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()

Virtualisierung

Abstraktion von 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: Interpreter der Maschinenbefehle, Optimierung: JiT-Compilation
Virtualisierung: native Ausführung von Nichtprivilegierten Befehlen Voraussetzung: Ausreichend ähnliche Prozessoren
Technik: Hypervisor fängt kritische Befehle ab, und ersetzt sie durch geeignete Systemaufrufe (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 oder 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 des Gast-BS offen

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 aaS: Cloud-Provider bietet Zugang zu Programmierungsumgebungen
SaaS = Software aaS: Cloud-Provider bietet Zugang zu Anwendungsprogrammen

Synchronisation

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

- Nur ein Prozess gleichzeitig in kritischem 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

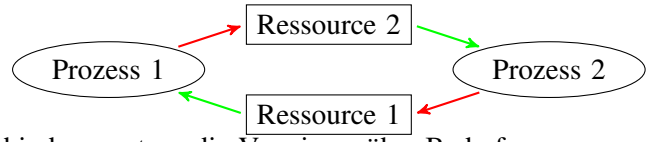
Bedingungen:

- 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

belegt von
wartet auf



- 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, nur in aufsteigender Reihenfolge anfordern
    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
  Standardausgabe zu Standardeingabe; Unidirektional, bidirektional über mehrere Pipes;
- Nachrichtenwarteschlangen (Message Queues)
- Gemeinsam genutzter Speicher (Shared Memory)
- Sockets (Ip-Loopback)

Speicherverwaltung

Lokalitätsprinzip: örtlich: nah beieinanderliegende Daten werden oft zusammen benötigt, zeitlich: Daten werden oft gleich wieder benutzt
Adressraum: benutzbare Adressen (z.B. 2^{32});
Anordung durch Compiler (z.B. | Code | Konstanten | Heap | Stack)
externe Fragmentierung: Speicher zerfällt in kleine Bereiche
interne Fragmentierung: Prozesse mehr speicher als nötig => Leere Bereiche

Cacheersetzungsstrategien

- 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
- Swapping: Prozesse werden im Ganzen ein/ausgelagert (Fragmentierung!)
- Virtueller Speicher: Aufteilung des Adressraums in Seiten, diese werden ein/ausgelagert. **(Paging Area)** => Adressraum kann größer sein als RAM.

Umrechnung durch MMU:
$Seiten = \frac{Adresse_{virtuell}}{Seitengröße}, Frame = \frac{Adresse_{real}}{Seitengröße}$
Offset = Adresse mod Seitengröße
$Adresse_{Virtuell}$ = Seite · Seitengröße + Offset;
$Adresse_{Real}$ = Rahmen · Seitengröße + Offset;
Seite $\Leftrightarrow$ Rahmen: Lookup in Seitentabelle

TLB (Translation Lookaside Buffer):MMU-Cache für Seiten->Frame zuordnung (ggf. mit 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/des Frames
Bsp: $\underbrace{01\ldots00}_{1.Level}\underbrace{11\ldots01}_{2.Level}\underbrace{1001\ldots0001}_{offset}$
invertierte Seitentabelle: eine Tabelle mit Zuordnung Rahmen->Seiten;
aufwändigere Suche, weniger Speicherbedarf (Lookup über Hashtabelle)

Ersetzungsstrategien

Demand Paging: nach Page-Fault

- Belady (optimal): ersetze am spätesten wieder verwendete Seite (Zukunft)
- FIFO: älteste Seite wird ersetzt, einfach zu implementieren (Verkettete Liste)
- Second Chance: ähnlich FIFO, aber: ist R-Bit gesetzt stelle hinten an und setze R=0
- NRU (Not Recently Used): **R**ead-Bit **M**odified-Bit; Auslagerungsreihenfolge: R=0,M=0; vor R=0,M=1; vor R=1,M=0; vor R=1,M=1
- LRU (Least Recently Used) Am längsten nicht genutzt wird ausgelagert
- NFU (Not Frequently Used) Zugriffszähler, auslagern des kleinsten Zählers

Aging:
- Zähler als Matrix: Bei Zugriff: Zeile auf 1, spalte auf 0

$$1: \begin{pmatrix}0&1&0\\0&0&0\\1&1&0\end{pmatrix}, Zugriff auf 2 => \begin{pmatrix}0&0&0\\1&0&1\\1&0&0\end{pmatrix}$$

- als Register: shifte die R-Bits von links ein

	R:	$Register$:		$Register$:
	1	1001		1100
Bsp:	0	1010	=>	0101
	1	0011		1001
	1	0001		1000

Prepaging: Working Set: gerade bearbeitete Seiten, τ = Zeitraum für ein Workingset
Working Set Clock: Seiten bilden Ring, lasse Zeiger laufen:
if R == 1:
    R=0, betrachte nächste Seite;
else
    if Alter > τ und M == 0 ersetzen und stop
    if Alter > τ und M == 1 sichere die geänderte Seite, betrachte nächste seite
Wurde jede Seite betrachtet: wenn seite ausgelagert wurde laufe bis zu sauberer Seite.
Wurde keine Seite ausgelagert: wähle Zufällige Seite (denn alle sind im Working Set)

Speicherbelegung

Suche nach freien Speicherbereichen;

- sequentielle Suche: erster Passender Bereich wird vergeben
- optimale Suche: möglichst genau passender Bereich wird vergeben
- Buddy-Technik: halbieren des Speichers zu passender größe => mehr externe, weniger interne Fragmentierung

Cleaning

Demand-Cleaning vs Precleaning
(z.B. Page-Buffering: Abarbeiten in Listen (Modified List, Unmodified List))

Dateiverwaltung

Dateien: abstrahierung der perisistenten Datenhaltung
Dateien, Pseudodateien (z.B. Paging Area), 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

Struktur (Unix)

Master Boot Record auf Sektor 0,
Partition Table,
Bootblock,
Superblock (enthält Verwaltungsinformationen zum Dateisystem (Anzahl der Blöcke,...)
Free Blocks (als Bitmap oder Liste)
Rootverzeichnis

Dateiimplementierung:

- Zusammenhängend => Fragmentierung
- Verkettung von Blöcken => Langsam
- Verkettet durch FAT im Arbeitsspeicher, hoher Platzbedarf
- I-Node:

Metadaten, Attribute
Direkte Verweise (siehe aufgabenstellung)
Adresse einfach Indirekter Block
Adresse doppelt Indirekter Block
Adresse 3fach Indirekter Block

  dreifach Indirekte -> doppelt Indirekte -> einfach Indirekte -> Blöcke
  Anzahl der Verweise pro Indirektem Block: $\frac{Blockgrösse}{Adressgrösse}$
- NTFS: Alles steht in MFT (Master-File-Table), Dateien als Listen von Serien (Start-Block & anzahlBlöcke)

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(schrumpft) vs Bitmap(feste größe)

Konsistenzsicherung

Blockebene: Zähle vorkommen eines blocks in dateien, und vorkommen 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 Logdatei; => Aktion kann nach absturz wiederholt werden.
Logging Dateisysteme: ähnlich Journaling, aber es werden nur die Änderungen geschrieben + Checkpoints; Garbage Collection
Vorteil: schnelleres Schreiben, passt zu Flash
Nachteil: (Fragmentierung), Garbage Collecting

Dateicache

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, ggf. mehrstufig;
Invertierte Block Map: 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

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** (Prozesse unterbrechbar)
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, Non-Preemptive
- Round-Robin-Scheduling (RR): vgl. Zeitscheibe, 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