sollte ein anderes Programm *laden* (load) und *ausführen* (execute) können. So kann ein neuer Prozess entstehen, der entweder erst beendet sein muss, bevor der Vaterprozess weiterrechnen kann, oder neben seinem Vaterprozess existiert; hierfür ist oft ein besonderer Systemaufruf *Prozess!erzeugen* (create process) zuständig.[45] Manchmal will der Vaterprozess beim Kindprozess bestimmte *Attribute setzen* (set attributes), etwa einen Adressraum im Adressraum des Vaters oder eine maximale Rechenzeit. Es kann auch notwendig sein, den Kindprozess zu *terminieren* (terminate).

Dateien

Den Begriff *Datei* (file) haben wir schon früher verwendet, ohne genau zu sagen, was damit gemeint ist. Eine Datei ist, grob gesagt, eine Sammlung zusammengehöriger Information. Dabei kann es sich um ein Programm in Maschinencode handeln, ein druckbares PostScript-Dokument, eine Videosequenz oder

[44] Dabei legen wir uns nicht auf ein spezielles Betriebssystem fest; in UNIX und Linux sind jedenfalls Aufrufe dieser Art vorhanden.

[45] In UNIX und Linux wird solch ein Prozess mit *fork* erzeugt und mit *exec* ausgeführt.

um eine HTML-Seite. Dateien befinden sich im Sekundär- und Tertiärspeicher. Die Magnetplatte eines PC enthält nicht selten einige tausend Dateien; einen großen Anteil bilden oft das Betriebssystem und die Systemprogramme selbst.

Aus der Sicht des Benutzers sind Dateien *abstrakte Datentypen*, ähnlich wie virtuelle Geräte; vergleiche Abschnitt 1.3.2. Sie sind in *logische Datensätze* (records) aufgeteilt, die einzelne Informationseinheiten darstellen und oft dieselbe Länge haben. So kann zum Beispiel in einer Personaldatei für jede Mitarbeiterin ein Datensatz vorhanden sein. Es gibt Operationen, um Dateien zu erzeugen, zu löschen und um einzelne Datensätze zu lesen oder zu schreiben. Dabei kann der Zugriff auf die Datensätze sequentiell oder wahlfrei sein. Manchmal kann man über einen *Schlüssel* zugreifen: Bei der Personaldatei gibt man dann die Personalnummer vor und erhält den Datensatz des entsprechenden Mitarbeiters.[46]

Das Dateisystem hat die Aufgabe, diese logisch-abstrakte Sicht auf die physische Organisation einer Datei in Blöcke abzubilden; vergleiche den Anfang von Abschnitt 1.4.2; das kann oft dadurch geschehen, dass jeweils k Datensätze gleicher Größe in einen Block geschrieben werden.

anlegen und löschen

Man benötigt dazu Systemaufrufe, mit denen sich eine Datei physisch *anlegen* (create) oder *löschen* (delete) lässt. Vor dem ersten Zugriff muss man eine Datei *öffnen* (open), nach dem letzten Zugriff *schließen* (close). Durch das Öffnen wird die Datei in eine spezielle Liste eingetragen; bei den folgenden Zugriffen braucht dann nicht jedesmal aufs neue die physische Adresse der Datei bestimmt zu werden. Außerdem lässt sich damit verhindern, dass zwei Benutzer sich beim Zugriff auf dieselbe Datei stören. Es gibt Systemaufrufe zum *Lesen* (read) und zum *Schreiben* (write) eines Blocks. Ferner lässt sich eine Datei vor- oder zurückspulen, um auf einen bestimmten Block direkt zugreifen zu können; die Vorstellung des Umspulens stammt dabei von Magnetbandgeräten.

öffnen und schließen

lesen und schreiben

Information

Zeit und Datum

Systemaufrufe zur Beschaffung von *Information* können vom System die *Zeit* erfragen oder das *Datum*. Manche Systeme stellen Aufrufe bereit, welche die Nummer der laufenden Betriebssystemversion oder eine Liste mit den Nummern aller vorhandenen Prozesse liefern.

Kommunikation

Die *Kommunikation* zwischen Prozessen kann auf zwei verschiedene Arten erfolgen: über den Versand von *Nachrichten* (message passing) oder durch einen *gemeinsamen Speicherbereich* (shared memory).[47]

Versand von Nachrichten

Beim *Versand von Nachrichten* gibt es zwei Varianten: Der *verbindungslose* (connectionless) Datenaustausch entspricht dem Versenden eines Briefs: Man gibt die Adresse des Empfängers und des Absenders an und überlässt die Übermittlung dem System. Ein Beispiel für den verbindungslosen Austausch ganz kurzer Nachrichten sind die Signale, die oben schon erwähnt wurden. Für längere Nachrichten gibt es Systemaufrufe zum *Senden* (send) und *Empfangen*

[46] Diese datenbankähnliche Zugriffsmethode wird von der Programmiersprache COBOL unterstützt; trotzdem wollen wir hier keine Empfehlung für diese altertümliche Sprache geben.

[47] Bei Rechnernetzen kommt nur der Versand von Nachrichten für die Kommunikation in Frage.

(receive), die besonders beim *Client-Server-Betrieb* wichtig sind.

Beim *verbindungsorientierten* (connection-oriented) Datenaustausch wird dagegen erst eine Verbindung zwischen Sender und Empfänger aufgebaut, über die dann die Kommunikation erfolgt; dem entspricht ein Telefongespräch. Hierfür braucht der Sender Systemaufrufe zum *Öffnen* und *Schließen* der Verbindung; der Empfänger kann schon auf die Verbindung *warten* (wait) oder sie doch wenigstens *akzeptieren* (accept). Sobald die Verbindung zustande gekommen ist, kann man Nachrichten *schreiben* und *lesen*. In UNIX und Linux werden solche Verbindungen durch *pipes* realisiert.

gemeinsamer Speicherbereich

Viel effizienter ist es, für die Kommunikation zwischen Prozessen einen *gemeinsamen Speicherbereich* zu verwenden, in den der Sender schreibt und von dem der Empfänger liest; denn dann kann man die schnellen CPU-Befehle für den Hauptspeicherzugriff benutzen. Hier treten aber zwei Schwierigkeiten auf: Zum einen achtet das Betriebssystem ja eigentlich darauf, dass verschiedene Prozesse getrennte Adressräume haben; siehe Abschnitt 1.3.5. Hier braucht man Systemaufrufe, um vorübergehend einen Teil des einen Adressraums in den anderen abzubilden. Zum anderen kann es Probleme geben, wenn sich die Prozesse beim Lesen und Schreiben stören. Hier sind *Synchronisationsmechanismen* erforderlich, wie wir sie in Abschnitt 2.1.2 behandeln werden.

Übungsaufgabe 1.16 Welcher gravierende Unterschied besteht zwischen einer verbindungsorientierten und einer verbindungslosen Übertragung, wenn eine sehr lange Nachricht vor dem Absenden in mehrere Teile zerlegt werden muss?

1.4.4 Benutzerschnittstelle

In Abschnitt 1.4.3 haben wir uns mit der Programmierschnittstelle eines Betriebssystems beschäftigt; nun wenden wir uns der *Benutzerschnittstelle* zu. Sie dient dem Benutzer zusammen mit den Schnittstellen der Anwendungsprogramme zur Kommunikation mit dem Rechner. Wie angenehm es sich mit einem Rechner arbeiten lässt, hängt entscheidend von der Gestaltung der Benutzerschnittstelle ab. Weit verbreitet sind heute graphische Benutzeroberflächen mit Fenstern, Menüs und Maussteuerung.

Systemprogramme

Während die Programmierschnittstelle durch die Gesamtheit aller Systemaufrufe gegeben ist, besteht die Benutzerschnittstelle aus der Gesamtheit aller *Systemprogramme* (system programs). In Abbildung 1.11 sieht man am Beispiel von UNIX, wie die Systemprogramme über die Programmierschnittstelle auf die Systemaufrufe zugreifen.

Auch die Systemprogramme lassen sich grob den folgenden Aufgabenbereichen zuordnen:

- Programme,
- Dateien und Verzeichnisse,
- Information und
- Kommunikation.

Sie sehen: Diese Liste entspricht der Einteilung der Systemaufrufe in Abschnitt Abschnitt 1.4.3. Der Unterschied besteht darin, dass Systemprogramme oft auf einer höheren Abstraktionsebene angesiedelt sind als Systemaufrufe, und dass sie in eine Benutzeroberfläche integriert sind. Ein Beispiel für eine solche integrierte Oberfläche ist das Common Desktop Environment (CDE)[48]. Es vereinigt viele Systemprogramme mit den nachfolgend beschriebenen Funktionen in sich; man kann auch CDE selbst als ein großes Systemprogramm auffassen.

Programme

Gehen wir die einzelnen Bereiche durch. Von Systemprogrammen werden alle Benutzeraktivitäten unterstützt, die mit dem Herstellen und Ausführen von *Programmen* zu tun haben. Dazu gehört ein *Editor* zum Schreiben der Programmtexte,[49] ein *Compiler* oder *Interpreter* für die verwendete Programmiersprache, ein *Binder* (linker), der einzelne Module zu einem Lademodul zusammenfasst, und schließlich den *Lader* (loader), der das Lademodul in den Hauptspeicher bringt; vergleiche Abschnitt 1.3.5. Außerdem ist ein *Debugger* (bug = Wanze) bei der Fehlersuche hilfreich.

editieren, übersetzen, binden, laden

Fehler suchen

Dateien und Verzeichnisse

Systemprogramme helfen dem Benutzer, Dateien *anzulegen*, zu *kopieren, zu drucken, umzubenennen* und zu *löschen*. Auch das Setzen der *Zugriffsberechtigung*, d. h. die Festsetzung, welcher Benutzer wie auf eine Datei zugreifen darf, kann mit einem Systemprogramm erfolgen.[50] Zusammengehörige Dateien kann man in *Verzeichnissen* (directories) zusammenfassen. Ein Verzeichnis kann selbst andere Verzeichnisse enthalten. So lassen sich hierarchische Strukturen aufbauen, in denen man sich gut zurechtfindet. Für die Verwaltung von Verzeichnissen gibt es ähnliche Systemprogramme wie für Dateien. Wir werden uns hiermit in Abschnitt 2.2 ausführlicher beschäftigen.

Zugriffsrechte setzen

verwalten

Information
Systemdaten

Neben *Zeit* und *Datum* lassen sich wichtige Systemdaten abfragen wie *freier Speicherplatz*, *CPU-Auslastung* und eine Liste der anderen Benutzer, die zur Zeit am Rechner arbeiten.

Kommunikation

Während die Systemaufrufe insbesondere mit der Kommunikation zwischen Prozessen zu tun haben, sind Systemprogramme auf höherer Ebene für die Kommunikation zwischen Benutzern zuständig; diese können an voneinander weit entfernten Rechnern arbeiten. Besonders bekannt sind *elektronische Post* (electronic mail), *Dateitransfer* (file transfer, z.B. mit dem *file transfer protocol*) und *Benutzerzugriff auf entfernte Rechner* (remote login). Hieran sind auch die Netze beteiligt, die die Rechner miteinander verbinden. Näheres hierzu finden Sie in den Kurseinheiten 3 und 4 dieses Kurses.

email, ftp und remote login

[48] oder die freien Linux Oberflächen KDE und GNOME.

[49] Bei Editoren gibt es ein breites Spektrum, vom zeilenorientierten *vi* in UNIX bis hin zu graphischen Editoren mit komfortabler Maussteuerung. Für das Programmieren in einer bestimmten Sprache kann ein *syntaxgesteuerter* Editor zweckmäßig sein, der Verstöße gegen die Regeln der Sprache schon beim Eingeben erkennt, die Programmzeilen sinnvoll einrückt und Schlüsselwörter fett druckt, z. B. *XEmacs*.

[50] An diesem Beispiel lässt sich der Unterschied zwischen Systemaufruf und Systemprogramm verdeutlichen: In UNIX und Linux gibt es den Systemaufruf *chmod*, mit dem der Besitzer einer Datei festlegen kann, wer außer ihm – d. h. seine Gruppe oder alle Benutzer – die Datei lesen, beschreiben oder ausführen dürfen. Unter CDE lässt sich im File Manager ein Menü öffnen, das die Zugriffsrechte einer ausgewählten Datei anzeigt und zu ändern erlaubt.

Kommando-interpreter

Ein besonders wichtiges Systemprogramm ist der *Kommandointerpreter* (command interpreter), der die Eingaben des Benutzers entgegennimmt und ihre Ausführung veranlasst. Während die Kommandos früher eingetippt werden mussten, kann heute für viele Aufgaben die Maus eingesetzt werden; so kann man zum Beispiel eine Datei!löschen, indem man ihr Bild mit der Maus in einen Papierkorb bewegt. Noch komfortabler ist der Start des zugehörigen Anwendungsprogramms durch Doppelklicken auf eine Datei: so wird bei einer Textdatei das Textprogramm gestartet, mit dem sie erstellt wurde, und bei einer Tondatei ein Abspielprogramm; Dateien in Maschinencode werden geladen und ausgeführt.

Früher waren die Programme zur Ausführung der Benutzerkommandos selbst Teil des Kommandointerpreters, wodurch dieser unhandlich und Änderungen mühsam wurden. Heute kann man in UNIX und Linux jedem Benutzer leicht einen maßgeschneiderten Kommandointerpreter – eine sogenannte *shell* – zur Verfügung stellen. Um ein neues Kommando namens *neukommando* einzuführen genügt es, eine Datei mit dem Namen *neukommando* anzulegen, in der das zugehörige ausführbare Systemprogramm steht, und dem System beim Aufruf mitzuteilen, in welchem Verzeichnis sich diese Datei befindet.

1.5 Andere Systeme

In den vorangegangenen Abschnitten haben wir angenommen, dass der Rechner nach dem von-Neumann-Modell aufgebaut ist, also nur über eine einzige CPU verfügt; siehe Abschnitt 1.2. Am Ende dieser Kurseinheit wollen wir einen kurzen Blick auf andere Rechnerarchitekturen werfen.

1.5.1 Parallelrechner

Ein naheliegender Gedanke ist es, die Bearbeitungszeit für die Lösung komplexer Probleme dadurch zu verkürzen, dass man die Arbeit auf mehrere Prozessoren verteilt; dieser Ansatz führt zum Konzept des *Parallelrechners*; oft findet sich auch die Bezeichnung *Multiprozessorsystem* (multiprocessor system).

Parallelisierung

Hier sollte man sich allerdings vor übergroßem Optimismus hüten: Ein Rechner mit k Prozessoren löst ein Problem nicht unbedingt k mal so schnell wie ein Einprozessorsystem![51] Denn zum einen setzt das Zusammenwirken mehrerer Prozessoren *Kommunikation* voraus; auch sie benötigt Zeit. Zum anderen ist zunächst zu prüfen, ob sich ein gegebenes Problem überhaupt gut parallelisieren lässt. Bei Schachproblemen und dem Brechen von Codes geht das zum Beispiel recht gut, aber wie steht es mit dem Problem, n Zahlen der Größe nach zu sortieren?[52]

Einsparung

Neben der möglichen Effizienzsteigerung bieten Parallelrechner zwei weitere Vorteile: Indem man viele CPUs in einem gemeinsamen Gehäuse unterbringt, lassen sich Kosten einsparen. Und man gewinnt eine gewisse Sicherheit gegen

[51] Dies wird durch alltägliche Erfahrung bestätigt: Ein Team von 30 Programmierern schafft auch nicht an einem Tag, wozu ein einzelner einen Monat braucht.

[52] Wer sich für solche Fragen interessiert, sei auf den Kurs *Parallele Algorithmen* verwiesen.

Störungen; wenn nämlich eine CPU ausfällt, können die anderen ihre Arbeit mit übernehmen. Man kann sogar einen spontan auftretenden Rechenfehler einer CPU neutralisieren, wenn man dieselbe Berechnung zur Kontrolle auch von anderen Prozessoren des Systems durchführen lässt. Solche *fehlertoleranten Systeme* (fault tolerant systems) werden dort eingesetzt, wo es besonders auf Zuverlässigkeit ankommt.

Fehlertoleranz

Bei einem Parallelrechner teilen sich die Prozessoren einen gemeinsamen großen Hauptspeicher. Der Zugriff erfolgt entweder über einen Bus, oder der gemeinsame Hauptspeicher wird in kleinere Stücke zerlegt und ein *Schalterwerk* sorgt dafür, dass jeder Prozessor auf jeden Teil des Speichers zugreifen kann.

bus- oder schalterbasiert

1.5.2 Verteilte Systeme

Bei einem *verteilten System* hat jeder Prozessor seinen eigenen Speicher, auf den nur er selbst zugreifen kann. Die Kommunikation zwischen den Prozessoren kann also nur über den Versand von Nachrichten erfolgen, wie in Abschnitt 1.4.3 beschrieben. Es kann sein, dass die Prozessoren im selben Rechnergehäuse untergebracht sind und zwischen einigen von ihnen feste Verbindungen bestehen. Ein verteiltes System kann aber auch aus vielen Einprozessorrechnern bestehen, die auf der ganzen Welt verteilt sind; dies ist zum Beispiel beim World Wide Web der Fall.[53]

Ein Beispiel für den ersten Fall ist die *Hypercube-Architektur*. Beim d-dimensionalen Hypercube gibt es für jeden der 2^d möglichen Vektoren, die man aus d Nullen und Einsen bilden kann, einen Prozessor. Nun ist aber nicht jeder Prozessor direkt mit jedem anderen verbunden, sondern nur mit den d Prozessoren, deren Vektor sich an genau einer Stelle vom eigenen Vektor unterscheidet. Von (10011000) nach (10011010) gibt es also eine Verbindung, nach (10111010) aber nicht.

Hypercube

Übungsaufgabe 1.17 Über wieviele Zwischenstationen muss man beim d-dimensionalen Hypercube höchstens laufen, um von einem Prozessor zu einem anderen zu gelangen?

Für verteilte Systeme Software zu entwickeln, ist nicht einfach. Ein Grund liegt darin, dass viele Prozesse gleichzeitig ablaufen, die sich durch Nachrichten gegenseitig beeinflussen können, dass sich aber nicht genau planen lässt, welcher Prozessor zu welcher Zeit welche Anweisung ausführt.

Wer hierüber mehr erfahren möchte, sei auf die Thematik *Rechnernetze und Verteilte Systeme* und die Kurse *Betriebssysteme*, *Verteilte Systeme* und *Sicherheit im Internet* verwiesen.

1.5.3 Realzeitsysteme

Bei unseren bisherigen Betrachtungen war unser Ziel, ein Programm korrekt ablaufen zu lassen und dabei die Rechenzeit möglichst gering zu halten. Es

[53] Für die Geschwindigkeit der Kommunikation bedeutet das natürlich einen erheblichen Unterschied.

gibt aber eine ganze Reihe von Anwendungen, bei denen das allein noch keine befriedigende Lösung garantiert. Wenn zum Beispiel die Sensoren eines mobilen Industrieroboters den Kontakt mit einem festen Hindernis melden, muss die eingebaute Steuerung einen Haltebefehl auslösen, bevor der Roboter die Wand durchbricht.

Hier ist es Teil der Korrektheitsanforderung, dass das Ergebnis *innerhalb einer vorgegebenen Zeit* berechnet wird. Mit gängigen Multitaskingsystemen lassen sich solche harten Anforderungen nicht erfüllen; man denke nur an die Folgen, wenn im kritischen Moment die Prozedur zum Halten des Antriebs erst von der Magnetplatte geladen werden muss, oder wenn andere Prozesse mit höherer Priorität die CPU für sich beanspruchen.

Aus diesem Grund gibt es spezielle *Realzeitsysteme*, bei denen große Teile der Software in ROMs untergebracht sind. Die Konstruktion von Realzeitsystemen ist ein interessantes Spezialgebiet.

Literatur

[1] SUSE Supportdatenbank, Updates, Links, Bestellungen etc. http://www.suse.de.

[2] Webressourcen zum Thema Linux, u. a. Liste von Linux-Distributoren. http://www.linux.org.

[3] M. Beck, H. Böhme, M. Dziadzka, U. Kunitz, R. Magnus, D. Verworner. *Linux-Kernel-Programmierung*. Addison-Wesley, 1997.

[4] J. Gulbins, K. Obermayr. *UNIX – Konzepte, Kommandos, Schnittstellen*. Springer-Verlag, 1995.

[5] S. Hetze, D. Hohndel, M. Müller, O. Kirch. *Linux: Anwenderhandbuch und Leitfaden für die Systemverwaltung*. LunetIX Softfair, Berlin, 1997.

[6] M. Kofler. *Linux: Installation, Konfiguration und erste Schritte*. Addision-Wesley, 8., überarbeitete Auflage, 2008.

[7] J. Nehmer, P. Sturm. *Systemsoftware – Grundlagen moderner Betriebssysteme*. dpunkt Verlag, 2001.

[8] A. Silberschatz, P. B. Galvin, G. Gagne. *Operating System Concepts*. John Wiley & Sons, seventh edition, 2005.

[9] A. S. Tanenbaum. *Moderne Betriebssysteme*. Prentice Hall, Pearson Studium, 2., überarbeitete Auflage, 2002.

[10] A. S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, Pearson Education, third edition, 2008.

[11] A. S. Tanenbaum, A. S. Woodhull. *Operating Systems: Design and Implementation*. Prentice Hall, third edition, 2006.

Weitere WWW-Adressen zum Thema Betriebssysteme und Linux finden sie unter

http://www.fernuni-hagen.de/ks/1801/

Lösungen der Übungsaufgaben

Übungsaufgabe 1.1, Seite 12: Dass 10011101 die Binärdarstellung der Zahl 157 ist, kann man direkt nachrechnen, indem man die Zweierpotenzen addiert, deren Koeffizient in der Binärdarstellung gleich eins ist, also

$$2^7 + 2^4 + 2^3 + 2^2 + 2^0 = 157.$$

Allgemein berechnet man die Binärdarstellung einer Zahl $n \geq 1$ folgendermaßen: Der letzte Koeffizient, a_0, ist der Rest von n bei Division durch 2 (d. h. a_0 ist 1, wenn n ungerade ist, für gerade n ist a_0 gleich 0). Jetzt bildet man

$$n' := \frac{n - a_0}{2} = n \operatorname{div} 2$$

und iteriert das Verfahren. Es ist also

$$a_1 = n' \bmod 2$$

und so fort. Die Binärdarstellung von 160 kann man auch direkt aus der Darstellung von 157 gewinnen, indem man dreimal eins addiert und die Überträge wie in der gewöhnlichen Dezimalrechnung behandelt. Es ergibt sich 10100000.

Übungsaufgabe 1.2, Seite 12: Teilt man 16 MByte in Worte der Länge 2 Byte auf, so entstehen

$$\frac{16 \cdot 2^{20}}{2} = 8 \cdot 2^{20} = 2^{23}$$

viele Worte. Der Adressbus muss also die Breite 23 haben.

Übungsaufgabe 1.3, Seite 14: Sei c die Zeitdauer eines Schreib-/Lesezugriffs auf den Cache; dann fällt also in 80 % aller Datenzugriffe ein Zeitaufwand von c Einheiten an. Bei den übrigen 20 % der Fälle muss auf den Hauptspeicher zugegriffen werden, das kostet zusätzlich $9c$ Zeiteinheiten. Danach muss die Daten noch in den Cache abgelegt werden, das benötigt noch c Zeiteinheiten. Im Mittel entsteht also bei Verwendung des Cache ein Zeitaufwand in Höhe von

$$\frac{80}{100}c + \frac{20}{100}(9c + 2c) = \frac{80 + 220}{100}c = 3c$$

gegenüber $9c$ beim Betrieb ohne Cache. Bei den hier zugrunde gelegten Werten geht es mit Cache also drei mal so schnell.

Übungsaufgabe 1.4, Seite 16: Bei der Strategie FCFS ergibt sich als Gesamtdistanz

$$(98 - 32) + (185 - 32) + \cdots + (107 - 19) = 565.$$

Wendet man die Strategie SSTF an, so werden nacheinander die Spuren 98, 107, 126, 80, 32, 19, 185 besucht, was eine Gesamtdistanz von 301 ergibt. Bei einem SCAN, der von Spur 98 zunächst nach außen führt bis zur Spur 0 und dann wieder nach innen bis Spur 185, beträgt die Gesamtdistanz 283. Bei

diesem Beispiel schneidet also die Strategie SCAN am besten ab. In der Praxis sind die Verhältnisse komplizierter, weil nicht alle Schreib-/Leseaufträge im voraus bekannt sind; vielmehr treffen immer neue Aufträge ein, während die ersten bereits abgearbeitet werden.

Übungsaufgabe 1.5, Seite 18: Nach aufsteigender Zugriffszeit geordnet, hatten wir Prozessorregister, Cache, Hauptspeicher, RAM-disk, Magnetplatte, BD und HD DVD, CD/DVD, Diskette und Magnetband betrachtet.

Übungsaufgabe 1.6, Seite 27: Es stimmt zwar, dass beim regelmäßigen Überprüfen des Unterbrechungseingangs ein Abfragebetrieb vorliegt; diese Abfragen erfolgen aber im Innern der CPU und benötigen deshalb viel weniger Zeit als das Lesen des Statusregisters des Controllers. Denn selbst bei speicherabgebildeter Ein-/Ausgabe werden hierfür mehrere CPU-Befehle benötigt.

Übungsaufgabe 1.7, Seite 28: Auch bei Einsatz des DMA-Verfahrens kann es zu Verzögerungen in der Arbeit der CPU kommen, weil der DMA-Controller während der Übertragung den Bus besetzt hält, die CPU also nicht auf den Hauptspeicher zugreifen kann.

Übungsaufgabe 1.8, Seite 31: Läge der Unterbrechungsvektor im Adressraum eines Benutzerprogramms, so könnte der Benutzer die Anfangsadresse einer Unterbrechungsroutine so ändern, dass, wenn diese Unterbrechung eintritt, statt der Unterbrechungsroutine ein Teil des Benutzerprogramms gestartet wird. Die CPU wäre dann im Systemmodus, und der Benutzer hätte damit die vollständige Herrschaft über den Rechner erlangt.

Übungsaufgabe 1.9, Seite 31: Der Befehl zum Außerbetriebsetzen der Unterbrechungsleitung muss auf jeden Fall privilegiert sein. Wenn nämlich ein Benutzerprogramm diesen Befehl ausführen könnte und anschließend in eine Endlosschleife geriete, wäre es nur noch durch Abschalten des Rechners zu stoppen.

Übungsaufgabe 1.10, Seite 33: Betrachten wir eine Ausführungsreihenfolge, bei der ein langer Prozess L vor einem kurzen Prozess K an die Reihe kommt. Wenn wir die beiden miteinander vertauschen, brauchen alle dazwischen liegenden Prozesse nicht mehr so lange zu warten. Für L verlängert sich bei diesem Tausch zwar die Wartezeit, aber die Verkürzung der Wartezeit von K ist größer! Alle übrigen Prozesse sind nicht betroffen. Also verkürzt solch ein Tausch die Gesamtwartezeit. Jede von SJF verschiedene Ausführungsreihenfolge lässt sich also noch verbessern; folglich ist SJF optimal.

Übungsaufgabe 1.11, Seite 34: Bei SJF werden kürzere Prozesse vor längeren bearbeitet. Je mehr kurze Prozesse generiert werden, desto länger müssen die langen Prozesse warten. Wenn also immer neue kurze Prozesse erzeugt werden, kommen die langen Prozesse nie an die Reihe. Dieses Problem wird *starvation* genannt. Eine einfache Möglichkeit, starvation zu verhindern ist, beim Scheduling zusätzlich zu den erwarteten Laufzeiten auch Prioritäten zu berücksichtigen. Alle im System eintreffenden Prozesse erhalten zunächst die gleiche Priorität, die dann jedoch erhöht wird, je länger ein Prozess auf Prozessorzuteilung wartet. Der lange Prozess hätte dann irgendwann eine hohe

Priorität, und der Scheduler kann erkennen, dass der Prozess schon lange wartet und ihn bearbeiten, obwohl kürzere Prozesse im System sind. Diese Prioritätsanpassung wird auch als aging bezeichnet.

Übungsaufgabe 1.12, Seite 34: Der Zugriff auf das Register, das die Anzahl der verbleibenden Zeiteinheiten der aktuellen Zeitscheibe enthält, muss privilegiert sein! Denn sonst könnte ein Prozess sich selbst zusätzliche Rechenzeit verschaffen.

Übungsaufgabe 1.13, Seite 35: Prozesse mit großem Ein-/Ausgabebedarf können von extrem langen Zeitscheiben nicht profitieren: Meist blockieren sie lange bevor ihre Zeitscheibe abgelaufen ist, etwa um auf die nächste Benutzereingabe zu warten. Sie können aber sehr darunter leiden, wenn sie rechenzeitintensive Prozesse vor sich haben, die ihre dicken Zeitscheiben voll ausnutzen. Darum sind zu dicke Zeitscheiben nicht wünschenswert. Aber zu dünne Zeitscheiben sind auch ineffizient: Die CPU verbringt dann einen zu großen Teil ihrer Zeit mit unproduktiven Kontextwechseln, während es mit den Prozessen nur langsam voran geht.

Übungsaufgabe 1.14, Seite 35: Zwischen dem Compilieren und dem Ausführen eines Programms kann ein längerer Zeitraum liegen, in dem die Dateien verändert werden. Deshalb kann man logische Adressen erst zur Laufzeit auf physische Adressen abbilden.

Übungsaufgabe 1.15, Seite 37: Wollte man die gelesenen Daten sofort in den Adressraum des Prozesses P übertragen, der sie angefordert hat, so müsste die Zieladresse vom Betriebssystem über den Treiber an den Controller weitergegeben werden. Außerdem dürfte der Adressraum von Prozess P im Hauptspeicher nicht bewegt werden, bis der Lesevorgang abgeschlossen ist. Beides wäre unzweckmäßig! Dem Controller die Zieladresse mitzuteilen, widerspricht den Prinzipien von Kapselung und Aufgabenteilung (separation of concerns). Blockierte Prozesse nicht bewegen oder auf die Magnetplatte auslagern zu können, würde die effiziente Verwaltung des Hauptspeichers behindern.

Übungsaufgabe 1.16, Seite 40: Wenn man eine lange Nachricht in mehreren Teilen überträgt, ist beim verbindungsorientierten Datenaustausch sichergestellt, dass die Teile in der richtigen Reihenfolge beim Empfänger ankommen. Beim verbindungslosen Austausch kann das nicht garantiert werden.

Übungsaufgabe 1.17, Seite 43: Zwei beliebige Vektoren der Länge d unterscheiden sich im schlimmsten Fall an allen d Stellen. Wenn man also bei jedem Schritt einen Unterschied verschwinden lässt, indem man den entsprechenden Nachbarn aufsucht, sind insgesamt d Schritte über $d - 1$ Zwischenstationen erforderlich.