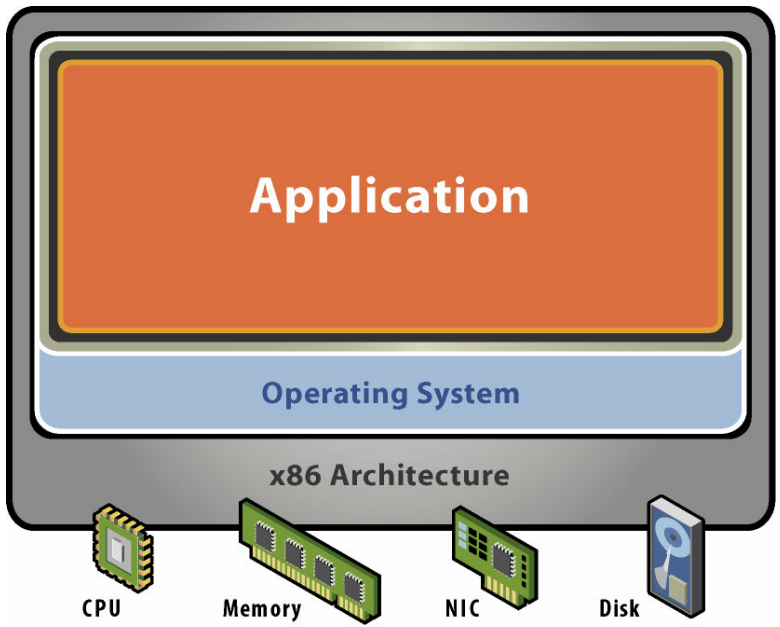
Operating Systems

NSWI170 Computer Systems

Jakub Yaghob, Martin Kruliš

Operating system – role



- Abstract machine
  - Presented by kernel API
    - System calls
    - Wrapped in C libraries
  - Hide HW complexity/diversity
- Resource manager
  - All HW managed by OS
  - Sharing HW among applications
    - Allocation (memory)
    - Time sharing (CPU)
    - Abstraction (disk, network)

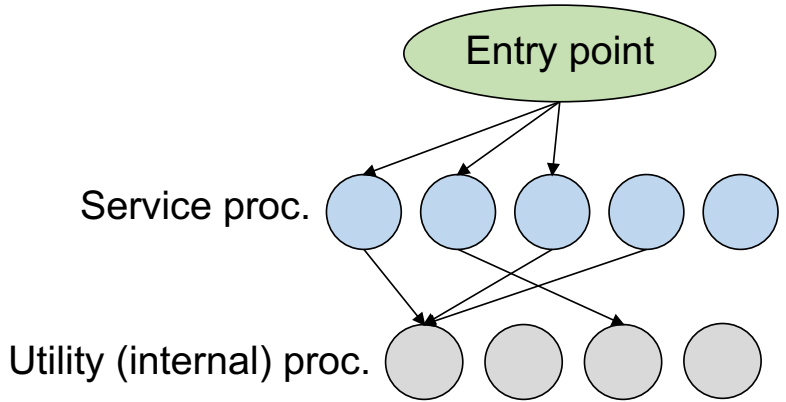
CPU modes

- User mode
  - Available to all application
  - Limited or no access to some resources
    - System registers, instructions
- Kernel (system) mode
  - More privileged (all registers and instructions are available)
  - Used by OS or by only part of OS
  - Full access to all resources
- Transition between the modes (especially user -> kernel)
  - Syscall (user instruction), jumps to explicit kernel entry point

Architecture – monolithic

- Monolithic systems
  - Big mess – no structure
  - “Early days”
    - Linux
  - Collection of procedures
    - Each one can call another one
  - No information hiding
    - Potentially error prone
  - Efficient use of resources, efficient code
  - Originally no extensibility
    - Now able to load modules dynamically (flexible, but even more error-prone)

All kernel mode



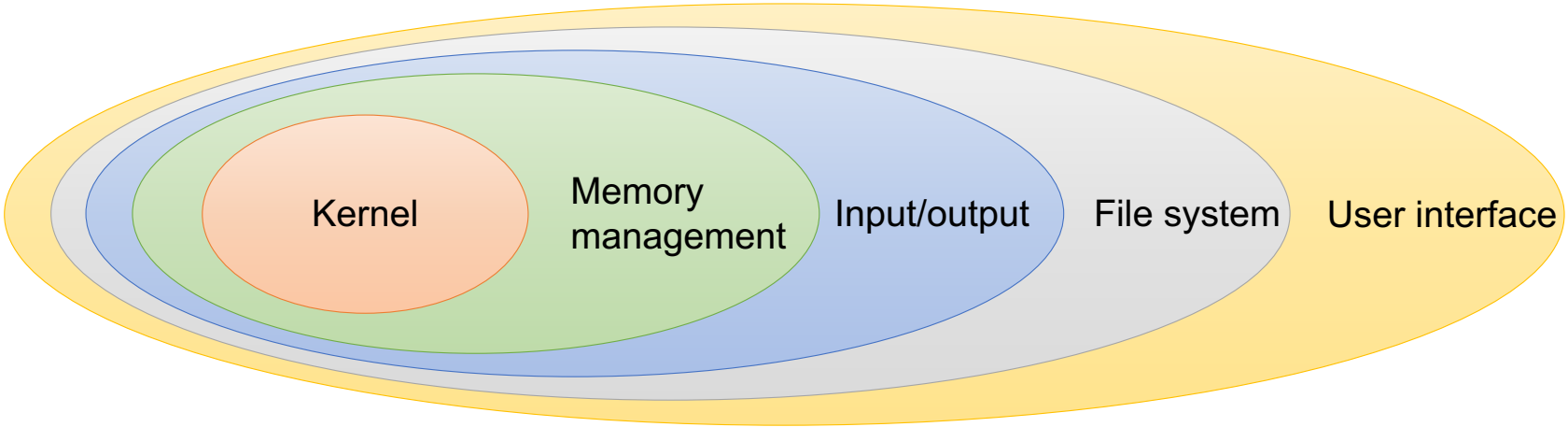
ze všech architektur je to nejrychlejší
Linux -> proto se používá
-> skrchno se to používá

(např.: se nedalo připojit žádný USB zařízení – nemělo to místo v paměti)

Architecture – layered

- Evolution of monolithic system
  - Organized into hierarchy of layers
  - Layer n+1 uses exclusively services supported by layer n
  - Easier to extend and evolve

To má zajistit bezpečnost a uzavřenost



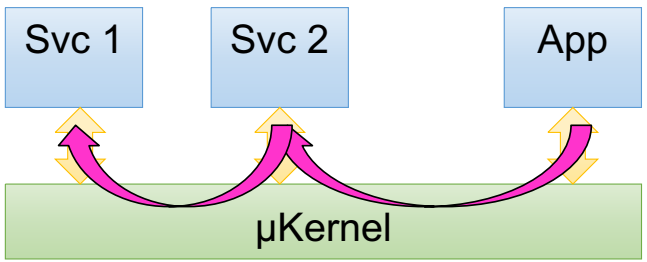
Architecture – microkernel

- Microkernel architecture
  - Move as much as possible from the kernel space to the user space
  - Communication between user modules
    - Message passing
    - Client/server
  - Extendable
  - Secure
  - Reliable

To je prakticky jediná velká věc, co dělá!

Posílají se: otázky / odpovědi

Ještě řeší nějaké low-level operace, je jich ale málo



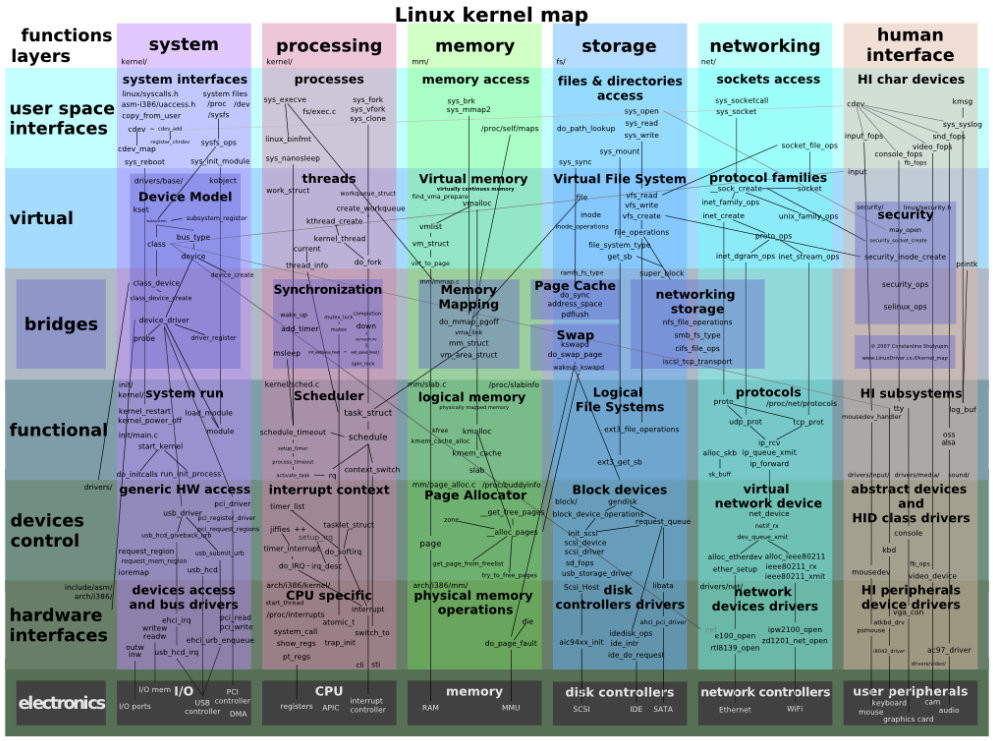
Každý servis je izolovaný

Např.: Otevření souboru zavolá jedna aplikace, µkernel to pošle jiný aplikaci, co to otevře a vrátí té původní

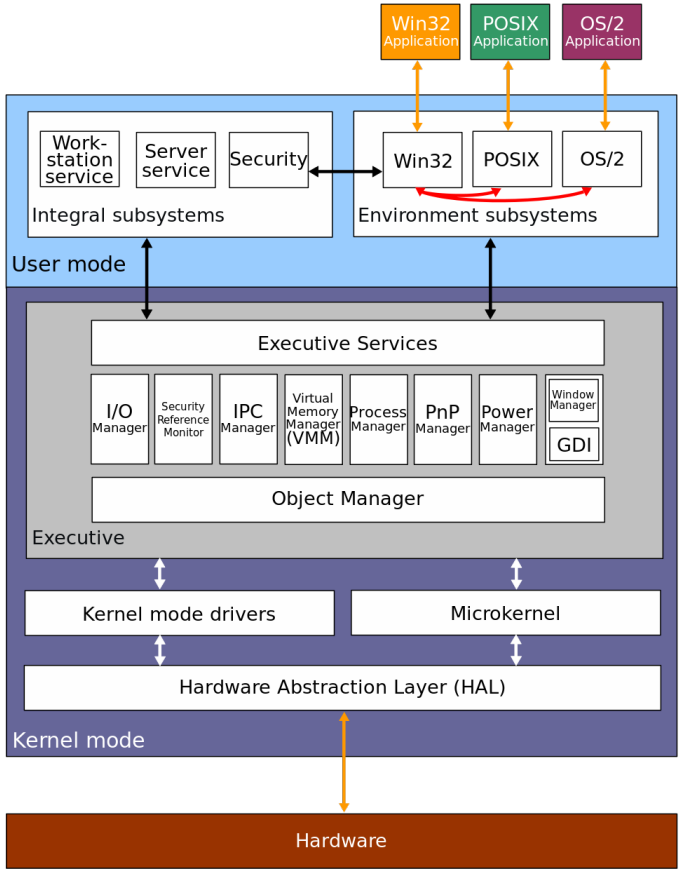
Posílání zpráv je drahé, takže je to pomalejší na úrovni komunikace

(Windows je µkernel)

Linux kernel architecture



Windows kernel architecture



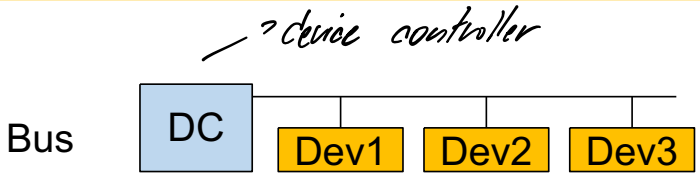
Až tohle je to, co známe jako Windows

Tohle je ale složitá práce, umí to spouštět úplně všechno možného

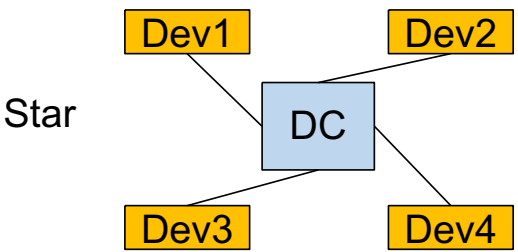
Devices

- Terminology
  - Device - “a thing made for a particular purpose”
  - Device controller
    - Handles connected devices electrically (signals “on wires”, A/D converters)
    - Devices connected in a topology
  - Device driver
    - SW component (piece of code), part of OS (module, dynamically loaded)
    - Abstract interface to the upper layer in OS
    - Specific for a controller or a class/group of controllers
  - BIOS/UEFI
    - Basic HW interfaces that allow to enumerate and initialize devices on boot

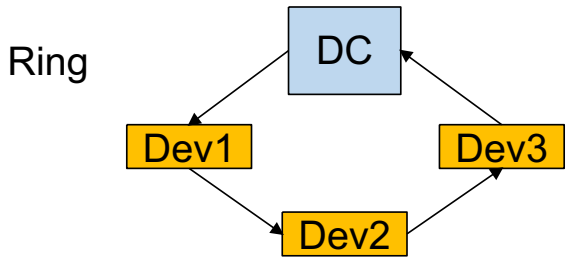
Devices topology



- Omezení kapacita sběrnice, zejména při vyšším počtu devices

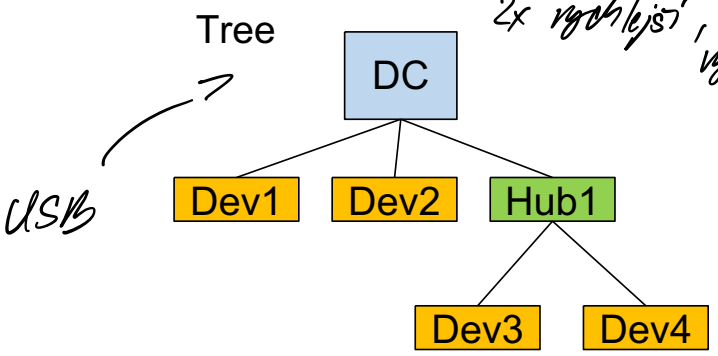


- tohle řeší kapacitu sběrnice
- řadič musí být schopný mít tolik vývodů



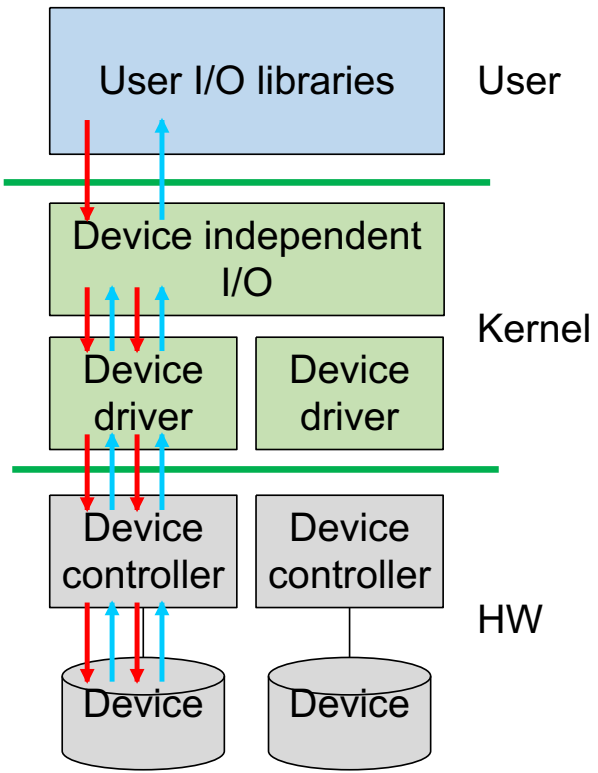
- kompromis mezi BUS / STAR

2x rychlejší, může mít vyšší latency



Device handling

1. Application issues an I/O request
2. Language library makes a system call
3. Kernel decides, which device is involved
4. Kernel starts an I/O operation using device driver
5. Device driver initiates an I/O operation on a device controller
6. Device does something
7. Device driver checks for a status of the device controller
8. When data are ready, transfer data from device to the memory
9. Return to any kernel layer and make other I/O operation fulfilling the user request
10. Return to the application



Device intercommunication

- Polling
  - CPU actively checks device status change
- Interrupt
  - Device notifies CPU that it needs attention
  - CPU interrupts current execution flow
  - IRQ handling
  - CPU has at least one pin for requesting interrupt
- DMA (Direct Memory Access)
  - Transfer data to/from a device without CPU attention
  - DMA controller
  - Scatter/gather

Interrupt types

- External
  - HW source using an IRQ pin
  - Masking
- (Hardware) Exception
  - Unexpectedly triggered by an instruction (when the instruction is completed)
  - Trap (trigger exception after) or fault (instruction rollbacks, trigger before)
  - Predefined set by CPU architecture
- Software
  - Special instruction
  - Can be used for system call mechanism

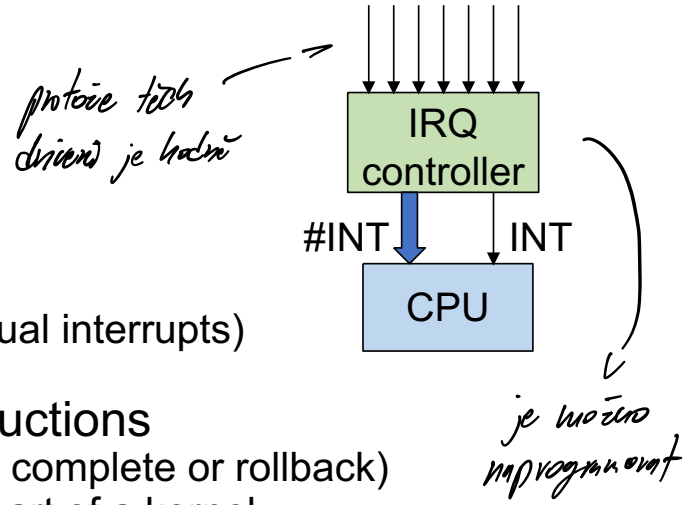
x86 Exceptions

- x86 Architecture
  - 256 interrupts
  - First 32 reserved for exceptions
  - Remaining are IRQ or SW interrupts

Interrupt	Exception description
0x00	Division by zero
0x01	Single-step interrupt (see trap flag)
0x02	NMI
0x03	Breakpoint (which benefits from the shorter 0xCC encoding of INT 3)
0x04	Overflow
0x05	Bound Range Exceeded
0x06	Invalid Opcode
0x07	Coprocessor not available
0x08	Double Fault
0x09	Coprocessor Segment Overrun *(386 or earlier only)*
0x0A	Invalid Task State Segment
0x0B	Segment not present
0x0C	Stack Segment Fault
0x0D	General Protection Fault
0x0E	Page Fault
0x0F	*reserved*
0x10	x87 Floating Point Exception
0x11	Alignment Check
0x12	Machine Check
0x13	SIMD Floating-Point Exception
0x14	Virtualization Exception
0x15	Control Protection Exception (only available with CET)

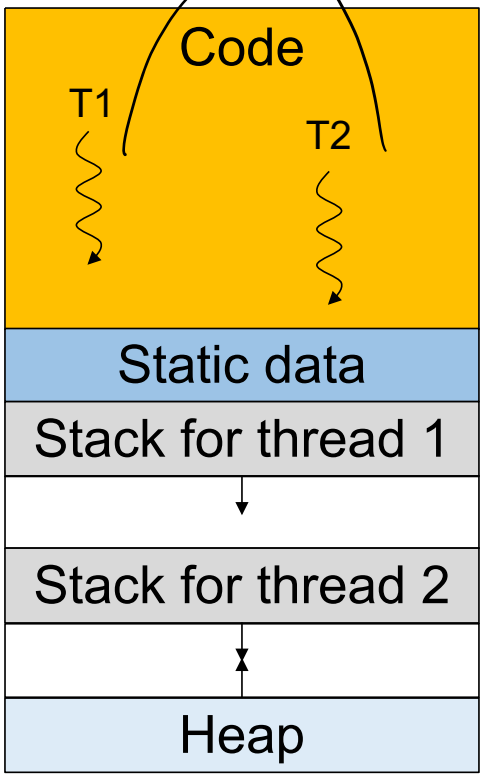
Interrupt request handling

- What happens, when an interrupt occurs?
  - CPU decides the source of the interrupt
    - Predefined
    - IRQ controller
  - CPU gets an address of interrupt handler
    - Fixed (defined by ISA)
    - Interrupt table (array of pointers of handlers for individual interrupts)
  - Current stream of instructions is interrupted, CPU begins execution of interrupt handler's instructions
    - Usually **between** instructions (current instruction must complete or rollback)
    - Privilege switch usually happens, interrupt handler is part of a kernel
  - Interrupt handler saves the CPU state
  - Interrupt handler do something useful
  - Interrupt handler restores the CPU state
  - CPU continues with original instruction stream



Processing

- Program
  - A passive set of instruction and data
  - Created by a compiler/linker
- Process
  - An instance of a program created by OS
  - Program code and data
    - Process address space
  - The program is “enlivened” by an activity
    - Instructions are executed by CPU
  - Owns other resources
- Thread
  - One activity in a process
  - Stream of instructions executed by CPU
  - Unit of kernel scheduling
- Fiber
  - Lighter unit of scheduling
  - Cooperative scheduling
    - Running fiber explicitly yields



Process vs. Thread

- Process
  - Code (loaded in memory)
  - Memory space
  - Other system resources
    - File handles
    - Network sockets
    - Synchronization primitives
    - …

- Thread
  - Position in code (program counter)
  - Own stack (rest is shared)
  - Access to some system resources may require synchronization
  - CPU state
    - Must be saved when thread is removed from CPU core and reloaded when the thread resumes

Creating a process

- Windows

```
STARTUPINFO si;
PROCESS_INFORMATION pi;
ZeroMemory(&si, sizeof(si));
si.cb = sizeof(si);
ZeroMemory(&pi, sizeof(pi));
// ...

bool ok = CreateProcess(NULL,
   cmdline,
   NULL, NULL, FALSE, 0, NULL, NULL,
   &si, &pi);
// ...

WaitForSingleObject(
   pi.hProcess, INFINITE);
```

- Linux

```
pid_t p = fork();
if (p == -1) {
   // handle error
} else if (p == 0) {

   // new process
   execv(pathToExecutable, args);

} else {

   // original process
   int status;
   waitpid(p, &status, 0);

}
```

Creating a thread

- C++ example (but C#/Java are similar)

```
void thread_code(/* params */) {
   // ...
}

int main() {
   // ...
   std::thread t(thread_code, /* params */);
   // here main thread and t run concurrently
   t.join();
   // ...
}
```

This will translate in some low-level system calls

Processing

- Scheduler
  - Part of OS
  - Uses scheduling algorithms to assign computing resources to scheduling units (CPU cores)
- Multitasking — střídání procesy v čase, aby to vypadalo, že běží „současně"
  - Concurrent executions of multiple processes
- Multiprocessing — ideálně, že vláknům zůstávají na jádře (mimo jiné kvůli cache)
  - Multiple CPUs (cores) in one system
  - More challenging for the scheduler
  - Affinity

Processing

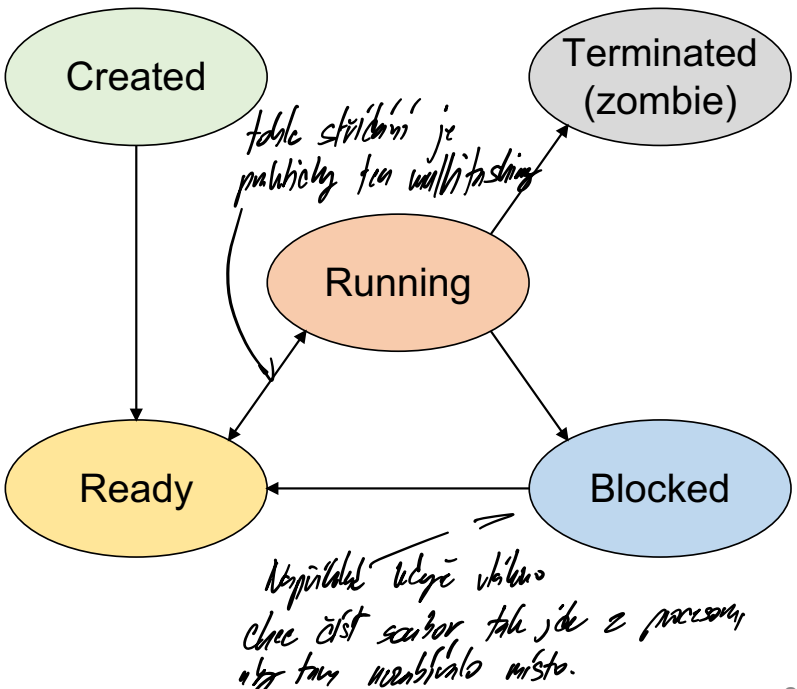
- Context
  - CPU (and possibly other) state of a scheduling unit
    - Registers (including PC, specialized vector registers)
    - Additional units (x87 coprocessor)
    - Virtual memory and address-space related context
      - Page tables, TLB (will be covered later)
      - Memory caches are transparent (not part of the context, but may affect performance)
- Context switch To dělá multitasking, switchuje mezi contexty procesů
  - Process of storing the context of a scheduling unit (on suspend) and restoring the context of another scheduling unit (on resume)
  - Quite costly (hundreds-thousands of instructions)

Real-time scheduling

- Real-time scheduling
  - RT process has a start time (release time) and a stop time (deadline)
  - Release time – time at which the process must start after some event occurred
  - Deadline – time by which the task must complete
    - Hard – no value to continue computation after the deadline
    - Soft – the value of late result diminishes with time after the deadline

Unit of scheduling state

- Created
  - Awaits admission
- Terminated
  - Until parent process waits for result
- Ready
  - Wait for scheduling
- Running
  - CPU assigned
- Blocked
  - Wait for resources



Multitasking

- Cooperative
  - Unit of scheduling must explicitly and voluntarily yield control
  - All processes must cooperate
    - Special systems
  - Scheduling in OS reduced on starting the process and making context switch after the yield
    - OS does not initiate context switch
- Preemptive
  - Each running unit of scheduling has assigned **time-slice**
  - OS needs some external source of **interrupt** (HW timer)
  - If the unit of scheduling blocks or is terminated before the time -slice ends, nothing of interest happens
  - If the unit of scheduling consumes the whole time-slice
    - interrupted by the external source
    - changed to READY state
    - OS will make context switch

Scheduling

- Objectives
  - Maximize/optimize CPU utilization (based on workload)
  - Fair allocation of CPU
  - Maximize throughput
    - Number of processes that complete their execution per time unit
  - Minimize turnaround time
    - Time taken by a process to finish
  - Minimize waiting time
    - Time a process waits in READY state
  - Minimize response time
    - Time to response for interactive applications

Scheduling – priority

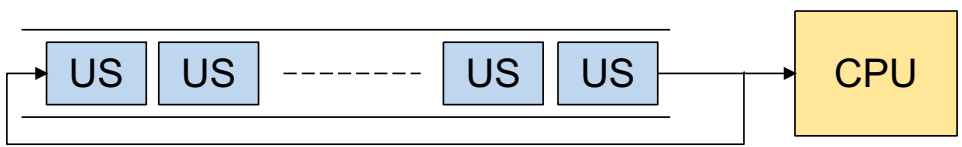
- Priority
  - A number expressing the importance of the process
  - Unit of scheduling with greater priority should be scheduled before (or more often than) unit of scheduling with lower priority
  - The priority of the process is the sum of a static priority and dynamic priority
    - Static priority
      - Assigned at the start of the process
        - Users hierarchy or importance
    - Dynamic priority – tohle je opatření, aby se mi něco nezamrzlo
      - Adding fairness to the scheduling
      - Once in a time the dynamic priority is increased for all READY units of scheduling
      - The dynamic priority is initialized to 0 and is reset to 0 after the unit of scheduling is scheduled for execution

Scheduling algorithms – non-preemptive

- First Come, First Serve (FCFS)
  - Single FIFO queue
  - Process enters the queue on the tail, the head process is running on CPU
  - Afterwards, there is removed from the queue
- Shortest Job First
  - Maximizes throughput
  - Expected job execution time must be known in advance
- Longest Job first

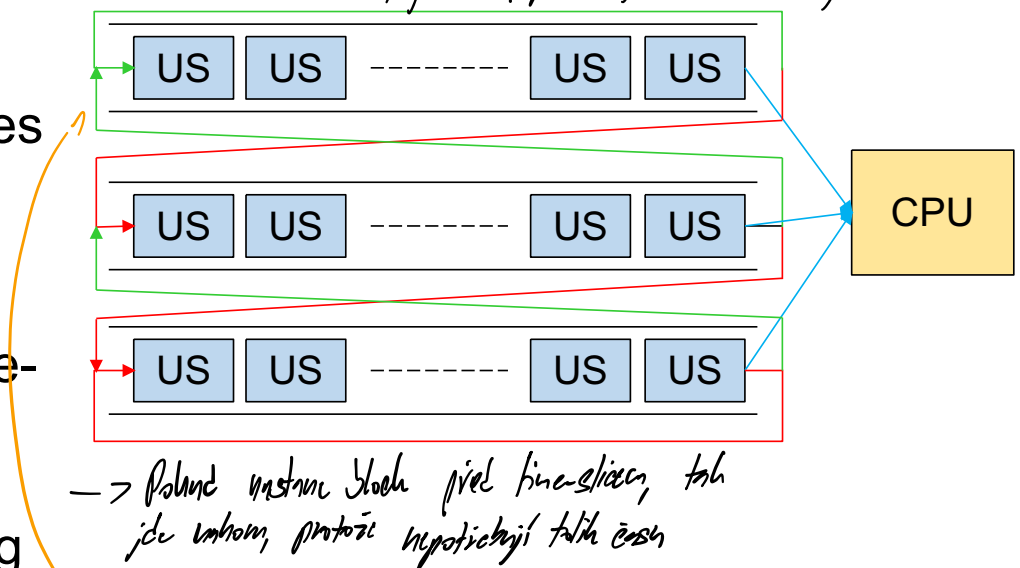
Scheduling algorithms – preemptive

- Round Robin
  - Like FCFS (but preemptive)
  - Single queue
  - Each unit of scheduling has assigned time-slice
  - If the unit of scheduling consumes whole time-slice or is blocked, it will be assigned to the tail of the queue



Scheduling algorithms – preemptive

- Multilevel feedback-queue
  - Multiple queues
    - Time-slice defined by queue (increasing)
  - If the unit of scheduling consumes the whole time-slice, it will be assigned to the lower queue
  - If the unit of scheduling blocks before consuming the whole time-slice, it will be assigned to the higher queue
  - Schedule head unit of scheduling from the highest non-empty queue



Scheduling algorithms - preemptive

- Completely fair scheduler (CFS)
  - Implemented in Linux kernel
    - Currently the default scheduler
  - SUs are stored in red-black tree
    - Indexed by their total execution time (called *virtual runtime*)
  - *Maximum execution time*
    - A time-slice calculated for each unit
    - Total waiting time divided by the current number of processes
      - The longer it waits, the greater
- Scheduling algorithm
  - The leftmost node in RB tree is selected (lowest execution time)
  - If the process completes its execution, it is removed from scheduling
  - If the process reaches its *maximum execution time* or is somehow stopped or interrupted, it is reinserted into the tree with new execution time
    - Actual time spent on CPU is added to virtual runtime
  - Repeat until the tree is empty

File

- File
  - Data organization unit
  - Collection of related information
    - Abstract stream of data (bytes)
      - Kernel does not understand file formats
    - Typically stored on secondary storage, but there are other possibilities
  - File identification
    - System uses numeric identifiers
    - File name and path – a named reference to the file identifier in organized tree structure
      - So that humans can find the files
      - Some parts of the file name may have special meaning (leading dot, extension)

File operations

- Std. lib in C

```
#include <stdio.h>

FILE *fp = fopen("file.txt", "r");
if (!fp) { /* error */ }

fseek(fp, 42, SEEK_SET);
fgets(buf, 16, fp);

fclose(fp);
```

- POSIX

```
#include <unistd.h>

int fd = open("file.txt", O_RDONLY);
if (fd == -1) { /* error */ }

lseek(fd, 42, SEEK_SET);
read(fd, buf, 16);

close(fd);
```

File operations

- Additional operations
  - Create, truncate, delete, flush, change attribu
- File handle
  - Process-specific sequentially assigned, kernel holds translation table
- Buffering
  - To increase performance, multiple levels (system, language runtime)
  - Sequential vs random access
- Alternatives
  - Memory mapping (will become more clear after memory management)
  - Async file I/O

It is no coincidence that **stdin**, **stdout**, and **stderr** file descriptors have handles 0, 1, and 2.

File attributes

```
$> stat ./smart.py
  File: ./smart.py
  Size: 11534           Blocks: 24         IO Block: 4096   regular file
Device: fd05h/64773d    Inode: 120575467360  Links: 1
Access: (0664/-rw-rw-r--)  Uid: (29345/krulm3am)   Gid: (29345/krulm3am)
Access: 2021-05-17 18:25:06.734717566 +0200
Modify: 2021-05-16 15:58:16.000000000 +0200
Change: 2021-08-04 16:11:32.853190394 +0200
 Birth: -
```

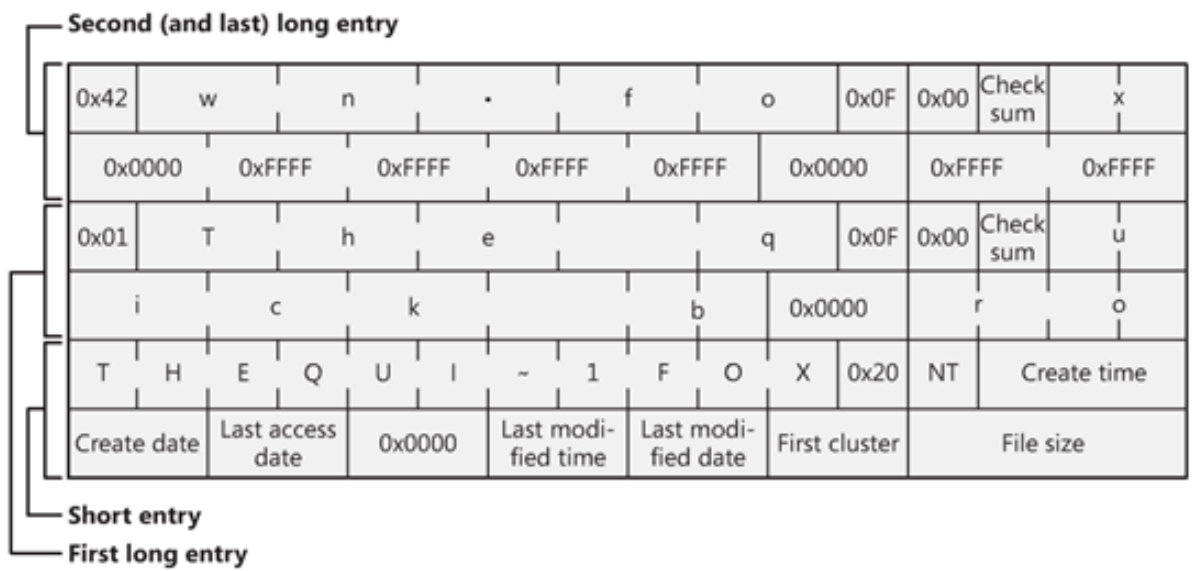
File directory

- Directory
  - Collection (list) of file entries
    - Efficiency – a file can be located more quickly
    - Naming – better navigation for users
    - Grouping – logical grouping of files
  - Usually represented as a file of a special type
  - Store (some of) the file attributes
  - Hierarchy or structure
    - Root
  - Operations
    - Create/delete/rename file/subdirectory
    - Search for a name
    - List members

Moving file on the same file system
=
moving only file entry between directories

File directory example

- FAT directory entry
  - File name “The quick brown fox”



https://social.technet.microsoft.com/wiki/contents/articles/6771.the-fat-file-system.aspx

File storage

- Traditional storage
  - File system in secondary or external storage (persisted)
  - File system in RAM (e.g., for temporary files)
- Network storage
  - Protocols for performing FS operations remotely over network
- Virtual (system) files
  - Using file abstraction to provide additional (system) features
    /dev/null
    /dev/urandom
    /proc/cpuinfo

File links

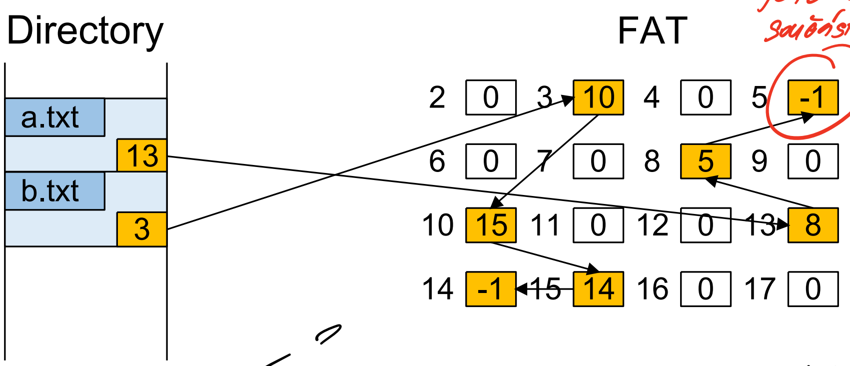
- Links (hard links)
  - Multiple directory entries refer to the same physical file (same file ID)
  - Most operations are transparent (no special handling required)
  - Saves space (in some situations), creates additional problems
    - E.g., file deletion should not always remove the file data
- Symlinks (soft links)
  - Special files, text content holds path to another file
    - Does not refer to file IDs
  - Requires special handling in path processing (“follow symlinks”)
    - Often hidden in basic system tools or programming runtime libraries

File system

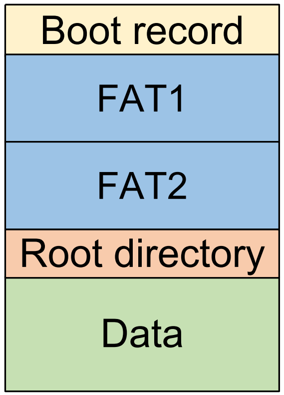
- File system
  - How and where data are stored
    - Formats, protocols
  - Implementation of an abstraction for files and directories
  - Responsibility
    - Name translation (directory format)
    - Data blocks management
      - Allocated vs. free blocks
      - Bitmap, linked list, B-tree, …
    - File data management
      - Sequence of data blocks
- Local file system
  - Stored on HDD, SSD, removable media
  - FAT, NTFS, ext234, XFS, …
- Network file system
  - Protocol for accessing files and directories over the network
  - NFS, CIFS/SMB, …

FAT

- File Allocation Table (FAT)
  - Simple, old, MS-DOS, many variants used today
  - One structure (FAT) for managing free blocks and file data location
  - Directory
    - Sequence of entries with fixed size and attributes
      - Starting cluster, name+ext, size, timestamps, attributes
    - Root in fixed position



Tohle už není součástí velikosti souborem



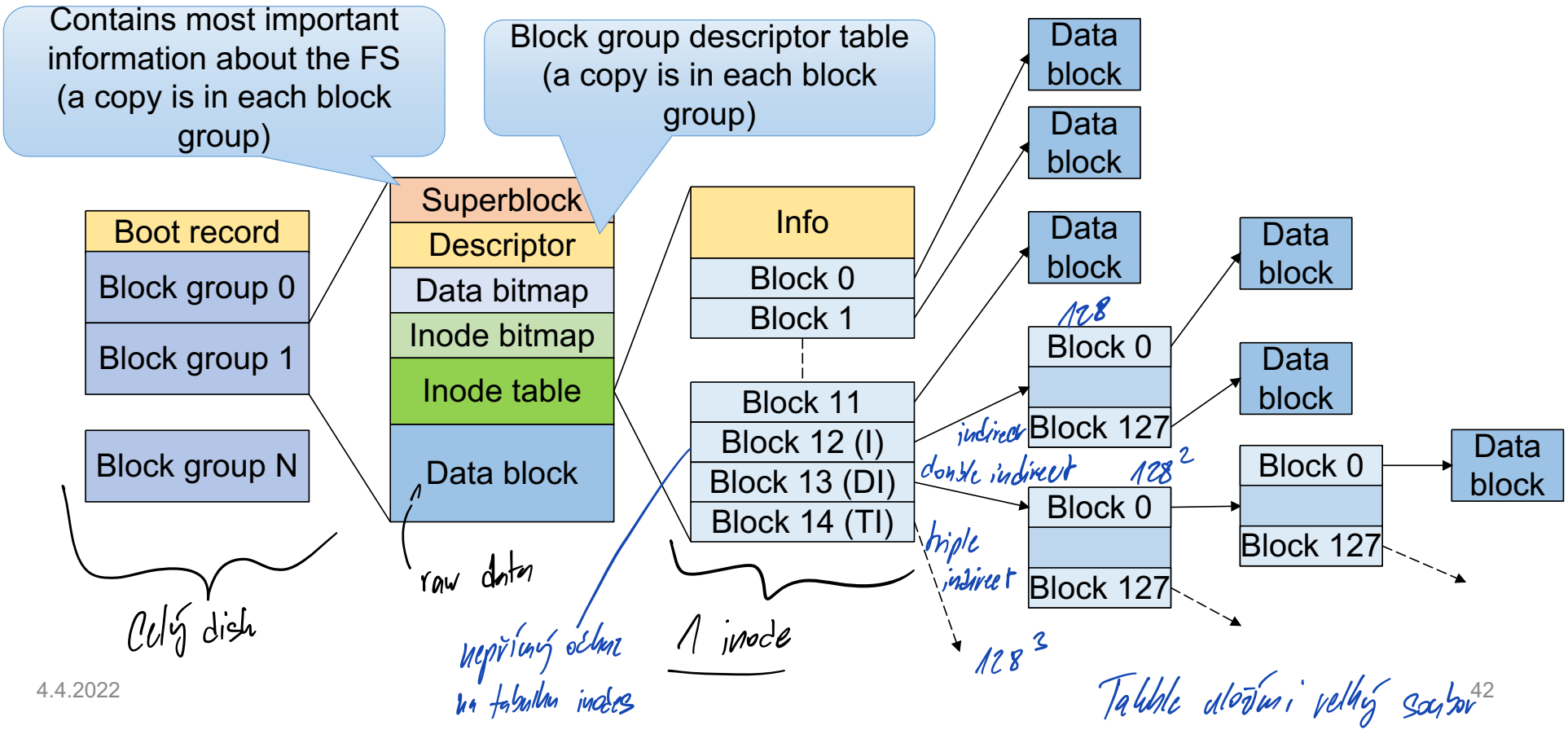
Mám dvě kopie, takže když to spadne uprostřed zápisu, tak mám zálohu

Funguje to jako mapa k blokům na disku

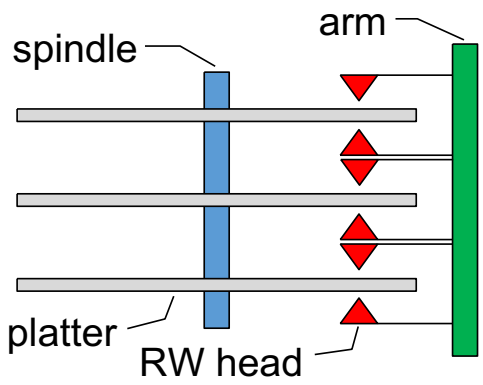
Ext2

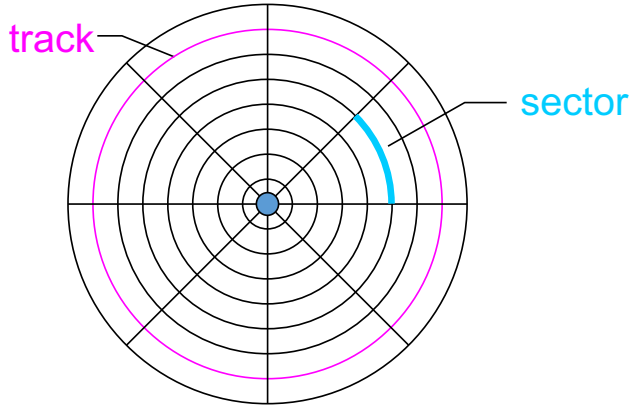
- Second extended file system (ext2)
  - Simple, old, Linux
    - Ext3 – added journal to improve persistence
    - Ext4 – improvement of ext3, larger individual files (16T) and FS (1 EB)
  - Inode (index node)
    - Represents one file/directory
    - Tree-like hierarchy with block references (faster than linked list)
      - Smaller files are represented more efficiently
    - Holds most of the attributes
  - Directory
    - Sequence of entries with fixed structure
      - Inode number, file name

Ext2


Contains most important information about the FS (a copy is in each block group)
Block group descriptor table (a copy is in each block group)
Boot record
Block group 0
Block group 1
Block group N
Superblock
Descriptor
Data bitmap
Inode bitmap
Inode table
Data block
Info
Block 0
Block 1
Block 11
Block 12 (I)
Block 13 (DI)
Block 14 (TI)
Data block
Data block
Data block
128
Block 0
Block 127
Data block
Data block
indirect
double indirect
128 2
Block 0
Block 127
Block 0
Block 127
Data block
triple indirect
128 3
raw data
Celý disk
nepřímý odkaz na tabulku indexů
1 inode
Takhle uložíme velký soubor

Hard disc mechanics





- Additional terminology
  - Block – the same sector on all platters
  - Cluster – the same track on all platters
  - Flying height – distance between head and platter (~5 nm)
  - Rotational speed – 5400, 7200, 10k, 15k rpm

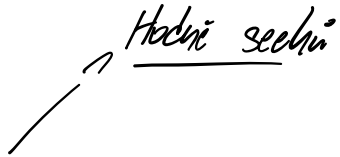
Disk scheduling algorithms

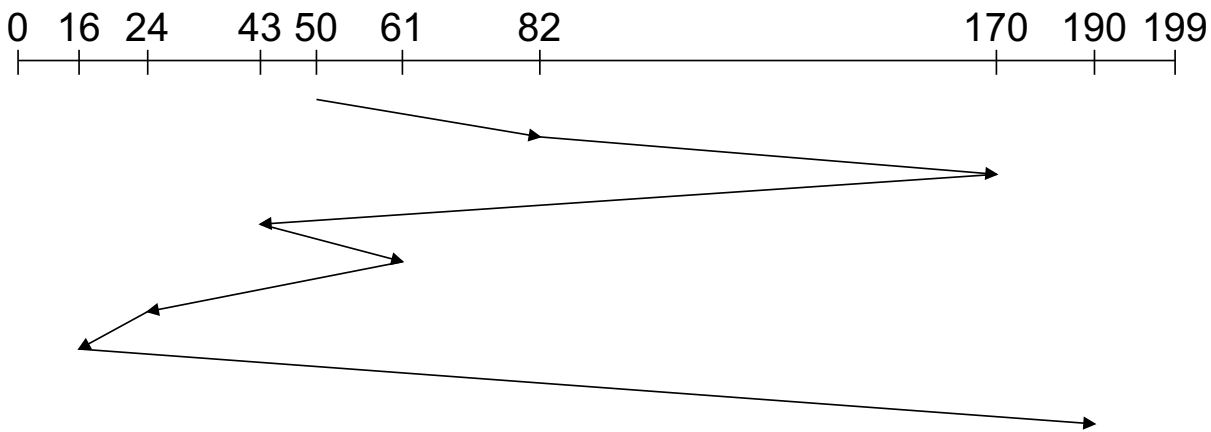
- What?
  - Scheduling of I/O requests for the disk
  - Originally done by OS, now by disk itself
- Why?
  - Disk access time = Seek Time + Rotational Latency + Transfer time
    - Seek Time – time to locate the arm to a track (~ms)
    - Rotational latency – time to rotate a sector to the fixed position of heads
    - Transfer time – time to transfer data
  - Minimize disk access time for multiple I/O requests
- Examples
  - All algorithms demonstrated with the same pattern of I/O requests and initial position
    - I/O requests - 82, 170, 43, 61, 24, 16, 190
    - Initial position - 50

→ Seek může být opravdu dlouhý, protože ta hlava není rychlá

Disk scheduling algorithms

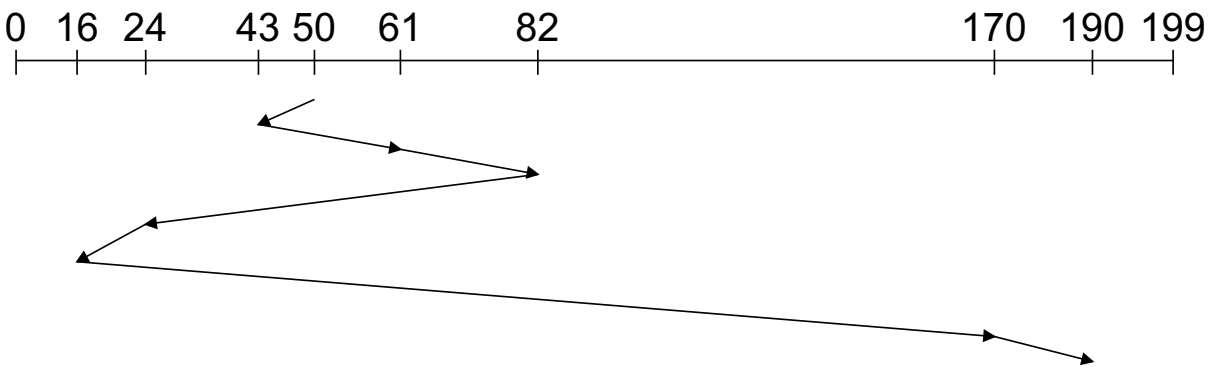
- FCFS (First Come First Served)
  - Pros
    - Fair chance for all requests
    - Simple, good for light load
  - Cons
    - No optimization – usually not the best





Disk scheduling algorithms

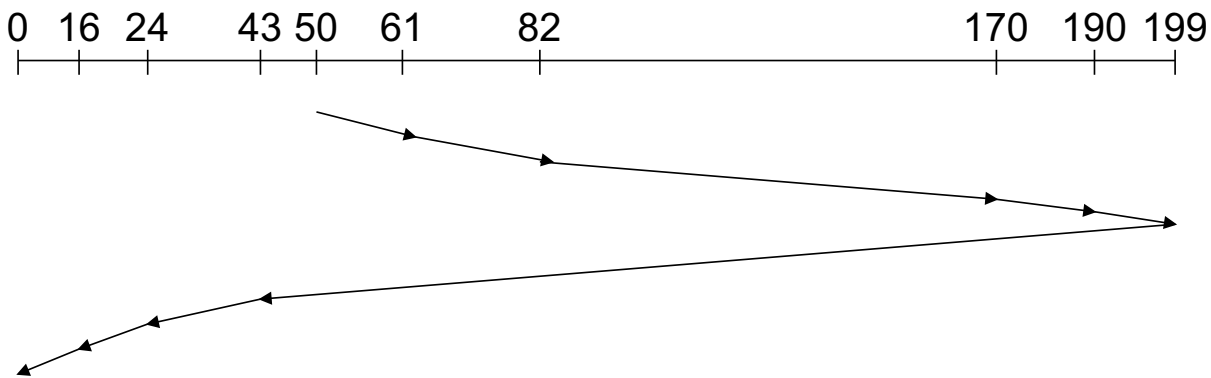
- SSTF (Shortest Seek Time First)
  - Pros
    - Average access time decreases
    - Increased throughput
  - Cons
    - Possible starvation for distant requests, when new short seek requests arrive



Disk scheduling algorithms

- SCAN (a.k.a . Elevator algorithm)
  - Keeps direction (as long as request exists)
  - Pros
    - High throughput – good for heavy loads
    - Low variance in access time
  - Cons
    - Long waiting times for new request just visited by the arm

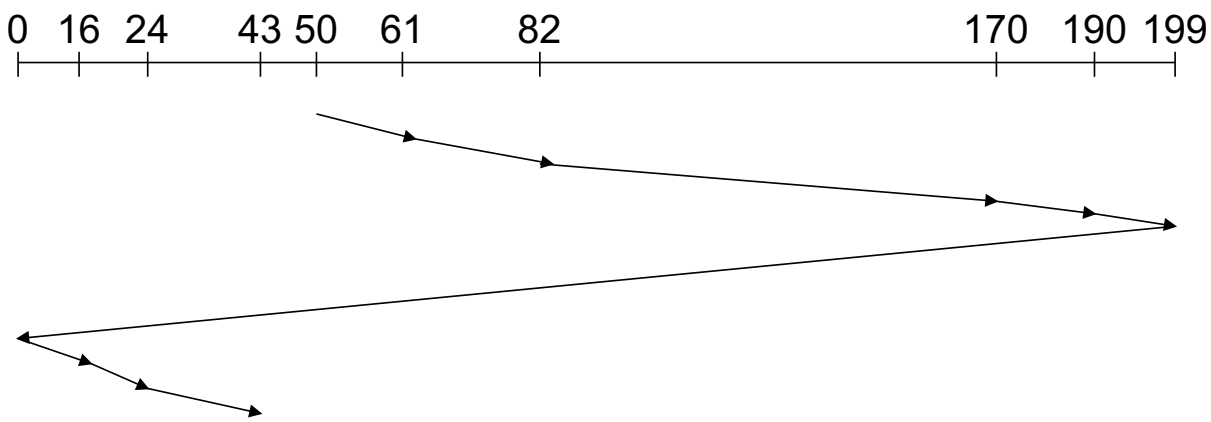
Jedí se na konec, na nikoho nečekáme. Jen je problém, že když zrovna ujetý objel, bude dlouho čekat



Disk scheduling algorithms

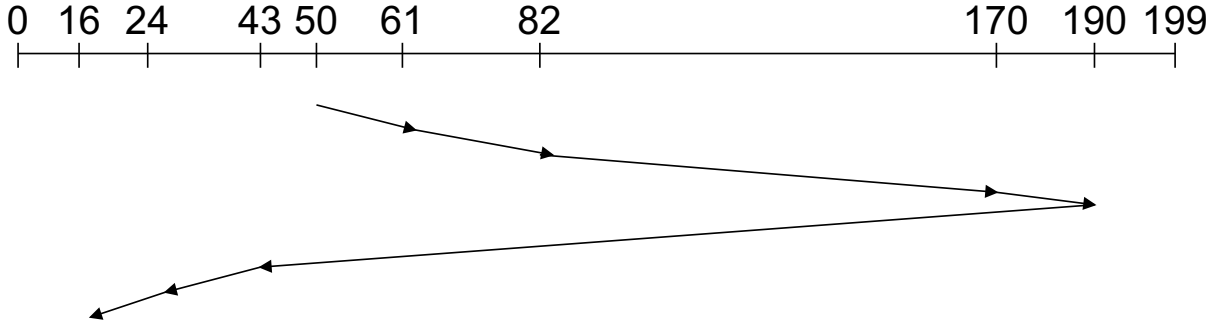
- CSCAN
  - Circular SCAN
  - Pros
    - More uniform time compared to SCAN

ještě lepší výtah, furt jezdí až do krajních pater.



Disk scheduling algorithms

- LOOK/CLOOK → Úplně super, nejezdí ani do krajů, ale fungují jako výtah
  - Like SCAN/CSCAN but does not visit ends of the disk
- FSCAN
  - Two queues
  - Compute algorithm only for 1st queue, new requests are inserted to the 2nd one

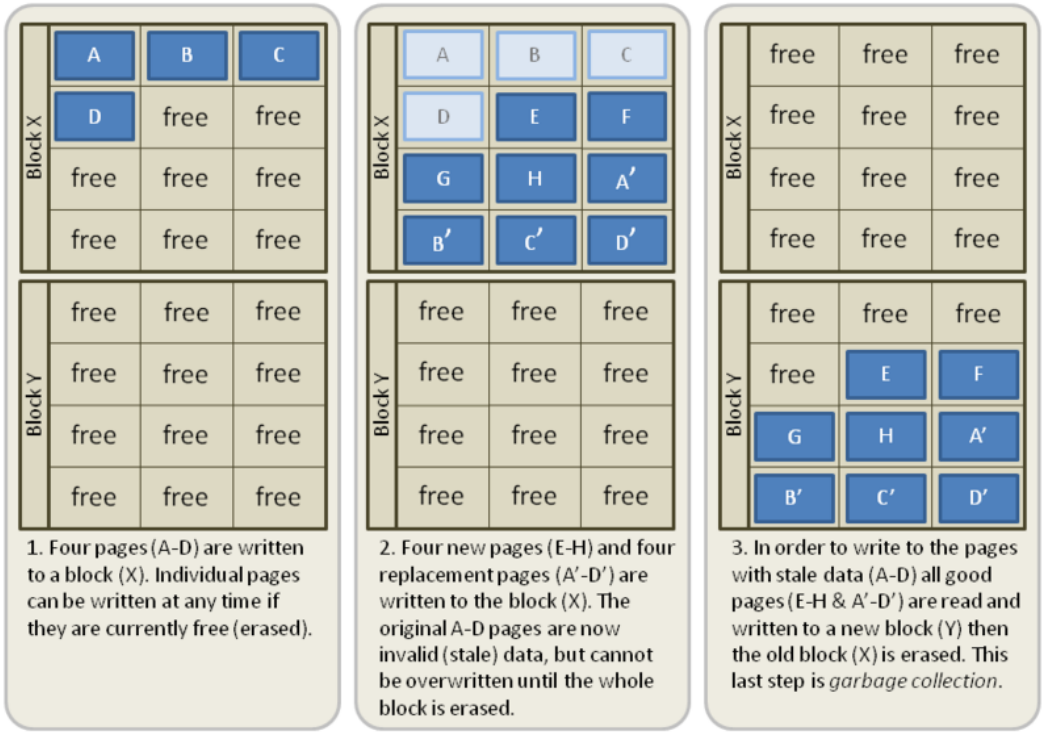


Solid state disk

- Solid-state disk (SSD)
  - Disk without moving parts (only electric circuits)
  - NAND flash, made of floating gate transistors
    - Similar to RAM, but slower and the transistors retain the charge without refreshing
    - Reads/writes damage the transistors over time (writes and erases far more than reads)
      - 1k-100k program-erase (P/E) cycles (MTBF)
  - Grid structure organized as block of pages
    - Page is ~ 2-16 KiB long, block has ~ 128-256 pages
      - 1-4 bits per cell (SLC, MLC, TLC QLC), more bits = cheaper, but less efficient
    - Read/write per page, erase per block (!)
  - Complex controller that handles the I/O operations
    - And data caching in internal RAM

Solid state disk

- SSD data updates
  - Writing new data is OK
  - Page update cannot erase only one page
    - Erase is performed on blocks
  - Update invalidates old page and writes a new one
  - Garbage collection preforms data consolidation
    - Copy only valid pages of a block
    - Erase entire block



https://en.wikipedia.org/wiki/File:Garbage_Collection.png

Solid state disk

- SSD issues
  - Write amplification problem
    - A page is re-written many times due to garbage collection
    - Each write/erase create cumulative damage to NAND flash
  - HW solutions
    - Wear leveling – elaborate algorithm that remaps the blocks
    - Over-provisioning – the SSD is larger than it declares
  - Operating systems solutions
    - Special file systems designed for SSDs
      - Flash-friendly FS (F2FS), BtrFS, log-structure file systems (e.g., LFS)
    - TRIM operation – special command how OS can render blocks invalid (file is deleted)

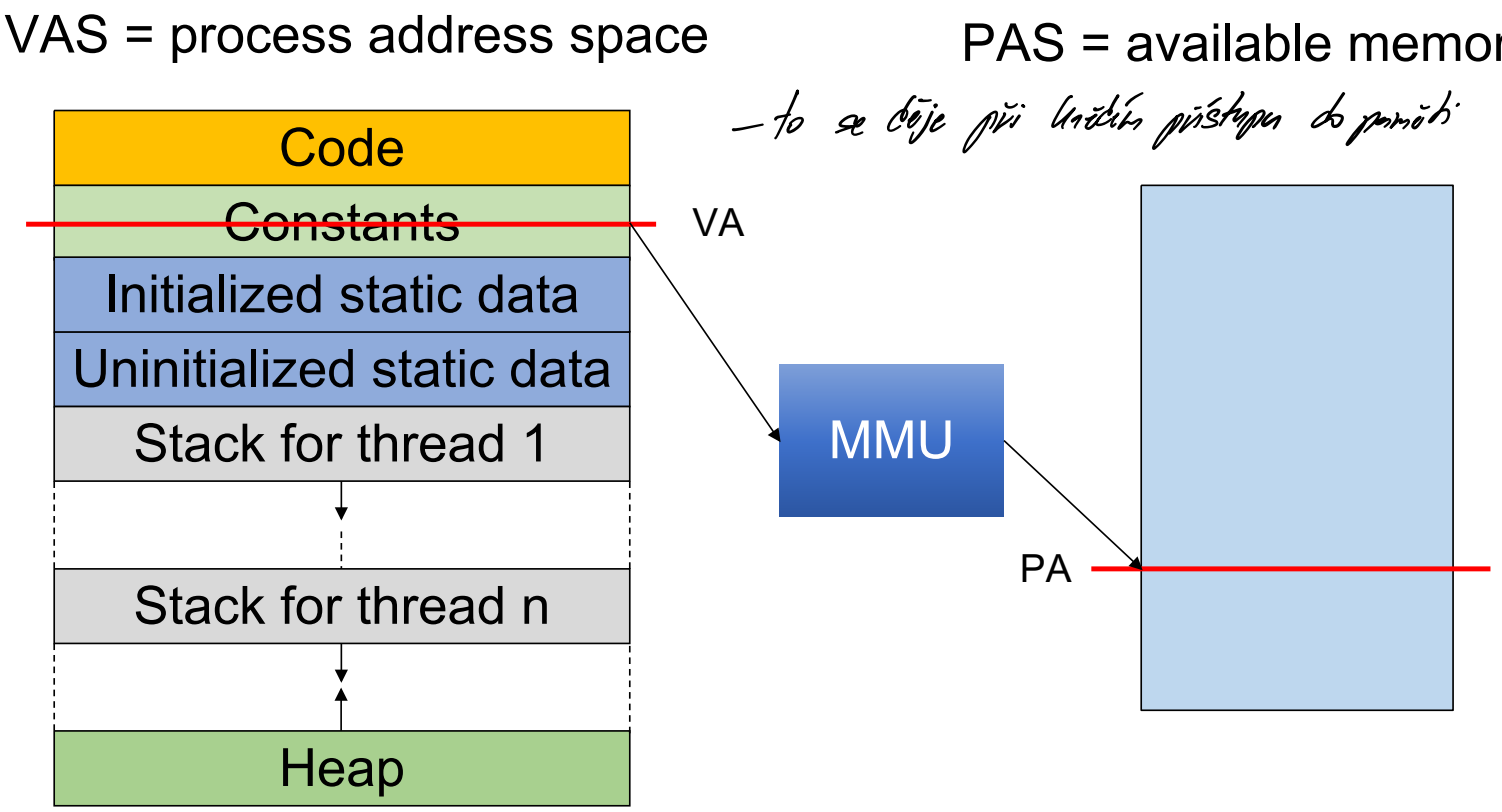
File system(s) on HDD(s)

- HDD Partitioning
  - Division of physical drive into multiple logical drives
    - Each may have its own file system
  - Mounted to paths in root tree (Linux), or presented separately (Windows)
- Redundant Array of Inexpensive Disks (RAID)
  - A way to interconnect multiple HDDs into one
    - Typically at hardware level, but OS can implement it as well
  - Main objective is to increase reliability (and possibly R/W speed)
    - RAID 0 – two disks, per-sector interleaving (better speed, worst reliability)
    - RAID 1 – two disks completely mirrored
    - RAID 5 – each data block is divided among N disks + 1 checksum is created

Virtual memory

- Basic concepts
  - All memory accesses from instructions work with virtual address
    - Virtual address space
    - Even instruction fetch
  - Operating memory provides physical memory
    - Physical address space
    - Always 1-dimensional
    - Memory controller uses physical addresses
  - Translation mechanism
    - Implemented **in HW** (MMU embedded in CPU)
    - Translates a virtual address to a physical address
    - The translation (mapping) **may not exist -> exception (fault)**
    - Two basic mechanisms – segmentation, paging

Virtual memory



Virtual memory

- Why?
  - More address space
    - VAS can be larger then PAS (an illusion of having large memory)
      - Today, IA-32 can have larger PAS then VAS
    - Add a secondary storage as a memory backup/swap
    - This is no longer the primary reason today
  - Security
    - Process address space separation
    - “Separation” of logical segments in a process address space (read-only, executable, …)
  - Specialized (advanced) operations
    - Memory mapped I/O (e.g., memory mapped file)
    - Controlled memory sharing

Segmentation

- Concepts
  - Virtual (process) address space divided into logical segments
    - Segments are numbered
    - may have different sizes
  - Virtual address has two parts
    - [segment number; segment offset]
    - Offsets 0-based for each segment
  - Segment table (translation data structure)
    - In memory, for each process
    - Stores base physical address, length, and attributes for each segment
    - Indexed by the segment number
    - Segment fault (if translation or validation of access fails)

2D pole

Segmentation

- Schema

Code
Constants
offs
Initialized static data
Uninitialized static data
Stack for thread
Heap

VA
seg
offs
BPA
len
Seg table
Heap
Constants
offs
Stack for thread
Code

↳ Pokud není nastaven, pak není v paměti; je ve swapu.

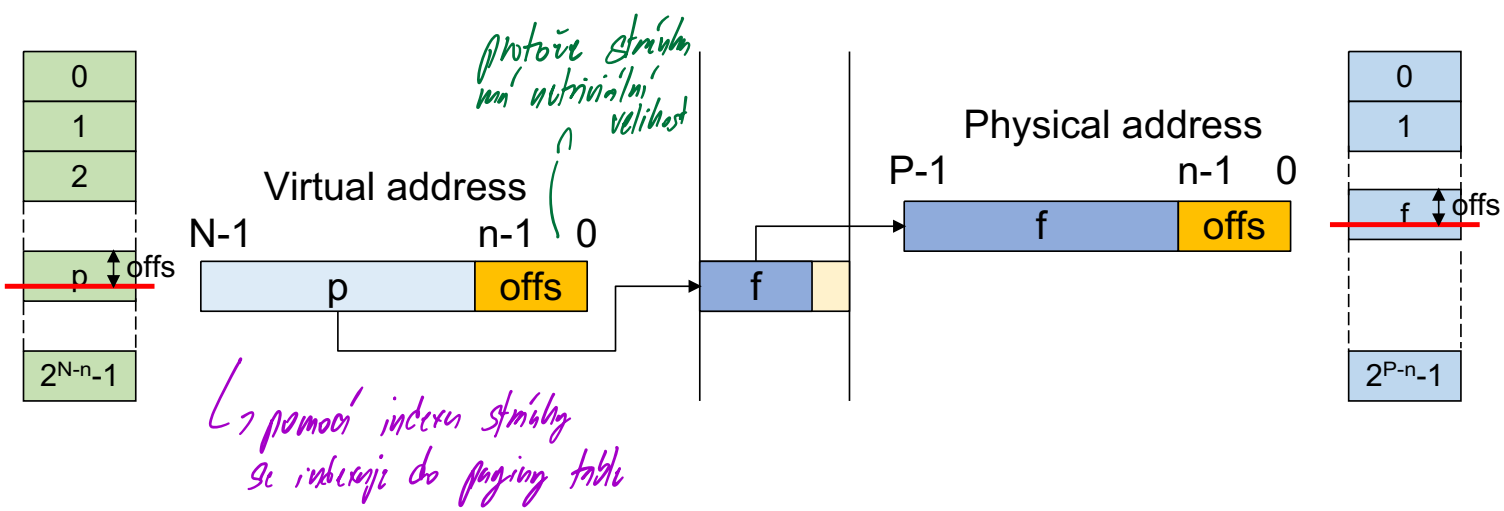
Paging

- Concepts
  - VAS divided into **equal** parts
    - Page, 2^n size
  - PAS divided into **equal** parts
    - Frame, equal size with page (i.e., one page fits exactly one frame)
  - VA 1-dimensional
  - Page table (translation data structure)
    - In memory, for each process
    - Indexed by a page number
    - Each entry contains a frame number and attributes (P)
    - Page fault

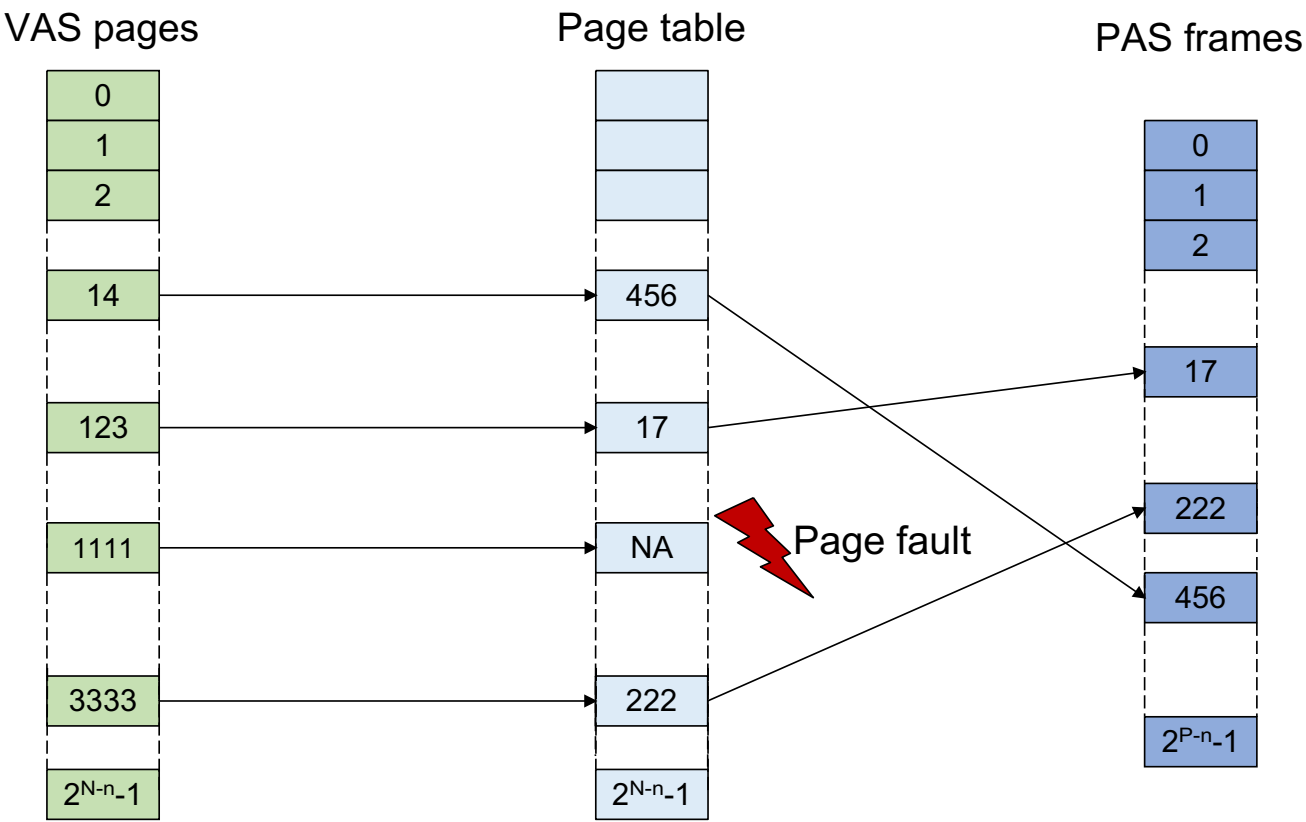
This is very important!

Paging

- Concepts



Page table – 1-level



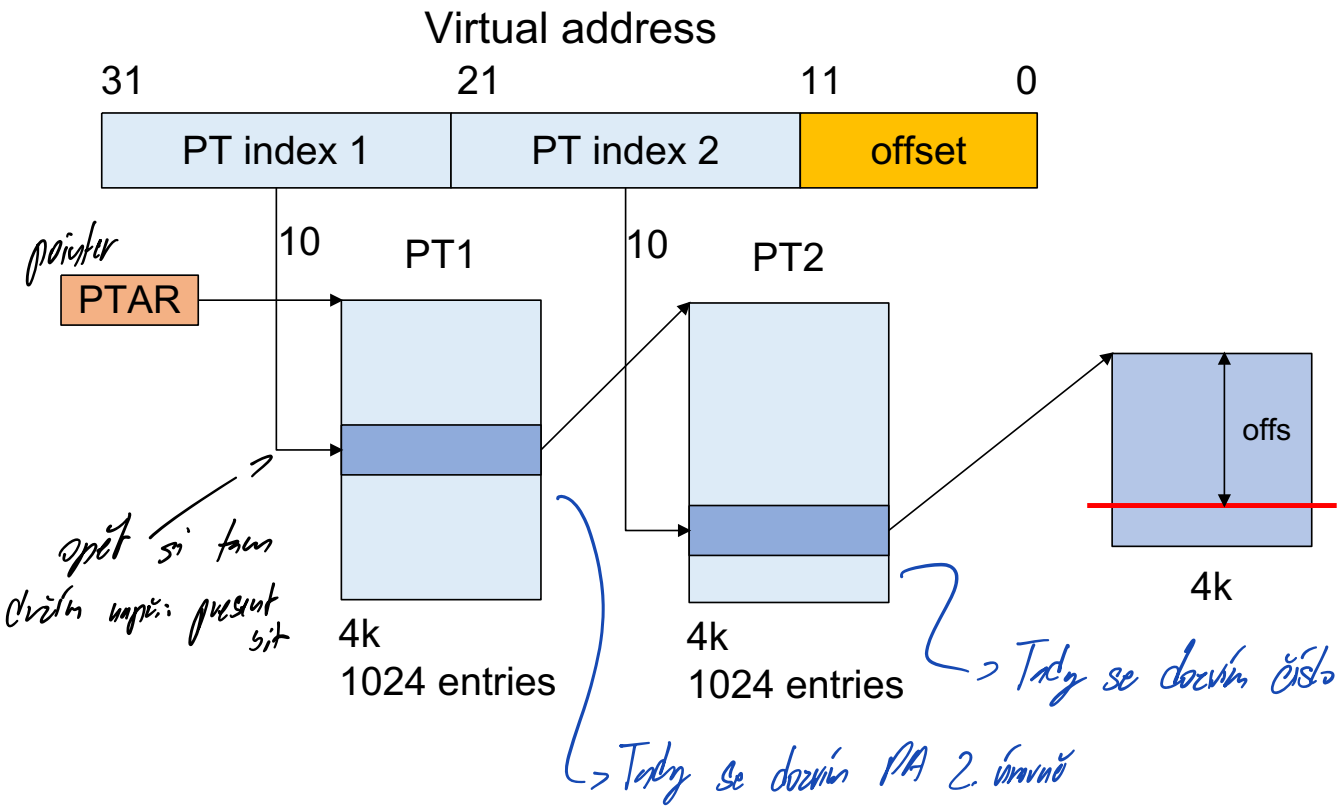
Většinou je jeden logický segment procesu rozdělen do spousty VAS pages.

V PAS ale musí být volné sekvenčně.

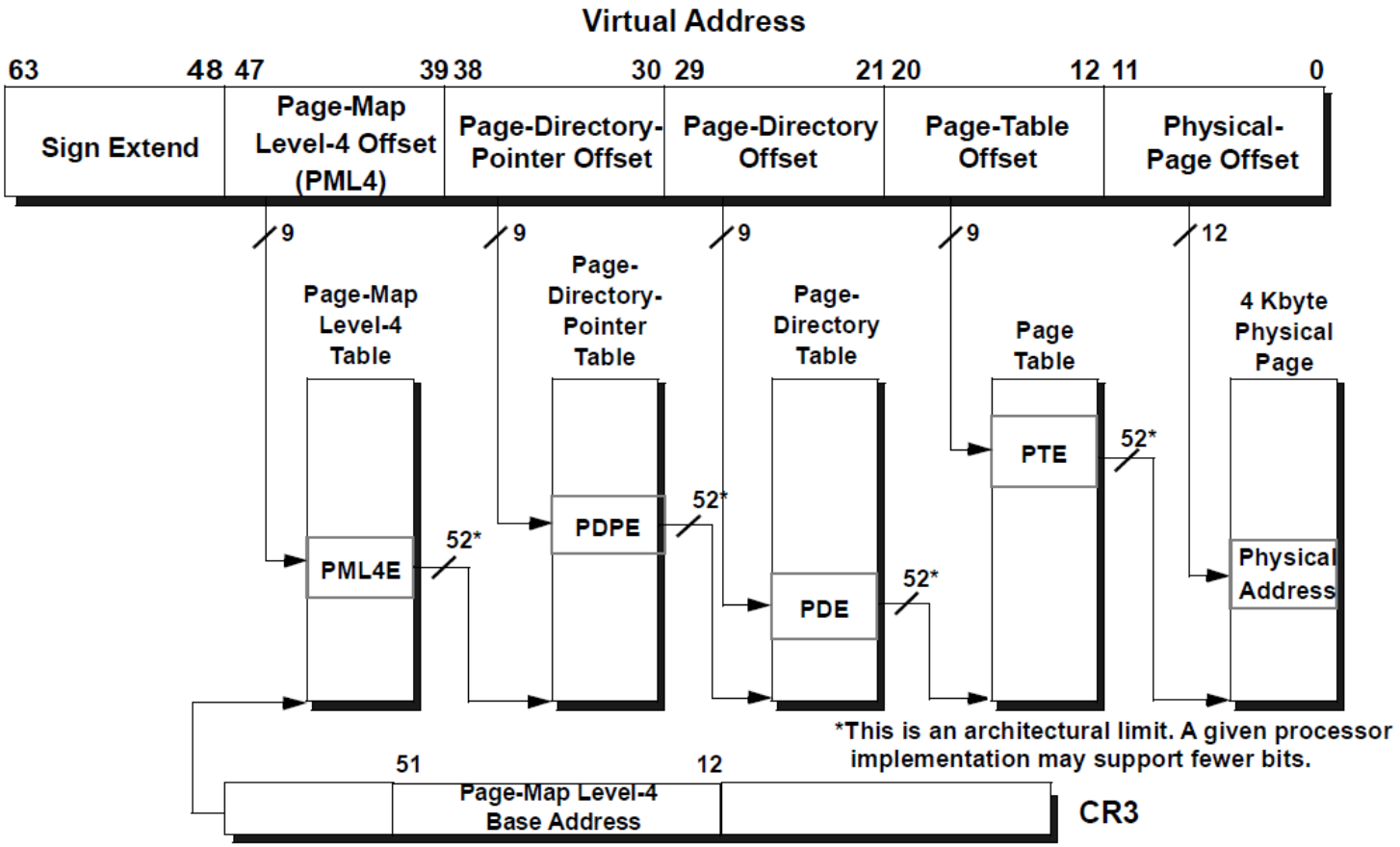
Page table – problems

- Size
  - 1-level page table, 32-bit VA/PA, 4k pages/frames (12 bits)
    - Size of the page table entry?
    - Size of the page table?
  - Do we really need the whole VA?
  - Multilevel page tables
    - 1st level always in memory
    - Individual tables on other levels may be missing (i.e., we are saving space)
- Speed
  - Each memory access from an instruction means at least one other memory access to the page table
  - TLB (Translation Lookaside Buffer)
    - Associative memory
    - Cache for translating page number to a frame number
    - 0-level page tables (MIPS)

Page table – 2-level



Page table – real AMD64 example



Paging – address translation

- Steps for address translation
  - Take page number from VA and keep offset (separately)
  - Check TLB for mapping
    - If exists, retrieve frame number, otherwise continue
  - Go through the page table
  - Update A(ccessed) and D(irty) bits in page table/TLB
  - Assemble PA by “gluing” the retrieved frame number and the original offset from VA
- Go through the page table
  - Divide page number into multiple PT indices
  - Index 1st level PT
    - If there is no mapping for 2nd level PT, raise page fault exception
    - Retrieve PA for 2nd level PT and continue
  - Go through all levels of PTs
    - If there is no mapping in any PT level, raise page fault exception
    - If all PT levels are mapped, retrieve frame number
  - Save retrieved mapping to TLB

Paging – page fault exception handling

- An instruction raises the page fault exception
- OS interrupt handler
- Determine the cause of the fault
  - Unauthorized access
    - Out of allocated virtual space, store to R/O page, access to kernel memory, …
  - Valid address but not mapped
- Create mapping
  - Find a free frame
  - Load content to the free frame
  - Construct/fill corresponding page tables
- Return back from handler and retry the instruction

- Find a free frame
  - Either there is one unoccupied
  - Or find a victim (for swapping)
    - Page replacement algorithms
    - Save dirty victim frame
    - Remove mapping from TLB

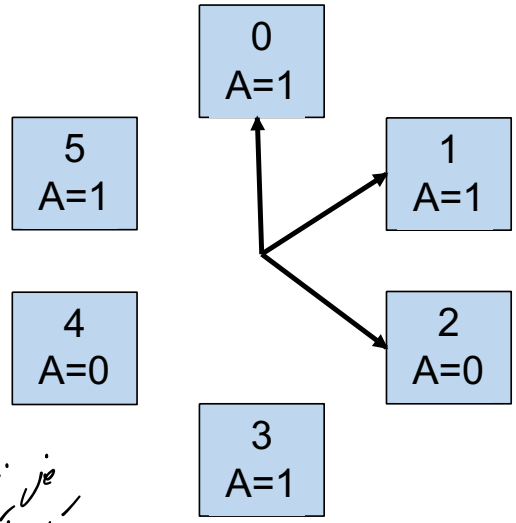
Ty, které nejsou dirty, nemusíme zapisovat do swapu, protože je umíme znovu najít

Page replacement algorithms

- Cache-replacement algorithms
  - Any situation, when you need to find a victim from a limited space
  - Frames, TLB, cache, …
- Optimal page algorithm
  - Replace the page that will not be used for the longest period of time
  - Lowest page-fault rate
  - Theoretical, we do not have an oracle for foretelling the future

Page replacement algorithms

- Clock
  - Frames organized in a circular manner
  - Clock hand points to the next frame to replace
  - Each page has A(cessed) bit
    - A is the accessed flag which is set to 1 whenever the page is touched (by HW)
  - If the frame has **A != 0**, set **A = 0** and advance the hand
  - If the frame has **A == 0**, select this frame



Page replacement algorithms

- NRU (Not Recently Used)
  - Each page has A(ccessed) and D(irty) bits
  - Clears A bits periodically (e.g., once a minute)
    - Bit D is not touched
  - Uses A and D bits to classify frames into four classes
  - Selects a random frame from the lowest non-empty class

Class	A	D
0	0	0
1	0	1
2	1	0
3	1	1

Page replacement algorithms

- LRU (Least Recently Used)
  - Uses the recent past as a prediction of the near future
  - Replaces the page that has not been referenced for the longest time
  - Existing HW implementations
    - Cache
    - Bit matrix
  - SW implementation
    - Move-to-front algorithm
      - Can be implemented by linked list or heap data structure
    - Too complicated and space consuming
    - Approximation algorithms exist

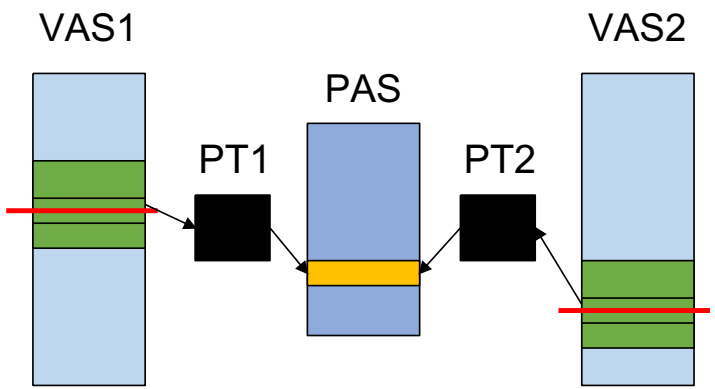
Page replacement algorithms

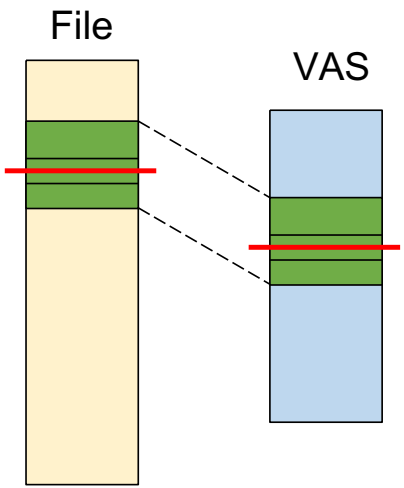
- NFU (Not Frequently Use)
  - Rough approximation of LRU
  - Each frame has a counter (typically small, several bits)
  - Periodically scan page table and increase the counter for a frames with **A==1**
    - Always clear A
  - Select the frame with lowest counter
  - Problems
    - Newly occupied frames may be swapped before they get used
    - Frames that were previously heavy used will never be selected
  - Aging
    - Periodically divide counters by 2 (i.e., shift by 1)

I stránky, co hodně jedi na začátku, tedy rychle vyhnájou

Advanced paging

- Shared memory
  - Part of a virtual memory space shared amongst processes
    - The block is probably placed on different starting virtual address
- Memory-mapped files
  - File as a backing store for paging
  - Direct access to the file content using CPU instructions
  - Problems with file size and with appending data

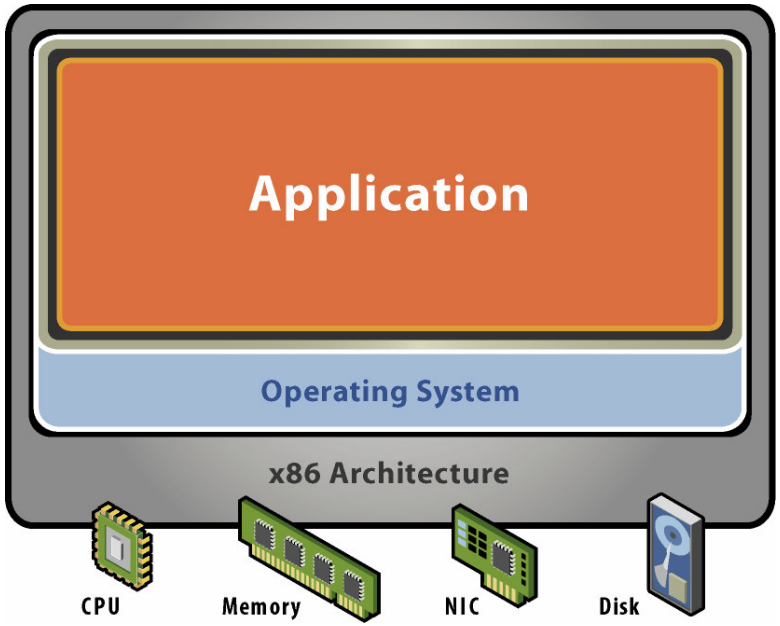




Virtual machine and containers

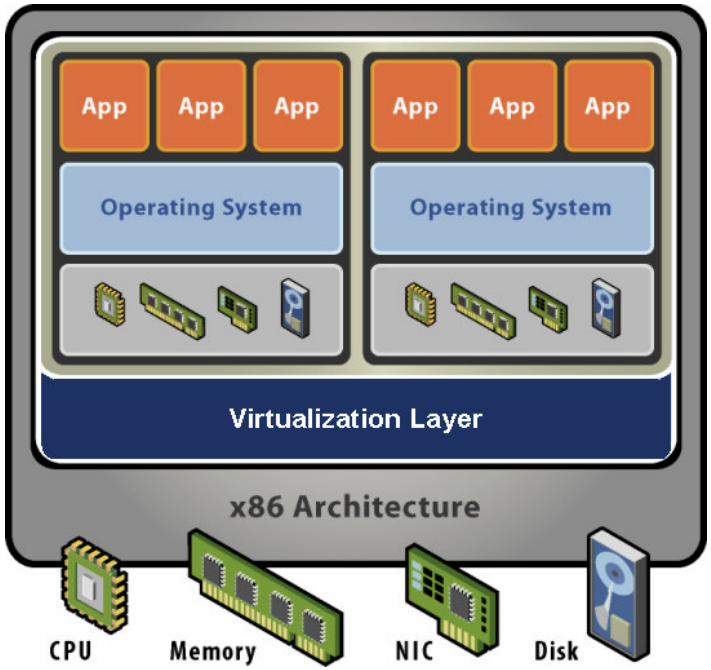
- VM = Emulation of a computer system
  - Full virtualization
    - Substitute for a real machine, allows execution of entire OS
    - Hypervisor shares real HW, native execution, virtual HW
    - Isolation, encapsulation, compatibility
  - Process VM
    - Runs as an application inside OS
    - Provides platform-independent programming environment
      - Abstract machine (instructions, memory, registers, …)
      - Java VM, .NET CLR
    - Slow execution
      - JIT, AOT
- Container = OS-level virtualization
  - OS kernel allows existence of multiple isolated user space instances

Physical machine



- Physical HW
  - CPU, RAM, disks, I/O
  - Underutilized HW
- SW
  - Single active OS
  - OS controls HW

Virtual machine



- HW-level abstraction
  - Virtual HW: CPU, RAM, disks, I/O
- Virtualization SW
  - Decouples HW and OS
  - Multiplexes physical HW across multiple guest VMs
  - Strong isolation between VMs
  - Manages physical resources, improves utilization
  - Encapsulation – VM represented as a set of files, can be easily distributed

Discussion