Kommandos

```
exit /<Strg> D      # beenden der Shell
expr               # Arithmetische Ausdrücke, bei Vergleichen 0 = false
ls                 # Verzeichnis ausgeben
wc                 #  worte zählen
echo               #  ausgabe
cd
cat
man / –help
read var1 var2 ...  #  lesen von eingabe in variable
-gt                # größer als
-lt                # kleiner
```

Ströme

```
stdin: 0
stdout: 1
stderr: 2
```

Umleitung

```
>                  # Datei überschreiben
> >                # an Datei anhängen
<                  # aus Datei lesen (in stdin)
|                  # Pipe
&                  # Prozess im Hintergrund starten ⇔ Parallele Verarbeitung
;                  # Sequenz
```

Variablen

```
var=12             # definition
echo $var          # ausgabe
set                # ausgabe aller Umgebungsvariablen
```

Parameter

```
$#                 # Anzahl der Parameter
$-                 # übergebene Schalter (z.B. -a)
$@                 # Alle Parameter (einzeln)
$*                 # Alle Parameter (zusammengefasst)
$?                 # Wert des letzten ausgeführten Kommandos
$_                 # letztes Argument des letzten Kommandos
$$                 # PID dieser Shell
$!                 # PID des letzten Hintergrundkommandos
```

Kontrollstrukturen

if

```
# wahr: 0, falsch !=0
if Bedingung       # z.B. if [ "$v1≟ "$v2"]
  then
  elif Bedingung
  then
  else
fi
```

Mehrfachauswahl

```
case $var in
  1) echo wert = 1
  c) echo wert = c
  *) echo Ungueltig
esac
```

for

```
for x in $Liste      #  for F in ‘find -name bla*‘
  do

done
```

Variante: Zählschleife

```
for (( i = 1; i <= $max; i++ ))
```

while

```
while Bedingung
  do
done
```

until

```
until Bedingung
  do
done
```

Allgemein

Kernel: wesentliche Dienste des Betriebssystems, möglichst immer im RAM
Usermodus: Ablaufmodus für Anwendungsprogramme, kein Zugriff auf Kernel
Kernelmodus: Privilegiert für Kernelausführung, Wechsel über Syscall/Trap
Monolithisch: alle Module des Kernels in einem Adressraum (ein Prozess)

Schichtenmodell: Monolithischer Kern, Module durch Schichten strukturiert
Pro: effizient Contra: Modul kann zu Absturz führen (Blue-Screen)
Mikrokern: Module (z.B. Speicherverwaltung) als eigener Serverprozess, Kernel sorgt nur für Kommunikation zwischen Anwendung und Serverprozess;
Pro: läuft stabiler Contra: Häufiger Wechsel zwischen User/Kernelmodus
Teilhaberbetrieb: Viele User nutzen ein Programm
Teilnehmerbetrieb: Viele User nutzen eigene Programme

Interrupts

Polling: Busy waiting abfrage des Zustandes (Schleife)
Interrupt:

1. Unterbrechen des aktuellen Prozesses
2. Ausführung der Interrupt Service routine
3. Fortsetzen des Prozesses

Interrupt-Service-Routine: Kernel-Code der auf interrupt reagiert, aufruf durch index in Interrupt-Vector-Table
Systemcall/Trap: Software-Interrupt zur Kommunikation mit BS, z.B. fork()

Prozess

Prozesskontext: Zustandsinformation zum Prozess,
Speicherung in Process-Control-Block: Programmzähler, Prozesszustand, Priorität ,Verbrauchte Prozessorzeit seit dem Start des Prozesses, Prozessnummer (PID), Elternprozess (PID), Zugeordnete Betriebsmittel z.B. Dateien, ...
zustände: bereit, aktiv, blockiert
fork() klont Prozess, return 0 für kind, return kind-PID für Eltern

Scheduling

Ziele: Fairness, Effizienz, Antwortzeit, Verweilzeit (Durchlaufzeit), Durchsatz
Non-Preemptive Scheduling vs **Preemptive Scheduling** (Prozess kann unterbrochen werden)
Zeitscheibe/Quantum: zuteilung von Zeit, z.B. 3ms, an Prozesse, wechseln nach ablauf der Zeit/Blockieren;

Strategien

- First Come First Served (FCFS): In Ankunftsreihenfolge, Non-Preemptive
- Shortest Job First (SJF); Theoretisch Optimal, kürzester Gewinnt, Non-Preemptive
- Shortest Remaining Time Next (SRTN): kürzeste Restlaufzeit gewinnt, nicht preemtiv
- Round-Robin-Scheduling (RR) = Rundlauf-Verfahren: Der Reihe nach, Preemptiv
- Priority Scheduling (PS) statisch/dynamisch: höchste Priorität gewinnt, preemptiv
- Shortest Remaining Time First (SRTF): wie SRTN aber preemptiv
- Lottery Scheduling: Zufällige Vergabe von CPU-Zeit nicht preemptiv

Echtzeit-Betriebssystem:

garantierte zeiten; Tasks in endlosschleife
Parameter: Computation time C $\leq$ Deadline D $\leq$ Period T
Major Cycle / Hyperperiode: Wiederholung nach kleinstem gemeinsamen Vielfachen der Perioden
Rate Monotonic Scheduling (RMS): kürzeste Periode gewinnt, preemptiv
Earliest-Deadline First (EDF): nächste Deadline zuerst, preemptiv

Synchronisation

Race Conditions: geteilte Ressource, Ergebnis abhänging von Ausführungsreihenfolge
Kritische Abschnitte: logisch ununterbrechbare Code-bereiche;
Kriterien von Dijkstra:

- Keine zwei Prozesse gleichzeitig im gleichen kritischen Abschnitt (mutual exclusion)
- Keine Annahmen über die Geschwindigkeit, Anzahl der Prozesse bzw. Prozessoren
- Kein Blockieren durch Prozesse außerhalb eines kritischen Abschnittes
- kein ewiges Warten (fairness condition)

Methoden:

- busy waiting/spinlock: testen einer Variablen bis zutritt erlaubt
- interrupts maskieren: nur bei Einkernern, sehr ungünstig
- Hardwareunterstützung durch atomare Befehle
- Semaphore/Mutex

```
Semaphore x = new Semaphore();
```

```
x.Down(); // kritischer Abschnitt besetzt?
   c=counter.read(); // kritischer Abschnitt
   c++;
   counter.write(c);
x.Up();  // Verlassen des kritischen Abschnittes
```

- Erzeuger-Verbraucher

```
Erzeuger:                          Verbraucher:
While (true) {                     While (true) {
   produce(item);                     Down(belegt);
   Down(frei);                        Down(mutex);
   Down(mutex);                          getFromBuffer(item);
      putInBuffer(item);              Up(mutex);
   Up(mutex);                         Up(frei);
   Up(belegt);                     }
}
```

- Monitor: Ein Betriebsmittel aus Prozeduren und Daten, geshared zwischen Prozessen, aber nur von einem gleichzeitig nutzbar (bsp. synchronized in Java)
  Methoden: Enter, Leave, Wait, Pulse

Deadlock

Darstellung: Belegungsgraph; Prozess -> Ressource

Bedingungen:

- Mutual exclusion: Ressourcensharing nicht möglich (DVD-Brenner)
- Hold-and-wait: Prozesse belegen Ressourcen und wollen weitere
- No preemption: Entzug nicht möglich
- Circular waiting: gegenseitiges Warten

Strategien:

- Ignorieren (wenn selten)
- Erkennen und beheben (Erkennen: Zyklus im Belegungsgraph) :
  Unterbrechung, Rollback Prozessabbruch Transaktionsabbruch
- Dynamisches Verhindern: notwendig Vorwissen über Bedarf
  z.B. Bankiers-Algorithmus: prüfen, ob es eine Zuteilungsreihefolge gibt, bei der der Bedarf erfüllt werden kann
- Vermeiden: vermeiden einer der Deadlock-Bedingungen
  - Mutual exclusion: z.B. virtualisieren mit Spooling
  - Hold-and-wait: anfordern aller benötigten Ressourcen auf einen Schlag, oder freigabe alter Ressourcen bevor weitere Angefordert werden
  - Circular waiting: nummerieren der Ressourcen, anforderung nur in aufsteigender Reihenfolge
    z.B. in Echtzeitsysteme: Priority Ceiling Protocol
    Ressource hat Ceiling Priorität = maximale Priorität der Tasks, die sie verwenden werden. Der sie nutzende Task hat während der Nutzung diese Priorität

Kommunikation

Nachrichten: verbindungsorientiert vs verbindungslos, Synchron(Blockierend) vs Asynchron
Speicher: gemeinsamer Adressraum (Threads), Shared Memory, Datei (Prozess)

Interprozesskommunikation

- Pipes und FIFOs (Named Pipes) als Nachrichtenkanal
- Nachrichtenwarteschlangen (Message Queues)
- Gemeinsam genutzter Speicher (Shared Memory)
- Sockets (Ip-Loopback)

Pipes

Unidirektional, bidirektional über mehrere Pipes; Standardausgabe zu Standardeingabe

```
int fds[2] / Filedescriptoren für Pipe
pipe(fds);
if (fork() == 0) {
   // 1. Kindprozess, Standardausgabe auf Pipe-Schreibseite (Pipe-Eingang)
   dup2(fds[1], 1); // 1 = Standardausgabe
   close(fds[0]);
   write (1, text, strlen(text)+1);
}
```

```
else{
   if (fork() == 0) {
      // 2. Kindprozess, Pipe-Leseseite (Pipe-Ausgang) auf
      // Standardeingabe umlenken und Pipe-Schreibseite
      // (Pipe-Eingang) schließen
      dup2(fds[0], 0); // 0 = standardeingabe
      close(fds[1]);
      while (count = read(0, buffer, 4))
      {
         // Pipe in einer Schleife auslesen
         prozess Pipe
         buffer[count] = 0; // String terminieren
         printf(‘‘%s‘‘, buffer) // und ausgeben
      }
   else {
      // Im Vaterprozess: Pipe an beiden Seiten schließen und
      // auf das Beenden der Kindprozesse warten
      close(fds[0]);
      close[fds[1]);
      wait(&status);
      wait(&status);
   }
   exit(0);
}
```

Speicherverwaltung

Lokalitätsprinzip: örtlich: nah beieinanderliegende Daten werden oft zusammen benötigt, zeitlich: Daten werden oft sofort wieder benutzt
Adressraum: benutzbare Adressen (z.B. 2^{32}); Anordung durch Compiler (z.B.

Code	Konstanten	Heap	Stack)

Cache

write through (sofort) vs write back (bei auslagern) vs write on demand (erst durch expliziten Befehl)

Ersetzungsstrategien

- LRU (Least Recently Used) ältester Zugriffszeitstempel
- LFU (Least Frequently Used) Zähler, hochzählen pro zugriff, niedrigster wird zuerst ausgelagert;
  mit altern: setzen auf 0 nach intervall
- LRL (Least Recently Loaded) analog zu FIFO, ältester Einlagerunszeitstempel wird ausgelagert
- Zufällige Seite wird ausgelagert; billig umzusetzen

Speicherverwaltung

- Monoprogramming: Ein Programm hat gesamten RAM (bis auf BS)
- feste Partitionierung: Aufteilen des RAM in feste Bereiche, Prozesse bekommt einen zugeteilt
- Swapping: Prozesse werden im Ganzen ein/ausgelagert (Fragmentierung!)
- Virtueller Speicher: Aufteilung des Adressraums in Seiten, diese werden ein/ausgelagert. => Adressraum kann größer sein als RAM.

Umrechnung durch MMU:
$Adresse_{Virtuell}$ = Seite + Offset;
$Adresse_{Real}$ = Rahmen + Offset;
Seite $\Leftrightarrow$ Rahmen: Lookup in Seitentabelle

Ein/Auslagern in **Paging Area** auf Festplatte;
Ggf. Caching durch **TLB** (Translation Lookaside Buffer);
TLB-Einträge beinhalten PID
Page Fault: Seite nicht im RAM -> Einlagern nötig
Mehrstufige Adresstabellen: die bitgruppen gibt den index in der entsprechenden Tabelle an, diese enthält die Nummer der nächsten Seitentabelle
Bsp: $\underbrace{001100}_{1.Level}\underbrace{110101}_{2.Level}\underbrace{1001010001}_{offset}$
invertierte Seitentabelle: eine Tabelle mit Zuordnung Rahmen->Seiten; aufwändigere Suche, weniger Speicherbedarf (Lookup über Hashtabelle)

Ersetzungsstrategien

Demand Paging: nach Page-Fault

- Belady (optimal): ersetzung der Seiten die am spätesten in der Zukunft wieder verwendet wird

- FIFO: älteste Seite wird ersetzt, einfach zu implementieren (Verkettete Liste)
- Second Chance: ähnlich FIFO; ist R-Bit gesetzt hänge hinten an und setze R=0, sonst auslagern
- NRU (Not Recently Used): **R**ead-Bit **M**odified-Bit; Auslagerungsreihenfolge: R=0,M=0; R=0,M=1; R=1,M=0; R=1,M=1
- LRU (Least Recently Used) Am längsten nicht genutzt wird ausgelagert
- NFU (Not Frequently Used) Zähler für Zugriffe, auslagerung des Eintrags mit kleinstem Zähler
  Aging:
  - als Matrix: setze Bei Zugriff alle in Zeile auf 1, alle in spalte auf 0 Bsp: $\begin{pmatrix}0&1&0\\0&0&0\\1&1&0\end{pmatrix}, Zugriff auf 2 => \begin{pmatrix}0&0&0\\1&0&1\\1&0&0\end{pmatrix}$

1	1001	1100
0	1010	0101

  - als Register: shifte die R-Bits von links ein Bsp: $R1 Register: 0011 => 1001$

1	0001	1000
0	1101	0110

Prepaging: Working Set: aktuell Bearbeitete Seiten, versuch daraus die benötigten Seiten zu ermitteln; τ = Zeitraum für ein Workingset Bsp: Working Set Clock: wenn R == 1 { R=0, nächste Seite; } sonst { wenn Alter > τ und M == 0 überschreiben wenn Alter > τ und M == 1 sichere die geänderte Seite, betrachte nächste seite }
Ist der Zeiger wieder am Anfang: wenn seite ausgelagert wurde laufe weiter bis zur nächsten sauberen Seite. Wurde keine Seite ausgelagert: wähle Zufällige Seite (denn alle gehören zum Working Set)

Speicherbelegung

Suche nach freien Speicherbereichen;

- sequentielle Suche: erster Passender Bereich wird vergeben
- optimale Suche: möglichst genau passender Bereich wird vergeben um Fragmentierung zu vermeiden
- Buddy-Technik: schrittweises halbieren des Speichers => externe Fragmentierung sinkt, interne Steigt;

Cleaning

- Demand-Cleaning: Bei Bedarf
- Precleaning: Präventives Zurückschreiben, wenn Zeit ist
- Page-Buffering: Verwaltung in Listen (Modified List, Unmodified List)

Dateiverwaltung

Dateien: abstrahiert perisistente Speicherung
Dateien, Pseudodateien (Freiliste), Verzeichnisse, Gerätedateien (abstraktion von geräten, z.B. /dev/sda)
Sequentieller vs Wahlfreier Zugriff
Metadaten: informtionen über die Datei (Zugriffsrechte, Erstelldatum, zugriffsdatum...)
Operationen: create • delete • open • close • read • write • append • seek • get attributes • set attributes • rename
Memory-Mapping: einblenden der Datei in den Adressraum eines Prozesses
Master Boot Record auf Sektor 0, Partition Table, Bootblock, Superblock enthält Verwaltungsinformationen zum Dateisystem (Anzahl der Blöcke,...), Free Blocks (z.B. Bitmap) gibt die freien Blöcke des Dateisystems an, Rootverzeichnis enthält den Inhalt des Dateisystems, I-nodes;
Dateiimplementierung:

- Zusammenhängend => Fragmentierung
- Verkettung der Blöcke => Langsam
- Verkettet durch FAT im Arbeitsspeicher, hoher Platzbedarf
- I-Node: enthält: Metadaten, direkte Blockverweise, verweise auf ein-, zwei und dreifach-Indirekte Blöcke

Finden einer Datei: suche Verzeichnis anhand Pfad, Verzeichnis enthält I-Node-nummer, bzw. Nummer des 1. Blocks
Hard-Link: Verzeichnisse zeigen auf den selben I-Node; Symbolic Link: Datei vom Typ LINK enthält Pfad
Virtuelles Filesystem: abstraktionsschicht zwischen Systemaufrufen und Dateisystemen (Windows: Laufwerksbuchstabe, Linux: mounten im Verzeichnisbaum
Blockgröße: je größer desto bessere Datenrate, je kleiner desto Speichereffizienter
Freiblockverwaltung: Liste vs Bitmap

Konsistenzsicherung

Blockebene:
Zähler vorkommen eines blocks in dateien vs Zähler Block in Freiliste

- Fehlender Block => einfügen in Freibereichsliste
- Doppelt in Freibereich => Freibereich anpassen
- 1x Frei 1x Belegt => Aus Freibereich entfernen
- Belegt in 2 Dateien => Kopiere Block, ordne ihn einer der Dateien zu; ausgabe für den Benutzer

Dateiebene: Prüfe den Linkzähler aller Dateien und passe ihn an
sonstiges, z.B. unsinnige Zugriffsrechte, Dateien größe 0, etc.

Journaling

Vor ausführen einer Aktion: Eintrag in Log; => Aktion kann nach absturz wiederholt werden.

Cache

Puffern der Blöcke, zugriff über hash, auslagerungsstrategie meist LRU, angepasst nach heuristik;
Konsistenzrelevante Änderungen sofort rausschreiben, Write-Through oder regelmäßiges Rausschreiben von Änderungen sinnvoll

Flash

wichtig: gleichmäßige Abnutzung => zurückschreiben nicht an selber stelle (Copy-on-write) => "wandering Trees"
Flash Translation Layer: Abbildung Dateisystemadresse auf Sektor
Virtual Block Map: zuordnungstabelle im RAM, mehrstufig; Invertierte: jeder Sektor speichert seine Adresse, oder reservierter sektor pro bank enthält tabelle => scannen beim Mounten
Superblock: reservierte Bänke + Zeittempel, oder durchsuche gesamten Datenträger
löschen: informieren des Controllers mit TRIM-Kommando über gelöschte Blöcke, damit sie wiederverwendet werden können
Logging Dateisysteme: ähnlich Journaling, aber es werden nur die Änderungen geschrieben + Checkpoints; Garbage Collection
Vorteil: schnelleres Schreiben, passt zu Flash Nachteil: (Fragmentierung), Garbage Collecting

NTFS

Alles steht in MFT (Master-File-Table), Blöcke in Serien angegeben (von-anzahlBlöcke)

Virtualisierung

Abstraktion der Ressourcen durch Softwareschicht
BS-Virtualisierung: Softwareschicht: Hypervisor oder VMM
Vorteile von VMs: unterschiedliche Betriebssysteme gleichzeitig auf einem Rechner, Hardwareauslatung, Test nicht vertrauenswürdiger Programme, Einfaches Sichern / Wiederaufsetzen /Klonen (Snapshot)
Nachteile: VMs müssen sich reale Ressourcen teilen, Rechnerausfall => Ausfall aller VMs, probleme mit spezialhardware, ca. 10% Leistungsminderung
Emulation: interpretation der Maschinenbefehle, mögliche Optimierung: Just-In-Time Compilation
Virtualisierung: native Ausführung von Nichtprivilegierten Befehlen Voraussetzung: Ausreichend ähnliche Prozessoren
Technik: Hypervisor fängt bestimmte Befehle ab, und ersetzt sie durch geeignete Befehle (Implementiert durch Interrupt-Service-Routinen, Hypervisor im Kernelmodus, Gast-BS im Usermodus)
Mindestvorraussetzungen: Unterscheidung zwischen Kernel- und Anwendungsmodus durch Prozessor, MMU ,Kriterien nach Popek und Goldberg:
Popek und Goldberg: Es gibt privilegierte Befehle, diese Lösen Trap aus wenn nicht im Kernelmodus. alle sensitiven Befehle: (zustandsverändernd z.B. Zugriff auf I/O oder die MMU,etc.) müssen privilegiert sein.
Lösung falls nicht alle sensitiven Befehle priviligiert sind: Ersetzung zur Laufzeit/JiT-Compilation

Typen

Typ-1: Hypervisor als mini-Betriebssystem, ohne Wirt, Treiber ggf. problematisch
Typ-2: Hypervisor als Prozess des Wirtsbetriebssystem, nutzt Wirtstreiber mit
Paravirtualisierung: ersetzung kritischer Befehle im Gast Betriebssystem durch Hypercalls => schneller, vorraussetzung: Quellcode Gast-BS

Speicher

Virtuelle Adresse Gast -> Virtuelle Adresse Host -> Reale Adresse Host
Schattentabelle: tabelle für umsetzung $Virt_{Host} -> Real_{Host}$
alternativ Hardwarelösung EPT (Extended Page Table) in MMU

Treiber

Geräte-Emulation: Nutzen des Treibers des Hypervisors oder Host-BS, Standardgerät für den Gast;
schlechter Durchsatz
Direktzuweisung eines Geräts: Gast nutzt Gerät exklusiv mit eigenem Treiber
Effizienteste Möglichkeit, Erfordert spezielle Hardware
Verschieben einer VM: kopieren von Dateisystem + RAM von Quelle zum Ziel; Wiederholt die geänderten Teile umkopieren; anhalten Quelle, kopieren des Rests, Umleiten Netzwerk, Ziel-VM starten, Quell-VM löschen

Cloud

IaaS = Infrastructure as a Service: Cloud-Provider bietet Zugang zu virtualisierten Rechnern (inkl. BS) und Speichersystemen
PaaS = Platform as a Service: Cloud-Provider bietet Zugang zu Programmierungsumgebungen
SaaS = Software as a Service: Cloud-Provider bietet Zugang zu Anwendungsprogrammen