Operating System

Abstraktionsebene zwischen Hard- und Software

Modular vs Monolyth

Modular: + kleiner Kernel − complexity (IPC)

Monolythic: + consistency − Bloatware (unused stuff)

Aufgaben

Ressourcenmanagement CPU/RAM/Disks/IO

Hardware Abstraktion → I/O devices

Booting

1. BIOS durchläuft POST (Power On Self Test)
2. Master Boot Record wird eingelesen & ausgeführt
3. MBR wählt einen Bootloader (GRUB) aus und lädt ihn
   a. OS wird gestartet, lädt Treiber aller Geräte
   b. Initialisiert OS Management Structure
   c. Creates system services (sys calls, etc.)
   d. Spawns a (User) interface (GUI)

Booting in Linux

1. System Startup / HW initialization
2. MBR wird eingelesen und ausgeführt
3. Kernel (Linux OS): OS start-up Code
4. INIT process (run levels)
5. User prompt (Shell or GUI)

BIOS vs. UEFI

UEFI (→ modular) ist im Gegensatz zum BIOS ein eigenes kleines OS. BIOS braucht den MBR und UEFI eine Partition. MBR wird mit GPT ersetzt für UEFI. UEFI nutzt eine Architektur unabhängige VM, dadurch lassen sich EFI binaries ausführen (modular, arbitrary number of partitions)

UEFI → EFI Bootloader → Kernel

ID	Name	Description
0	Halt	Shuts down the system
1	Single user-mode	Admin tasks
2	Multi user-mode	
3	" with networking	starts the system normally
4	Not used/user-definable	Special purposes
5	Start system normally with appropriate GUI	runlevel 3 + GUI
6	Reboot	

RAMDISK

SW needed to load Kernel is stored on that disk itself.

→ initrd stored in same area as Kernel. Contains kernel + basic device-special files.

systemd

System & service (→ daemons) manager for Linux. Provides dep-system between units. Unit (object) states: (in)active, (de)activating, failed. Unit types: service (OS service), target (like runlevel → group units). Targets don't offer additional functionality → group units

default config: /etc/systemd/system.conf

Unit files: /lib/systemd/system $\xrightarrow{\text{override}}$ etc/systemd/system

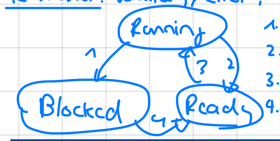
Processes

run in user-mode → may trap into kernel-mode (system calls)

○ associated owner, reside within an ENV

Creation: Boot, User Request, fork(), exec(), Interrup, Cron Jobs

Termination: Voluntary, error, murder (killed)



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes avail

Context vs. Mode Switch

- Mode switch: no sched involved (e.g. ISR)
- Mode/context switch: sched involved

Threads

Per process items: Address space, global variables, open files, child procs

Per thread items: Stack + PC, Registers, State

→ created & owned by a process. Linux Thread is a process that shares a configurable set of resources with other processes.

Kernel thread: runs exclusively in system mode,

Kernel vs. Linux Thread

Kernel: ⊕ Kernel kennt Threads → gut für viele I/O calls, (→ Scheduling auf Thread level) blocking (un-interruptible) ⊖ Threadwechsel = 2 Moduswitch

Thread: ⊕ effizienter, cheaper ⊖ IO = blocking process

Orphan vs. Zombie

Zombie: state until child sends exit state to parent and he acks it

Orphan: Parent terminated without waiting for children

Scheduling

sorts a queue according to some policy.

→ dispatcher moves the task from head to CPU (⇒ context switch)

− FIFO / FCFS → simple, non-preemptive

Round Robin

- One queue with time-slicing (slice = quantum)
- pre-emptive, no starvation, simple

1. running task interrupted every slice (arriving has prio)
2. if queue not empty, task adjourned to end of queue
3. task at head of queue gets dispatched

Heuristics

allows user to influence task priorities

Rate Monothic

→ highest prio given to task with highest repetition rate

guranteed schedule if: $U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq \underset{\ln(2) = 0{,}6931}{\underline{n(2^{1/n} - 1)}}$

$U \leq 1 \Rightarrow$ Sched possible ; $U > \ln(2) \Rightarrow$ nicht garantiert

Linux Scheduler ⇒ Completely Fair Scheduler

gibt jedem Task die gleiche CPU Zeit

nice: [−20, 19] → higher value = lower prio

Optimales Scheduling

CPU soll so ausgelastet sein, dass die Jobs in der kürzesten Zeit fertig sind.

Starvation: Prozess/Thread ist ready, bekommt aber keine CPU Zeit

(Non)preemtive Scheduling

In preemtive scheduling (→ RR, multi-level) wird die CPU einem Prozess für eine limitierte Zeit zur Verfügung gestellt, während in Non-preemptive ein Prozess die CPU hat bis er fertig/waiting ist.

Realtime vs. preemtive Scheduling

Realtime → Prozesse haben eine gewisse Zeit, in der sie ausgeführt werden müssen

Preemtive → CPU kann Prozesse unterbrechen (→ Starvation) und gibt Prozesse nur eine geringe CPU time

cgroups

Teil des Linux Kernels, mit dem sich die Nutzung von Ressourcen durch Prozesse beschränken + überwachen lässt.

→ "a cgroup is a collection of processes that are bound to a set of limits or parameters defined via the cgroup Filesystem" → Bsp.: CPU, memory

V1 vs. V2

V1: each V1 controller may be mounted against a seperate cgroup filesystem. Also possible to co-mount (multiple controllers against same cgroup). Controllers require at least one cgroup filesystem, can only be mounted to one at the same time. Each cgroup is represented by a directory. Task cannot be a member of two different groups in the same hierarchy.

V2: all mounted controllers reside in a single unified hierarchy

Operations

Creation: eine neue cgroup wird erstellt, indem man einen Ordner unter einem entsprechenden Controller erstellt.

```
mkdir /sys/fs/cgroup/cpu/newgroup
```

Löschen: cgroup darf keine child cgroups haben + cgroup darf keine non-zombie Prozesse beinhalten. → rmdir -rf ....

Memory

Swapping

Datenverschiebung zwischen RAM & Festplatte. Festplatte wird als temp storage für Prozesse verwendet, wenn der RAM voll ist. Lange nicht mehr gebrauchte Daten werden temporär auf die HDD geschrieben. → Multitasking ist möglich dank swapping

Placement Algorithms

→ Zuordnung von Prozessen im bestehenden Memory

1. First fit: first fragment that fits
2. Best fit: Zuordnung zum Fragment, dessen von der Grösse am besten passt → viele kleine ungenutzte Fragmente
3. Next fit: erstes Fragment ausgehend vom letzten Zugriff
4. Buddy: teilt memory / 2 bis best fit erreicht ist. Freie memory Blöcke werden wieder zu grösseren Blöcken gemerged.

Virtual Memory

Speicheranforderung von Prozessen übersteigt den physisch vorhandenen Speicher. OS gaukelt Prozessen vor unbegrenzt Memory zur Verfügung zu haben. → Prozesse werden auf der Harddisk gespeichert. Falls der Prozess aktiv ist, wird er ins Memory geladen.

Page Table: Daten sind im Memory verteilt oder auf die Festplatte ausgelagert (paged out). Die page table mappt bei einer Speicherabfrage eines Prozesses die virtuellen Adressen auf physische Adressen.

Page (virtual mem block) → Frame (physical mem block)

Memory Management Unit - MMU

- MMU schaut, dass Prozesse nur auf Memory zugreift, welcher ihnen zugeteilt wurde
- Address Translation: virtuelle Adresse → physische Addr.

Virtual address (page #, offset) 1. TLB hit or page table (law page #) 2. returns (frame #, offset)

Translation Lookaside Buffer - TLB

cached die letzten 64 Page Table Zugriffe. MMU schaut bei Zugriffen immer zuerst, ob die Entries im Cache sind. Falls nicht → check page table.

Page Replacement

→ wenn der physische Speicher im RAM überläuft.

1. FIFO: ersetzt die älteste Page in der Queue
2. Optimal: ersetzt page, welche in Zukunft am wenigsten häufig gebraucht wird. → nicht umsetzbar
3. LRU $\to$ Linux: ersetzt page, welche am längsten nicht gebraucht wurde

Page fault

→ Wenn ein Prozess auf eine Speicheradresse zugreift, welche momentan nicht ins Memory gemapped ist.

Input / Output

Device categories

Block: blocks with fixed size. → Harddisk

Character: operates on char streams

HW architecture

Southbridge: all I/O devices, PCI Bus, Flash rom (slow devices)

Northbridge: PCI express, CPU, memory bus (fast devices)

x86 architecture: map peripheral I/O registers to separate address range and buffers into memory

Direct Memory Access - DMA

CPU kommuniziert mit dem Speicher über einen Bus mit hoher Bandbreite. → DMA controller bearbeitet IO requests, damit die CPU weiterarbeiten kann

Access

Continuous: Polling, sync

Event-based: Interrupts, async

Blocking: Request → wait → receive

Non-blocking: Request → receive

} feature of the provider → instant access or not

Buffered IO: decouples data access from generation

Custom Kernel

+ tiny kernel → load only what is needed
- support for HW that is hot-pluggable

File systems

a file is an abstraction, a logical unit of information created by a process.

File Types

$\overset{d}{\text{Dirs}}$, $\overset{-}{\text{regular}}$ files, $\overset{c}{\text{char}}$ special files (serial I/O devices), $\overset{b}{\text{block}}$ special devices (disks)

Links

Hardlink: dir entry that maps a name with the i-node of an existing file → cannot be created for dirs

Symbolic: file that points to a file/dir but doesn't mirror the other's file data → different i-node

Tmpfs

file system that stores files in RAM. → use for fast access to non-critical data

Logical volume management - LVM

Copy on Write: Optimierungsmethode zur Vermeidung unnötiger Kopien. Kopie wird erst dann erstellt, wenn file beschrieben wird.

Snapshot: read-only copy of the file system state

B-tree file system / RAID

Disks sind unzuverlässig → mit mehreren Disks einen zuverlässigen Speicher realisieren

RAID 0: Daten auf 2 HD splitten und in beiden nur die Hälfte reinschreiben → + 2x schneller - Datenverlust

RAID 1: Daten auf 2 HD gespiegelt schreiben → + Datensicherheit - Effizienz

RAID 10: RAID 0, aber man spiegelt jede Hälfte auf 2 HDDs wie in RAID 1 → + 2x schneller, Ausfallsicher für 2 HDD, - teuer $\leftarrow$ 4 HDD

Networking (0-4: Kernel ; 5-7: User)

Requirements: Local IPC, Protect machine from malicious traffic, Route traffic to different network interfaces

L1: connect physical network interface card to rest of the system via system-bus (kernel-space)

L2: Frame from network interface → memory, Bridging

L3: firewall, routing → IP packet sanity checks

L7: API to tap into OS networking stack (Berkeley)

Linux Commands

System info

kernel version: `uname -a` cores: `lscpu`

memory: `free -k` manufacturer + feat.: `lshw`

PCI devices: `lspci` (`-k` for kernel drivers)

USB devices: `lsusb` IO stats: `iostat (-dev /dev/...)` or `iotop`

Booting

available target units: `systemctl list-units --type target`

default target: `systemctl get-default`

default target unit file: `cat /etc/systemd/system/default.target`

recursively list deps.: `systemctl list-dependencies default.target`

service status: `systemctl status ....service` or `journalctl -u ..`

see unit file: `systemctl cat ssh.service` ⇒ requires: units this depends on, started when this unit is started

Wants: weaker than "requires" → listed units are started, curr unit not stopped if any of them fail ; immediately enable & start service: `sudo systemctl enable --now my-service.service`

Processes (PID 0 = child, -1 = error, else parent)

current PID: `pidof tail` procs hierarchy: `pstree -p PID`

running procs: `ps xawf -eo pid,user,cgroup,args` monitor syslog: `tail -f /var/log/syslog` proc to PID: `ps PID` kernel threads: `ps -eo pid,ppid,cmd,state | grep [k]worker`

Scheduling

change prio: `nice -n [value] [executable]` change niceness of running proc: `renice -n [value] -p [PID]` scheduler of a proc: `chrt .. -p PID`

cgroups

display subsystems: `ls /sys/fs/cgroup` hierarchies: `systemd-cgls`

disable CPU 2 & 3: `sudo chcpu -d 2-3` create cgroup: `mkdir /sys/fs/cgroup/cpu/newgroup` add proc to cgroup: `PID > /sys/fs/cgroup/cpu/newgroup/cgroup.procs`

limit proc memory: `echo 2M > /sys/fs/cgroup/memory/.../memory.limit_in_bytes`

mounting: `mkdir /cgroups/cpu` → `mount -t tmpfs -o size=2G tmpfs /cgroups/cpu` → `mount -t cgroup -o cpu none /cgroup/cpu`

Files

`ls -al ...` ⇒ (owner/group/others), Links, Owner/Group, size, ...

permissions: `chmod u+rwx file` (user/group/others) ; change group: `chown user:group path` partitions: `fdisk -l` blk devices: `lsblk`

soft/hard link: `ln (-s) file link` LVM: prepare disk: `pvcreate /dev/vdb1`

create volume group: `vgcreate NAME1 /dev/vdb1 /dev/vdb2`

new volume: `lvcreate -L 1G -n NAME2 NAME1`

volume manipulation: $\overset{\text{lvreduce}}{\text{lvextend}}$ `-resizefs -L 800M /dev/NAME1/NAME2`

Kernel

load/remove module: $\overset{\text{is}}{\text{rmmod}}$ filename

Threads