



# 02. Koncepce OS

## Procesy, vlákna

ZOS 2012

---

# Vektor přerušení

Kód	Číslo přerušení	Popis
B	INT 00H	Dělení nulou
B	INT 01H	Krokování
B	INT 02H	Nemaskovatelné přerušení
B	INT 03H	Bod přerušení (breakpoint)
B	INT 04H	Přetečení
B	INT 05H	Tisk obrazovky
B	INT 06H	Nesprávný operační kód
B	INT 07H	Není koprocessor
B	INT 08H IRQ0	Přerušení od časovače
B	INT 09H IRQ1	Přerušení od klávesnice
	INT 0aH IRQ2	EGA vertikální zpětný běh
	INT 0bH IRQ3	COM2
	INT 0cH IRQ4	COM1
	INT 0dH IRQ5	Přerušení harddisku
B	INT 0eH IRQ6	Přerušení řadiče disket
	INT 0fH IRQ7	Přerušení tiskárny
B	INT 10H	Služby obrazovky
	INT 11H	Seznam vybavení
	INT 12H	Velikost volné paměti
B	INT 13H	Diskové vstupně-výstupní operace

Příklad možného mapování (IBM PC), může být různé

všimněte si IRQ0 je zde na INT 08H, na vektoru 08H bude adresa podprogramu k vykonání



---

# IRQ – Interrupt Request

**IRQ** – signál, kterým zařízení (časovač, klávesnice) žádá procesor o přerušení zpracovávaného procesu za účelem provedení důležitější akce

**IRQL** – priorita přerušovacího požadavku  
(Interrupt Request Level)

**NMI** – nemaskovatelné přerušení, např. nezotavitelná hw chyba (non-maskable interrupt)

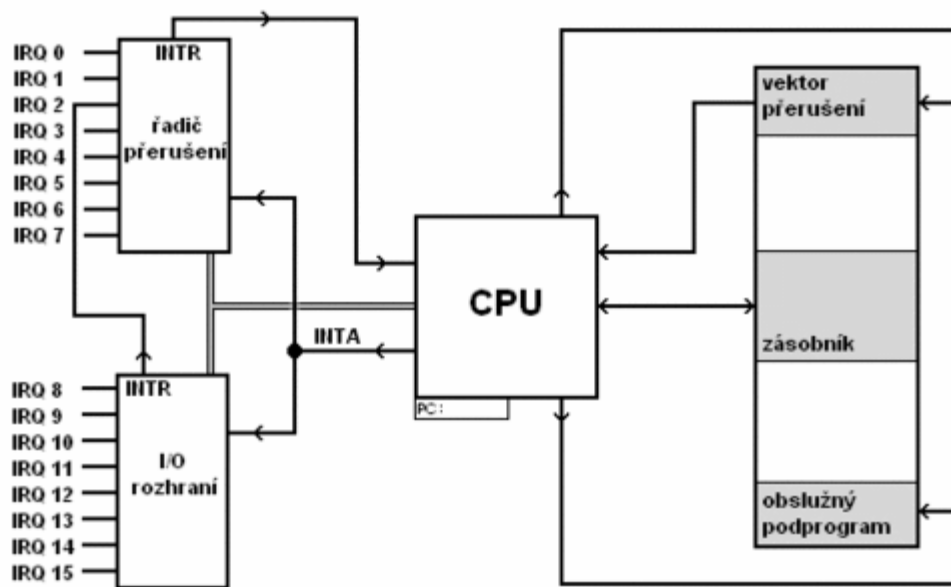
---



# Obsluha HW přerušeni

1. zařízení sdělí **řadiči přerušeni**, že potřebuje přerušeni
  2. řadič upozorní CPU, že jsou čekající (**pending**) přerušeni
  3. až je CPU ochotné přijmout přerušeni tak přerušeni výpočet a zeptá se řadiče přerušeni, které nejdůležitější čeká a spustí jeho obsluhu
  4. uloží stav procesu, provede základní obsluhu zařízení, informuje řadič o dokončení obsluhy, obnoví stav procesu a pokračuje se dále
-

# Řadič přerušení



2 integrované obvody Intel 8259

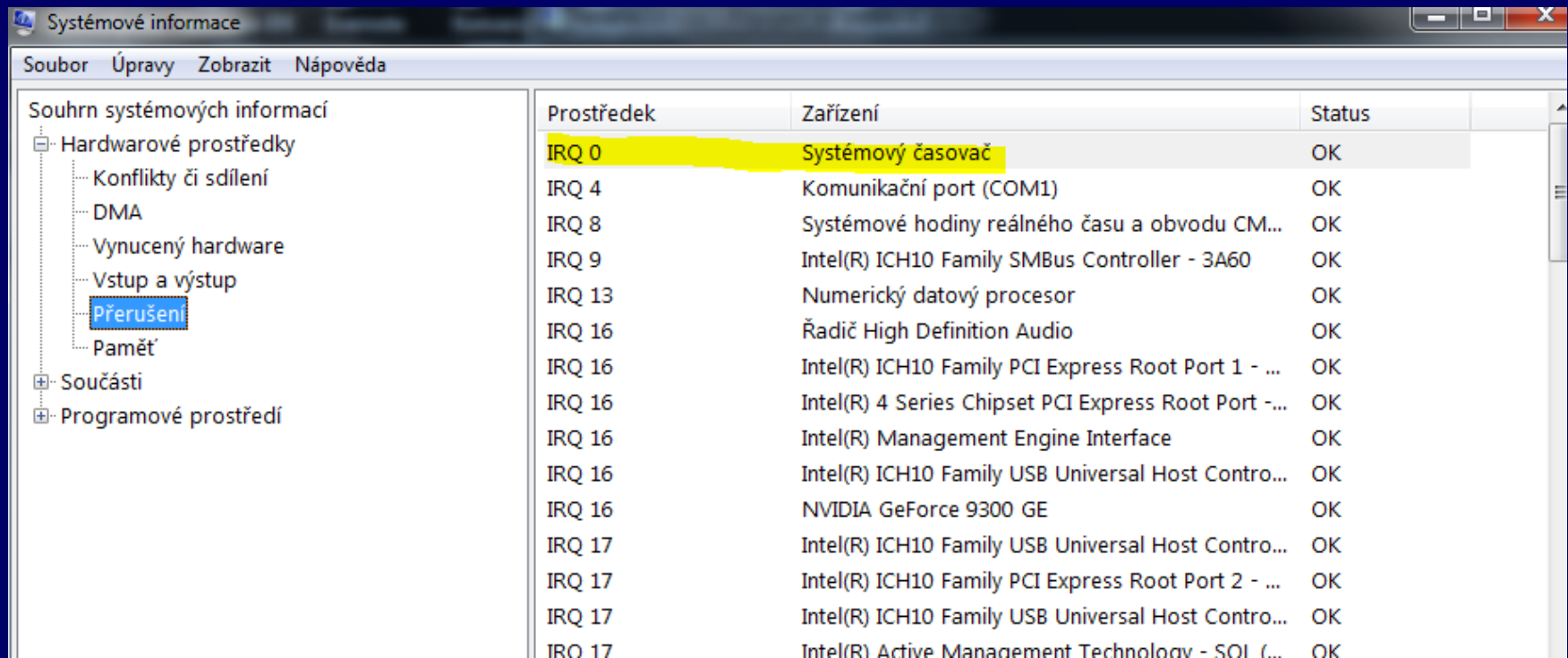
1. spravuje IRQ 0 až 7

(master, na IRQ2 je připojen druhý)

2. spravuje IRQ 8 až 16

novější systémy Intel APIC Architecture (typicky 24 IRQ)

# IRQ pod Win: msinfo32.exe



Prostředek	Zařízení	Status
IRQ 0	Systémový časovač	OK
IRQ 4	Komunikační port (COM1)	OK
IRQ 8	Systémové hodiny reálného času a obvodu CM...	OK
IRQ 9	Intel(R) ICH10 Family SMBus Controller - 3A60	OK
IRQ 13	Numerický datový procesor	OK
IRQ 16	Řadič High Definition Audio	OK
IRQ 16	Intel(R) ICH10 Family PCI Express Root Port 1 - ...	OK
IRQ 16	Intel(R) 4 Series Chipset PCI Express Root Port - ...	OK
IRQ 16	Intel(R) Management Engine Interface	OK
IRQ 16	Intel(R) ICH10 Family USB Universal Host Contro...	OK
IRQ 16	NVIDIA GeForce 9300 GE	OK
IRQ 17	Intel(R) ICH10 Family USB Universal Host Contro...	OK
IRQ 17	Intel(R) ICH10 Family PCI Express Root Port 2 - ...	OK
IRQ 17	Intel(R) ICH10 Family USB Universal Host Contro...	OK
IRQ 17	Intel(R) Active Management Technology - SQL (...)	OK



---

# Koncepce OS

## □ Základní abstrakce:

- procesy
  - soubory
  - uživatelská rozhraní
-



# Procesy

- **Proces** – instance běžícího programu
  - **Adresní prostor** procesu
    - **MMU (Memory Management Unit)** zajišťuje soukromí
    - **kód** spustitelného programu, **data**, **zásobník**
  - S procesem sdruženy **registry** a další info potřebné k běhu procesu = **stavové informace**
    - **registry** – čítač instrukcí **PC**, ukazatel zásobníku **SP**, univerzální registry
-





# Registry (příklad architektura x86)

- malé úložiště dat uvnitř procesoru
  - **obecné registry**
    - **EAX, EBX, ECX, EDX** .. jako 32ti bitové
    - **AX, BX, CX, DX** .. využití jako 16ti bitové (dolních 16)
    - **AL, AH** .. využití jako 8bitové
  - **obecné registry - uložení offsetu**
    - **SP** .. offset adresy vrcholu **zásobníku (!)**
    - **BP** .. pro práci se zásobníkem
    - **SI** .. offset zdroje (source index)
    - **DI** .. offset cíle (destination index)
-



# Registry

## □ segmentové registry

- CS code segment (kód)
- DS data segment (data)
- ES extra segment
- FS volně k dispozici
- GS volně k dispozici
- SS stag segment (zásobník)



# Registry

## □ speciální

- **IP** .. offset vykonávané instrukce (CS:IP)
- **FLAGS** .. zajímavé jsou jednotlivé bity
  - **IF** .. interrupt flag (přerušeni zakázáno-povoleno)
  - **ZF** .. zero flag (je-li výsledek operace 0)
  - **OF, DF, TF, SF, AF, PF, CF**

bližší info např. [http://cs.wikipedia.org/wiki/Registr\\_procesoru](http://cs.wikipedia.org/wiki/Registr_procesoru)  
jde nám o představu jaké registry a k jakému účelu jsou

# Registry (x86-64)

64bit

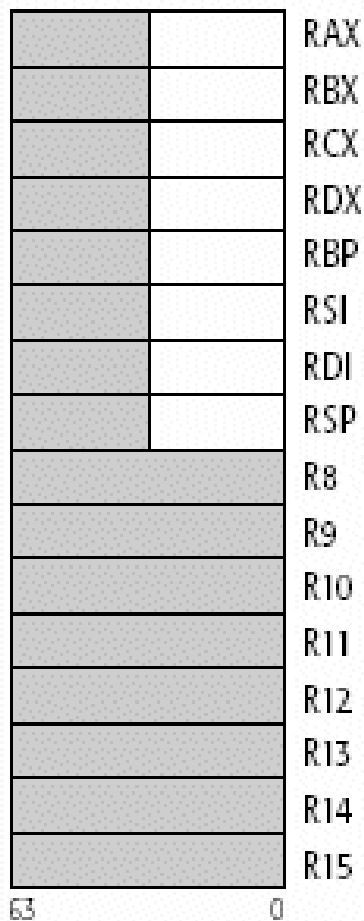


- For 16-bit operations, the two bytes of Register A are addressed as AX
- For 32-bit operations, the four bytes of Register A are addressed as EAX
- For 64-bit operations, the eight bytes of Register A are addressed as RAX

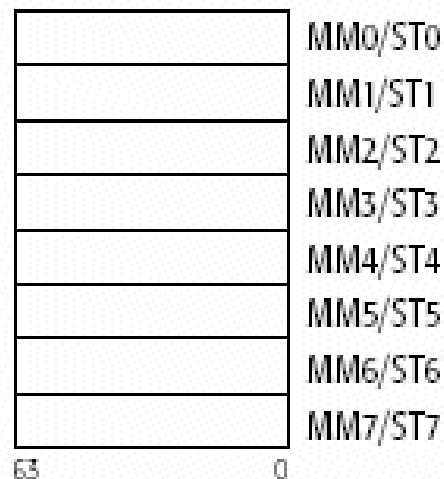
zdroj:

[http://pctuning.tyden.cz/index2.php?option=com\\_content&task=view&id=7475&Itemid=28&pop=1&page=0](http://pctuning.tyden.cz/index2.php?option=com_content&task=view&id=7475&Itemid=28&pop=1&page=0)

### General-Purpose Registers (GPRs)



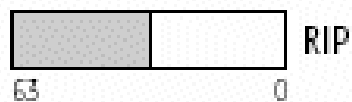
### Multimedia Extension and Floating-Point Registers



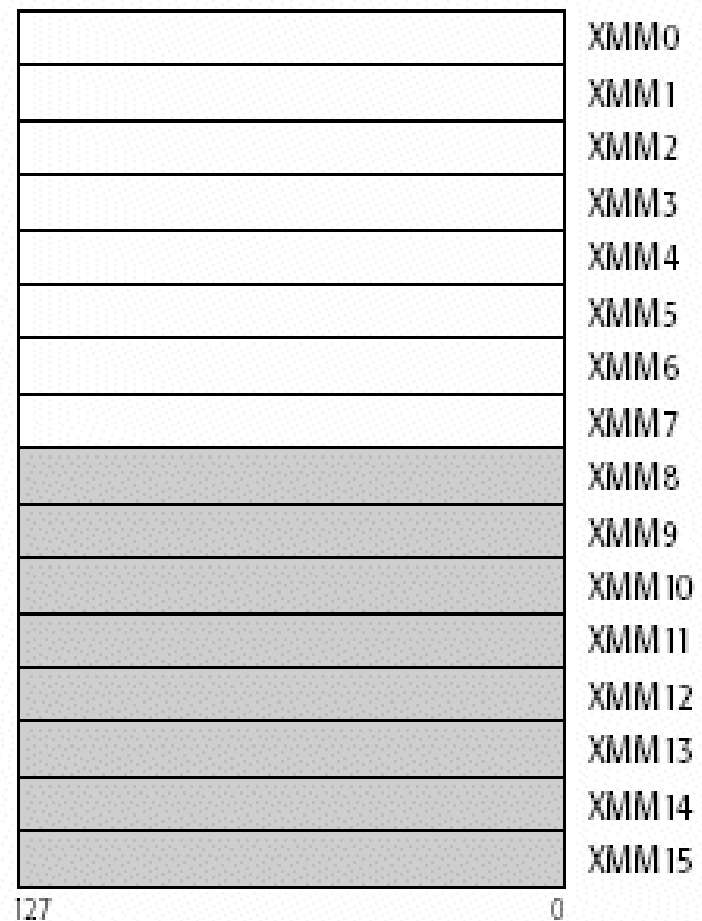
### Flags Register



### Instruction Pointer



### Streaming SIMD Extension (SSE) Registers



-  Legacy x86 Registers, supported in all modes
-  Register Extensions, supported in 64-Bit Mode



# Základní služby OS pro práci s procesy

## □ Vytvoření nového procesu

- `fork` v UNIXu, `CreateProcess` ve Win32

## □ Ukončení procesu

- `exit` v UNIXu, `ExitProcess` ve Win32

## □ Čekání na dokončení potomka

- `wait (waitpid)` v UNIXu,
  - `WaitForSingleObject` ve Win32
-



# Další služby - procesy

- **Alokace** a **uvolnění** paměti procesu
  - **Komunikace** mezi procesy (IPC)
  - **Identifikace** ve víceuživatel. systémech
    - identifikátor uživatele (**UID**)
    - skupina uživatele (**GID**)
    - proces běží s **UID** toho, kdo ho spustil
    - v UNIXu – **UID**, **GID** – celá čísla
  - Problém **uvíznutí** procesu
-



# Soubory

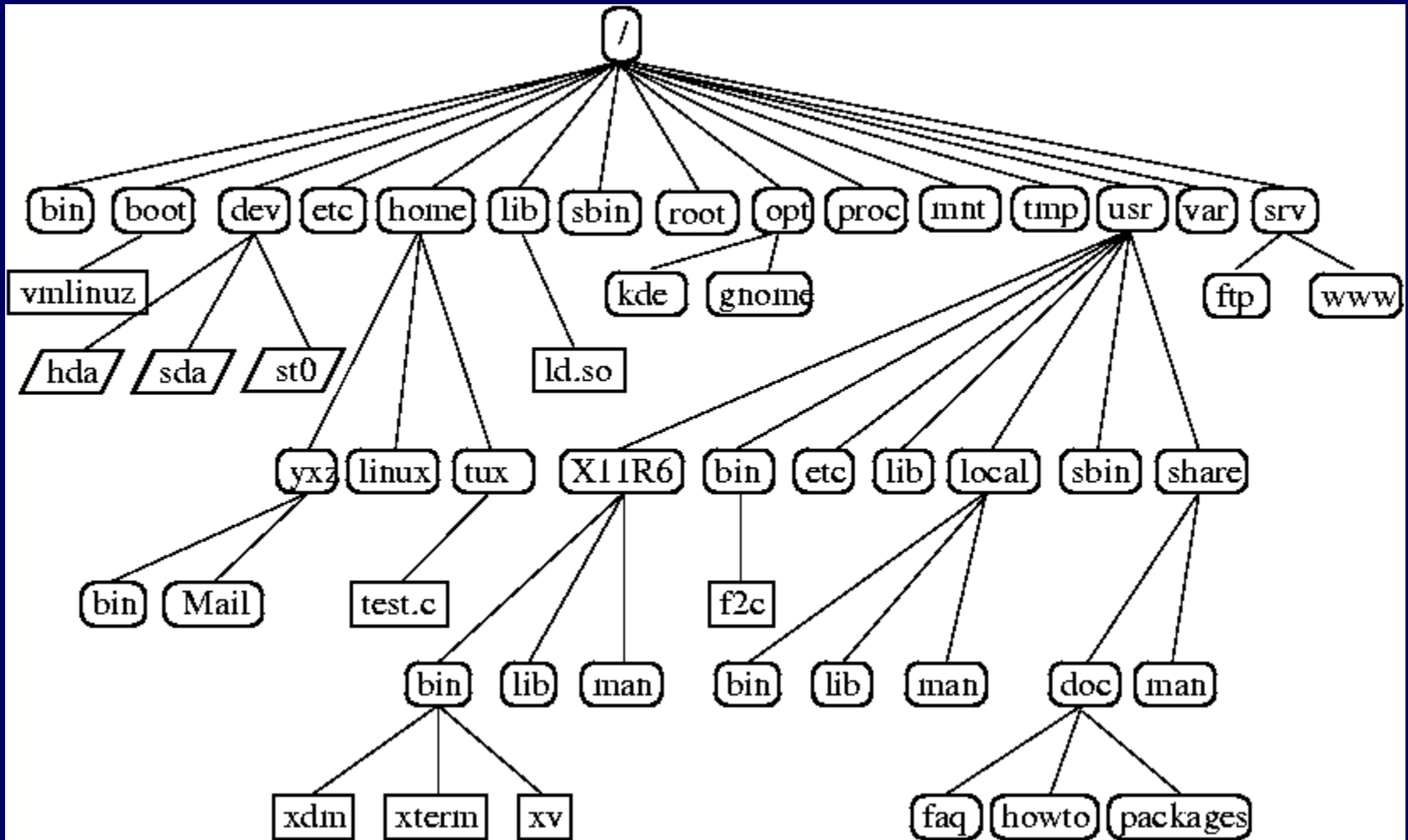
- Zakrytí podrobností o discích a I/O zařízení
  - Poskytnutí abstrakce – **soubor**
  - **Systemová volání**
    - vytvoření, zrušení, čtení, zápis
  - Otevření a uzavření souboru – **open, close**
  - **Sekvenční** nebo **náhodný** přístup k datům
  - Logické sdružování souborů do **adresářů**
  - **Hierarchie** adresářů – stromová struktura
-





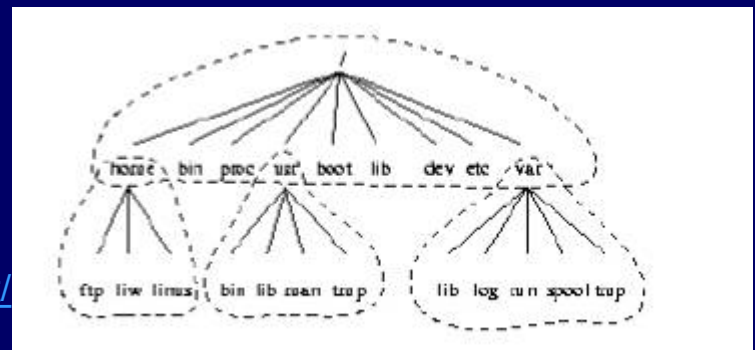
# Soubory II.

- Ochrana souborů, adresářů **přístupovými právy**
    - kontrola při otevření souboru
    - pokud není přístup – chyba
  
  - **Připojitelnost** souborových systémů
    - Windows – disk určený prefixem *C:*, *D:*
    - Unix – *kamkoliv* v adresářovém stromu
-



## Linux – filesystem

zdroj: <http://www-uxsup.csx.cam.ac.uk/pub/doc/suse/suse9.0/userguide-9.0/ch24s02.html>  
<http://www.cs.wits.ac.za/~adi/courses/linuxadmin/content/module2doc.html>



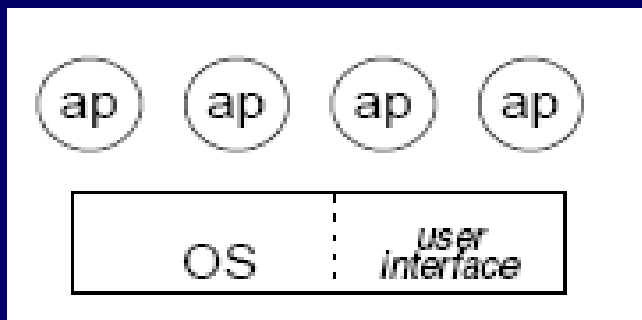


---

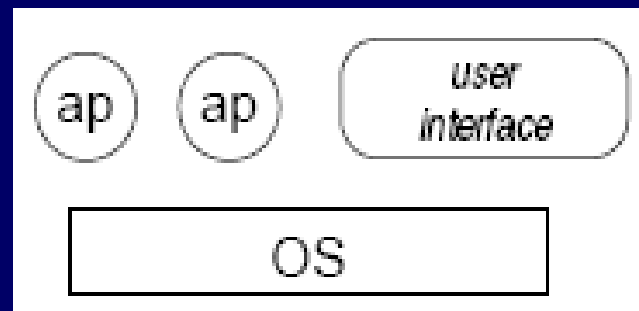
# Uživatelské rozhraní

- řádková – CLI (Command Line Interface)
  - grafická uživ. rozhraní (GUI)
  
  - původně UI součást **jádra**
  - v moderních OS – jedním z programů, možnost náhrady za jiné
-

# UI – obrázky



**UI jako součást jádra**



**UI v uživ. režimu**

Kolik přepnutí kontextu je potřeba?  
vs. vliv na stabilitu jádra OS



# Uživatelské rozhraní - příklady

## □ GUI Linux

- systém **XWindow** (zobrazování grafiky) a grafické prostředí (**správci oken**,...) – programy v uživatelském režimu

## □ Windows NT,2000,XP

- grafická část v jádře
  - logická část (v uživatelském režimu)
  - výkon vs. stabilita
-

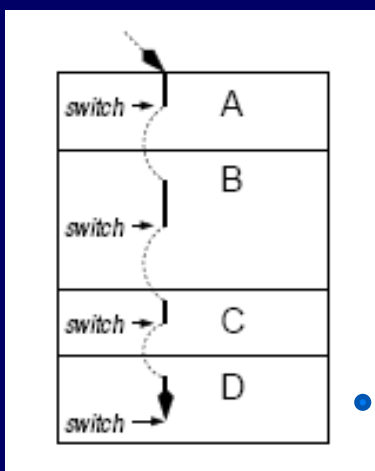


# Proces jako abstrakce

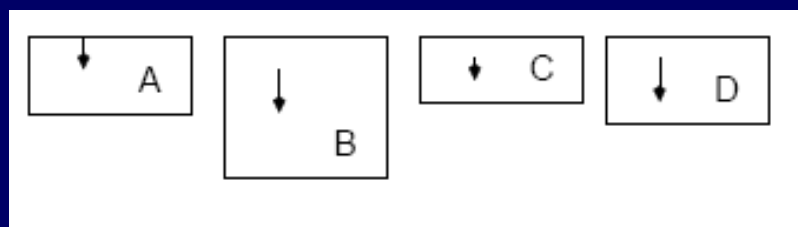
- Běžící SW – organizován jako **množina sekvenčních procesů**
  - **Proces** – běžící program včetně obsahu čítače instrukcí, registrů, proměnných; běží ve vlastní paměti
  - Konceptně každý proces – vlastní **virtuální** CPU
  - **Reálný** procesor – přepíná mezi procesy (multiprogramování)
  - Představa množiny procesů běžících (pseudo)paralelně
-

# Ukázka

4 procesy, každý má vlastní bod běhu (čítač instrukcí)  
pseudoparalelní běh x paralelní (více CPU)



1 procesor



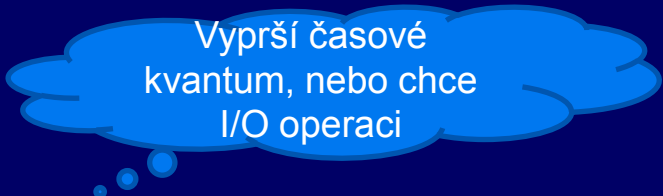
4 procesory

z pohledu uživatele se nám jeví jako paralelní vykonávání procesů



# Pseudoparalelní běh

- **Pseudoparalelní** běh – v jednu chvíli aktivní pouze **jeden** proces

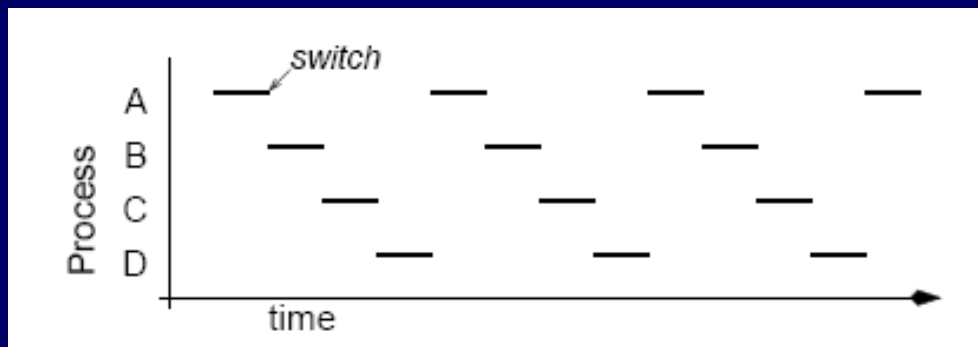


Vyprší časové kvantum, nebo chce I/O operaci

- Po určité době **pozastaven** a spuštěn další
  - Po určité době všechny procesy vykonají část své činnosti
-



# Pseudoparalelní běh



Procesy A, B, C, D se střídají na procesoru  
Z obrázku se zdá, že na procesoru stráví vždy stejnou dobu, ale nemusí tomu tak být – např. mohou požadovat I/O operaci a „odevzdají“ procesor dříve



# Rychlost procesů

- Rychlost běhu procesu **není konstantní**.
- Obvykle **není** ani **reprodukovatelná**.
- Procesy nesmějí mít vestavěné **předpoklady o časování (!)**
- Např. doba trvání I/O různá.
- Procesy **neběží stejně rychle**.

Proces běží v reálném systému, který se věnuje i dalším procesům, obsluze přerušení atd., tedy nesmíme spoléhat, že poběží vždy stejně rychle..



# Stavy procesu

- Procesy často potřebují **komunikovat** s ostatními procesy:
- *ls -l | more* . . .
- proces *ls* vypíše obsah adresáře na std. výstup
- *more* zobrazí obrazovku a čeká na klávesu
- More je připraven běžet, ale **nemá** žádný **vstup** – **zablokuje se** dokud vstup nedostane

Oba jsou spuštěny  
současně



# Kdy proces neběží

Nemůže,  
na něco  
čeká

- **Blokování procesu** – proces nemůže pokračovat, protože **čeká** na zdroj (vstup, zařízení, paměť), který není dostupný – proces nemůže **logicky** pokračovat

Chtěl by,  
ale není  
volný  
CPU

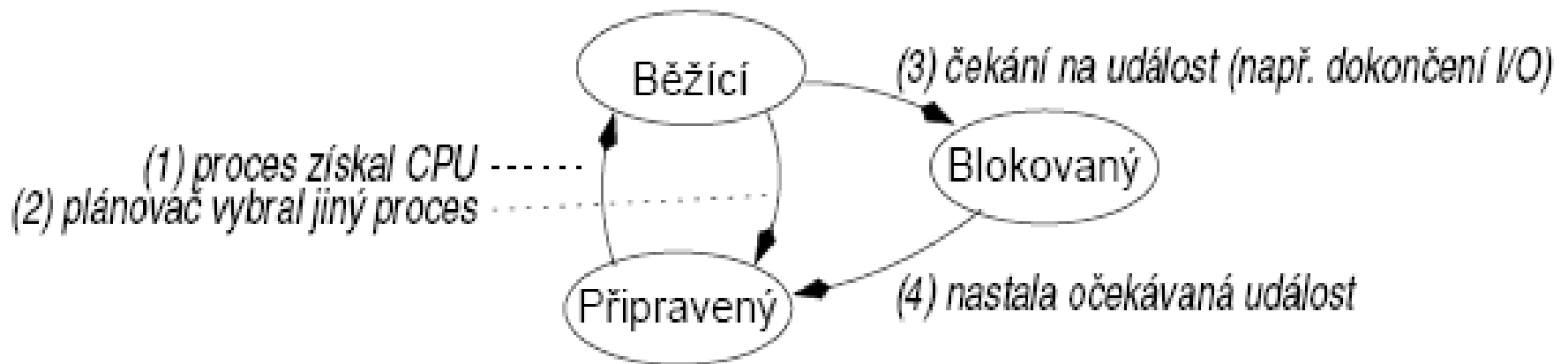
- Proces může být **připraven pokračovat**, ale CPU **vykonává jiný** proces – musí počkat, až bude CPU „volné“



# Základní stavy procesu

- **Běžící (running)**
  - skutečně využívá CPU, vykonává instrukce
- **Připravený (ready, runnable)**
  - dočasně pozastaven, aby mohl jiný proces pokračovat
- **Blokovaný (blocked, waiting)**
  - neschopný běhu, dokud nenastane externí událost

# Základní stavy procesu (!!)





# Přechody stavů procesu

1. Plánovač **vybere** nějaký proces
  2. Proces je **pozastaven**, plánovač vybere jiný proces (typicky - vypršelo časové kvantum)
  3. Proces se **zablokuje**, protože čeká na událost (zdroj – disk, čtení z klávesnice)
  4. **Nastala** očekávaná **událost**, např. zdroj se stal dostupný
-

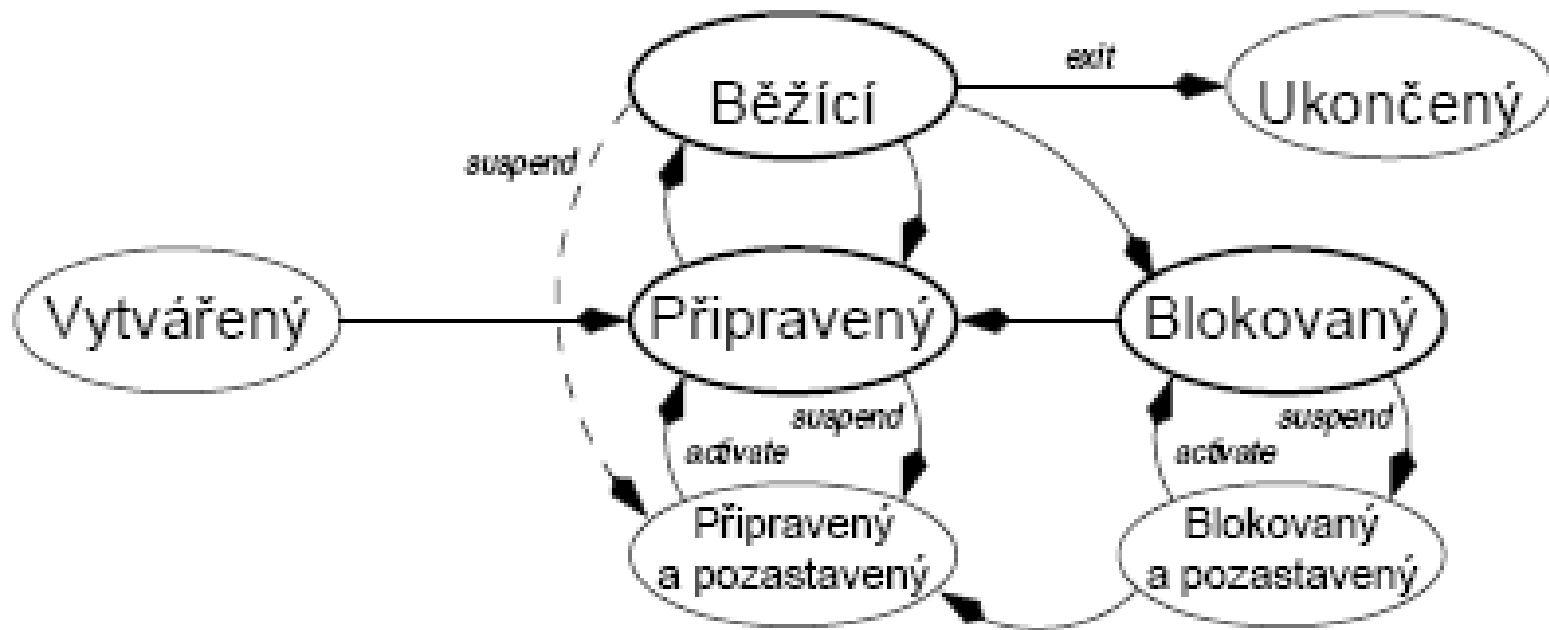


# Stavy procesů

- Jádru OS obsahuje plánovač
  - Plánovač určuje, který proces bude běžet
  - Nad OS řada procesů, střídají se o CPU
  
  - Stav procesu **pozastavený**
  - V některých systémech může být proces **pozastaven** nebo **aktivován**
  - V diagramu přibudou **dva** nové stavy
-



# Stavy procesů





# Tabulka procesů

OS si musí vést evidenci, jaké procesy v systému v danou chvíli existují.

Tato informace je vedena v **tabulce procesů**.

Každý proces v ní má záznam, a tento záznam se nazývá **process control block (PCB)**.

Na základě informací zde obsažených se plánovač umí rozhodnout, který proces dále poběží a bude schopen tento proces spustit ze stavu, v kterém byl naposledy přerušen.



# PCB (Process Control Block) !

- OS udržuje tabulku nazývanou **tabulka procesů**
  - Každý proces v ní má položku zvanou **PCB** (Process Control Block)
  - PCB obsahuje všechny informace potřebné pro **opětovné spuštění** přerušenoého procesu
    - Procesy se o CPU střídají, tj. jeho běh je přerušovaný
  - Konkrétní obsah PCB – různý dle OS
  - Pole správy **procesů**, správy **paměti**, správy **souborů** (!!)
-



# Položky - správa procesů

- Identifikátory (číselné)
    - Identifikátor procesu - PID
    - Identifikátor uživatele - UID
  - Stavová informace procesoru
    - Univerzální registry,
    - Ukazatel na další instrukci - PC
    - ukazatel zásobníku SP
    - Stav CPU – PSW (Program Status Word)
  - Stav procesu (běžící, připraven, blokován)
  - Plánovací parametry procesu (algoritmus, priorita)
-



---

# Položky – správa procesů II

- **Odkazy** na rodiče a potomky
  - **Účtovací** informace
    - Čas spuštění procesu
    - Čas CPU spotřebovaný procesem
  - Nastavení meziprocesové **komunikace**
    - Nastavení signálů, zpráv
-



---

# Položky – správa paměti

## □ Popis paměti

- Ukazatel, velikost, přístupová práva
  - 1. Úsek paměti s **kódem programu**
  - 2. **Data** – hromada
    - Pascal – new release
    - C – malloc, free
  - 3. **Zásobník**
    - Návrátové adresy, parametry funkcí a procedur, lokální proměnné
-



---

# Položky – správa souborů

## □ Nastavení prostředí

- Aktuální pracovní adresář

## □ Otevřené soubory

- Způsob otevření – čtení / zápis
  - Pozice v otevřeném souboru
-

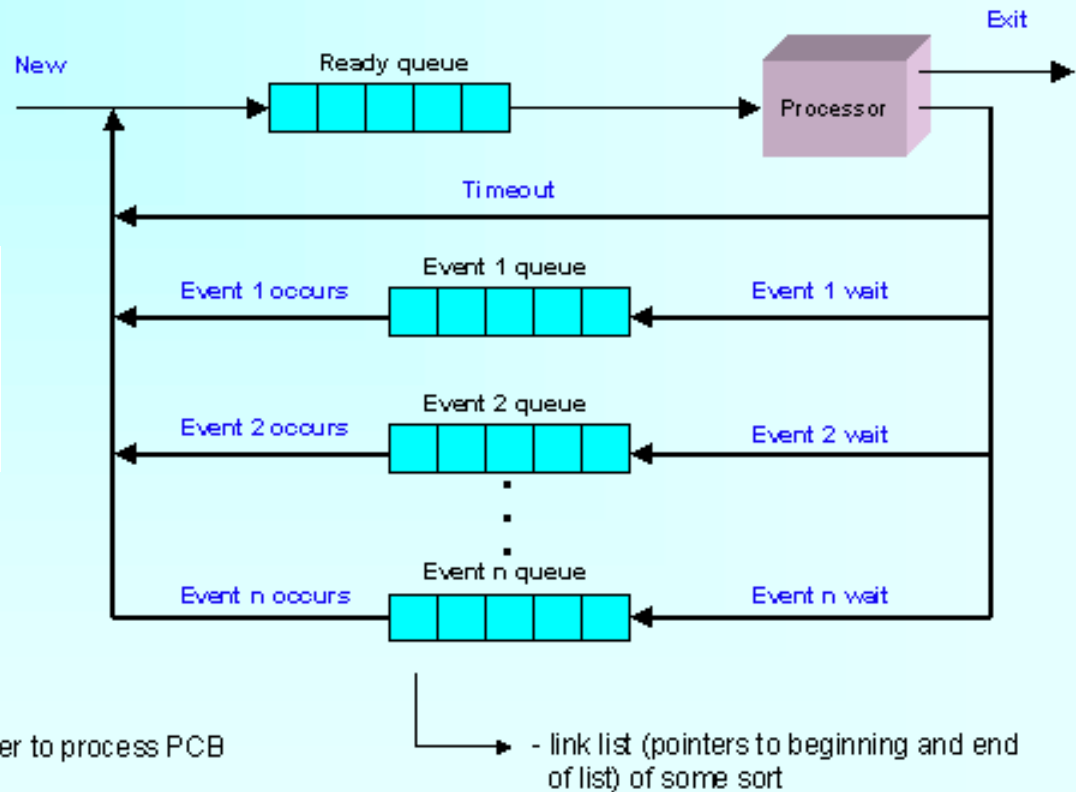
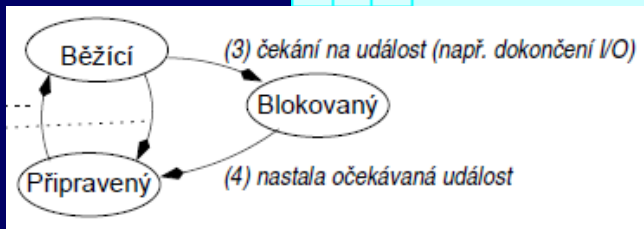


# PCB

Pointer	Process state
Process number	
Program counter	
<b>Registers</b>	
Memory limits	
List of open files	
...	



# Data Structures (again)...



Yair Amir

Fall 00 / Lecture 2

8

Viz <http://www.cs.jhu.edu/~yairamir/cs418/os2/sld007.htm>



# Poznámky

## □ Stav **Nový**

- Proces přejde z nový do stavu Připravený

## □ Stav **Ukončený**

- Přejchod ze stavu běžící do ukončený, např. voláním exit

Častou chybou je, že lidé kreslí přechod ze stavu Nový do stavu Běžící, napřed se musí jít přes Připravený !  
Stejně tak, do stavu Ukončený jdeme ze stavu Běžící.



# Ukončení procesu

- Proces úspěšně vykoná kód programu 😊
  - Skončí rodičovský proces a OS nedovolí pokračovat child procesu (záleží na OS, někdy ano někdy ne)
  - Proces překročí limit nějakého zdroje
-



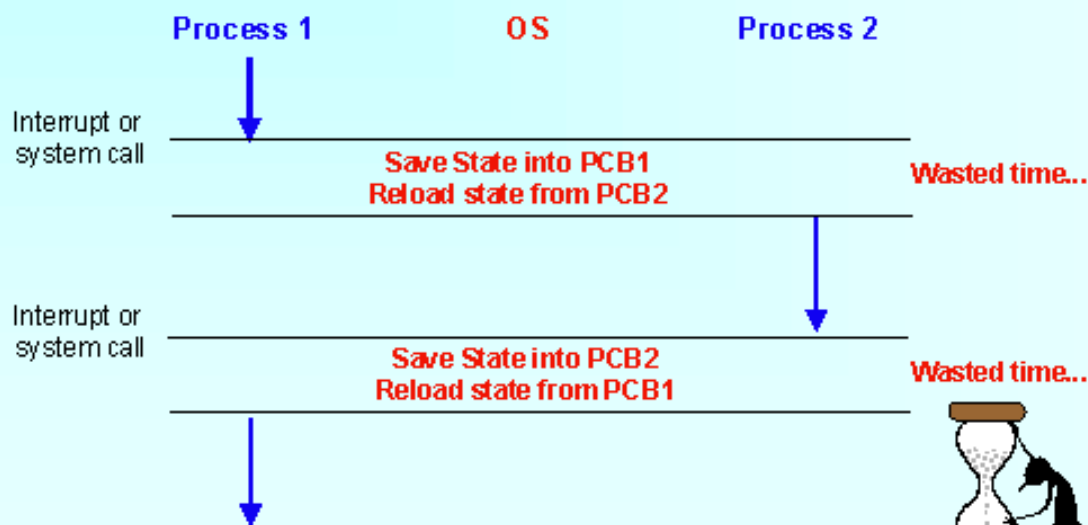
# Přepnutí procesu - průběh

- Systém **nastaví časovač** – pravidelně přerušení
  - Na předem definovaném místě – **adresa** obslužného programu přerušení
  - **CPU po příchodu přerušení provede:**
    - Uloží čítač instrukcí PC do zásobníku
    - Načte do PC adresu obsluž. programu přerušení
    - Přepne do režimu jádra
-

# Přepnutí procesu

## Context Switch

Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process.





# Přepnutí procesu - II

- **Vyvolána obsluha přerušení:**
  - Uloží obsah registrů do zásobníku
  - Nastaví nový zásobník
- Plánovač **nastaví** proces jako ready, vybere nový proces pro spuštění
- **Přepnutí kontextu**
  - Nastaví mapu paměti nového procesu
  - Nastaví zásobník, načte obsah registrů
  - Provede návrat z přerušení – RET (do PC adresa ze zásobníku, přepne do uživatelského režimu)

K přeplánování procesu nedojde při každém tiku časovače, ale až, když jich je tolik, že vyprší časové kvantum



# Rychlost CPU vs. paměti

Cílem následující vsuvky je říci, že výkon systému může degradovat nejenom časté střídání procesů, protože se pořád musí **přepínat kontext**, ale i fakt, že informace v cache se po přepnutí na jiný proces stane neaktuální, a cache paměti chvíli trvá, než se naplní aktuálními daty, což má také vliv na výkon systému.



# Rychlost CPU vs. paměť

## □ CPU

- Rychlost – počet instrukcí za sekundu
- Obvykle **nejrychlejší** komponenta v systému
- Skutečný počet instrukcí závisí na rychlosti, jak lze instrukce a data přenášet z a do hlavní paměti

## □ Hlavní paměť

- Rychlost v paměťových cyklech (čtení, zápis)
  - O řád pomalejší než CPU
  - Proto důvod používat cache paměť
-





# Rozdíly rychlostí – „pyramida“

- **CPU** registry – rychlé – zápisníková paměť, 32x32 nebo 64x64 bitů, žádné zpoždění při přístupu
  - **Cache** – malá paměť s vysokou rychlostí,
    - princip lokality,
    - pokud jsou data v cache – dostaneme velmi rychle, 2 tiky hodin
  - **RAM**
  - **Vnější paměť**
    - Mechanická, pomalejší, větší kapacita, levnější cena za bit
-

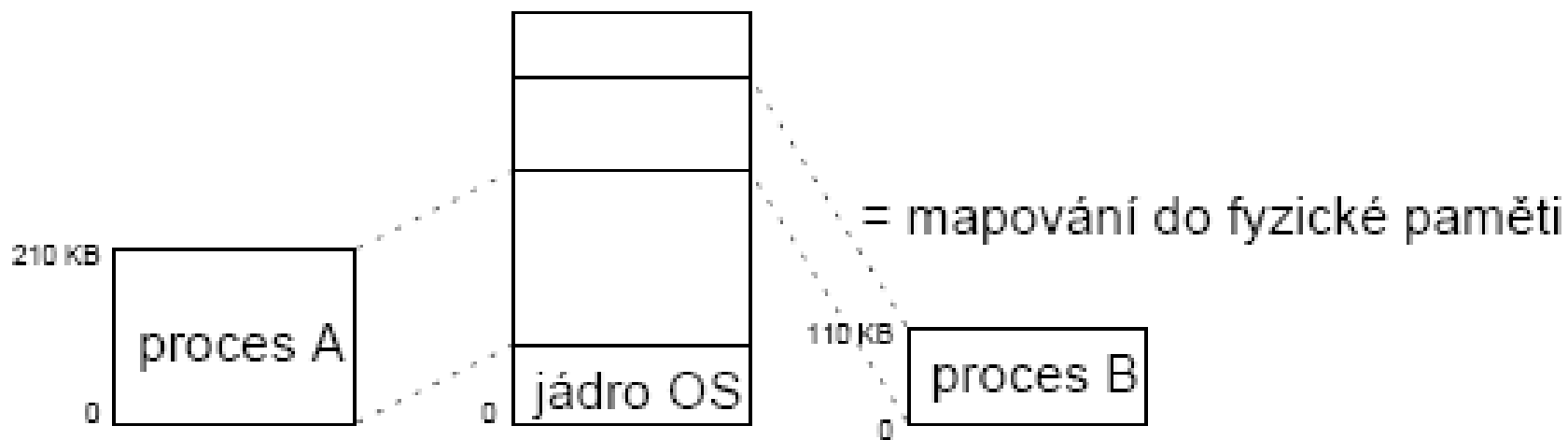


# MMU – Memory Management Unit

- Více procesů v paměti
    - Každý proces paměť pro sebe, např. od adresy 0 (relokace)
    - Ochrana – nemůže zasahovat do paměti ostatních procesů ani jádra
  - Mezi CPU a pamětí je **MMU**
    - Program pracuje s **virtuálními** adresami
    - MMU je převede na **fyzické** adresy
-

# MMU

fyzická paměť





# Výkonnostní důsledky

- Pokud program **nějakou dobu** běží – v cache **jeho** data a instrukce – dobrá výkonnost
  - Při přepnutí na jiný proces – převažuje přístup do hlavní paměti (keš není naučená)
  - Nastavení MMU se musí změnit
  
  - Přepnutí mezi úlohami i přepnutí do jádra (volání služby OS) – relativně drahé (čas)
-



# Služby pro práci s procesy

## □ Jednoduché systémy

- Všechny potřebné procesy spuštěny při startu systému
  - Běží po celou dobu běhu systému – žádné služby nepotřebujeme
  - Některé zapouzdřené (embedded) systémy
-



# UNIX a Linux

- Služba **fork()** – vytvoří přesnou kopii rodičovského procesu
  - Návratová hodnota – rozliší mezi rodičem a potomkem (potomek dostane 0)
  - *pid = fork();*
  - *if (pid == 0) potomek else rodic*
  - Potomek může činnost ukončit pomocí **exit()**
  - Rodič může na potomka čekat – **wait()**
-



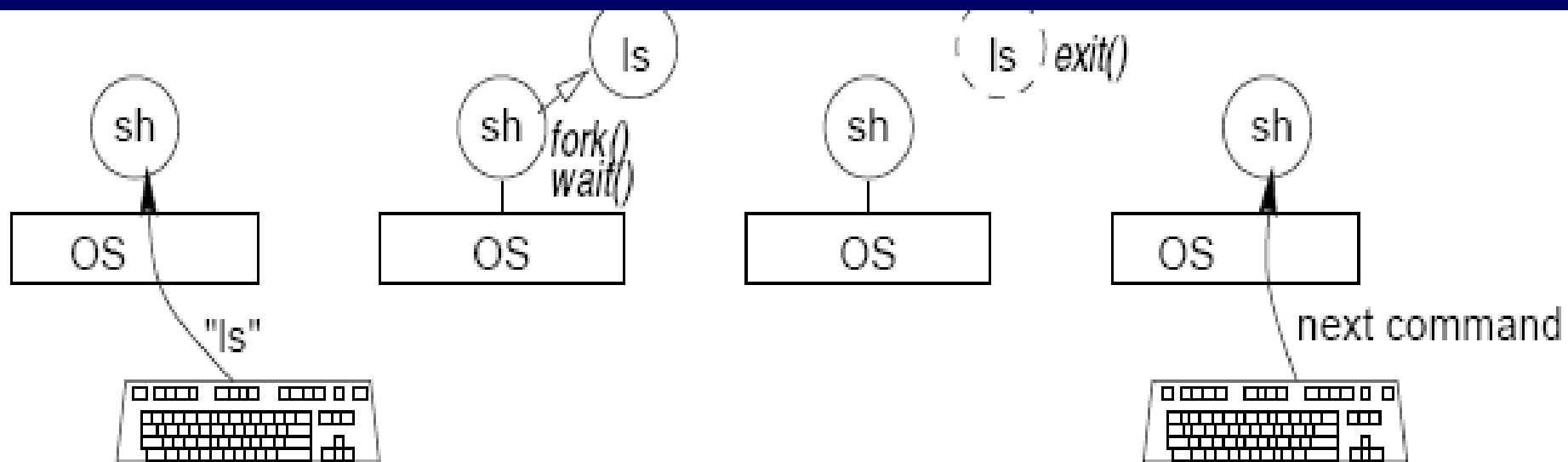
# UNIX

□ Potomek může místo sebe spustit jiný program – volání **execve()** – nahradí obsah paměti procesem spouštěným ze zadaného souboru

1. *if (fork() == 0)*
2. *execve("/bin/ls", argv, envp);*
3. *else*
4. *wait(NULL);*

# Příkazový interpret

- Spouští příkaz – vytvoří nový proces, čeká na jeho dokončení; ukončení – volání `sl. systému`







---

# Win32

- Vytvoření procesu službou  
CreateProcess
  - Mnoho parametrů – vlastnosti procesu
-



# Win32 ukázka

```
STARTUPINFO StartInfo; // name structure
PROCESS_INFORMATION ProcInfo; // name structure
memset(&ProcInfo, 0, sizeof(ProcInfo)); // Set up memory block
memset(&StartInfo, 0, sizeof(StartInfo)); // Set up memory block
StartInfo.cb = sizeof(StartInfo); // Set structure size
int res = CreateProcess(NULL, "MyApp.exe", NULL, NULL, NULL, NULL, NULL, NULL, &StartInfo, &ProcInfo); // starts
MyApp
if (res)
{
    WaitForSingleObject(ProcInfo.hThread, INFINITE); // wait forever for process to finish
    SetFocus(); // Bring back focus
}
```

příklad viz

[http://msdn.microsoft.com/en-](http://msdn.microsoft.com/en-us/library/windows/desktop/ms682512%28v=vs.85%29.aspx)

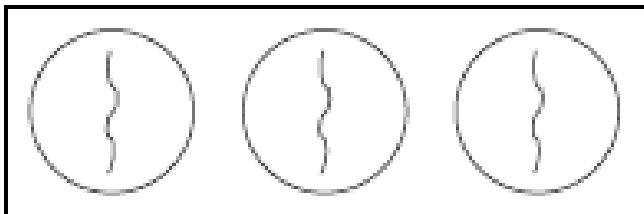
[us/library/windows/desktop/ms682512%28v=vs.85%29.aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms682512%28v=vs.85%29.aspx)



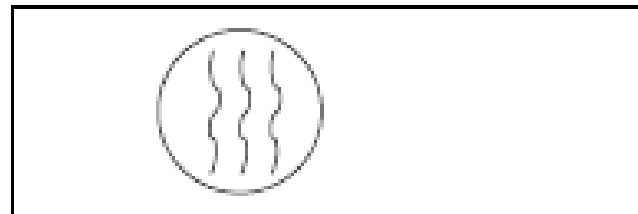
# Procesy a vlákna

- Tradiční OS – každý proces svůj vlastní adresový prostor a místo kde běží (**bod běhu**)
  - Často výhodné – více bodů běhu, ale ve stejném adresovém prostoru
  - Bod běhu – **vlákno** (thread, lightweight process)
  - Více vláken ve stejném procesu - **multithreading**
-

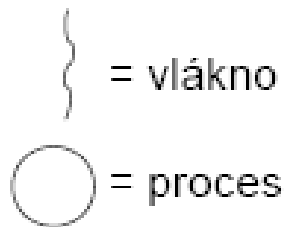
# Procesy a vlákna



a) tradiční procesy



b) proces jako kontejner na vlákna





# Vlákna (!!)

- Vlákna v procesu **sdílejí** adresní prostor, otevřené soubory (atributy procesu)
- Vlákna **mají soukromý** čítač instrukcí, obsah registrů, soukromý zásobník
  - Mohou mít soukromé lokální proměnné
- Původně využívána zejména pro VT výpočty na multiprocesech (každé vlákno vlastní CPU, společná data)



# Vlákna – použití dnes

- Rozsáhlejší výpočet a rozsáhlejší i/o
  - **Interaktivní procesy** – jedno vlákno pro komunikaci s uživatelem, další činnost na pozadí
  - **www prohlížeč** – jedno vlákno příjem dat, další zobrazování a interakce s uživatelem
  - **Textový procesor** – vstup dat, přeformátování textu
  - **Servery www** – jedno vlákno pro každého klienta
-



# Multithreading

- Podporován většinou OS
    - Linux, Windows
  - Podporován programovacími jazyky
    - Java, knihovny v C, ...
  - Proces začíná svůj běh s **jedním vláknem**, ostatní vytváří za běhu programově (konstrukce vytvoř vlákno)
  - Režie na vytvoření vlákna a přepnutí kontextu menší než v případě procesů (!)
-



# Poznámka (terminologie)

- Jeden proces – více vláken
    - Ošetření souběžného přístupu ke sdílené paměti
  - Více procesů sdílejících paměť
    - Ošetření souběžného přístupu ke sdílené paměti
  
  - V literatuře např. při řešení synchronizace, se většinou nerozlišuje, zda uvažujeme souběžný přístup vláken nebo procesů ke společné paměti
-





# Programové konstrukce pro vytváření vláken

## □ Statické

- Proces obsahuje deklaraci pevné množiny podprocesů (např. tabulka)
- Všechny spuštěny při spuštění procesu

## □ Dynamické

- Procesy mohou vytvářet potomky dynamicky
- častější

## □ Pro popis – **precedenční grafy**

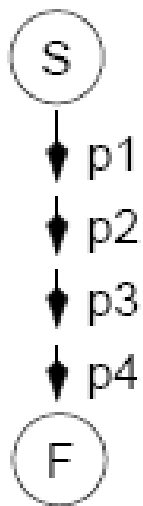
---



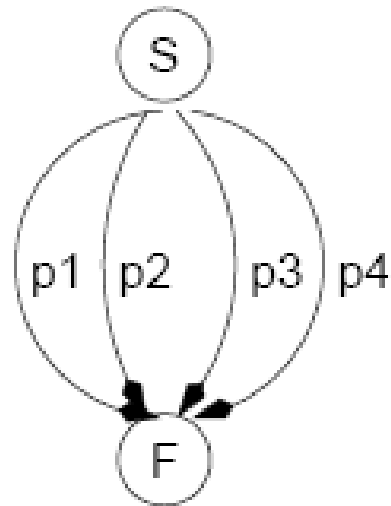
# Precedenční grafy

- Popis pro vyjádření různých relací mezi procesy
  - Process flow graph
  
  - Acyklický orientovaný graf
  - **Běh** procesu  $p_i$  – orientovaná **hrana grafu**
  - **Vztahy** mezi procesy – seriové nebo paralelní spojení – **spojením hran**
-

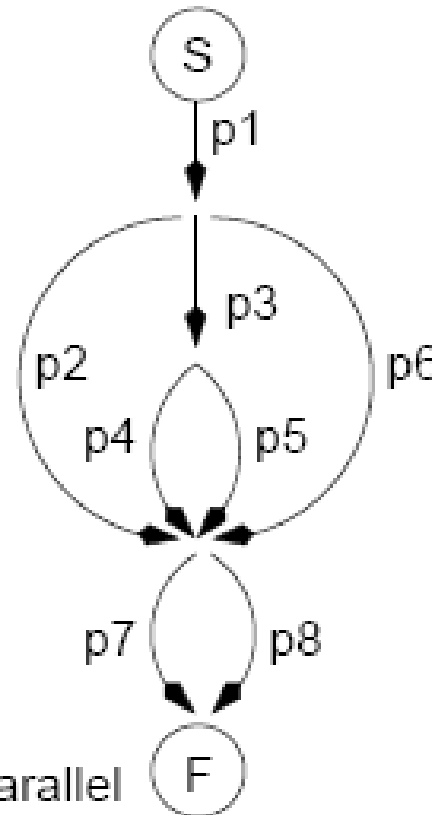
# Precedenční grafy



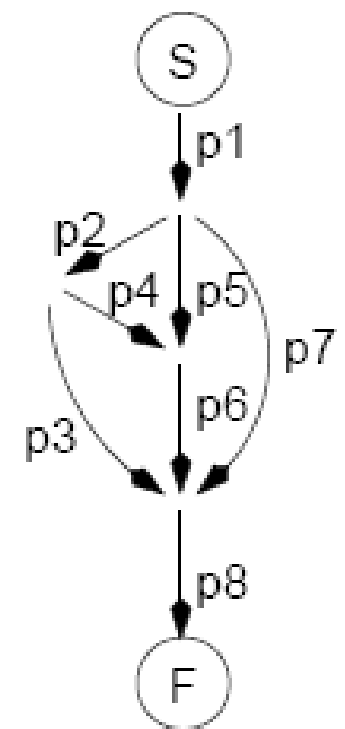
a) Series



(b) Parallel



(c) Series/Parallel



(d) General

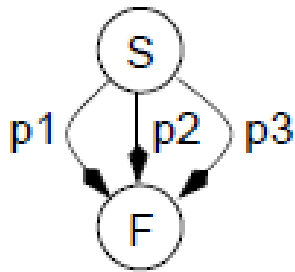


# Fork, join, quit

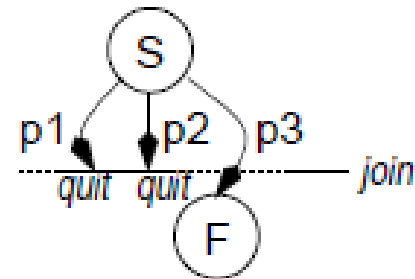
Mechanismus pro obecný popis paralelních aktivit

primitivum	funkce
<b>fork X;</b>	Spuštění nového vlákna od příkazu označeného návěstím X; nové vlákno poběží paralelně s původním
<b>quit ;</b>	Ukončí vlákno
<b>joint t, Y;</b>	Atomicky (nedělitelně) provede: $t = t - 1$ ; if ( $t == 0$ ) then goto Y;

# Běh procesů odpovídající precedenčnímu grafu



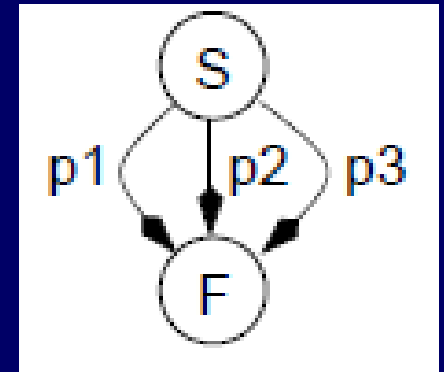
a) precedenční graf



b) skutečný běh

Nevíme, který z procesů doběhne první a který poslední,  
jen jeden z možných běhů

# Zápis pomocí fork-join-quit



```
n=3; // tři procesy
fork L2; // spustíme vlákno od L2
fork L3; // spustíme vlákno od L3
p1; join n, L4; quit; // jen 1. vlákno
L2: p2; join n, L4; quit; // jen 2. vlákno
L3: p3; join n, L4; quit; // jen 3. vlákno
L4: .... // zde jen poslední
F: ....
```



---

# Poznámky k fork-join-quit

- + obecný zápis
- špatná čitelnost (přehlednost)

V některé literatuře se neuvádí quit, a předpokládá se  $\text{join} = \text{join} + \text{quit}$

---



# Správně vnořené precedenční grafy

$S(a,b)$  – sériové spojení procesů  
(za procesem **a** následuje **b**)

$P(a,b)$  – paralelní spojení procesů **a** a **b**

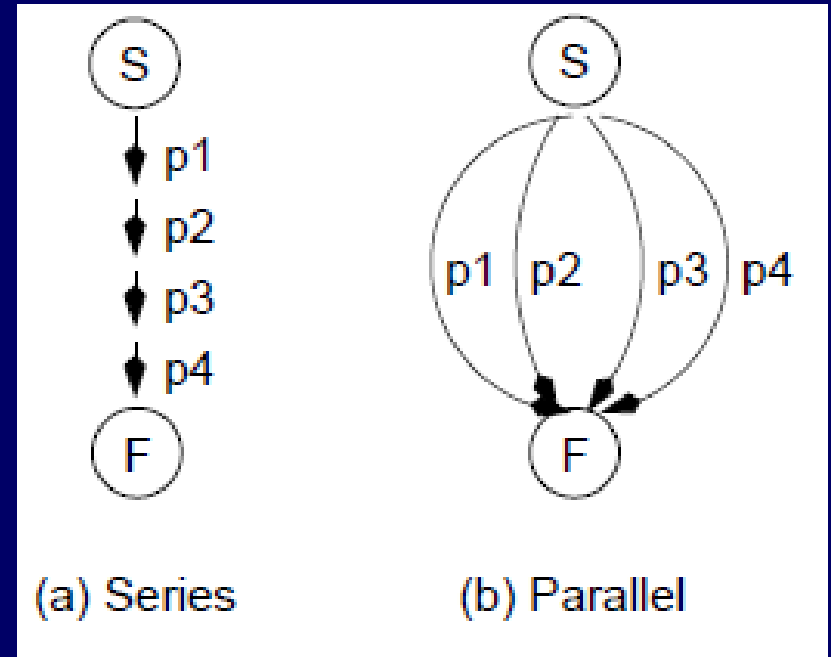
Precedenční graf je **správně vnořený**,  
pokud může být popsán kompozicí  
funkcí **S** a **P**



# Příklady správně vnořených grafů

$S(p_1, S(p_2, S(p_3, p_4)))$

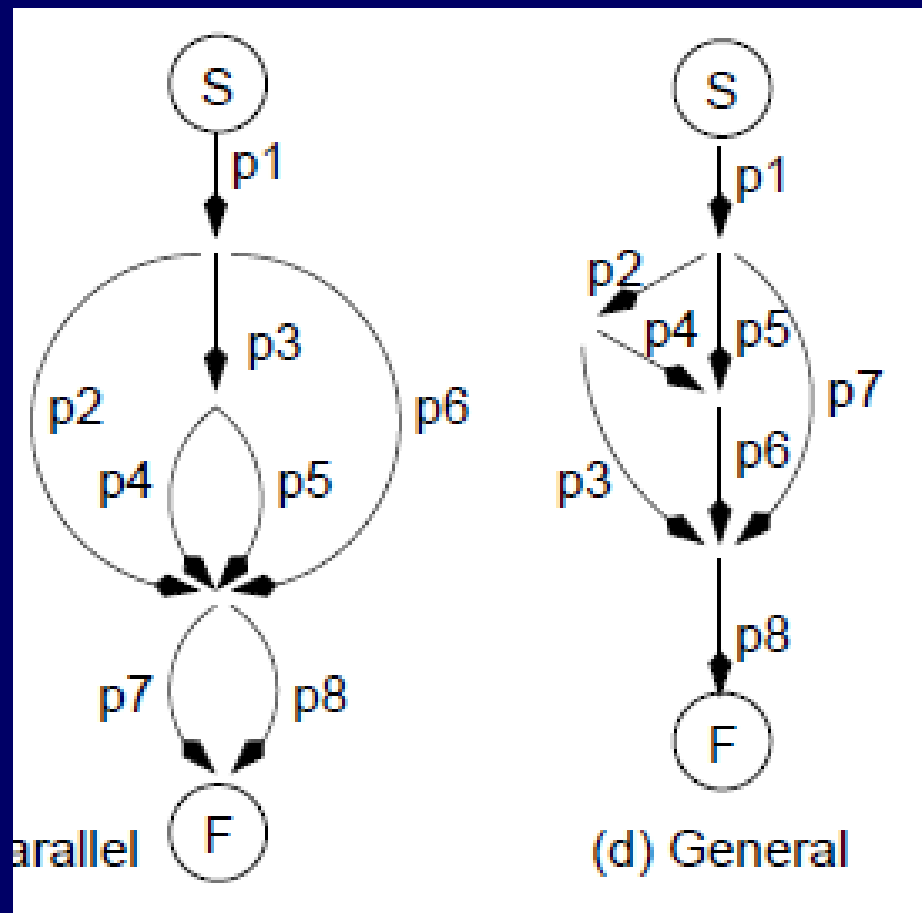
$P(p_1, P(p_2, P(p_3, p_4)))$





Graf (d) není  
správně vnořený  
Nelze jej popsat  
kompozicí S a P

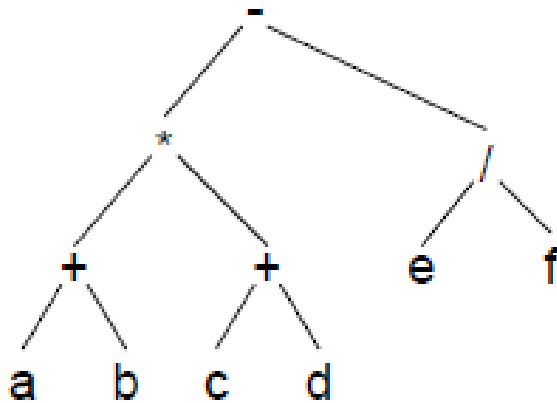
Graf vlevo lze:



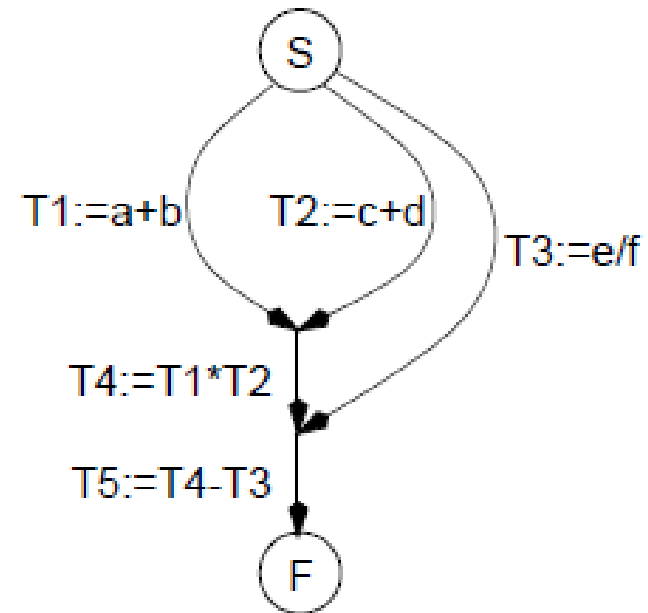
$S(p1, S(P(p2, P(S(p3, P(p4, p5))), p6)), P(p7, p8))$

# Příklad vyhodnocení aritmetického výrazu

$$(a + b) * (c + d) - (e / f)$$



a) expression tree



b) process flow graph

Vznikají správně vnořené procesy; dodržet maximální paralelismus !



# Abstraktní primitiva **cobegin, coend**

- Dijkstra (1968), původně parbegin,..
- Specifikuje sekvence programu, která má být spuštěna paralelně

**cobegin**

$C_1 \parallel C_2 \parallel \dots \parallel C_n$

**coend**

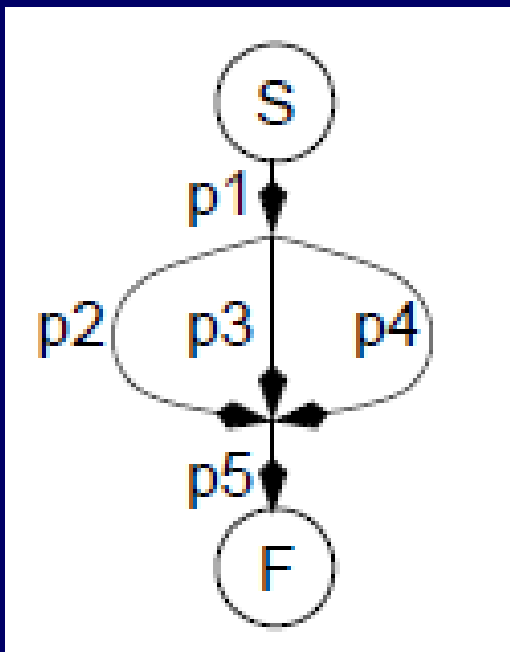
Každé  $C_i$  ... autonomní segment kódu (blok)

Samostatné vlákno pro všechna  $C_i$

$C_i$  běží nezávisle na ostatních

Program pokračuje za coend až po skončení posledního  $C_i$

# Příklad – cobegin, coend



```
begin  
  C1;  
  cobegin  
    C2 || C3 || C4  
  coend  
  C5  
end
```



# Vztah cobegin/coend a funkcí P, S

- Každý segment kódu  $C_i$  lze dekomponovat na sekvenci příkazů  $p_i$ :  
 $S(p_{i1}, S(p_{i2}, \dots))$
- Konstrukce `cobegin  $C_1$  ||  $C_2$  .. coend` odpovídají vnoření funkcí:  
 $P(C_1, P(C_2, \dots))$

# Příklad – aritmetický výraz

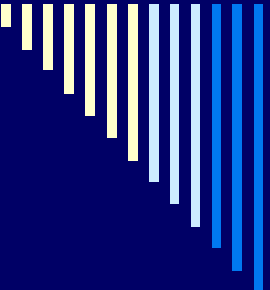
$$(a+b) * (c+d) - (e/f)$$

```
begin
  cobegin
    begin
      cobegin
        T1 = a+b || T2 = c+d
      coend
      T4 = T1 * T2
    end
    || T3 = e/f
  coend
  T5 = T4 - T3
end
```

**Maximální  
paralelismus**

Část výpočtu  
spustím ihned  
jak je to možné

Např. T1,T2,T3



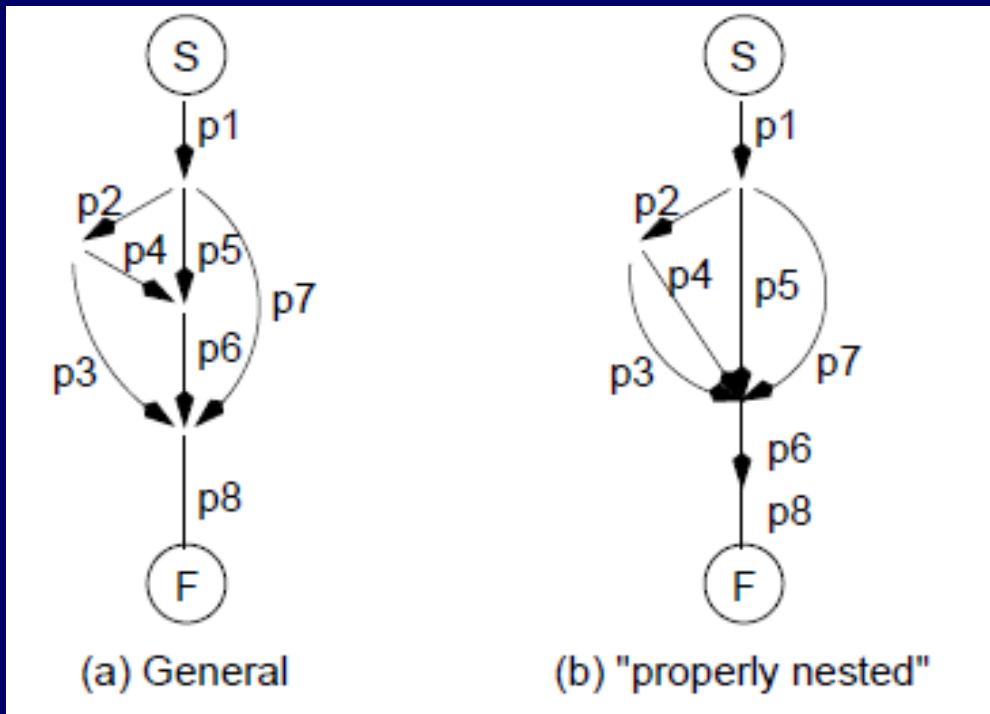
# Příklad – fork, join, quit

## $(a+b) * (c+d) - (e/f)$

```
n := 2;  
fork L3;  
m := 2;  
fork L2;  
t1 := a + b;      join m, L4; quit;  
L2: t2 := c + d;   join m, L4; quit;  
L4: t4 := t1 * t2; join n, L5; quit;  
L3: t3 := e/f;     join n, L5; quit;  
L5: t5 := t4 - t3;
```



# Lze nesprávně vnořený graf upravit?



Můžeme „beztrestně“  
posunout proces p6?

Ne vždy !!

Pokud jsou závislé, a p6  
musí běžet paralelně s p3 a  
p7, např. si vyměňují  
zprávy, pak toto nelze.

Fork – join – quit  
popíše i nesprávně  
vnořené grafy



# Př. iterace

```
    for i:=1 to m do
      for j:=1 to n do
        fork E;

    quit;
E: A[i][j]:= ...
    join t, R;
    quit;
R: ...
```

Soukromé kopie proměnných rodičovského vlákna  
Každé vlákno vytvořené fork E má soukromou kopii i, j  
Deklarace typu „private“



---

## Ada – statická deklarace podprocesu

**process p**

```
    deklarace .. // mohou být další definice  
begin          podprocesů, spuštěny při  
    ...        spuštění p  
end
```



---

## Ada – dynamická deklarace podprocesu

```
process type p2           // šablona
    deklarace ..
begin
    ...
end
begin
    q = new p2;
end
```

---



---

# Vlákna v systému UNIX a jazyce C

- Knihovna **libpthread**
  - Jako vlákno se spustí určitá funkce
  - Návratem z této funkce vlákno zanikne
-



# Základní funkce

funkce	popis
t = pthread_create(..f..)	Podprogram f se spustí jako vlákno vrací id vlákna
pthread_exit ()	Odpovídá quit, může předat návratovou hodnotu
x = pthread_join t	Čeká na dokončení vlákna t vrací hodnotu předanou voláním exit
pthread_detach (t)	Na dokončení vlákna se nebude čekat joinem
pthread_cancel (t)	Zruší jiné vlákno uvnitř stejného procesu

```
#include <stdio.h>
#include <errno.h>
#include <pthread.h>

void *vlakno(void *m) /* podprogram pro vlákno */
{
    int i;

    for (i=0; i<10000; i++)
        write(1, m, 1);
    return NULL;
}

int main(int argc, char *argv[])
{
    pthread_t th1, th2;

    pthread_create(&th1, NULL, vlakno, "**"); /* vytvoří vlákno */
    pthread_create(&th2, NULL, vlakno, ".");
    pthread_join(th1, NULL); /* čeká na dokončení vlákna */
    pthread_join(th2, NULL);

    return 0;
}
```



---

# Java – základ práce s vlákny

- Třída `java.lang.Thread`
- Programátor vytvoří podtřídu s vlastní metodou `run()` .. činnost vlákna
- Spustí se vytvořením instance podtřídy a spuštěním metody `start()`

```
MyThread t = new MyThread();  
t.start();
```

---





---

# Java – rozhraní Runnable

- Rozhraní **Runnable**
  - Třída může definovat metodu **run()**, ale sama nemusí být potomkem třídy Thread
  - Viz pozdější cvičení
-

---



# Další materiály

- Viz texty k přednáškám: *p2proc.pdf*
-