

01 Intro

only 45 years of experience

Mechanism

Policy

02 System Overview

Layer: User / Apps / Tasks / Threads / Scheduler-Dispatcher /// CPU(s) / Memory / (DMA) / Devices / Filez

Hardware elements: Instruction Pointer (Program Counter), Interrupts, Adress spaces

Wishes: Understandable, Explainable, Correct, Robust, Reusable, Efficient, Secure, Structured, Scalable, Modularized **more wishes:** Useful & Extendible, Quality of Service, Correct & secure

Patterns: generic-fast-simple and customizable-complicated-slower /// simple = general

System dependencies: Calling, Serializing constraints, Communicating, Master/Slave, Peers, Client/Server

Kernel: IPC, Thread creation & killing

Monolithic	Mü-Kernel
Apps vs. Kernel	Kernel: Dispatcher & Basic IPC Apps & Services: Device Driver, File System, Memory Allocation, etc..

Bottlenecks and Timing (Sec, ms, mü-s, ns) Think about others in round-robin / KGV-Trick

03 Hardware

Stack

Stack starts at ffffffff (lowest on a sheet of paper) and goes up to 0 (on top of the paper).

Return address / old frame pointer / local values / parameter for caller /// ??? / ???

Parameter passing

Register: Agree on register usage

Register/mem: Non-reentrant code

Heap: caller "where is my data?"

IRQ

Polling

Heavy CPU-load, keyboard

Interrupt driven I/O

Medium load: mouse

DMA

Low CPU-Usage, MemBus still flooded, Multimedia

IRQ Parade

Nested / Hierarchical handling

+ priorities ok

- low prio needs long

Serial handling

+ simple

- time critical!

Interrupt Masking

- can kill the system

Cache

Write-Through = Sofort durchschreiben (race conditions?)

Write-Back = Achtung bei mehreren Memory-Layern: L1 cache ganz nach unten durchschreiben bedeutet erstmal L2 ganz runter etc...

04 Threads 9x

General keep-in-mind: multiprocessor, blocking, buffered, mode switches

TCB (Instanzenleitblock) /// **Management:** TCBs linked, queues for each state

Thread States: [new] ready - running - blocked (on event i) - [exit] /// swapped out

Kernel level threads	User level threads (coroutine)
Multiple scheduling policies possible Only single threads are blocked, app more lively Generic TCB quite big	Multiprocessors unused Mapping user level threads to kernel level threads (blanced)

Switch if:

Sync. = Exception = Internal behaviour

termination / I/O wait / wait for (server)thread(message) / sleep() / page fault

Async. = Interrupts = System policy

a) end of time slice

b) high prio thread by: interrupt handling, no longer sleep, creation

Switch: User mode To - Kernel mode - User Mode Tn

Thread Lib: Create & destroy / message & data passing / scheduling / save & restore context

Race conditions: load, calc, store are atomic

Need for IPC because of these calssical problems:

n-Producer/m-Consumer (bounded buffer) Problem / Barbershop Problem / Dining Philosophers / Reader/Writer Problem / Cigarette Smoker Problem / Monkey Rock Problem

Requirements for scheduling solution:

Mutual Exclusion: at most one can enter CS

Progress Threads in RS are not scheduled, Cs waiting preferred "soon"

Bounded Waiting no one gets tricked (starvation)

IPC

Solutions via: flags & turn (Peterson's) // **busy waiting**

	Synchronization Special role for last mover	Mutual Exclusion
Cooperative Scheduling Via yield(){ save old, switch sp, load new } Stack: Local Vars To / Paramater Tn / return adress Tn / Local Vars of yield		
Signals Block(T) & unblock(T) => Signal() & wait()		
Semaphore (counting / binary) P wait(s) proberen V signal(s) verhoegen (increment)	... wait(s) /// other thread: signal(s), maybe two semas Main s=0	... wait(s) CR signal(s) ... main: s=1
Machine instructions Working on multiprocessors systems Test & Set (i) = spin lock Exchange (reg,mem)	High prio can lead to starvation of low prio threads => deadlocks if dependency	
Monitors =Igloo Interface / Init / local private data	Cwait() & csignal()	
Shared Memory No practicable because of admin.		

<p>Messages <i>Tricks: Training packets to synchronize data flow / Timeouts / via register, reference, value, mapping</i> Synchronization / Direct/Indirect Addressing / Communication Objects to System/Partners / Ownership of Communication / Organization of Data Transfer / Format of Messages / Buffering / Internal Scheduling</p>	<p>"go!" or not</p>	<p>"mayproduce!", "mayconsume!"</p>
--	---------------------	--

Deadlock / Starvation = Livelock

Incorrect program & race cond. => deadlock

Necessary Cond:

Exclusiveness (of resource)

Hold and Wait

No Preemption (no take away) ANTI: virtualization & multiplexing

Circular Wait ANTI: **linear order**

Deadlock Avoidance:

Resource vector, Vrei Vector

Claimed matrix, Allocated matrix (thread\res)

Invariants:

All resources are either available or allocated

no thread claims more than the total amount of resources in the system

no thread is allocated more resources than initially claimed

Safe System:

A possible execution sequence with Banker Alg. In $O(n^2m)$

Transactions

BEGIN_TRANSACTION

END_TRANSACTION

return "COMMIT" or "ABORT"

remarks: on highest com level for correctness, on lower levels for performance

used for: replicated servers, distributed apps

ACID-Concept

A tomicity = All or Nothing Concept

Atomic

C onsistency = State(t_i) State(t_{i+1})

Consistency by Data Base User

I solation (serializability)= hide intermediate results

Isolation = like threads, [interleaving]

D urability = persistant result if committed

Durability

Isolation feature leads to: modularity, extendibility, software design help, conditions hold

Problems: concurrency, undo faulty transactions

Implementation:

Transaction

→ *Transaction manager:* start, read & write, result: abort & commit

→ *Recovery Manager:* read & write, abort & commit → Buffer Manager

→ *Data Base:* read & write

Transaction States: Active → part. Done → more done → complete, can always undo

Conflicts

Dirty read => rollback if dirty read aborts // dirty write

Commutative transactions can be exchanged => conflict serializable

Detecting conflicting & commutative transactions

Conservative: locking
Optimistic: timestamps & abort

Locking

+locking granularity → more concurrency & more deadlocking danger

Two Phase Locking Protocol (acquire locks, release locks)

- 1.&2.** Wanna read/write? Get a read/write lock first! Conservative: get all locks at begin
- 3.** RW, WR, WW conflict => block transaction
- 4.** Can not acquire a lock after releasing one => no "starvation"
→ transaction serialization
- 5. strict:** Hold all locks till end of transaction → **recoverable**

Logging (A,D)

<Ti, start>

...

<Ti TID of the transaction, data object, value (old, new)> Before each write
for performance reasons some: <checkpoint>s

...

<Ti, commit> => redo after crash

=> undo(Ti) & redo(Ti) idempotent

Caching

Cell = cached disk block

Buffer directory: data item → cell number

Buffer: cell number, dirty bit, content

Flush(c), Fetch(x), Pin(c), Unpin(c)

Shadowing gurantees atomic commit

05 Memory Management

LAS - Logical Adress Scope (adress withd of CPU) access by task => Protection!

8 Bit => 256 Byte / 16 Bit => 640 KB / 32 Bit => 4 GB / 50 Bit => 1 PetaByte

Requirements: Relocation, Protection, Sharing, Logical Organization, Physical Organization

Contiguous Adress Space (adress range aka segment) vs. Dispersed Adress Space

Mapping Problems

Fragmentation:

internal = in blocks => Garbage Collection

external = between blocks => Compaction

Mapping information

logical address space in a paging system needs a page table for MMU

Page table + entry for a 16 Bit Processor with MMU with
up to 64 KB main memory (frame size = 1 KB)

Single threaded address space sizes up to 16 KB
have to be mapped completely.

Code should begin at logical Address 0, stack (size < 2 KB) must
reside at the highest logical addresses

Addressing errors within the address space must be detected.

Address Scope versus Address Space

Mapping Possibilities, orthogonal

Complete Contiguous LAS to ...

.... Contiguous Memory Partition (without direct mapping)

Complete Non Contiguous LAS to...

.... Contiguous Memory Partition (specific simple segmentation, rarely implemented)

.... Non Contiguous Memory Partition (Simple Segmentation)

Variable sized Portions of Non Contiguous LAS to ...

... Non Contiguous Memory Partition (Overlay or VM Segmentation)

Fixed sized Portions of Non Contiguous LAS to ...

... Non Contiguous Memory Partition (VM Paging)

Memory Management

Sequence of allocate/release-operations

FIFO, LIFO, arbitrary

Size of memory portions

Identical size, fixed size, exponential size, arbitrary

Management data structures

Integrated, special memory portion(s)

Allocating policy

First-[works better than best fit], Next-, Best-, Worst-, ..., Nearest-Fit

Reunification of Released Portions

Immediate, lazy reunification = wait for a proper moment for reunification

Boundary Tag System	Buddy System
Linked list of blocks	Array of heads pointing to free blocks of equal size internal frag.is ~ 25% / blocks at least 50% occupied
Arbitrary sized portions Integrated management data structures External fragmentation Immediate reunification	Allocated portions of $2^0, 2^1, 2^2, 2^3, \dots$ Explicit management data structures Internal + external fragmentation Immediate (or lazy) reunification
Operations in arbitrary order Allocation according to Best-Fit	

06 Virtual Memory 2x

Segmente vs. Seiten

Page fault - thread request page in mem which is only virtual

Collided page wait (accessed by other thread)

Free page wait

Page/Segment-System

Page#Offset=(TLB)=> Frame#Offset = Adress

Prinzip: Page# lookup in page table (nested?)

Cache-System

Adress = Tag | Rest => Cache or Main Memory

Special Bits:

Present Bit
Modified Bit[^]
Defined Bit (page belongs to task)
Pinned Bit
Access Rights (read,write)
Protection level (user/kernel)

Access:

Fetch page table entry, calc cell adress
Fetch mem cell
TLB: cell adress cache

Page Tables

Nested Page Tables
Inverted Page Tables

Page Size

Larger =>
+ enforces transfer of large blocks
+ more TLB hits due to less pages
Smaller =>
- more pages required (larger page tables)
+ less internal fragmentation

Page fault

Glockenkurve
Frames per Task (log-Kurve)
Page Buffering: examine free & modified free list (buffered write to disk), set present or exchange

Policies:

demand paging, pre-paging / demand cleaning, pre-cleaning (pages will be modified again!)

Allocation Strategies

fixed/variable / local or global replacement
working set strategy => page fault frequency strategy

Load Control

Clever: adjust multiprogramming so, that page faults occur at highest handable rate

Split page fault handler: page fault handler & pager thread

07 Scheduling 2x

Special **Scheduling policies** for each level of execution

Objectives?

Single-/multi-processor system
homogenous or heterogeneous multi-processor system,
e.g. same speed and/or same instruction set or not
Static or dynamic set of executable units (i.e. threads, tasks, etc.)
On-line or off-line scheduling
Known or unknown execution times
With or without preemption
With or without precedence relations
Do we have to pay regard on communication costs or not
With or without priorities
With or without deadlines
What's the objective (response time, turnaround time, throughput, etc.)

Common Objectives:

Length of the schedule, make span, maximal turnaround time
Maximal response time
Weighted medium turnaround time
Medium response time
Minimal number of involved CPUs
Throughput
CPU Utilization
Maximal Lateness or Tardiness

CPU-Scheduling:

Long-term: which thread to admit, mix CPU & io-bound threads
Medium-term: swapping (activate or deactivate) & allocation & load control
Short-term: = *dispatcher* which ready thread to execute next, on event

Dispatching

Multiple ready queues
Dispatcher chooses from highest prio if not empty
Thread gets higher prio as it ages => prevent starvation

Selection function - which thread?

Decision mode - when switch? (preemptive / non-preemptive)

"jobs" need lots of cpu time

Scheduling Policies

FCFS = FIFO

- high response time, penalize short processes

LCFS

Round-Robin = time slicing / preemptive

- low throughput if quantum is too small /
- poor use of i/o → virtual round robin

SPN shortest process next / How long runs a process???

- high overhead, penalize long processes, possible starvation

→ **SRT shortest remaining time** / preemptive

- high overhead, penalize long processes, possible starvation

→ **HRRN highest response ratio next / using age / RR = $w+s/s$**

- high overhead, penalize long processes

Feedback = move down cpu-junkies / preemptive / multilevel prio with 2^i time slices

- may favor io-bound processes, possible starvation

cpu junkies prevent io-user from completion => give i/o-threads higher prio

Multiprocessor and Real-Time-Scheduling

tightly coupled multiprocessors share main mem (but not the caches), one OS
Master/Slave vs. Symmetric "Clan"

Interrupts handled on: invoker, cpu in kernel mode due to int., idle cpu

Caches ←→ Scheduling on CPUs

Probs: Precedence relations / com. Costs

Anon queue, queue per cpu, choose always highest prio or matching thread

Five Characteristics: Determinism (time to int) / Responsiveness (serve int) / User control / Reliability / Fail-soft operation

Solutions: short term task scheduler, very responsive

#####

08 I/O Management

cycle stealing

DMA & interrupt breakpoints

Generality & Efficiency

No / Single / Double / Circular Buffer

Disk

Seek time, rotational delay => Access time

Access Strategies

First in, first out (FIFO)

Priority

LIFO → little arm movement

Shortest service time first

SCAN = Arm hin und her bewegen

C-SCAN = one way arm reading/writing ?stupid!?

N-Step SCAN = range queues

Raid

Disk Buffer in Mem using LRU / LFU

09 Files

Platten: Schwenkstrategie vs. Fahrstuhlstrategie

File create, read, write, change, close, destroy [move pointer]

Access rights

File format

Organization

Locking!

inode

file name / type

owner / access rights = execution, reading, appending, updating, deletion

date of creation / last access

actual size / statistical data

links to data blocks (physical addresses)

record

blocking method: fixed blocking, variable blocking: spanned / unspanned

secondary storage management

e.g. FAT

Free space management

Bit tables, chained portions, indexing

file methods

pile, good for small data, bad for retrieving

sequential, good for fixed, bad for interactive

contiguous, chained, indexed = little fat hidden in block

indexed seq. shit

Indexed, medium

Hashed, poor ##### extensible hashing $2^{\text{generation}}$

Directory = file owned by os

Search, create, delete, list

Disk layout

Physical disk -> logical partitions -> boot block, super block, inode table and data blocks
UNIX 3 level file system !!!

10 Security & Protection 3x

Attacks:

Interruption, interception, modification, fabrication

Identification & authentication

Auditing → Intrusion detection

D.Denings proposal:

Subject -- action → object => exception resource usage, time stamp

Protection

Access matrices (ACL acc. Control lists)

Capabilities = tickets for actions

Protection via hardware (\$\$\$!), special segments, splitted segments

- Difficult to take rights away later

reference monitors

Trusted Systems

Bell-LaPadula-Model = read down, write up

11 Distributed Systems 3x

distributed ↔ connect

#####

A. Open Questions about..

Exercise 1:

Why do I have "0xfff fff0" and "value at bla bla between paramters and stack frame adress?"

Exercise 3:

Why do they calc. 900/640 ? What about 640/9?

Exercise 8b

How does this buffered interleaving works?

Exercise 15

How does this program works?

B. Vokabeln / Begriffe

Verdrängung - Swapping
Abwicklung - Prozeßablaufsteuerung
SMP - symmetric multi-processor system
Spin lock - register/mem-exchange check
Postpone - verschieben

C. Unklare Begriffe

Interne / Externe Parallelität
Wartegraph
Unsichere Betriebsmittelsituation
Boundary tag (Randkennzeichnungsverfahren)

D nicht gewußt

4 deadlock reasons
unsafe ! -> deadlock, why??
User vs. Kernel threads?
Entwurfsparameter fuer Speicher

Quickie

Wishes

Understandable, Explainable, Correct, Robust, Reusable, Efficient, Secure, Structured, Scalable,

Stack

Return address / old frame pointer / local values / parameter for caller /// ??? / ???

Threads

General keep-in-mind: multiprocessor, blocking, buffered, mode switches

Switch if:

Sync. = Exception = Internal behaviour

termination / I/O wait / wait for (server)thread(message) / sleep() / page fault

Async. = Interrupts = System policy

a) end of time slice

b) high prio thread by: interrupt handling, no longer sleep, creation

Switch: User mode T_0 - Kernel mode - User Mode T_n

Thread Lib: Create & destroy / message & data passing / scheduling / save & restore context

Requirements for scheduling solution:

Mutual Exclusion: at most one can enter CS

Progress Threads in RS are not scheduled, Cs waiting preferred "soon"

Bounded Waiting no one gets tricked (starvation)

IPC

Solutions via: flags & turn (Peterson's) // **busy waiting**

Cooperative Scheduling

Via `yield()` { save old, switch sp, load new }

Stack: Local Vars T_0 / Parameter T_n / return adress T_n / Local Vars of yield

Signals

`Block(T) & unblock(T) => Signal() & wait()`

Semaphore (counting / binary)

P wait(s) proberen

V signal(s) verhoegen (increment)

Machine instructions - Working on multiprocessors systems

Test & Set (i) = spin lock

Exchange (reg,mem)

Monitors =Igloo

Interface / Init / local private data

Messages

Tricks: Training packets to synchronize data flow / Timeouts / via register, reference, value, mapping

Synchronization / Direct/Indirect Addressing / Communication Objects to System/Partners / Ownership of

Communication / Organization of Data Transfer / Format of Messages / Buffering / Internal Scheduling

Deadlock / Starvation = Livelock

Incorrect program & race cond. => deadlock

Necessary Cond:

Exclusiveness (of resource)

Hold and Wait

No Preemption (no take away) ANTI: virtualization & multiplexing

Circular Wait ANTI: linear order

Safe System:

A possible execution sequence with Banker Alg. In $O(n^2m)$

Transactions

com: on highest com level for correctness, on lower levels for performance
used for: replicated servers, distributed apps
Isolation feature leads to: modularity, extendibility, software design help, conditions hold
Problems: concurrency, undo faulty transactions
Transaction States

Implementation:

Transaction
→ *Transaction manager*: start, read & write, result: abort & commit
→ *Recovery Manager*: read & write, abort & commit → Buffer Manager
→ *Data Base*: read & write

Conflicts

Dirty read => rollback if dirty read aborts // dirty write
Commutative transactions can be exchanged => conflict serializable

Detecting conflicting & commutative transactions

Conservative: locking
Optimistic: timestamps & abort

Locking

+locking granularity → more concurrency & more deadlocking danger

Two Phase Locking Protocol (acquire locks, release locks)

Conservative: get all locks at begin
conflict => block transaction
no acquire a after releasing one => no "starvation"
strict: Hold all locks till end of transaction → **recoverable**

Logging (A,D)

<TID of the transaction, data object, value (old, new)> Before each write
for performance reasons some: <checkpoint>s
=> undo(Ti) & redo(Ti) idempotent

Caching

Cell = cached disk block
Buffer directory: data item → cell number
Buffer: cell number, dirty bit, content
Flush(c), Fetch(x), Pin(c), Unpin(c)

Shadowing gurantees atomic commit

Memory Management

LAS - Logical Adress Scope (adress wifth of CPU) access by task => Protection!

Requirements: Relocation, Protection, Sharing, Logical Organization, Physical Organization

Fragmentation:

internal = in blocks => Garbage Collection
external = between blocks => Compaction

Paging

logical address space in a paging system needs a page table for MMU

Mapping Possibilities, orthogonal

Contiguous, Complete LAS

→ Contiguous Memory Partition (without direct mapping)

Non-Contiguous LAS...

Complete

→ Contiguous Memory Partition (specific simple segmentation, rarely implemented)

→ Non Contiguous Memory Partition (Simple Segmentation)

Variable sized Portions

→ Non Contiguous Memory Partition (Overlay or VM Segmentation)

Fixed sized Portions

→ Non Contiguous Memory Partition (VM Paging)

Orthogonal Design

Sequence of allocate/release-operations

FIFO, LIFO, arbitrary

Size of memory portions

Identical size, fixed size, exponential size, arbitrary

Management data structures

Integrated, special memory portion(s)

Allocating policy

First-[works better than best fit], Next-, Best-, Worst-, ..., Nearest-Fit

Reunification of Released Portions

Immediate, lazy reunification = wait for a proper moment for reunification

Buddy System

Array of heads pointing to free blocks of equal size
internal frag.is ~ 25% / blocks at least 50% occupied
Internal + external fragmentation

Virtual Memory

Mechanism

1. Virtual address --(Page Table)→ real address

Page#Offset=(TLB)⇒ Frame#Offset = Address

2. real address → real data

Address = Tag | Rest ⇒ Cache or Main Memory

Page Table

Special bits per entry:

Present Bit, Modified Bit, Defined Bit (page belongs to task)

Pinned Bit, Access Rights (read,write), Protection level (user/kernel)

⌘ Nested Page Tables / Inverted Page Tables

⌘ Page Size → page fault Glockenkurve, optimal point

Larger ⇒ + large blocks / more TLB hits due to less pages

Smaller ⇒ - more pages required ⇒ larger page tables / + less internal fragmentation

⌘ Allocation Strategies

fixed/variable / local or global replacement

working set strategy ⇒ *page fault frequency strategy*

frames per task → page fault log kurve

⌘ Policies:

demand paging, pre-paging

demand cleaning, pre-cleaning (pages will be modified again!)

Page Buffering

examine free & modified free list (buffered write to disk), set present or exchange

new thread problems:

Collided page wait (accessed by other thread)
Free page wait

Thread Load Control

Clever: adjust multiprogramming so, that page faults occur at highest at a **handable** rate

Implementation: page fault handler & pager thread

Scheduling

Special **Scheduling policies** for each level of execution

Orthogonal

- ⌘ **Single-/multi-processor** system (homogenous?, same speed? Instruction set?)
- ⌘ **Static or dynamic set of executable units** (i.e. threads, tasks, etc.)
- ⌘ **On-line or off-line**
- ⌘ **execution times known?**
- ⌘ **Preemption?**
- ⌘ **precedence relations?**
- ⌘ **communication costs?**
- ⌘ **Priorities?**
- ⌘ **Deadlines?**

Common Objectives:

Schedule length
Turnaround time: maximal, weighted medium
Response time: maximal, medium
Cpu usage: minimal number
Throughput
CPU Utilization
Maximal Lateness or Tardiness

CPU-Scheduling:

Long-term: which thread to admit, mix CPU & io-bound threads
Medium-term: swapping (activate or deactivate) & allocation & load control
Short-term: = *dispatcher* which ready thread to execute next, on event

Dispatching

Multiple ready queues
Dispatcher chooses from highest prio if not empty
Thread gets higher prio as it ages => prevent starvation

"jobs" = threads which need lots of cpu time

Implementation

1. **Decision mode** - when switch? (preemptive / non-preemptive)
2. **Selection function** - which thread?

Scheduling Policies

FCFS = FIFO

- high response time, penalize short processes

LCFS

Round-Robin = time slicing / preemptive

- low throughput if quantum is too small /
- poor use of i/o → virtual round robin

SPN shortest process next / How long runs a process???

- high overhead, penalize long processes, possible starvation

→ SRT shortest remaining time / preemptive

- high overhead, penalize long processes, possible starvation

→ HRRN highest response ratio next / using age / RR = w+s/s

- high overhead, penalize long processes

Feedback = move down cpu-junkies / preemptive / multilevel prio with 2^i time slices

- may favor io-bound processes, possible starvation

cpu junkies prevent io-user from completion => give i/o-threads higher prio

Multiprocessor and Real-Time-Scheduling

tightly coupled multiprocessors share main mem (but not the caches) , one OS

Master/Slave vs. Symmetric "Clan"

Interrupts handled on: invoker, cpu in kernel mode due to int., idle cpu

Caches \leftrightarrow Scheduling on CPUs

Probs: Precedence relations / com. Costs

Anon queue, queue per cpu, choose always highest prio or matching thread

Five Characteristics: Determinism (time to int) / Responsiveness (serve int) / User control / Reliability / Fail-soft operation

Solutions: short term task scheduler, very responsive

#####

08 I/O Management

cycle stealing

DMA & interrupt breakpoints

Generality & Efficiency

No / Single / Double / Circular Buffer

Disk

Seek time, rotational delay => Access time

Access Strategies

First in, first out (FIFO)

Priority

LIFO \rightarrow little arm movement

Shortest service time first

SCAN = Arm hin und her bewegen

C-SCAN = one way arm reading/writing ?stupid!?

N-Step SCAN = range queues

Raid

Disk Buffer in Mem using LRU / LFU

09 Files

Platten: Schwenkstrategie vs. Fahrstuhlstrategie

File create, read, write, change, close, destroy [move pointer]

Access rights

File format

Organization

Locking!

inode

file name / type
owner / access rights = execution, reading, appending, updating, deletion
date of creation / last access
actual size / statistical data
links to data blocks (physical addresses)

record

blocking method: fixed blocking, variable blocking: spanned / unspanned

secondary storage management

e.g. FAT

Free space management

Bit tables, chained portions, indexing

file methods

pile, good for small data, bad for retrieving

sequential , good for fixed, bad for interactive
contiguous, chained, indexed = little fat hidden in block

indexed seq. shit

Indexed, medium

Hashed, poor ##### extensible hashing 2^{generation}

Directory = file owned by os

Search, create, delete, list

Disk layout

Physical disk -> logical partitions -> boot block, super block, inode table and data blocks
UNIX 3 level file system !!!

10 Security & Protection 3x

Attacks:

Interruption, interception, modification, fabrication

Identification & authentication

Auditing → Intrusion detection

D.Denings proposal:

Subject -- action → object => exception resource usage, time stamp

Protection

Access matrices (ACL acc. Control lists)

Capabilities = tickets for actions

Protection via hardware (\$\$\$!), special segments, splitted segments
- Difficult to take rights away later

reference monitors

Trusted Systems

Bell-LaPadula-Model = read down, write up

11 Distributed Systems 3x

distributed ←→ connect

#####

A. Open Questions about..

Exercise 1:

Why do I have "0xfff fff0" and "value at bla bla between paramters and stack frame adress?"

Exercise 3:

Why do they calc. 900/640 ? What about 640/9?

Exercise 8b

How does this buffered interleaving works?

Exercise 15

How does this program works?

B. Vokabeln / Begriffe

Verdrängung - Swapping

Abwicklung - Prozeßablaufsteuerung

SMP - symmetric multi-processor system

Spin lock - register/mem-exchange check

Postpone - verschieben

C. Unklare Begriffe

Interne / Externe Parallelität

Wartegraph

Unsichere Betriebsmittelsituation

Boundary tag (Randkennzeichnungsverfahren)

D nicht gewußt

4 deadlock reasons

unsafe ! -> deadlock, why??

User vs. Kernel threads?

Entwurfparameter fuer Speicher