PB153
Operační systémy a jejich rozhraní

Něco málo o hardwaru

Historie: dávkové systémy

- První počítače
  - obrovské stroje zabírající patra budov
  - velice drahé
  - I/O zařízení: děrné štítky, děrná páska, magnetická páska
  - job: program + data + řídící informace
  - řídící informace
    - návod co má počítač s programem a daty dělat
    - typicky kompilace + sestavení (linkování knihoven) + běh programu + dump výsledku (příp. také paměti a registrů)
    - Vznikají JCL – Job Control Language
  - jeden uživatel: programátor a operátor v jedné osobě
  - OS
    - Neustále v paměti
    - Úkolem je pouze předání řízení z jednoho jobu na druhý
  - využití počítače nebylo efektivní

Historie: dávkové systémy (2)

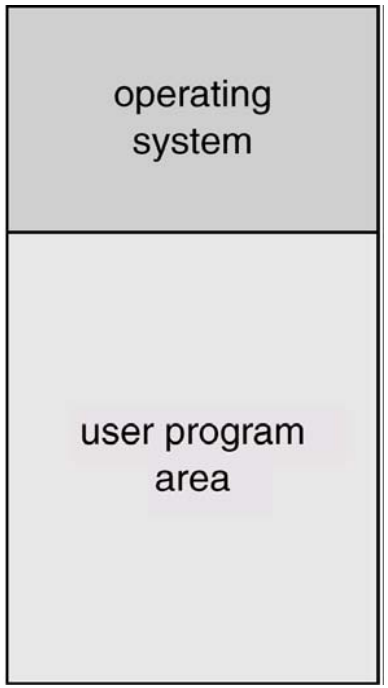
- Později
  - Počítače se zmenšují (střediskové počítače)
  - Oddělení role operátora a programátora
  - Operátoři úlohy (joby) přeskládají do dávek, tak aby se seskupily úlohy podobného typu
    - např. zavede se překladač a ten se využije pro přeložení hned několika programů za sebou
  - Přesto stále nebyly drahé počítače efektivně využity
    - I/O operace byly a stále jsou řádově pomalejší než procesor (CPU)
    - vždy, když program čeká na vyřízení I/O požadavku je CPU nevyužit
    - řešení: multiprogramování

Historie: multiprogramování

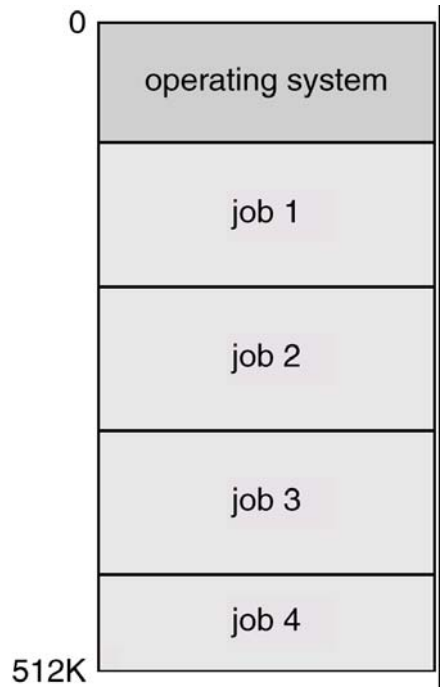
- Multiprogramování zvyšuje využití CPU
  - přidělování CPU jednotlivým úlohám tak, aby CPU byl využit (téměř) vždy
- V paměti je zároveň několik úloh současně
  - ne však nutně všechny
  - plánování úloh (job scheduling) – které úlohy umístit do paměti
  - musíme zajistit ochranu úloh navzájem
- CPU je přidělen úloze, jakmile úloha požádá o I/O operaci, je úloha pozastavena a CPU dostává jiná úloha
  - pro výběr úlohy, která dostane CPU musíme mít CPU plánovací algoritmus
  - jakmile je I/O operace dokončena, je úloha opět přemístěna do fronty úloh připravených ke spuštění
  - CPU je vytížen, dokud mám úlohy, které nečekají na dokončení I/O operací
  - dokud úloha nepožádá o I/O operaci, tak má CPU k dispozici
- Jde tedy „pouze“ o efektivnost využití CPU

Historie: obsazení paměti

- V paměti vždy jen jedna úloha



- V paměti několik úloh, běží jen jedna z nich



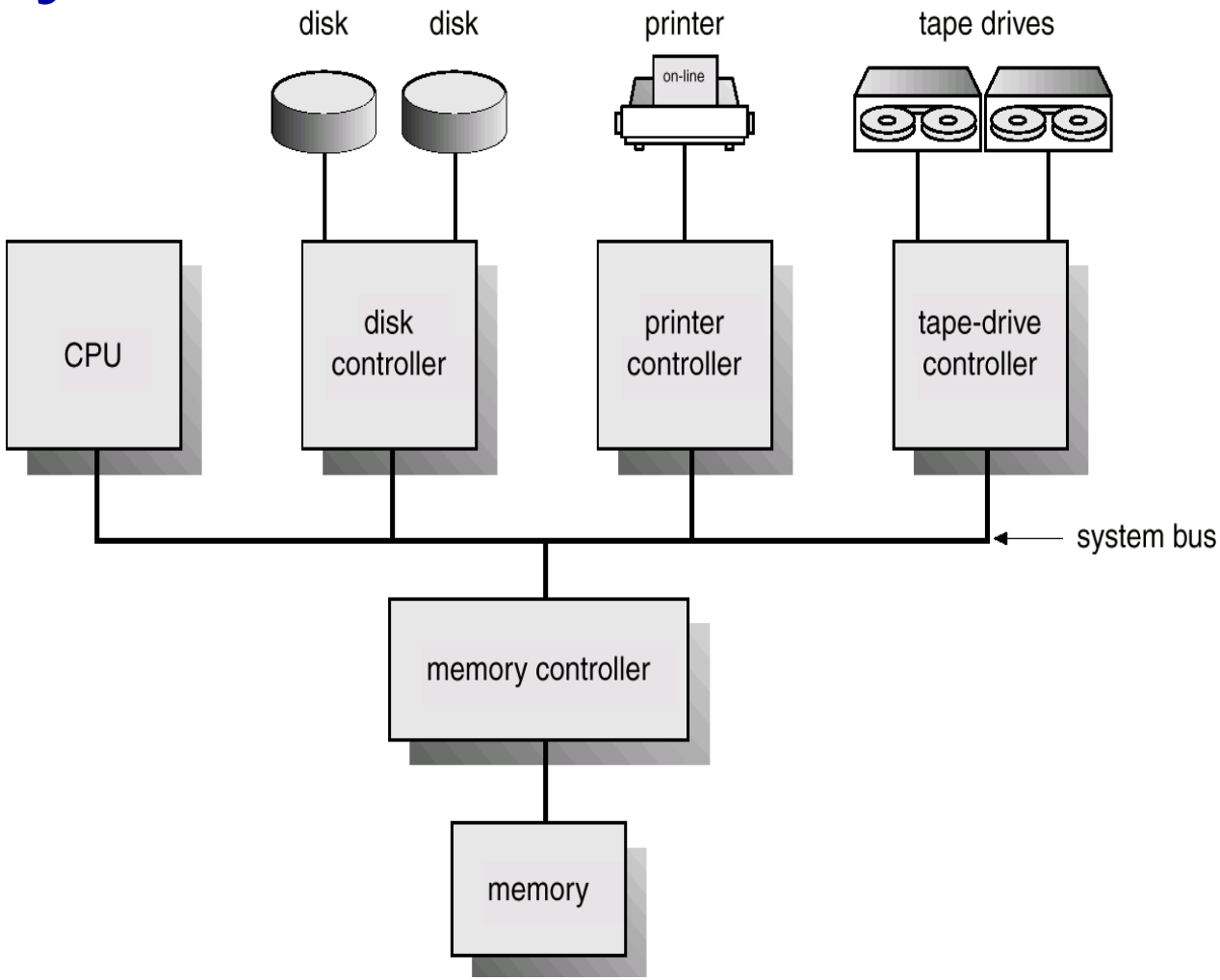
Historie: multitasking

- Time-sharing neboli multitasking
  - Logické rozšíření multiprogramování, kdy úloha (dočasně) ztrácí CPU nejen požadavkem I/O operace, ale také vypršením časového limitu
  - CPU je multiplexován, ve skutečnosti vždy běží jen jedna úloha, mezi těmito úlohami se však CPU přepíná, takže uživatelé získají dojem, že úlohy jsou zpracovávány paralelně

Historie: multitasking (2)

- Multitaskingový systém umožňuje řadě uživatelů počítačový systém *sdílet*
- Uživatelé mají dojem, že počítačový systém je vyhrazen jen pro ně
- Oproti pouhému multiprogramování snižuje dobu odezvy (response time) interaktivních procesů
- Multitaskingové systémy jsou již značně komplexní
  - správa a ochrana paměti, virtuální paměť
  - synchronizace a komunikace procesů
  - CPU plánovací algoritmy, souborové systémy

Architektura počítačového systému



Základní vlastnosti počítačového systému

- I/O zařízení typicky mají vlastní vyrovnávací paměť (buffer)
- CPU a I/O zařízení mohou pracovat paralelně
  - např. řadič disku může ukládat data na disk a CPU může něco počítat
  - pokud CPU a I/O zařízení pracují paralelně, měli by se nějak synchronizovat
    - neustálé zjišťování stavu
    - interrupt

Procesor bez přerušení

- Neustálý cyklus
  - loop
  - získej další instrukci
  - proveď instrukci
  - end loop
- Pokud chci provést I/O operaci a dozvědět se, že již byla dokončena, musím se periodicky ptát I/O zařízení
- CPU není v těchto chvílích efektivně využit nebo je programování extrémně náročné

Procesor s přerušením

- Základní cyklus je obohacen o kontrolu příznaku přerušení
  - loop
  - získej další instrukci
  - proveď instrukci
  - je-li nastaven příznak přerušení (a přerušení je povoleno), ulož adresu právě prováděného kódu a začni provádět kód obslužné rutiny
  - endloop

Přerušení

- Přerušení je signál od I/O zařízení, že se stalo něco, co by OS měl zpracovat
- Přerušení přichází asynchronně
  - OS nemusí aktivně čekat na událost (efektivní využití CPU)
  - při výskytu události se automaticky volá patřičná obslužná rutina
  - umístění obslužných rutin pro jednotlivé typy událostí je typicky dáno tabulkou (v paměti) adres rutin (tzv. vektor přerušení)
- Aby nedocházelo ke „ztrátě“ přerušení, je při zpracování přerušení další přerušení zakázáno (maskováno)
  - aby se nepřerušovala rutina obsluhující přerušení
  - časově náročnější obslužné rutiny však mohou další přerušení explicitně povolit

Přerušení (2)

- Operační systém je řízený přerušeními
  - (interrupt driven)
  - CPU zpracovává uživatelský proces, při příchodu přerušení je spuštěna obslužná rutina OS
    - návratovou adresu ukládá procesor
    - hodnoty použitých registrů ukládá ovládací rutina
  - Přerušení nemusí být generováno jen HW, přerušení je možné vyvolat i softwarovými prostředky (tzv. „trap“ – jde vlastně o synchronní přerušení)
    - chyby – dělení nulou
    - speciální instrukce (např. INT – typicky slouží k systémovému volání)

Příklad: tabulka přerušení Intel8086&MS-DOS

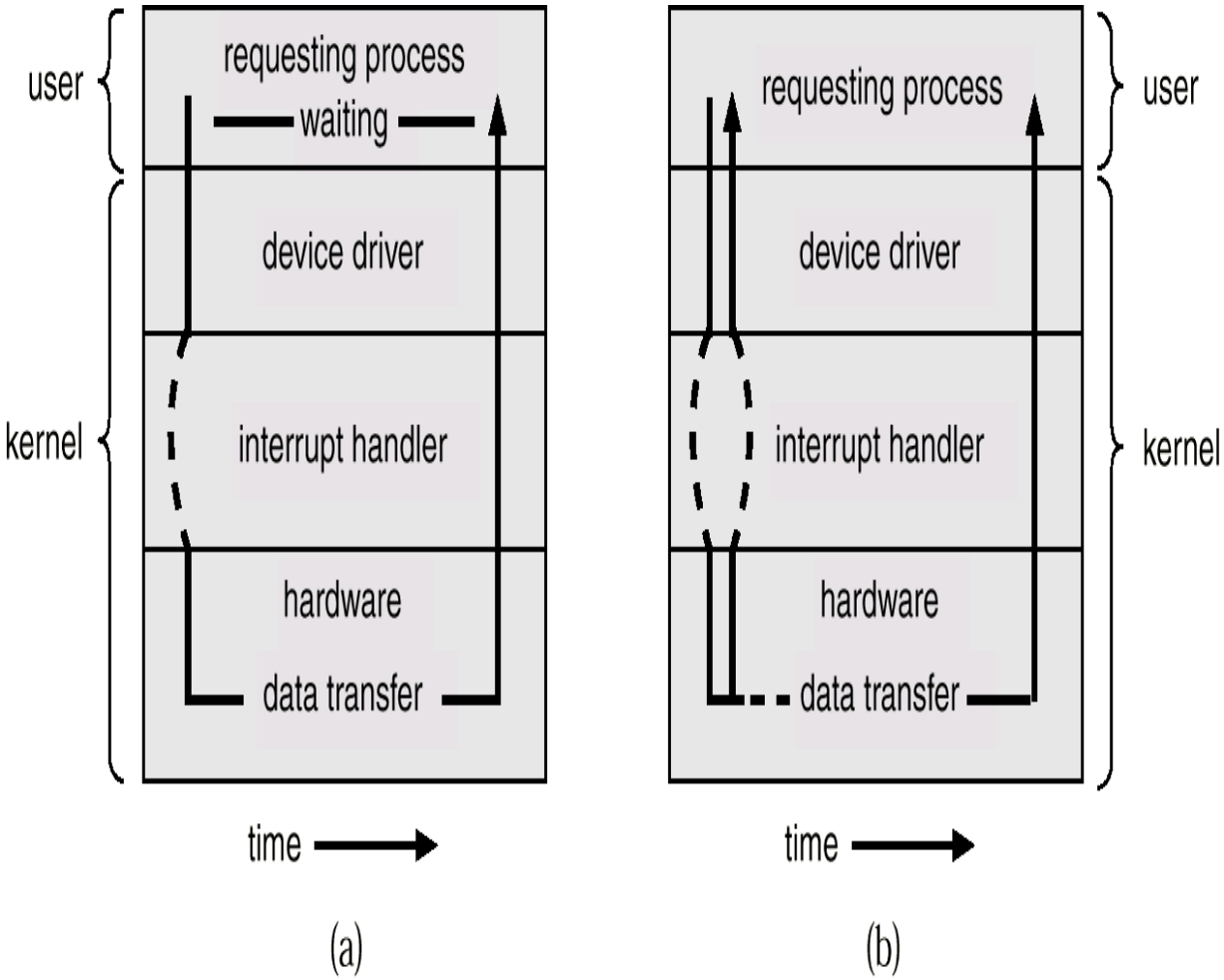
Interrupts	Services	Pointers
INT 00H Division by 0	INT 10H Video services	INT 1dH Video parms
INT 01H Single-step	INT 11H Equipment list	INT 1eH Diskette parms
INT 02H Non-Maskable	INT 12H Usable Mem Size	INT 1fH Graphics chars
INT 03H Breakpoint	INT 13H Disk I/O	
INT 04H Overflow	INT 14H Serial Port I/O	
INT 05H Print Screen	INT 15H AT Services	INT 41H hard disk 0 parms
INT 06H (reserved)	INT 16H Keyboard I/O	INT 46H hard disk 1 parms
INT 07H (reserved)	INT 17H Printer I/O	
INT 08H Timer	INT 18H ROM-BASIC	INT 43H EGA graphix chars
INT 09H Keyboard	INT 19H Bootstrap	
INT 0aH-0dH (hw ints)	INT 1aH Time I/O	INT 4aH user alarm addr
INT 0eH Diskette	INT 1bH Keyboard Break	INT 50H CMOS timer int
INT 0fH (hdwr int)	INT 1cH User timer Int	
	INT 20H-2fH DOS Interrupts	
	INT 33H Mouse Support	
	INT 67H Expanded Memory Fns	

Vektor přerušení začíná na adrese 0000:0000, přerušení je celkem 256

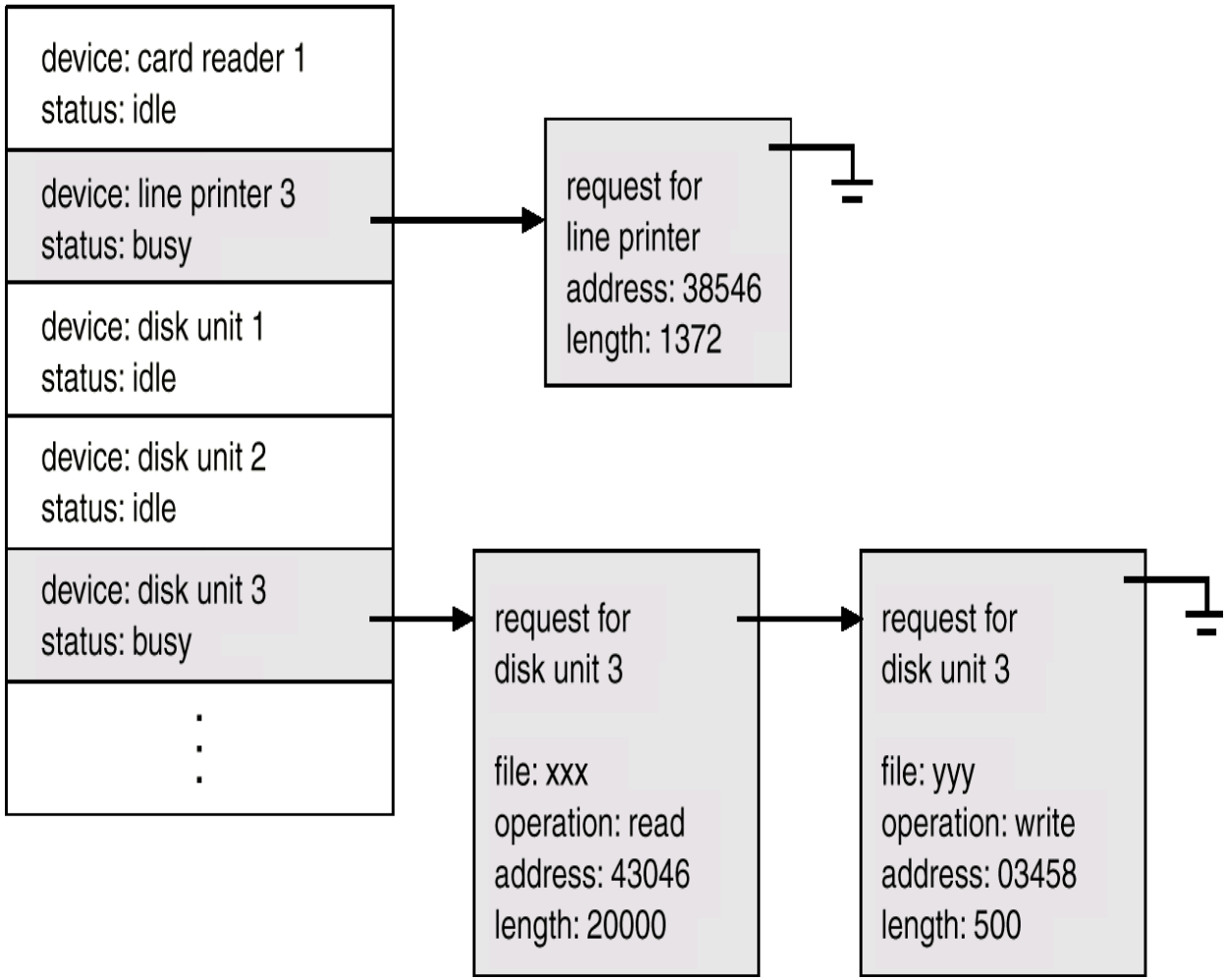
I/O: dva přístupy

- Synchronní
  - proces požádá o I/O operaci, řízení se uživatelskému procesu vrací až po ukončení I/O operace
  - výhody: jednoduchost, vyhýbám se souběžnému zpracování I/O požadavků
  - nevýhody: možná neefektivnost
- Asynchronní
  - proces požádá o I/O operaci a řízení se procesu vrací okamžitě
  - výhody: je možné paralelně pracovat s několika I/O zařízeními
  - nevýhody: komplexnější systém, potřebujeme tabulku stavu I/O zařízení, příkazy pro počkání na dokončení I/O operace apod.

Synchronní vs. asynchronní I/O



Tabulka stavu I/O zařízení



DMA, Direct Memory Access

- Způsob jak rychle přenášet data mezi I/O zařízením a pamětí
- CPU nemusí přenášet data po jednom bajtu, požádá řadič o přenos celého bloku dat
- CPU požádá o přenos dat, po skončení přenosu dat se o tom dozví pomocí přerušení
  (tj. 1 přerušení/blok dat ne 1 přerušení/bajt-slovo)
- V době přenosu dat soupeří I/O řadič a CPU o přístup do paměti (oba pracují se „stejnou“ pamětí)

Struktura paměti – primární paměť

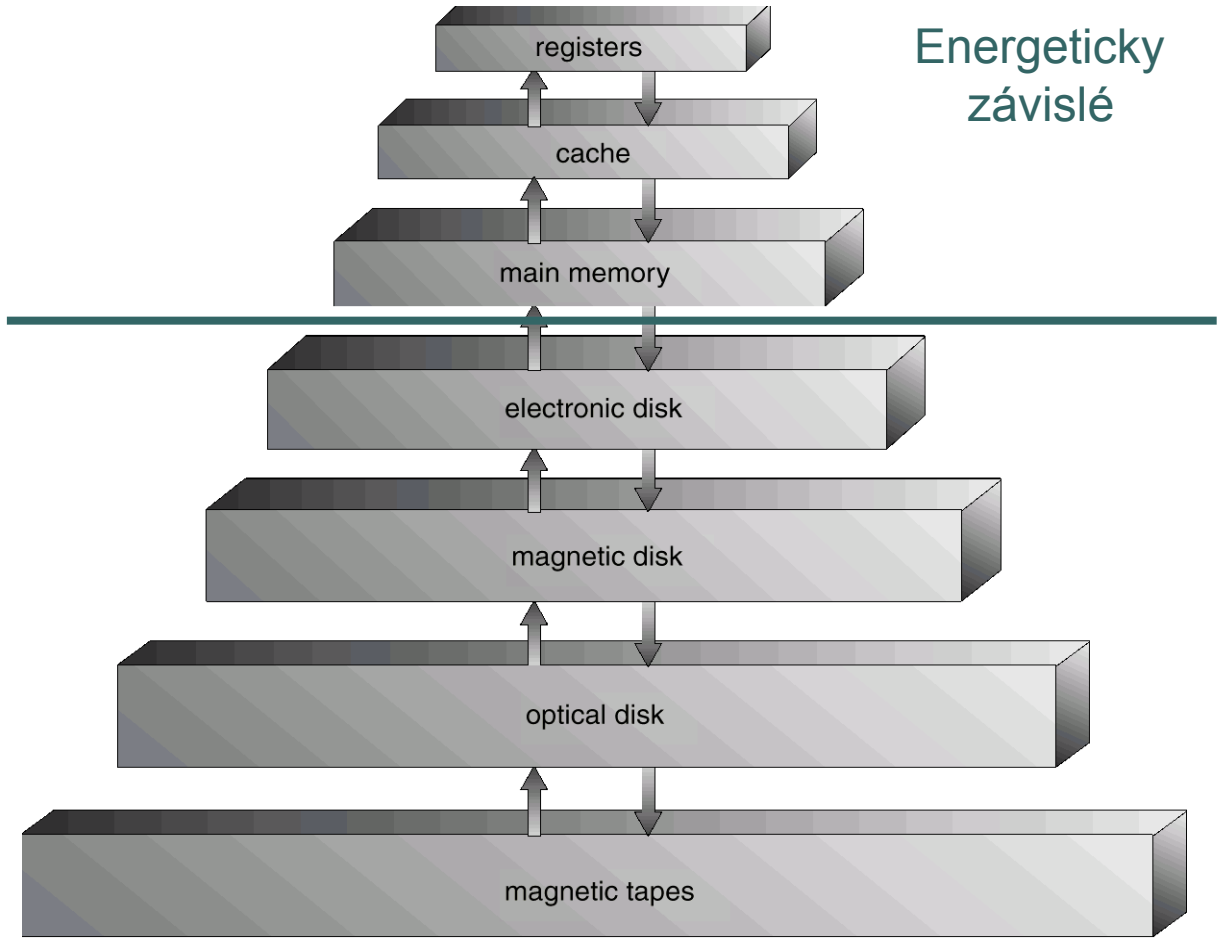
- Primární paměť (též operační paměť, hlavní paměť)
  - jediná větší paměť, kterou může CPU přímo adresovat
  - typicky energeticky závislá (volatilní)
  - současná kapacita: stovky MB až desítky GB, současná rychlost: nanosekundy

Struktura paměti – sekundární paměť

- Sekundární paměť rozšiřuje paměťovou kapacitu počítačového systému
  - energeticky nezávislá
  - vysoká paměťová kapacita (dnes gigabajty až terabajty)
  - relativně pomalá doba přístupu (velice variabilní – od mikrosekund až po jednotky sekund)
  - v dnešní době je nejběžnější magnetický disk

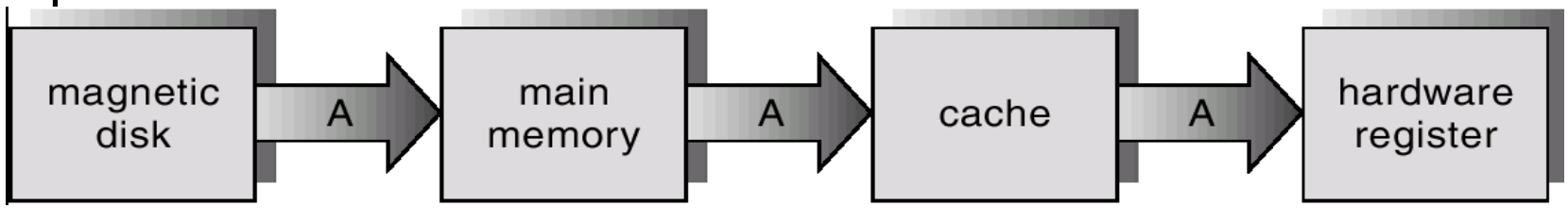
Hierarchie paměti

- Rychlost ↓
- Kapacita ↑
- Cena ↓



Kešování, cache, mezipaměť

- Použití rychlejší paměti pro uchování posledně použitých dat z paměti pomalejší
  - např. cache procesoru vs. hlavní paměť
  - nebo hlavní paměť vs. disková paměť
  - pokud jsou data v cachi použijí se tato data, pokud ne, musí se získat z pomalejší paměti, zároveň se přenesou i data okolní (princip časové a prostorové lokálnosti)
- Velikost cache paměti je omezená
- Data se zároveň nacházejí v několika úrovních paměti – problém udržení konzistence



Ochranné funkce HW

- Je-li v paměti několik procesů, nechceme, aby se procesy mohly navzájem negativně ovlivňovat
  - přepisování paměti
  - nespravedlivé získání času CPU
  - souběžný nekoordinovaný přístup k I/O prostředkům
- Proto OS musí tomuto ovlivňování zabránit
  - často to však nemůže zajistit sám a potřebuje podporu HW – např. při ochraně přístupu do paměti, přístupu k I/O portům

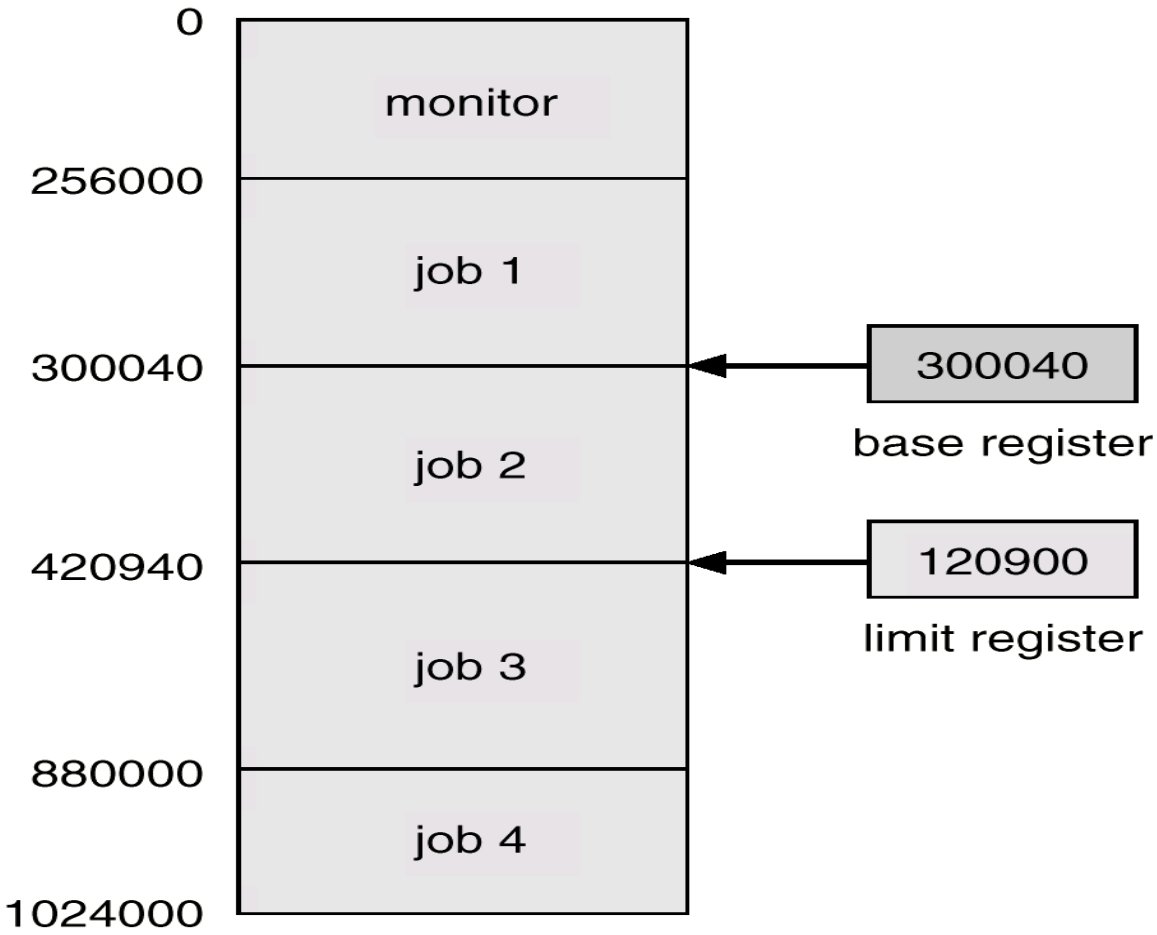
Režimy procesoru

- Běžný způsob ochrany je dvojí režim činnosti procesoru
  - uživatelský režim
  - privilegovaný režim
- Některé instrukce je možné provést jen v privilegovaném režimu
  - např. instrukce pro I/O, nastavování některých registrů (např. některé segmentové registry)
- Z privilegovaného režimu do uživatelského režimu se CPU dostane speciální instrukcí, z uživatelského režimu do privilegovaného režimu se CPU dostává při zpracování přerušení

Ochrana paměti

- Minimálně musíme chránit vektor přerušení a rutiny obsluhy přerušení
  - jinak by bylo možné získat přístup k privilegovanému režimu procesoru
- Každému procesu vyhradíme jeho paměť, jinou paměť nemůže proces používat
  - ochranu zajišťuje CPU na základě registrů/tabulek nebo principů nastavených OS
  - např. báze & limit – proces má přístup jen k adresám báze + 0 až báze + limit
  - přístup k nepovoleným adresám způsobí přerušení – to zpracovává OS a např. ukončí činnost procesu

Báze + limit



- Jednoduché na implementaci
  - dva registry, jejichž nastavování je privilegovanou operací
  - CPU kontroluje, zda adresy, které proces používá spadají do rozsahu daného registry
- Jak řešit požadavek procesu o přidělení dalšího bloku paměti?

Ochrana CPU

- Jak zaručit, že vládu nad procesorem (tj. jaký kód bude CPU vykonávat) bude mít OS?
- Časovač
  - časovač generuje přerušení
  - přerušení obsluhuje OS
    - ten rozhodne co dál
    - např. odebere jednomu procesu, vybere další připravený proces a ten spustí (změní kontext)
  - časovač může generovat přerušení pravidelně nebo je příchod přerušení programovatelný (privilegovanou instrukcí)