# Obsah přednášky

* Jazyk C a C++
* CPU
  + Architektura
  + Instrukční sady
  + Přerušení, DMA
* Paměť
  + Adresování
  + Zarovnání
  + Uspořádání
  + Cache
* Programovací jazyky
  + Kompilace
  + Správa paměti
  + Funkce, parametry funkcí
  + Halda
  + Runtime
  + JIT
* Operační systémy
  + Architektura
  + Procesy, vlákna
  + Virtuální paměť
* Paralelní programování
  + Synchronizace

# C / C++

* Procedurální programovací jazyk
* Statický typový systém
* Snadno se mapuje na strojový kód
* Použití
  + Operační systémy
  + HPC (High Performance Computing)
  + Mikrokontrolery a mikroprocesory
* Case sensitive
* Ignoruje mezery, tabulátory
* Vznik C 1971 (K&R C norma 1978)
* Nejnovější verze
  + C11 (2011)
  + C++20 (2020)
* V roce 2002 z něj vznikl C#

# Architektury

* Von Neumanovská
* Harvardská
  + Rozdělená paměť pro kód a pro proměnné
  + Můžu paralelně zapisovat do datové paměti a zároveň provádět instrukce
* Sběrnice
  + Sériové
    - Používají se dnes
    - Jsou rychlejší než paralelní
  + Paralelní
    - Používali se dříve

# Procesor

* + Hardwarová architektura
  + Architektura instrukční sady
    - ARM, X64, X86, PowerPC, Stark, MIPS
  + Implementace architektury instrukční sady prostřednictvím architektury hardwaru
    - Implementace záleží na výrobci
* Třídy instrukcí procesoru
  + Load, Store, Move
    - Práce s pamětí je pomalá
  + Aritmetické a logické operace
  + Jump
    - Podmíněné a nepodmíněné
    - Přímé, nepřímé, relativní
  + Call, return
* Registry
  + Obecné (na ukládání celých čísel, adres)
  + Příznakové 1bitové registry
  + Floating point registry
  + Predikátové registry
    - Každá instrukce má flag, zda se provádí nebo neprovádí
  + Vektorové registry
  + Aliasy
    - Aliasy pro menší registry (např. 16b) v rámci větších registrů (64b)
  + Klasifikace arichtektur
    - CISC – Complex Instruction Set Computer (x86, ARM)
    - RISC – Reduced Instruction Set Computer (MIPS)
    - VLIW – Very Large Instruction Word
      * Síťové switche
      * Jednodcuhé, ale účinné
    - EPIC – Explicitly parallel Instruction Computer

## Architektura MIPS

* 32 32bitových registrů
* Nemá zásobník, příznakový registr
* Jediný registr navíc je PC (program counter)
* 3 adresová architektura
  + Instrukce typu ADD $a1,$a2,$a3 – a1 = a2 + a3
* Délka každé instrukce 32 bitů (intel 1-15 bytů)

## Instrukce

* Překladač assembeler přeloží textové instrukce do binární podoby
  + Assembler je také jazyk nad strojovým kódem

## Zpracování instrukcí

* Pipeline
  + Zpracovává se více instrukcí paralelně (více stages)
  + Jedna se načítá, další se vykonává, třetí se ukládá …
* Pokud mám více pipeline → více instrukcí najednou
* Asymetrické
  + Různé pipeline umí zpracovávat různé typy instrukcí
* Instrukce se ve výkonných jednotkách zpracovávají v náhodném pořadí
  + Různé instrukce patří do různých jednotek
    - Dělené např. na práci s pamětí, logické operace …

# Paměť

* Data v paměti uložena v bitech
* Bity sdruženy do slov pevné délky
* Každé slovo je přístupné pod Nbitovou adresou
  + → lze uložit 2n slov
* Typické slovo dnes je byte – 8 bitů
* Adresový prostor – rozsah 0-2n-1
* V praxi se adresa (např. 32bitová) rozloží pro určení konkrétní pozice buňky
  + Row (14bits), Bank(3bits), Column(11bits), byteInBus(3bits)
* Přístup na řádku je pomalý, ale přesun mezi sloupci je rychlejší
  + Přesun mezi sloupci odpovídá sekvenčnímu čtení

## Reprezentace dat

* Celá čísla
  + Dvojkový doplněk
* Desetinná čísla
  + IEEE 754
* Endianita
  + Little-endian
    - Na nejnižší adrese je nejméně významný byte
  + Big-endian
* Většina zařízení Big-endian, Intel little-endian
* Nutné hlídat endianitu při přenosu dat (síťování, úložiště)

## **Zarovnání** paměti

* Adresy proměnných musí ležet na násobku velikost té proměnné
* Vnitřní zarovnání
  + Zarovnání mezi atributy jedné struktury
* Vnější zarovnání
  + Přidání zarovnání na konec struktury v poli
  + Konec struktury musí být na násobku největšího typu ve struktuře

## Alokace paměti

* Program si vyžádá od operačního systému úsek paměti
* Např. klíčové slovo new mi přidělí kus paměti – adresu
  + Uvolňování paměti
    - Explicitní C, C++
    - Garbage collector
* Fragmentace paměti
  + Interní
    - Runtime přiděluje programu paměť po blocích, např. 64 bytů
    - Ne vždy využívám celý blok
  + Externí
    - Nevyužitá paměť se hromadí v nevyužitých blocích

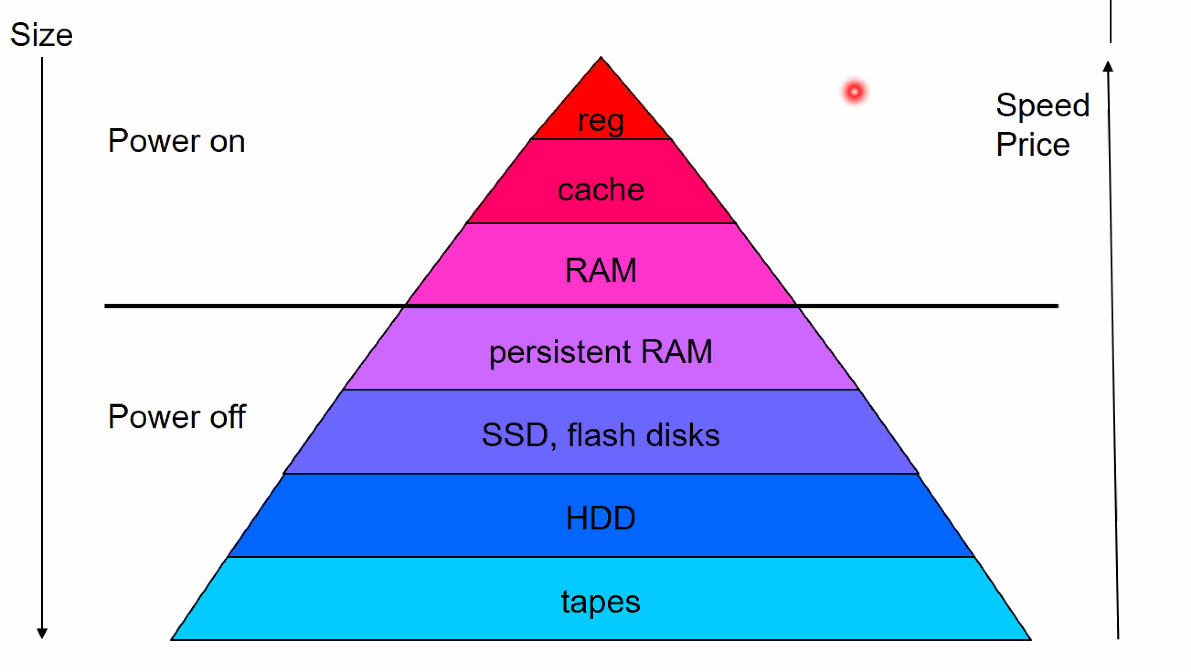
### Algoritmy pro dynamickou alokaci paměti

* **U zkoušky!!!!**
* FirstFit
* NextFit
  + Pamatuje si předchozí pozici
* BestFit
* RowstFit

### Buddy memory alocation

* Používá bloky velikosti 2n
* Slučuje volné bloky paměti do dvojic – buddy

### Hierarchie paměti

* 
* Reg
  + Rychlé
  + Velikost řádově 100B
  + Přítup v 1 taktu
* Cache
  + L1
    - Rychlost přístupu cca 4. takty
  + L2
    - 20 taktů
* RAM
  + Přístup procesoru do ram cca. 1000 taktů
  + Nejpomalejší součástí je čtení z paměti
* Persistent RAM
  + Intel Optane
* HDD
  + Pohyb hlavy - seek – jednotky mls
  + Nejrychlejší 15k otáček / min
* Pásky
  + Používají se na zálohování velkého množství dat

### Cache

* Na HW i SW úrovni
* Koherence cache v procesoru
  + Každé jádro má svou L1 a L2 cache
  + Nutné udržovat cache napříč jádry
  + Obvykle řešeno přesuny přes L3 cache, která je společná
* Cache line
  + Minimální blok paměti v cache
* Cache hit
  + Podaří se mi načíst data z cache
* Cache miss
  + Musím znovu přistoupit do paměti (RAM)

### Asociativní paměť

* Používá ve v procesorových cache
* Rychlý přístup podle klíče (např. adresa)

### Víceprocesorové systémy

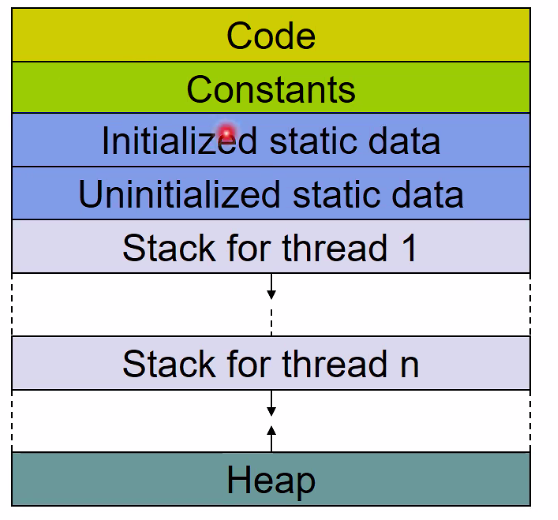
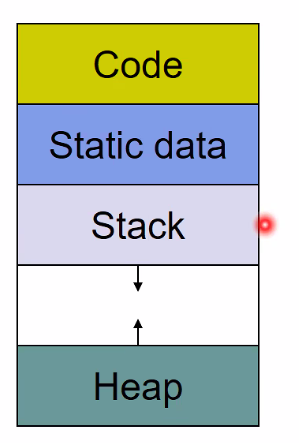
* SMP
  + Symetrické
  + Všechny procesory přistupují do RAM po jedné směrnici
* NUMA
  + Více procesorů
  + Každý má svou vlastní RAM
  + 2, 4 nebo 8 procesorů komunikujících po vlastní sběrnici

# Programovací jazyky

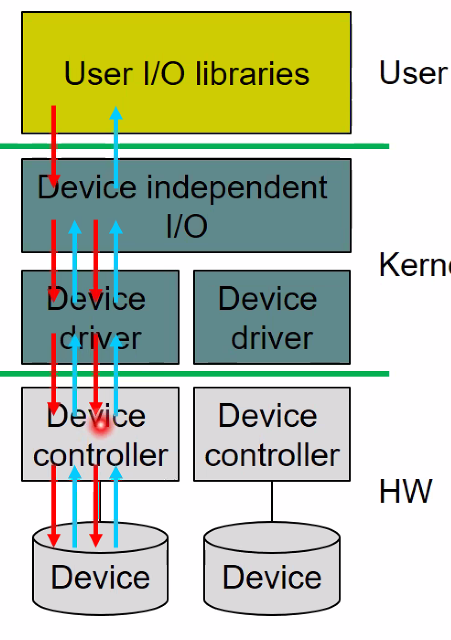
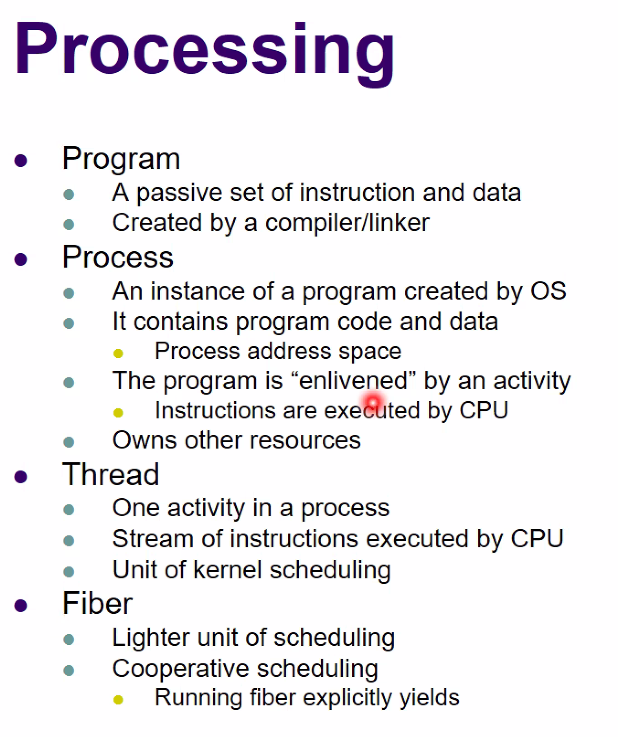
### Překladač

* Preprocesor
  + Textově nahrazuje části kódu
  + Např. #define v C++
* Překladač
  + Přeloží textový kód od preprocesoru do assembleru
    - Assembler = textový zápis instrukcí
* Assembler
  + Přeloží assebbler (textový zápis) do .obj formátu
* Knihova
  + Přeložená sada kódu → Moduly
  + Statické
    - Jeden soubor se všemi přeloženými moduly
    - Potřebné z ní se zkopíruje do .exe souboru
  + Dynamické
    - Při překladu se nekopíruje implementace, ale do .exe si poznamenám, kterou knihovnu potřebuji
    - Po spuštění souboru je loader nahraje do paměti a v .exe upraví adresy, aby se volali dané funkce z dynamické knihovny
* Linker
  + Připojí knihovny
  + Závislí na operačním systému
  + Vyrobí spustitelný soubor → program
* Operační systém
  + Spustí spustitelný soubor → proces
* Loader
  + Přečte .exe do paměti
  + Alokuje paměť
  + Vytvoří proces

### Paměť za běhu programu

* 
* Kód
  + Neměnný kus programu
* Statická data
  + Globální proměnné
  + Inicializované
    - Např. pokud dám glob. proměnnou int i = 5;
  + Neinicializované
* Zásobník
  + Při volání funkcí udržuje informaci, kam se vrátit
  + Ukládání lokálních proměnných ve funkci
* Halda
  + Není datová struktura! – spíš hromada
  + Paměť k dispozici pro běh programu
  + Alokace pomocí alokačních algoritmů vrátí ukazatel
* Vlákno
  + Podmnožina procesu
  + Vykonává vlastní instrukce → má vlastní zásobník pro volání a lokální proměnné
* **TODO tady jsem spal**
* Konvence volání funkcí
  + Mandlování veřejných jmen TODO
  + Sekvence volání
    - V jazycích C-like zodpovídá za předání parametrů a načtení výstupních hodnot funkce volající
  + Předávání parametrů
    - Hodnotou
    - Referencí
* Proměnné
  + Globální ve statických datech
  + Lokální v C na stacku
  + Dynamické na haldě
  + Např. python ukládá proměnné do slovníku
* Heap
  + Dynamická paměť
  + Volná alokace podle potřeby (na rozdíl od statických proměných)
  + Není vázaný na zásobník

# Operační systémy

* Vlastnosti
  + Abstrakce na HW prostřednictvím kernel API
  + Řízení různého HW, jeho správa a jeho zpřístupnění aplikacím
* Režimy procesoru
  + Uživatelský režim
    - Omezený přístup k registrům a instrukcím
  + Systémový režim
    - Přistup všude
* Architektury operačního systému
  + Monolitická
    - Běží v systémovém režimu
    - Např. linux
    - Výhody
      * Efektivní kód
    - Nevýhody
      * Bezpečností hrozba vniknutí do systémového režimu
      * Problém rozšiřitelnosti za běhu (např. ovladače)
  + Vrstvená
    - Praktické, ale složité na navržení
  + Mikrokernel
    - V systémovém módu pracuje minimum kódu
    - Nad jádrem běží systémové služby, které zpracovávají požadavky aplikací
    - Aplikace si předávají požadavky právě přes mikrokernel
    - Ovladače se spouští pouze jako další aplikace
    - Výhody
      * Bezpečné
    - Nevýhody
      * Předávání zpráv je pomalejší, než přímé volání funkcí
* Zařízení
  + Např. myš, klávesnice, tiskárna
  + Řadič zařízení
    - Elektrické (fyzické) připojení zařízení
      * Řešení kolizí, A/D převody
    - Zařazení zařízení do topologie
    - Dostupný v paměti na adrese
  + Ovladač zařízení
    - Program
    - Slouží k přístupu k zařízení na SW úrovni
  + Topologie
    - Sběrnicová
      * Jeden drát, problémy s kolizemi
    - Hvězdicová
      * Peer-to-peer
  + Správa zařízení  
    
  + Předání správy od zařízení procesoru
    - Polling
      * CPU se jednou za čas ptá zařízení, zda se děje něco nového
      * Nevhodné, zpoždění a nutná aktivní kontrola ze strany CPU
    - Přerušení
      * Řadič dá signál CPU, že nastala událost
      * CPU má pin pro IRQ (interrupt request)
      * CPU přestane vykonávat instrukce a vyřídí přerušení
    - DMA (Direct Memory Access)
      * Řadič po dokončení práce namísto upozornění procesoru přímo zapíše do paměti
      * Až po okopírování dat je upozorněn procesor přerušením
  + Typy přerušení
    - Externí
      * HW zdroj pošle přerušení na IRQ pin
      * Lze zakázat (např. když je CPU v systémovém režimu) např. nastavením nějakého registru nebo speciální instrukcí
    - Exception
      * Při chybě během zpracování instrukcí
      * Dva druhy
        + Trap – instrukce, která způsobila chybu nemůže doběhnout. Přerušení se provede za ní.
        + Fault – instrukce, která způsobila chybu může doběhnout. Přerušení se provede před ní. Očekává se od OS, že může provedení instrukce napravit.
    - Softwareová
* Processing  
  
  + Struktura
    - Program
      * Pasivní přeložená sada instrukcí
    - Proces
      * Instance programu vytvořená OS
    - Vlákno
      * Místo v procesu, kde se vykonávají instrukce
      * Každé vlákno má vlastní zásobník
    - Fiber
  + Úlohy
    - Scheduler
      * Real-time scheduling
        + Každý proces má čas, kdy se spustí a čas, do kterého musí skončit
        + Deadline

Hard – po překročení deadlinu už nemá cenu úlohu dokončovat

Soft – i přesto, že čas už uběhl, pořád á význam úlohu dokončit

* + - * + Typické pro systémy řízené událostmi
        + Nutné nasadit jazyky, kde je jasné, co se děje (např. C, ne Python)
      * Stavy úloh
        + Created
        + Terminated (slovy Yaghoba - zombáč)
        + Ready (pending)

Vlákno procesu může běžet, ale neběží … protože procesor jich má třeba jen 8, zatímco běžet by jich chtělo třeba 100

* + - * + Running

Vlákno běží na fyzickém vlákně procesoru

* + - * + Blocked

Vlákno čeká na nějakou službu OS, která trvá dlouho

* + - Scheduling (plánování)
      * Cíle
        + Různé aplikace mají různé cíle
        + Maximální vytížení procesoru
        + Nejrychlejší zpracování vláken
      * Priorita
        + Procesu lze přiřadit prioritu
        + Statická priorita

Přiřazena na začátku procesu

* + - * + Dynamická priorita

Zajišťuje spravedlivost plánování

Dokud jsou vlákna ve stavu Ready, tak se jim periodicky zvedá priorita. Po přechodu do stavu running se vynuluje.

* + - * + Celková priorita = statická + dynamická
      * Algoritmy pro plánování
        + Nepreemtivní systémy

First Come, First Serve (fronta)

Shortest Job First

Potřebuje znát očekávanou délku zpracování procesu

Longest Job First

* + - * + Preemtivní systémy (každé vlákno má omezený čas na průběh)

Round Robin (fronta)

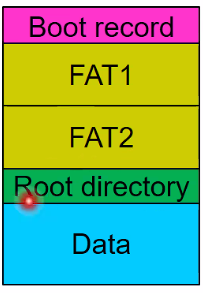
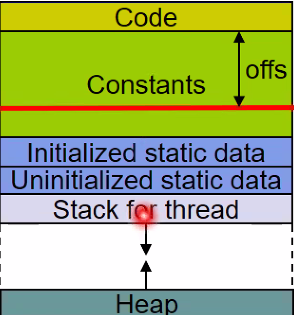
Víceúrovňová fronta

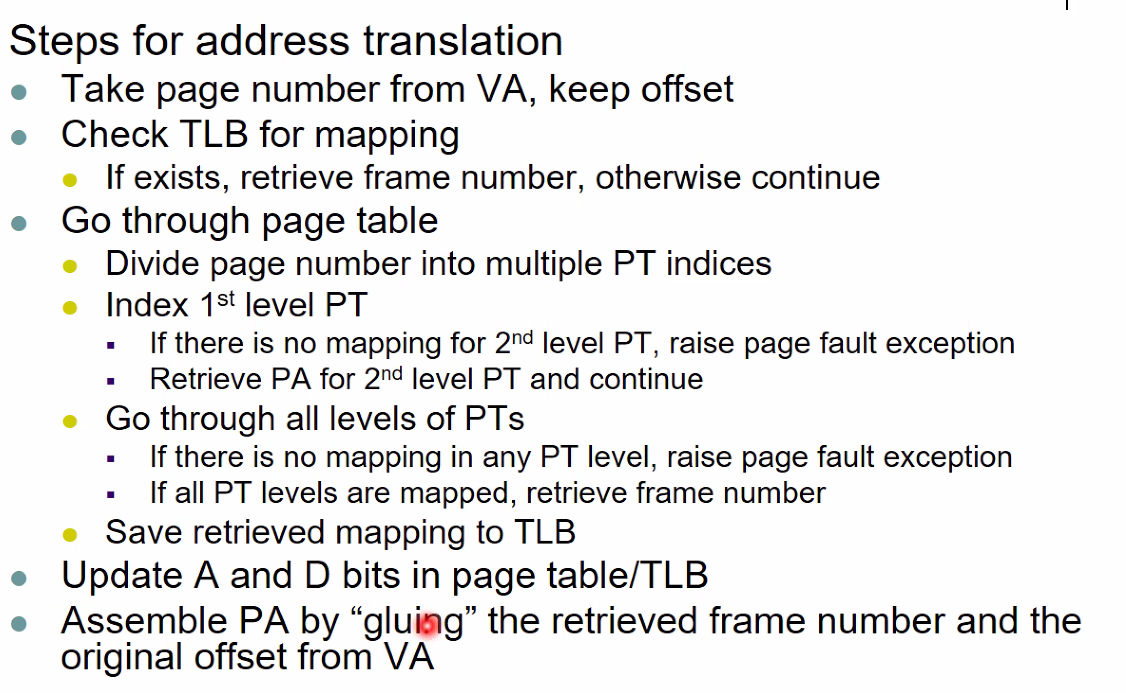
Reaguje na potřeby procesů

Sestupně klesá priorita procesů

Completly fair scheduler (CFS)

Procesy jsou v červeno-černém stromě

* + - Multitasking
      * Cooperative
        + Všechna vlákna spolupracují a ukončí se, aby uvolnilo místo dalším
      * Preemptive
        + Každé vlákno dostane časový slot
        + Na jeho konci je vlákno ukončeno přerušením
    - Multipreocessing
    - Context
    - ContextSwitch
* Soubor
  + Kolekce souvisejících dat
  + Operace
    - Open, Close, Read, Write, Seek
  + Přístup
    - Sekvenční, náhodný
* Adresář
  + Množina souborů
* File systém
  + Kontroluje, jak jsou data ukládána
  + Vytváří abstrakci pro soubory a adresáře
  + Adresář pouze speciální soubor
  + Funkce
    - Překlad jmen souborů (hledání)
    - Lokalizace souborů
    - Správa volných bloků
      * Bitmapa, linked list (ve volném je odkaz na dalšího)
  + Lokální
    - FAT (File alocation table)
      * Jednoduchý, původně na MS-DOS
      * Seznam souborů
        + Každý soubor má odkaz na první datový blok
      * Ochrana proti poškození při nečekaném přerušení zápisu
        + Vždy dvě kopie – do jedné zapisuji, když se to povede, přehodím do druhé (FAT1, FAT2)
        + 
    - Ext2
      * Linuxový, starý
      * INode
        + Představuje jeden soubor/adresář
      * Adresář
  + Síťové
    - NFS, SMB (samba)
  + Diskové plánovací algoritmy
    - First Come First Serve
    - Shortest Seek Time First
    - SCAN
    - Circular Scan
* Virtuální paměť
  + Koncepty
    - Všechny přístupy do paměti pracují s virtuálními adresami do paměti
    - Fyzická paměť tvoří vlastní paměťový prostor → fyzický
      * Jednorozměrná
      * Řadič paměti pracuje s fyzickými adresami (řadič na procesoru)
      * Velikost fyzické paměti odpovídá rozsahu procesoru
    - Součástí procesoru je MMU (memory managment unit)
      * Překlad virtuálních adres na fyzické (hardwarová záležitost, nikoliv OS)
      * Pokud se překlad virtuální adresy nepovede → výjimka (forma přerušení)
      * Mechanismy
        + Segmentace
        + Stránkování (pading)
  + Motivace
    - Větší adresový prostor
    - OS si může část virtuální paměti odložit třeba na pevný disk
    - Bezpečnost
      * Oddělení pamětí procesů
    - Možnost rozdělení paměti na části
      * Konstanty
      * Instrukce
      * Zásobník
      * …
  + Segmentace
    - Odpovídá segmentům uvnitř adresového prostoru procesu  
      
    - Segmenty jsou očíslované
    - Adresa do virtuální paměti má dvě části [číslo segmentu; offset od začátku segmentu] – 2rozměrná
    - Segmentační tabulka (deskriptor)
      * V paměti pro každý proces
      * Je v ní uložena první fyzická adresa, délka, atributy segmentů
      * Indexované číslem segmentu
    - Hardwarová implementace
  + Stránkování
    - Koncepty
      * VAS rozdělen na stejně velké části → stránky o velikost 2^n
      * PAS je také rozdělen na stejně velké části → framy o stejné velikosti jako stránky
        + Oblíbená velikost je 4 kb (12 bitů)
      * 1rozměrná virtuální adresa
      * Page table
        + V paměti pro každý proces
        + Indexovaná podle čísla stránky
    - Problémy
      * Velikost stránkovací tabulky … může existovat 220 stránek.
        + Řešení: víceúrovňové strákování
      * Rychlost přístupu
        + Pro každý přistup do paměti bych musel sáhnout dvakrát do paměti … → potřebujeme cache 😊
    - Algoritmus:



Tohle dělá hardware, nikoliv OS!