PB169 Počítačové sítě a operační systémy

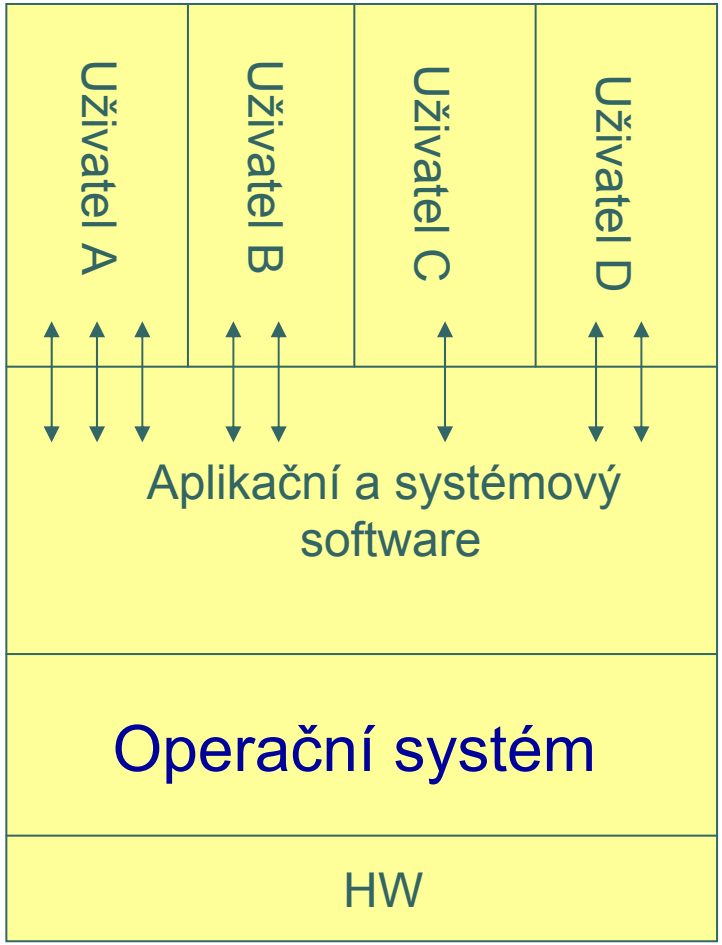
Zdeněk Říha
zriha@fi.muni.cz

Marek Kumpošt
kumpost@fi.muni.cz

Literatura

1. Přednášky
2. PPT prezentace
3. Prezentace z PB152, PB153, PA151, PA159, PA160, PB156, PV169
4. Silberschatz, Galvin, Gagne: *Operating System concepts*, 7th edition, Wiley, 2004, ISBN 0-471-69466-5
   PPT z PB169 jsou založeny na PPT k této knize a jsou modifikovány. © Silberschatz, Galvin and Gagne, 2005
5. Stallings: *Local and Metropolitan Area Networks,* Softcover, Prentice Hall, ISBN 0-13-018653-8

Počítačový systém



- Hardware
  - CPU
  - Paměti
  - I/O
- Operační systém
- Aplikační a systémový SW
- Uživatelé

Uživatelský pohled na OS

- Dnes používáme typicky desktopy vyhrazené pro jednoho uživatele
  - OS navržen pro jednoduché používání, výkon systému je brán na zřetel, ovšem na využití zdrojů není kladen důraz
- Dříve často terminály, OS plní požadavky programů řady uživatelů
  - důraz na využití zdrojů počítače
  - férové užívání zdrojů jednotlivými uživateli

Systémový pohled na OS

- OS především jako správce prostředků počítače
  - CPU, operační paměť, disková paměť, I/O zařízení
- Koordinátor, řídící složka
  - řídí spouštění programů, zabraňuje chybám a vzájemnému ovlivňování

Definice OS

- Neexistuje universální a všeobecně platná definice OS
- Stejně tak není jednotný názor na to, co všechno zahrnuje OS (jádro, systémové a aplikační programy)
  - OS = to co výrobce dá do krabice
  - OS = jádro (tj. část, která je neustále spuštěna)
- Raději definujeme OS tím co dělá, než tím co vlastně je.
- Analogie s „vládou“

Primární cíle OS

- Při návrhu OS jsou stanoveny podmínky/cíle, které má OS splňovat
  - uživatelská přívětivost
  - efektivní využití (drahých) zdrojů
  - ne všechny podmínky/cíle však implikují jasné způsoby návrhu/implementace (bezchybnost, spolehlivost)
- Za 45 let vývoje se OS značně změnily: od jednoduchých textově zaměřených po komplexní systémy s komfortním GUI.

Typy počítačových systémů

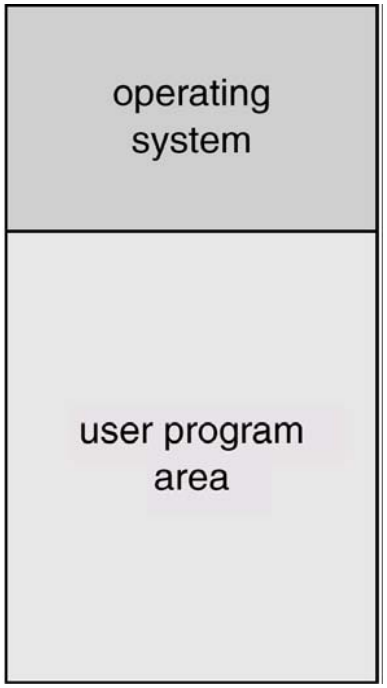
- Stolní systémy
  - Vyhrazené pro 1 uživatele
- Paralelní systémy
  - Sdílí paměť a hodinový signál, SMP, AMP
- Distribuované systémy
  - Vlastní paměť
- Real-time systémy
  - Pevně stanovené časové limity
- Kapesní systémy

Historie: multiprogramování

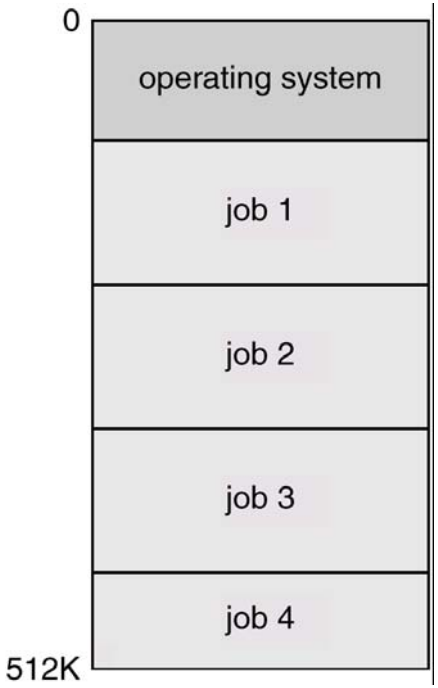
- Multiprogramování zvyšuje využití CPU
  - přidělování CPU jednotlivým úlohám tak, aby CPU byl využit (téměř) vždy
- V paměti je zároveň několik úloh současně
  - ne však nutně všechny
  - plánování úloh (job scheduling) – které úlohy umístit do paměti
  - musíme zajistit ochranu úloh navzájem
- CPU 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



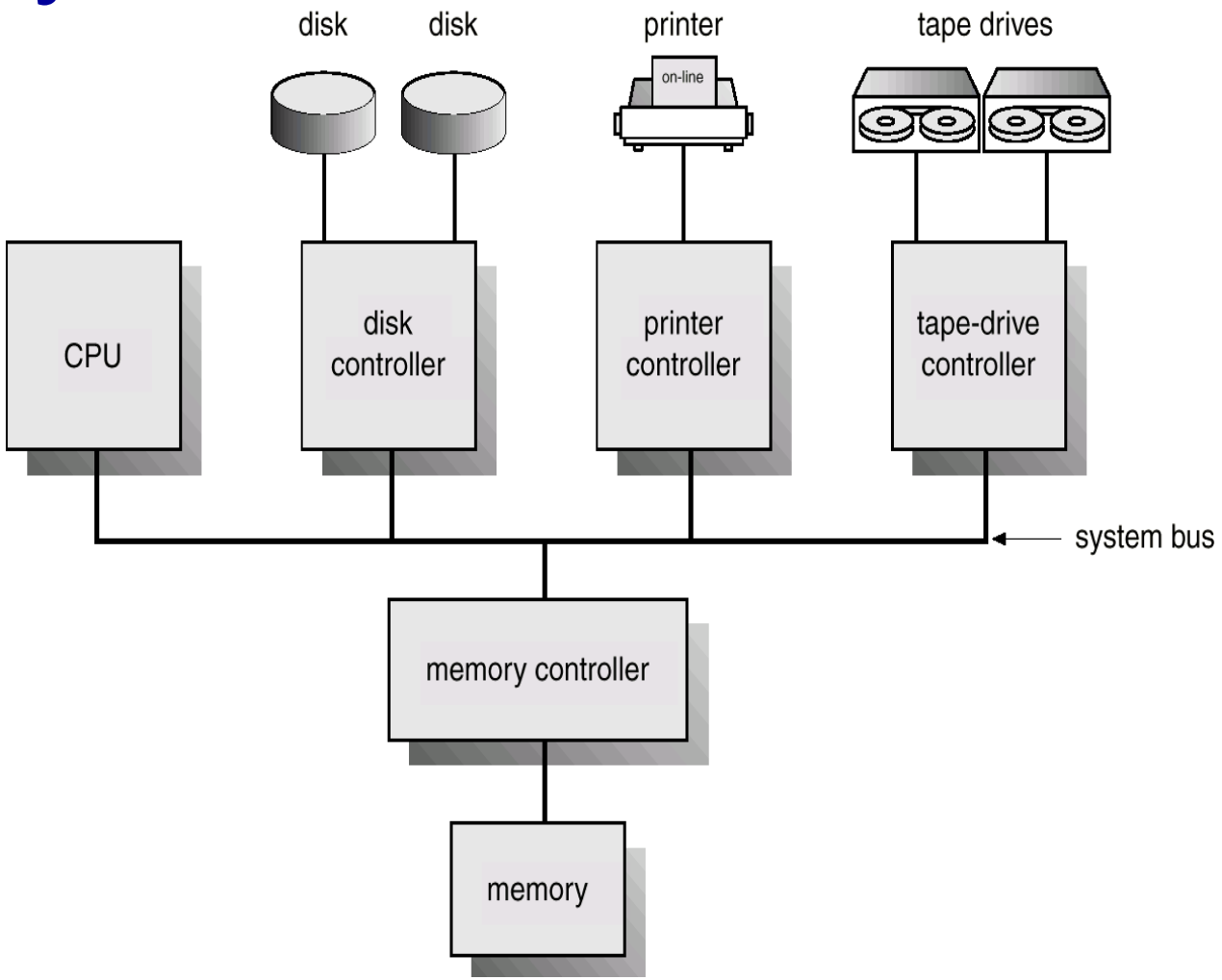
Historie: multitasking

- 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)

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)

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

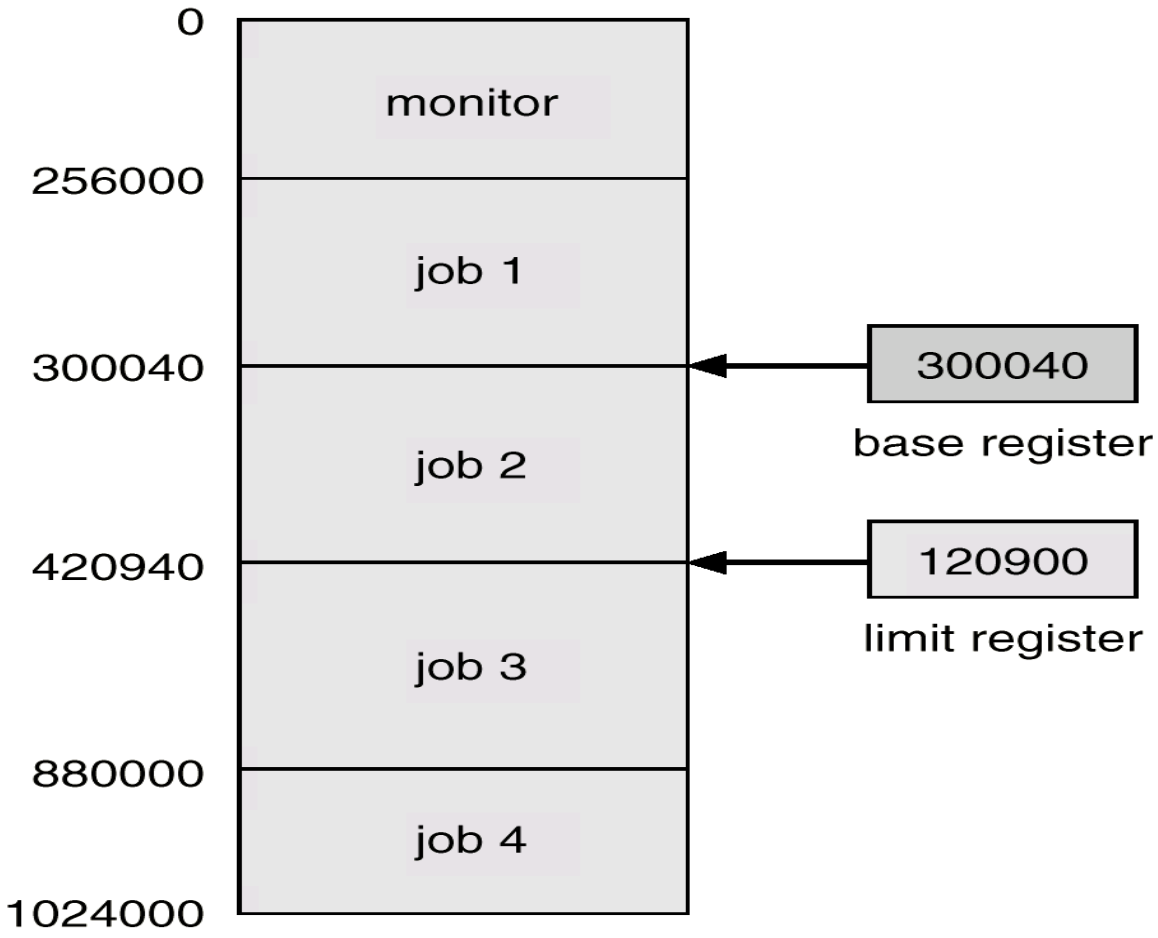
Režimy procesoru

- 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)

Komponenty OS

- Správa procesů
- Správa operační paměti
- Správa souborů
- Správa I/O zařízení
- Správa sekundární paměti
- Správa síťových služeb
- Ochranný systém
- Interpret příkazů (shell)

Systémová volání

- Systémová volání tvoří rozhraní mezi uživatelským procesem a OS
  - typicky jsou popsána jako instrukce assembleru a jsou uvedena v programátorském manuálu k OS
  - vyšší programovací jazyky obsahují některé funkce, které odpovídají systémovým voláním (např. open, write) a dále knihovní funkce, které poskytují vyšší funkčnost a v rámci této spouští (třeba hned několik) systémových volání (např. fopen, fwrite).

Systémová volání (2)

- Různé OS a různé HW platformy mívají různé způsoby jak volat služby OS a různou strukturu těchto služeb
- Nicméně existují určité standardy, které usnadňují přenositelnost programového kódu
  - v oblasti UNIXu: POSIX
  - V oblasti Windows: Win32 (Windows API)
- Teoreticky kód který bude psán podle standardu bude přeložitelný na kompatibilních platformách, v praxi však existuje celá řada verzí standardu a mnoho výjimek co je a není implementováno

Způsob předání parametrů

- Registry
  - `mov ah,01h`
  - `mov cx,2000h`
  - `int 10h`
- Zásobník
  - `mov ax,0001h`
  - `push ax`
  - `int 10h`
- Blok (struktura, tabulka) v paměti & pointer
  - `mov ax,0001h`
  - `mov [tabulka1],ax`
  - `mov bx,tabulka1`
  - `int 10h`

Příklad: Linux

- Systémová volání
  - standardně přes `int 0x80`
  - nově i přes instrukci `syscall (sysenter)`
- Číslo systémového volání
  - v registru `eax`
  - v jádře 2.2 přibližně 200 volání, v jádře 2.6 přes 300 volání
- Parametry systémového volání
  - v registrech `ebx, ecx, edx, esi, edi, ebp`
  - kromě fce 117 (parametrem je odkaz na strukturu)
- Výsledek
  - uložen v `eax`

Příklad: Linux (2)

- Hello world v assembleru pod Linuxem

```
section .text
global _start
msg db 'Hello, world!',0xa
len equ $ - msg
_start:
    mov edx,len ;message length
    mov ecx,msg ;message to write
    mov ebx,1 ;file descriptor (stdout)
    mov eax,4 ;system call number (sys_write)
    int 0x80 ;call kernel
    mov eax,1 ;system call number (sys_exit)
    int 0x80 ;call kernel
```

Komplexita OS

- Počet systémových volání OS / API volání

OS	rok	počet systémových volání
UNIX	1971	33
UNIX	1979	47
SunOS	1989	171
4.3 BSD	1991	136
SunOS	1992	219
SunOS	1997	190
Linux	1998	229
NT 4.0	1999	3443

Komplexita OS (2)

- Rozsah zdrojového kódu Windows

Verze	**rok**	**miliony řádků**
NT 3.1	1993	4-5
NT 3.5	1994	7-8
95	1995	10
NT 4.0	1996	11-12
98	1998	18
2000	2000	29
XP	2001	40
2003 Svr	2003	50