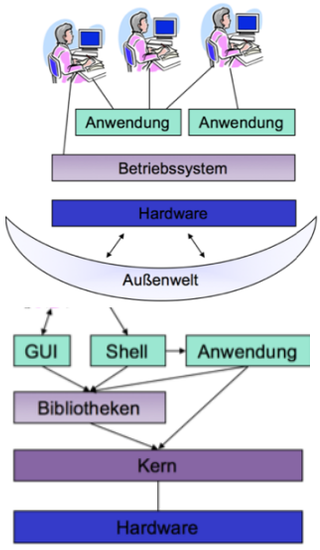
Kapitel 1: Einführung und die Shell

Aufgaben von Betriebssystemen

Wir betrachten Betriebssysteme (BS) aus der **Benutzerinnen- und Benutzersicht**.

Allgemeine Definition Betriebssystem (BS)

- Das BS ist ermöglich die **gleichzeitige**, **komfortable** und **hardwareunabhängige** Benutzung des Rechners.



Aufbau eines UNIX-Betriebssystems

Benutzerschnittstelle (GUI oder Shell):

- Aus Sicht des BS eine normale Anwendung nicht Teil des BS.
- Verschiedene Benutzerschnittstellen möglich, z. B. viele GUIs (KDE, Gnome, Unity, . . .), viele Shells (sh, bash, ksh, …)

Bibliotheken:

- Sammlung von nützlichen Funktionen, aus Anwendungen aufgerufen.

Betriebssystemkern (engl. kernel)

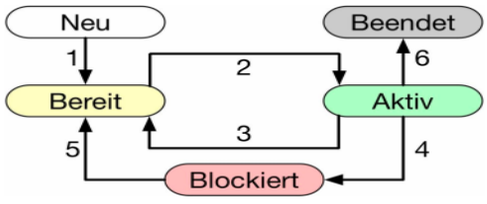
- Darf als einziger alles, kontrolliert alles („UNIX“ im engeren Sinn)

Rechnerhardware

- Die **CPU** ist das zentrale Element eines Rechners. Er kann Befehle in Maschinensprache ausführen.
- Die CPU kann im **Benutzermodus** oder im **Systemmodus** arbeiten.
  Im **Systemmodus** sind alle Befehle erlaubt
  Im **Benutzermodus** lösen einige kritische Befehle ein **Interrupt** aus.
- Im Rechner ist ein **RAM** eingebaut, in dem Byte an bestimmten Adressen gespeichert werden können.

Prozess = Programm : im RAM

Das Prozess-Konzept



- Ein **Prozess** ist ein laufendes Programm, es kann verschieden weit in der Ausführung vorangekommen sein.
- **dynamische Ausführung eines Programms** im Hauptspeicher und wechselt Zustände
- Jedem Prozess ist ein **Adressraum** zugeordnet.
  Dort liegen der Programmcode, die Daten, der Stack, eine „Halde“ (engl. **heap**).
- Schnelle Prozessoren (engl. CPU) wechseln zwischen ausführ-bereiten Prozessen und erlauben somit Multiprogrammbetrieb. Es scheint, als würden mehrere Programme „gleichzeitig“ laufen.

Adressräume

- **Hauptspeicher (RAM**) -> Eine Speicherzelle ist ein Byte groß und sie wird durch eine Nummer (**Adresse**) angesprochen.
- BS stellen jedem Prozess einen **virtuellen Adressraum** zur Verfügung.
- RAM gilt als Zwischenspeicher, dieser ist aber begrenzt
  => Lösung durch Memory Management Unit (MMU)
- **Virtuelle Adressräume** sind heute meist 264 Bytes groß.
- Eine Adresse ist also eine 64 Bit lange Binärzahl.
- **Mit Hilfe virtueller Adressierung können auch Programme ausgeführt werden, die mehr Speicher brauchen, als RAM im Computer eingebaut ist.**

Dateien und Dateisysteme

- **Festplatten** (HD) zur dauerhaften Speicherung von Daten. (in Sektoren **Datenblock speichern**)
- **Blöcke** sind meist 512 Bytes groß und werden durchnummeriert.
- Ein Dateisystem werden Dateien und Verzeichnisse zur Organisation angelegt.
- Im **Dateisystem** merkt sich das Betriebssystem wie Dateien heißen und in welchen Sektoren die Inhalte einer Datei gespeichert sind.
- Dateien können in **Verzeichnissen** zusammengefasst sein. Dadurch entsteht eine hierarchische Struktur (**Baum**).

Benutzer und Gruppen

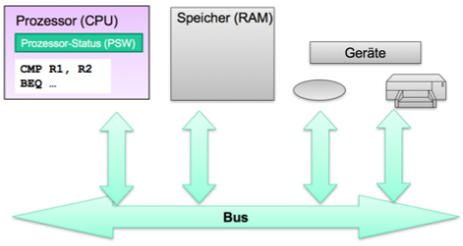
- Jede*r Benutzer*in hat eine **Benutzernummer** und meist dazu einen **Benutzernamen**
- Anmeldung *(User Name und Passwort eingeben)*: **Authentisierung**.
  Das Betriebssystem prüft die Anmeldedaten. **Authentifikation**.
  - Intern betrachten Betriebssysteme nur die Benutzernummern.
- **Zugriffskontrollen** definieren, welche Benutzer (**Subjekte**) auf welche **Objekte** (z. B. Dateien) wie (lesen, schreiben, usw.) zugreifen dürfen.
- Zur einfacheren Verwaltung dieser Zugriffsrechte werden
  Benutzer meist zu Gruppen zusammengefasst.
- Die Benutzerverwaltung in Rechnernetzen erfolgt meist zentral in einem **Benutzerverzeichnis** (engl. Directory)

Interrupts und Systemaufrufe

Rechneraufbau nach von Neumann

Ablauf bei Programmausführung:

1. In CPU-Register (PC) steht die Adresse des nächsten Befehls.
   Lade RAM[PC] in CPU.
2. CPU führt diesen Befehl aus, d. h.
   (1) sie lädt ggf. Operanden aus dem RAM,
   (2) berechnet das Ergebnis und
   (3) schreibt es ggf. zurück ins RAM.
3. Falls der Befehl ein Sprungbefehl war, dann PC = Sprungadresse, sonst PC = PC + 1
4. Weiter bei 1.



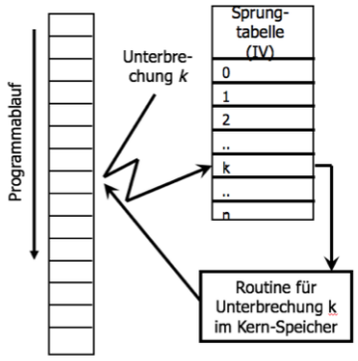
- CPU, RAM und Geräte sind an einen zentralen Transportbus angeschlossen.
- CPU liest Befehle und Daten aus dem RAM und führt sie aus.

Interrupts

- **Polling Verfahren**
  - CPU ständige Abfrage, ob Eingabe geschehen ist (Hohe CPU-Belastung)
- **Interrupt Verfahren**
  - Komponenten schicken sofort einen Interrupt-Request an CPU & jede Komponente hat einen eigenen Interrupt-Signal

- Betriebssysteme bieten eine abstrakte Schnittstelle zur Hardware, so dass Benutzer nur diese Schnittstelle kennen müssen.
- Der Zugriff auf Betriebssystemfunktion erfolgt über Systemaufrufe, d. h. über Interrupts und einen Wechsel in den System-Modus.

Ablauf der Interrupt Behandlung



1. CPU rettet die Rücksprungadresse,
   d. h. die Adresse der nächsten Anweisung (PC + 1).
2. In den PC wird die Einsprung Adresse der Interrupt-Routine kopiert.
3. CPU wird in den System-Modus geschaltet.
   Interrupt-Routine „rettet den CPU-Status“.
4. Die Interrupt-Routine behandelt nun das Ereignis.
5. Der alte CPU-Zustand wird restauriert.
6. PC = gerettete Rücksprungadresse, d. h. es geht an der Unterbrechungsstelle weiter.

Auslöser von Interrupts

- **Hardware-Ereignisse** können asynchron auftreten, also an jeder Stelle im aktuellen Programmablauf.
  - Eingabe-/Ausgabe-Operation beginnt oder endet.
    Beispiele: Taste gedrückt, Netzwerkpaket angekommen, usw.
  - Timer abgelaufen. Im Rechner gibt es eine Uhr, die regelmäßig eine Unterbrechung auslösen kann.
- **Software-Ereignisse** sind spezielle Befehle, die in einem Programm vorkommen können und damit eine Unterbrechung auslösen.
  - Ausnahmesituation bei Programmausführung, z. B. durch Division durch 0, illegaler Befehl, Zugriff auf verbotene Adresse, usw.
  - **Absichtlicher Betriebssystemaufruf** (engl. system call)

Besonderheiten bei Interrupts

- Interrupts können auch blockiert werden (maskiert).
- Wie verhält sich des Systems bei Interrupts während Ausführung einer Interrupt-Routine?
  - **Geschachtelte Interrupts**: Interrupt-Routine wird unterbrochen und dort geht es nach innerem Interrupt weiter.
  - **Sequentielle Interrupts**: erst wird laufende Interrupt-Routine zu Ende ausgeführt, danach erst das neue Interrupt.
- Details des Interrupt-Konzepts können von CPU-Herstellern selbst definiert und implementiert werden.
- Mit Hilfe von Interrupts kann der Zugriff auf bestimmte Systembestandteile (Befehle, Speicheradressen, usw.) kontrolliert werden.
- Interrupt-Routinen sind sehr kurz, damit sie sehr schnell ausgeführt werden können.
- Interrupts treten sehr häufig auf, mehrfach pro Sekunde.

Systemaufrufe

Wenn Benutzer die Abstraktionen eins Betriebssystems benutzen wollen, dann rufen sie Systemfunktionen auf.
Bei der Abarbeitung einer Systemfunktion findet folgendes statt:

1) Die Systemfunktion prüft die übergebenen Parameter.
2) Die Systemfunktion kopiert die Parameter an die richtigen Stellen.
3) Der Befehl für einen Software-Interrupt wird ausgeführt.
4) Die CPU wechselt in des System-Modus und führt die Interrupt-Routine aus.
5) Die Interrupt-Routine liest die Parameter und bearbeitet die gewünschte Funktion.
6) Funktionsergebnisse werden an die richtigen Stellen kopiert und der System-Modus wird beendet.
7) Hinter der Unterbrechungsstelle (Nr. 3) geht es weiter.

Der Kommandozeileninterpreter (engl. shell)

- Durch Befehle in der Shell kann man auf alle Betriebssystemobjekte zugreifen

Shell als Benutzerschnittstelle

Durch Tippen von Befehlen in der Shell kann ein Benutzer auf alle Betriebssystemobjekte zugreifen.

- Objekte haben Namen (oder Nummern), z. B. Dateien, Anwendungen, Benutzer, Prozesse, . . .
- Bei Befehlseingabe wird meist die gleichnamige Anwendung gestartet.
- ls *.pdf ruft das Programm /bin/ls auf und übergibt ls *.pdf als Argumente (Parameter)
    - /bin/ls wird bei Aufruf von ls aufgerufen, da /bin in der Umgebungsvariable $PATH enthalten ist. Die Shell durchsucht alle Pfade, die in der Variablen $PATH enthalten sind, nach dem aufgerufenen Programm ab.

Hauptaufgabe der Shell: Eingabezeile interpretieren und dann Anwendung ausführen.

Verarbeitungszyklus der Shell

1. Warte auf Eingabe einer Zeile
2. Interpretiere Zeile, d. h.
   (1) ersetze Shell-Metazeichen (*, $, `, etc.),
   (2) zerlege Zeile in Worte; erste Wort ist das Kommando, der Rest sind Parameter.
3. Führe das Kommando mit seinen Parametern aus.
4. Gehe zurück zu 1.

Kommandotypen:

- **Internes Kommando**: Auch eingebautes (engl. built-in) Kommando genannt. Wird von Shell selbst ausgeführt, d. h. Shell ruft direkt Bibliotheksfunktion auf.
  *(Mit dem internen Kommando **type** lässt sich prüfen, ob ein Kommando intern oder extern ist.)*
- **Externes Kommando**: Ist ein auf dem System installiertes Programm. Die Shell startet das Programm und wartet auf seine Beendigung.
  *(Mit dem externen Kommando **which** ermittelt man, wo ein installiertes Programm im Dateisystem steht.)*

Besonderheiten der Shell (s. auch Grundlagen der Informatik)

- [;] mehrere Kommandos in eine Zeiler eingeben.
  *Beispiel: echo Hallo; ls *.pdf*
- [&&] Das zweite Kommando wird nur dann ausgeführt, wenn das vorherige erfolgreich war.
- [$] beginnen Variablennamen.
  Beispiel: echo $PATH
- Externe Kommandos müssen mit vollständigem Pfadnamen angegeben werden.
  Beispiele: /home/w/bin/prog1 a1 oder ./bin/prog1 a1
- Vereinfachung durch die Umgebungsvariable **$PATH**. Wird nur ein Kommandoname eingegeben, dann sucht die Shell in den Verzeichnissen aus der Variablen **$PATH** nach dem Speicherort des Programms.
- Programmausgaben vom Bildschirm lassen sich in eine Datei umlenken.
  *Beispiel: ls -l *.pdf > allePDFdateien*
- Die Programmeingabe kann statt von der Tastatur aus einer Datei eingelesen werden.
  Beispiel: wc < eingabedatei
- Die Ausgabe eines Programms kann direkt als Eingabe eines weiteren Programms verwendet werden. Programme werden in diesem Fall über eine pipe verbunden.
  *Beispiel: ls -l *.pdf | wc*

Einige Kommandos:

touch <Datei>	Erstellt eine Datei
mkdir <Verzeichnis>	Erstellt ein Verzeichnis
pwd	Ausgabe working directory
cd <Verzeichnis>	Wechselt zum Verzeichnis
wc <Datei>	Zählt Wörter der Datei
man	Handbuch
grep “wort“ <Datei>	Sucht nach dem Wort in der Datei
cat	Zeigt Inhalt der Datei
less <Datei>	Ruft Textdateien auf
dig <dns>	IPv4 Adresse herausfinden
dig MX <dns>	Mailserver herausfinden

Besondere Zeichen in der Shell

- Kommandos und Parameter werden durch Zwischenräume getrennt.
  Zwischenräume sind Leerzeichen, Tabulatoren, Zeilenenden.
- Text in einfachen Anführungszeichen wird als ein Wort betrachtet.
  'abc $uvw' ist gleich abc $uvw
- Text in doppelten Anführungszeichen wird als ein Wort betrachtet. Variablennamen werden aber durch ihren Wert ersetzt.
  "abc $uvw" ist abc bsn1 falls Variable uvw den Wert bsn1 hat.
- Text in runden Klammern mit $ davor wird als Kommando interpretiert, Variablen werden ggf. durch ihre Werte ersetzt, das Ergebniswort dann ausgeführt und durch die Ausgabe des Kommandos ersetzt.
  *erg=$(ls *.pdf) speichert die Namen aller PDF-Dateien in der Variablen erg.*

Weitere Variablen:

- $# Anzahl der Parameter,
- $ Alle Parameter hintereinander.
- $? Beendigungsstatus des letzten Kommandos.

Shell-Skripte

Ein Shell-Skript ist eine normale Textdatei. Sie sollte ausführbar sein, d. h. das x-Bit muss gesetzt sein. Außerdem sollte sie im Pfad stehen.

```
#!/bin/bash
# Diese Datei heisst hs-shell
# Kommentarzeilen beginnen mit #
ssh -l alice -Y ssh.inform.hs-hannover.de
```

- Gestartet wird das Skript indem sein Dateiname eingegeben wird. *Beispiel: hs-shell*
- Auch ein Shell-Skript kann Parameter übergeben bekommen. Im Skript kann mit $1 auf den ersten Parameter zugegriffen werden. Zugriff auf Parameter 2–9 erfolgt analog. $0 ist der Name der Skript-Datei.

Sequenz

Damit Befehle hintereinander ausgeführt werden, werden diese hintereinander in eigene Zeilen des Skripts geschrieben.
In einer Zeile müssen Befehle durch Semikolon getrennt werden.

Bedingungen testen

Zahlen	Strings	Dateien
-eq bedeutet equal	== Gleichheit	-f $x ist $x eine Datei?
-ne bedeutet not equal	!= Gleichheit	-d $x ist$ x ein Directory?
-gt greater	-z $x ist $x leer?	-w $x ist $x beschreibbar?
-ge greater equal	< alphabetisch kleiner	-r $x ist $x lesbar?
-lt oder -le		

- boole'sche Operatoren: -a für AND, -o für OR und ! für NOT.

- Die Shell ist eine Terminal-basierte Schnittstelle zur Benutzung eines Computers.
- Shell-Skripte ermöglichen es, Abläufe zu automatisieren und Programme zu schreiben.

wc – word count
-n zeigt Zeilen an
^(...) = Anker sucht Anfang der line
$(..) = Anker am ende der line
-i = Groß o Klein Schreibung egal
-w = suche ein ganzes Wort – ende
-E = Erlaubt es regular expressions