HANSER

Leseprobe

Carsten Vogt

Nebenläufige Programmierung

Ein Arbeitsbuch mit UNIX/Linux und Java

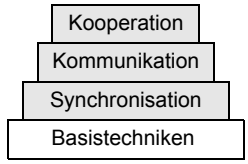
ISBN: 978-3-446-42755-6

Weitere Informationen oder Bestellungen unter

http://www.hanser.de/978-3-446-42755-6

sowie im Buchhandel.

2 Basistechniken



Kapitel 2 beschäftigt sich mit grundlegenden Techniken der Nebenläufigkeit. Im Mittelpunkt stehen dabei das Betriebssystem, insbesondere UNIX/Linux, sowie die Sprache Java.

2 Basistechniken

Der Begriff **Multitasking** – also das Erledigen mehrerer Dinge gleichzeitig – ist in der Alltagssprache angekommen. So wird diskutiert, ob Frauen dieses Multitasking besser beherrschen als Männer. Ältere wundern sich über das Multitasking Jüngerer, die gleichzeitig essen, trinken, telefonieren, SMS schreiben und über die Straße gehen. Und man beklagt das Multitasking im Arbeitsleben, in dem viele Dinge möglichst zur selben Zeit erledigt werden sollen.

Eine tiefere Wurzel hat das Multitasking jedoch in der Informatik. Das ist angemessen, denn Computer sind im gleichzeitigen (also **nebenläufigen**) Erledigen von Aufgaben wesentlich besser als wir Menschen. So ist es für heutige Personal Computer selbstverständlich, dass man mit ihnen zur selben Zeit Musik hören, Spiele spielen und E-Mails empfangen kann; Web Server können eine Vielzahl von Surfern gleichzeitig bedienen, und auf Großrechnern können alle Mitarbeiter einer Firma nebenläufig arbeiten:

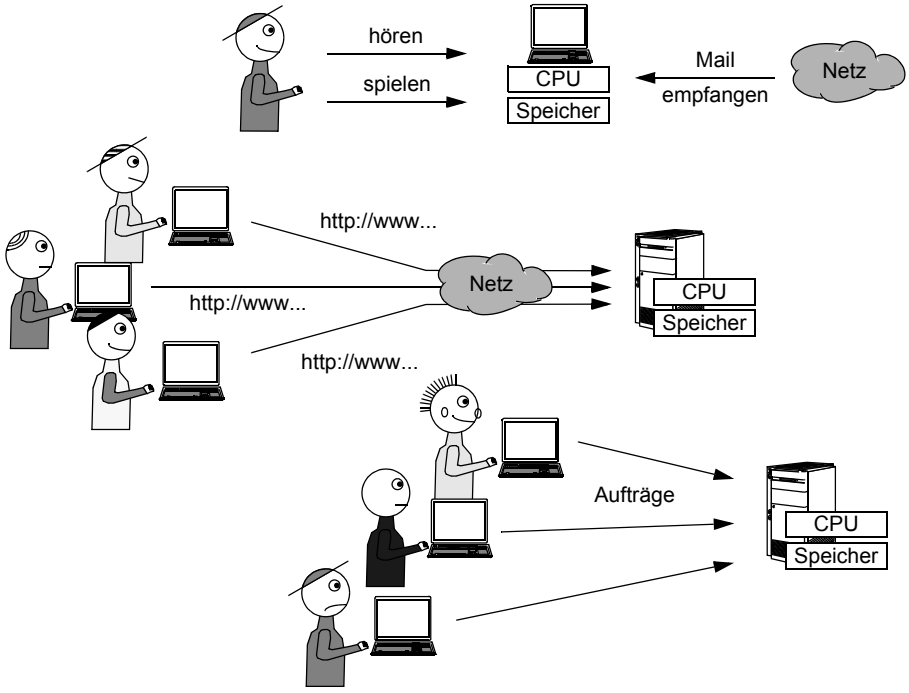


BILD 2.1
Nebenläufigkeit aus Benutzersicht

Dieses Kapitel zeigt, welche Formen der Nebenläufigkeit durch die Hard- und Software eines Computers realisiert werden, welche Rolle das Betriebssystem dabei spielt und welche grundlegenden Techniken UNIX/Linux und Java bieten, um nebenläufige Anwendungen zu programmieren.

2.1 Formen der Nebenläufigkeit

Nebenläufigkeit wird in Computern erstens durch Gruppen von *Hardware*komponenten realisiert, die echt gleichzeitig Aktionen ausführen können. Zweitens wird Nebenläufigkeit in *Software* implementiert: Mehrere Programmstücke können gleichzeitig ausgeführt werden, sofern es logisch nicht zwingend erforderlich ist, dass sie hintereinander ablaufen. Software-Nebenläufigkeit kann durch Hardware-Nebenläufigkeit unterstützt werden, muss es aber nicht: Eine nicht nebenläufige Hardware kann im raschen Wechsel zwischen nebenläufigen Softwarekomponenten hin- und hergeschaltet werden, so dass ein Beobachter den Eindruck bekommt, dass mehrere Dinge zur selben Zeit geschehen.

2.1.1 Hard- und Software – eine Kurzeinführung

Als Einstieg in diesen Abschnitt soll kurz das nötigste Basiswissen zur Rolle der Hard- und Software in Computern präsentiert werden. Wer sich hier „fit“ fühlt, kann die folgenden zweieinhalb Seiten überspringen und bei Bedarf später hierher zurückkehren.

2.1.1.1 Computer-Hardware

Die **Hardware** eines Computers gliedert sich in Zentraleinheit und Peripherie:

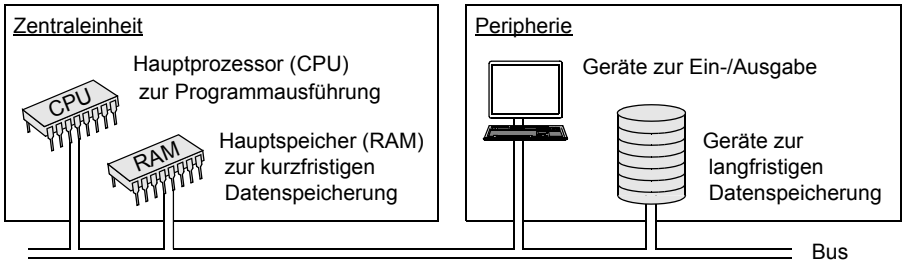


BILD 2.2
Eine einfache Hardware-Architektur

- Zur **Zentraleinheit** gehören der Hauptprozessor und der Hauptspeicher. Der **Hauptprozessor** (engl.: **central processing unit**, **CPU**) führt Programme aus, die als Folgen von **Maschinenbefehlen** vorliegen. Maschinenbefehle sind, im Gegensatz zu Befehlen höherer Sprachen, so einfach, dass sie von der Hardware unmittelbar ausgeführt werden können. Der Prozessor stützt sich dabei auf **Register** – kleine Speicherstellen, die die Ein- und Ausgabedaten des aktuell ausgeführten Befehls aufnehmen und zudem Informationen über die Programmausführung speichern. Zu den Prozessorregistern gehören insbesondere
  - der **Befehlszähler** (engl.: **program counter**, **instruction pointer**), der angibt, wo im Maschinenprogramm sich die Ausführung aktuell befindet,
  - das **Programmstatuswort**, das im Wesentlichen Informationen über das Ergebnis des zuletzt ausgeführten Befehls enthält (zum Beispiel, ob es gleich null oder negativ war), sowie

– der **Stack Pointer**, der die aktuelle Spitze des Aufrufstacks angibt. Der Aufrufstack ist ein Speicherbereich, der die Parameter und lokalen Variablen geschachtelter Funktionsaufrufe enthält.

Der **Hauptspeicher** (engl.: **main memory**, auch **random access memory**, **RAM**) speichert die Maschinenbefehle der aktuell laufenden Programme und ihre Daten. Maschinenbefehle können ihre Operanden durch Hauptspeicheradressen identifizieren, also durch Nummern von Hauptspeicherzellen, auf die der Prozessor dann direkt zugreift. Häufig benötigte Daten und Befehle können in einen **Cache** aufgenommen werden – einen kleinen, aber schnellen Speicher, der näher am Prozessor liegt. Bei Platzmangel können Hauptspeicherbereiche auf Hintergrundspeicher ausgelagert werden.

- Zur **Peripherie** gehören **Hintergrundspeicher**, wie insbesondere der Plattenspeicher, sowie **Ein-/Ausgabegeräte**. Auch Anschlüsse für **Datennetze** und für **Wechseldatenträger** (insbesondere USB-Sticks und CDs/DVDs) sind Teil der Peripherie.
- Die Komponenten von Zentraleinheit und Peripherie sind durch ein oder mehrere **Busse**, also lineare Leitungsbündel zur Datenübertragung, oder komplexer aufgebaute Leitungsnetzwerke miteinander verbunden.

2.1.1.2 Computer-Software

Die Hardware eines Computers wird durch **Software**, also durch **Programme** gesteuert. Üblicherweise werden Programme in einer **höheren Programmiersprache** wie C oder Java geschrieben. Die Befehle einer solchen höheren Sprache sind allerdings zu komplex, um direkt durch die Prozessor-Hardware ausgeführt zu werden. Sie müssen daher in **Maschinenprogramme** (also Folgen von Maschinenbefehlen) umgesetzt werden. Hierzu gibt es zwei Ansätze:

- Bei der **Übersetzung** (**Compilierung**) wird ein höhersprachiges Programm in ein äquivalentes Maschinenprogramm umgeformt, das anschließend durch den Prozessor ausgeführt werden kann. Ein Beispiel für eine höhere Sprache, deren Programme in Maschinensprache übersetzt werden, ist C.

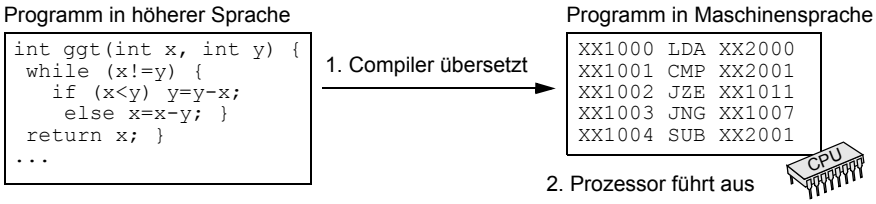


BILD 2.3
Übersetzung von Programmen

- Bei der **Interpretation** (→ Bild nächste Seite) wird ein höhersprachiges Programm Befehl für Befehl durchlaufen. Dabei wird für jeden Befehl unmittelbar ein entsprechendes kleines Maschinenprogramm ausgeführt. Der Durchlauf durch den Programmtext, seine Analyse und die Aufrufe der Maschinenprogramme werden durch einen **Interpreter**, ein Dienstprogramm, gesteuert. Ein Beispiel für eine höhere Sprache, deren Programme interpretiert werden, ist PHP.

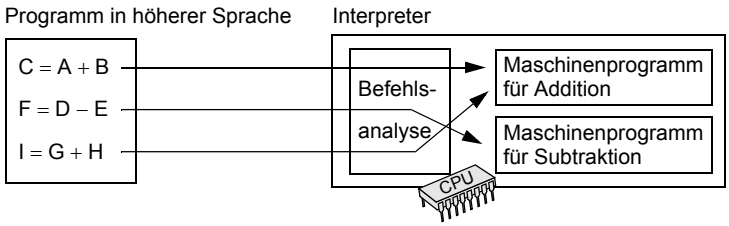


BILD 2.4
Interpretation von Programmen

Bei der Ausführung von **Java-Programmen** kommt eine Mischform aus Übersetzung und Interpretation zum Einsatz: Ein Java-Programm wird zunächst durch den Java-Compiler in maschinennahen **Bytecode** übersetzt. Die **Java Virtual Machine (JVM)** interpretiert dann diesen Bytecode, lässt ihn also auf der realen Hardware ausführen.

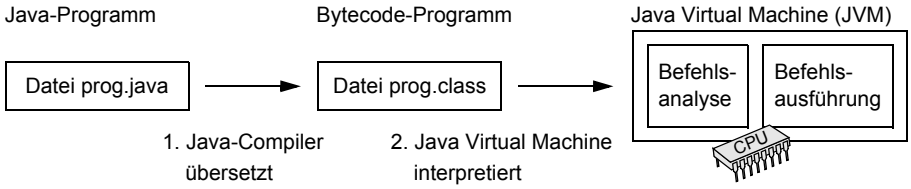


BILD 2.5
Übersetzung und Interpretation eines Java-Programms

Software, also insbesondere Maschinenprogramme, wird in Dateien auf Hintergrundspeichern oder Wechseldatenträgern gespeichert. Sie muss in den Hauptspeicher geladen werden und wird von dort (zusammen mit den zugehörigen Daten) Befehl für Befehl durch den Prozessor gelesen und ausgeführt.

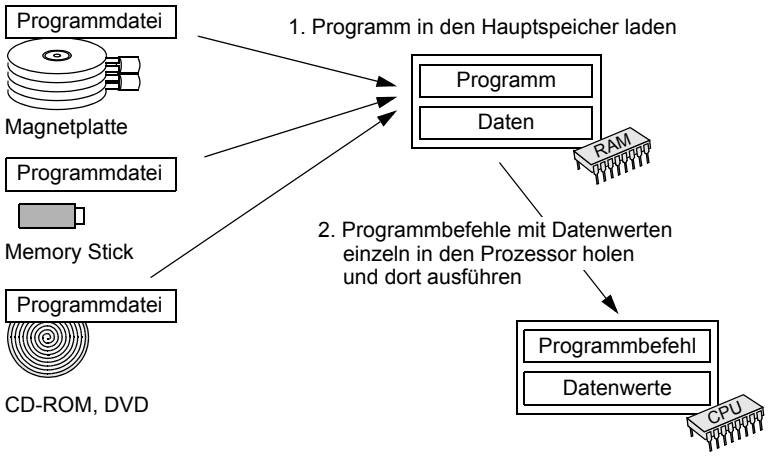


BILD 2.6
Laden und Ausführen von Software

2.1.2 Nebenläufigkeit in Hardware

Hardware-Nebenläufigkeit wird in heutigen Computern auf verschiedenen Ebenen realisiert. Eine grundlegende Rolle spielen dabei die Prozessoren (CPUs), die für die Ausführung von Programmen zuständig sind:

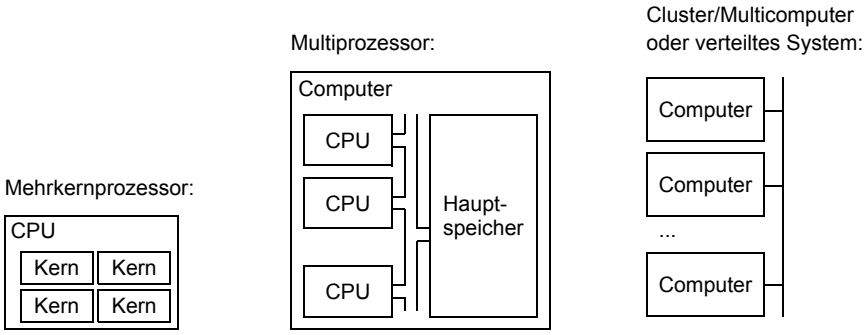


BILD 2.7
Nebenläufigkeit in Hardware

- Heutige Prozessor-Chips enthalten oft **mehrere Prozessorkerne** (kurz „Kerne“ genannt). Ein Kern umfasst zumindest ein Rechenwerk (ALU = Arithmetic-Logic Unit) zur Ausführung arithmetischer und logischer Operationen sowie Register zur kurzfristigen Speicherung von Daten und Speicheradressen. Dazu können noch weitere Komponenten wie beispielsweise Caches (kleine, schnelle Datenspeicher) kommen. Ein Kern ist damit in der Lage, Maschinenbefehle auszuführen. Sind also mehrere Kerne vorhanden, so können mehrere Programme oder Programmteile echt gleichzeitig ausgeführt werden.
- Mehrere Prozessor-Chips können zu einem **Multiprozessorsystem** zusammengefasst werden, das sich in einem einzelnen Computer befindet. Die Prozessoren sind über einen Bus (ein Leitungsbündel) oder ein komplexer aufgebautes Verbindungsnetz mit einem gemeinsamen Hauptspeicher verbunden und können so eng zusammenarbeiten.

  Da jeweils nur ein Prozessor gleichzeitig auf den Hauptspeicher zugreifen kann, kann dieser zum Engpass („Bottleneck“) werden. Als Gegenmaßnahme ordnet man den einzelnen Prozessoren private Caches zu, die Kopien der am häufigsten benutzten Hauptspeicherdaten enthalten und somit den Hauptteil der Speicherzugriffe abfangen.
- Schließlich können mehrere Computer („**Rechnerknoten**“) zu einem Gesamtsystem kombiniert werden. Die beteiligten Knoten haben jeweils ihre eigenen Prozessoren und Hauptspeicher und sind über ein Kommunikationsnetz miteinander verbunden. Über das Netz werden Daten und Aufträge ausgetauscht, so dass die Computer bei der Lösung komplexer Aufgaben kooperieren und die Arbeitslast untereinander verteilen können. Man kann hier zwei Architekturen unterscheiden:
  - Bei einem **Cluster** (auch **Multicomputer** genannt) sind die Computer eng miteinander verbunden: Sie stehen örtlich nah beieinander, werden möglicherweise durch ein gemeinsames Betriebssystem gesteuert und teilen sich eventuell Peripheriegeräte. Das Kommunikationsnetz ist schnell.
  - Bei einem **verteilten System** (zur Begriffsdefinition siehe auch → 2.2.1.3) ist der Verbund dagegen lockerer: Die Computer können räumlich weit verteilt sein und haben jeweils ihre eigenen Betriebssysteme und Peripheriegeräte. Das Kommunikationsnetz (meist das Internet) ist relativ langsam.

Offensichtlich realisieren alle Ansätze eine **echte Nebenläufigkeit**: Da mehrere Funktionseinheiten vorhanden sind, können *zum selben Zeitpunkt* mehrere Operationen ausgeführt werden, also **echt gleichzeitig**. Jedoch sind die Funktionseinheiten *unterschiedlich stark gekoppelt*: Die **Kopplung** bei Multiprozessorsystemen ist eng, da die Prozessoren über ein schnelles Kommunikationsmedium und einen gemeinsamen Hauptspeicher verbunden sind. Die Kopplung bei Clustern und mehr noch bei verteilten Systemen ist demgegenüber lose: Hier hat jeder Computer seinen eigenen Hauptspeicher und die Kommunikation läuft (unter anderem aufgrund der Entfernungen) langsamer als in einem Multiprozessor ab.

2.1.3 Nebenläufigkeit in Software

Auch die Nebenläufigkeit von Software kann auf verschiedenen Ebenen realisiert werden:

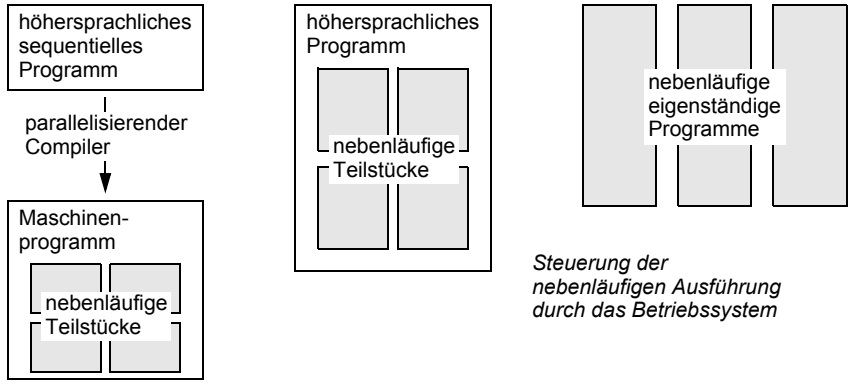


BILD 2.8 Nebenläufigkeit in Software

- Ein **Compiler** kann ein höhersprachliches sequentielles Programm *parallelisieren* und somit beschleunigen: Der Programmierer hat das Programm als Folge von Schritten geschrieben, von denen er denkt, dass sie hintereinander (also „sequentiell“) ablaufen sollen. Der Compiler ermittelt jedoch (anhand von Datenabhängigkeiten) voneinander unabhängige Programmstücke, die problemlos auch nebenläufig ausgeführt werden können, und übersetzt sie in entsprechende Teilfolgen von Maschinenbefehlen. Diese können dann auf die verschiedenen Prozessoren oder Prozessorkerne der Hardware (→ 2.1.2) gebracht werden.
- Der **Programmierer** kann sein Programm selbst in nebenläufige Teile untergliedern. Er benutzt dazu höhersprachliche Befehle, die so genannte *Prozesse* oder *Threads* erzeugen und steuern. Das Betriebssystem bringt dann diese Prozesse und Threads auf der Hardware nebenläufig zur Ausführung. Die Programmierung mit solchen Prozessen und Threads wird ein zentrales Thema dieses Buchs sein.
- Das **Betriebssystem** kann mehrere Programme, die nicht unbedingt etwas miteinander zu tun haben, gleichzeitig auf dem Computer ausführen lassen. Es teilt dabei die Programme den nebenläufigen Hardwarekomponenten zu und/oder schaltet die Hardware zwischen den Programmen um.

Die Nebenläufigkeit von Software ist oft „nur“ eine **Pseudonebenläufigkeit**: Sind mehr nebenläufige Software-Komponenten (z.B. Programme) als nebenläufige Hardware-Einheiten (z.B. Prozessoren) vorhanden, so können die Software-Komponenten nicht alle echt gleichzeitig ausgeführt werden. In diesem Fall wird die Hardware in derart hoher Frequenz zwischen den Programmen, Prozessen oder Threads hin- und hergeschaltet, dass ein Beobachter den Eindruck der gleichzeitigen Ausführung hat.

Fasst man die Betrachtungen dieses Abschnitts 2.1 zusammen, so kommt man zu der folgenden Definition der Nebenläufigkeit in Computern:

DEFINITION
„Nebenläufigkeit“

Nebenläufigkeit (engl.: **concurrency**) ist die gleichzeitige Ausführung von Aktivitäten. Der Begriff *Aktivität* bezeichnet hier eine Operationsfolge, die auf der Computer-Hardware in Bearbeitung ist. Bei **echter Nebenläufigkeit** können zum selben Zeitpunkt *t* Operationen mehrerer Aktivitäten auf der Hardware ausgeführt werden. Bei der **Pseudonebenläufigkeit** ist zu jedem Zeitpunkt *t* höchstens eine Operation auf der Hardware in Ausführung; die Hardware wird jedoch derart rasch zwischen den Aktivitäten umgeschaltet, dass ein menschlicher Betrachter den Eindruck einer echt gleichzeitigen Ausführung hat.

2.2 Die Rolle des Betriebssystems

Betriebssysteme sind *Verwalter* von Computern. Sie spielen damit bei der Realisierung von Nebenläufigkeit eine zentrale Rolle. Durch ihre *Betriebsart* bestimmen sie, welcher Grad von Nebenläufigkeit überhaupt möglich ist – hier reicht die Bandbreite von *strenger Sequentialität* (also Hintereinanderausführung einzelner Aufträge) bis zu *voller Nebenläufigkeit*. Zudem unterstützen sie das Konzept der *Prozesse* und *Threads*. Sie stellen so Dienste bereit, über die Benutzer und Anwendungsprogramme nebenläufige Aktivitäten starten und steuern können, und sie bringen diese Aktivitäten auf der realen Hardware zur Ausführung.

2.2.1 Systemarchitekturen

Zum Einstieg in diesen Abschnitt soll etwas Basiswissen über Betriebssysteme vermittelt werden, das im Zusammenhang mit Nebenläufigkeit relevant ist. Eine vertiefende Darstellung findet man in der einschlägigen Betriebssysteme-Literatur (z.B. [Silb10], [Stal12], [Tane08] oder [Vogt01]).

2.2.1.1 Aufgaben und Schnittstellen

In seiner klassischen Form setzt ein Betriebssystem auf die reale Hardware auf, nutzt also deren Dienste und steuert sie (→ Bild nächste Seite). Nach „oben“ bietet das Betriebssystem Dienste und Funktionen an, über die Benutzer und Anwendungsprogramme den Computer nutzen können. Ein Betriebssystem hat also zwei **Aufgaben**:

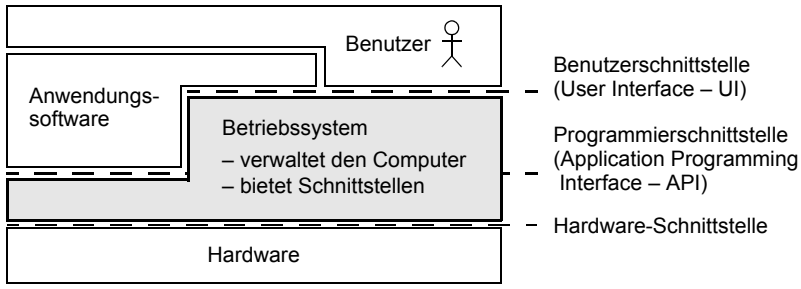
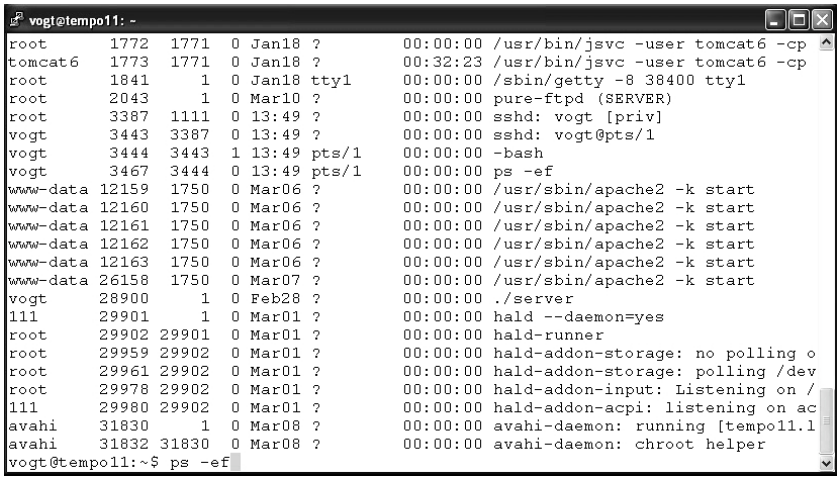


BILD 2.9
Position und Aufgaben des Betriebssystems

- Das Betriebssystem **verwaltet und steuert** die Komponenten des Computers. Dabei nutzt es die Funktionen der Hardware, die diese an ihrer Hardware-Schnittstelle bereitstellt. Zu den Kernaufgaben gehören hier
  - die Verwaltung von **Prozessorzeit**, also die „Scheduling"-Entscheidung (→ 2.2.4.3), wann welche Aktivität ausgeführt wird,
  - die Zuteilung von **Hauptspeicherplatz** und sonstigen Ressourcen an die einzelnen Aktivitäten und
  - die Steuerung des Zugriffs auf **Peripheriegeräte** und **Kommunikationsnetze**.

  Besonders anspruchsvoll sind diese Aufgaben bei einer Hardware mit mehreren Prozessoren und/oder Prozessorkernen (→ 2.1.2), da hier
  - die Aktivitäten auf die Prozessoren/Prozessorkerne verteilt,
  - deren Speicherzugriffe aufeinander abgestimmt („synchronisiert") und
  - die Datenübertragung zwischen den einzelnen Hard- und Software-Einheiten organisiert werden müssen.
- Das Betriebssystem stellt Benutzern und Anwendungsprogrammen **Schnittstellen** zur Verfügung, über die diese den Computer nutzen können. Zu den Schnittstellen gehören die Benutzer- und die Programmierschnittstelle (→ Bild nächste Seite):
  - Über die **Benutzerschnittstelle** (engl.: **user interface**, **UI**) kann der Benutzer mit dem Computer kommunizieren. Man unterscheidet textorientierte und grafische Schnittstellen, über die Kommandos in einer textuellen Kommandosprache bzw. über eine Maus oder einen berührungsempfindlichen Bildschirm eingegeben werden.
  - Die **Programmierschnittstelle** (engl.: **application programming interface**, **API**) ist eine Sammlung von Funktionen, die aus einem höhersprachigen Programm heraus aufgerufen werden können. Über die Programmierschnittstelle kann also ein Anwendungsprogrammierer auf die Dienste des Betriebssystems zugreifen.

    UNIX/Linux realisiert eine Programmierschnittstelle in der Programmiersprache C. Sie wird angeboten durch den UNIX- bzw. Linux-**Kern** (also den „innersten" Teil des Betriebssystems, der unmittelbar auf die Hardware aufsetzt) zusammen mit der **C-Bibliothek**. Das folgende Programm zeigt als Beispiel



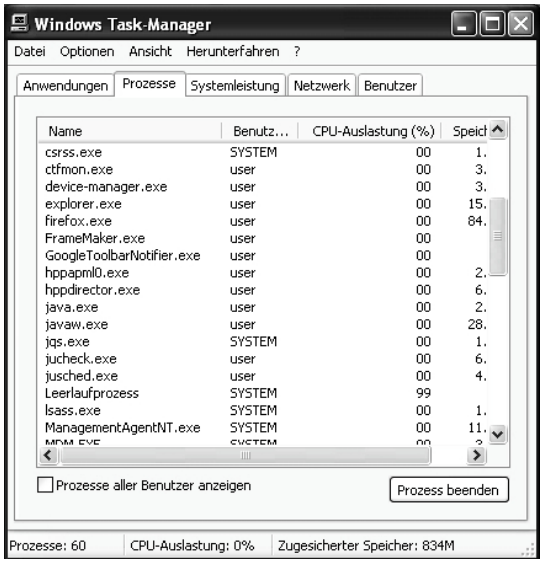


BILD 2.10
Textorientierte und grafische Benutzerschnittstelle

Anzeige der aktiven nebenläufigen „Prozesse“:
– an der textorientierten Linux-Schnittstelle (oben)
– an der grafischen Windows-Schnittstelle (links)

Aufrufe der UNIX/Linux-Schnittstellenfunktionen `open()`, `write()` und `close()`, durch die eine Datei neu erzeugt und mit einem Text gefüllt wird:

```
#include <stdio.h>  /* Standard-Ein-/Ausgabe */
#include <string.h> /* Verarbeitung von Zeichenketten */
#include <stdlib.h> /* Funktion exit() u.a. */
#include <errno.h>  /* Fehlerbehandlung */

main() {
 int fd;   /* Deskriptor für die geöffnete Datei */
 int err;  /* Zwischenspeicher für Fehlermeldungen */
 /* Eine Datei namens "myFile" neu erzeugen (O_CREAT)
    und zum Schreiben (O_WRONLY) öffnen.
```

PROG 2.1
Aufruf von Funktionen der UNIX/Linux-Programmierschnittstelle (API)