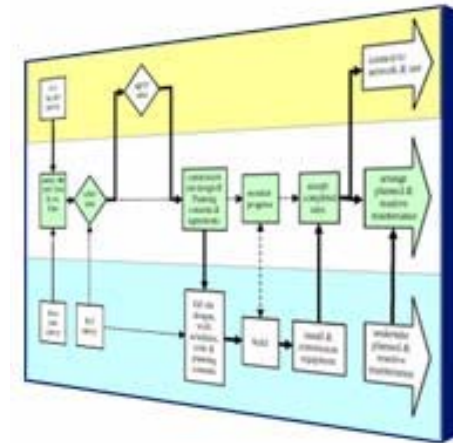




# PB153

# Operační systémy a jejich rozhraní

## Procesy





# Co je to proces

- Pro spuštěný program máme řadu pojmenování
  - dávkové systémy: úlohy, dávky, jobs
  - multiprogramové systémy: procesy (processes, tasks), vlákna (threads)
- Společné pojmenování pro spuštěný program je proces (někdy používáme synonymum task)
- Dále zavádíme pojem vlákno pro „dílčí“ proces v rámci „procesu“
- Proces obsahuje
  - čítač instrukcí
  - zásobník
  - datovou sekci
  - program (instrukční sekce)



# Proces v OS

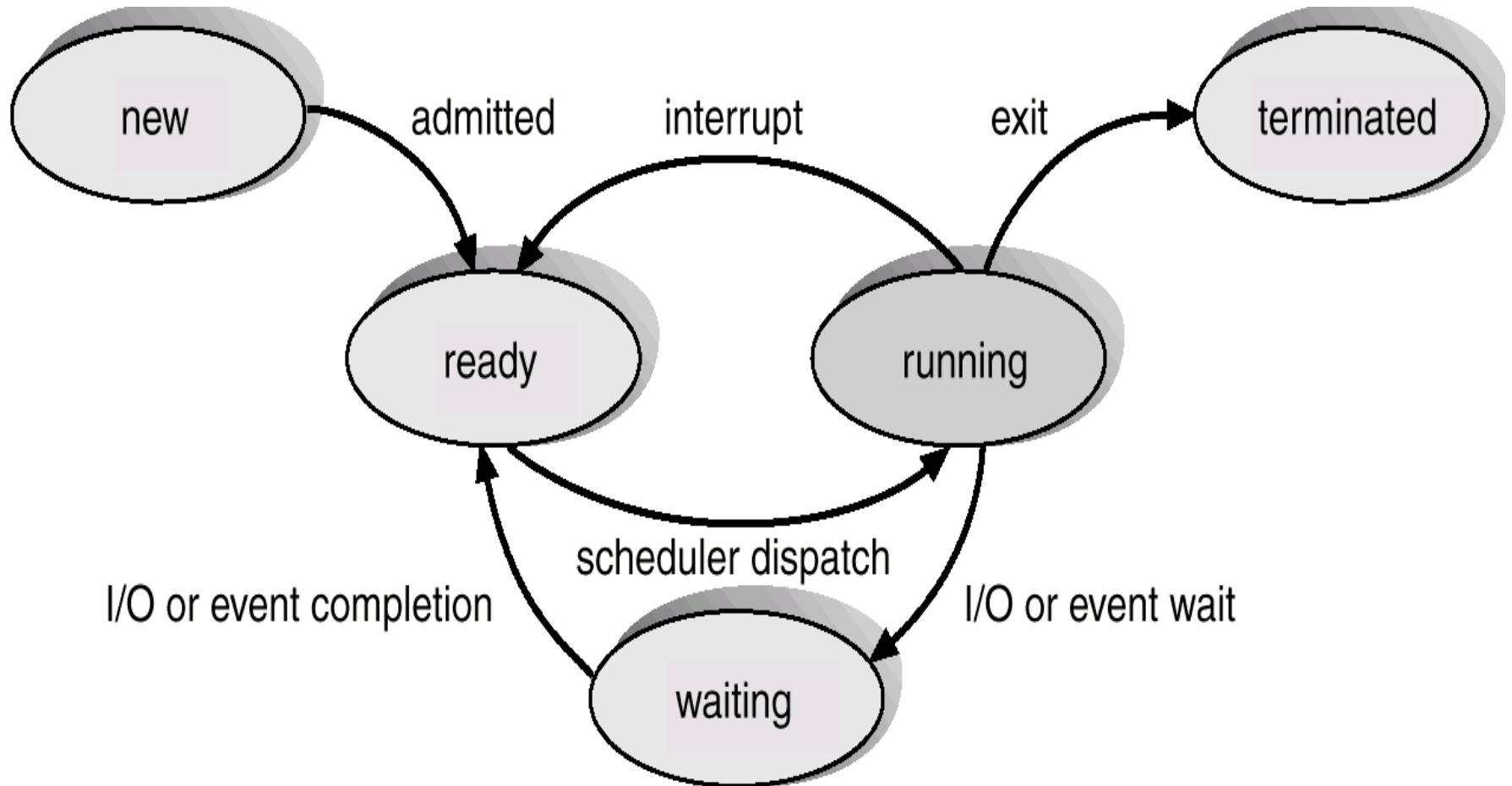
- Hierarchie procesů
  - rodič, potomek (proces vytvořený na žádost jiného procesu – rodiče)
  - sourozenci (procesy vytvořené jedním rodičem)
- Procesy a mutiprogramování
  - prokládáním běhů procesů maximalizujeme využití procesoru a minimalizujeme dobu odpovědi
  - procesu jsou přidělovány zdroje systému
    - bereme v úvahu priority a vzájemnou výlučnost operací
    - musíme zabránit „uváznutí“ procesů (deadlock)



# Stavy procesu

- Proces se může nacházet v jednom ze stavů:
  - nový (new): právě vytvořený proces
  - běžící (running): některý procesor právě vykonává instrukce procesu
  - čekající (waiting): čeká na určitou událost
  - připravený (ready): čeká na přidělení času procesoru
  - ukončený (terminated): ukončil své provádění

# Stavy procesu

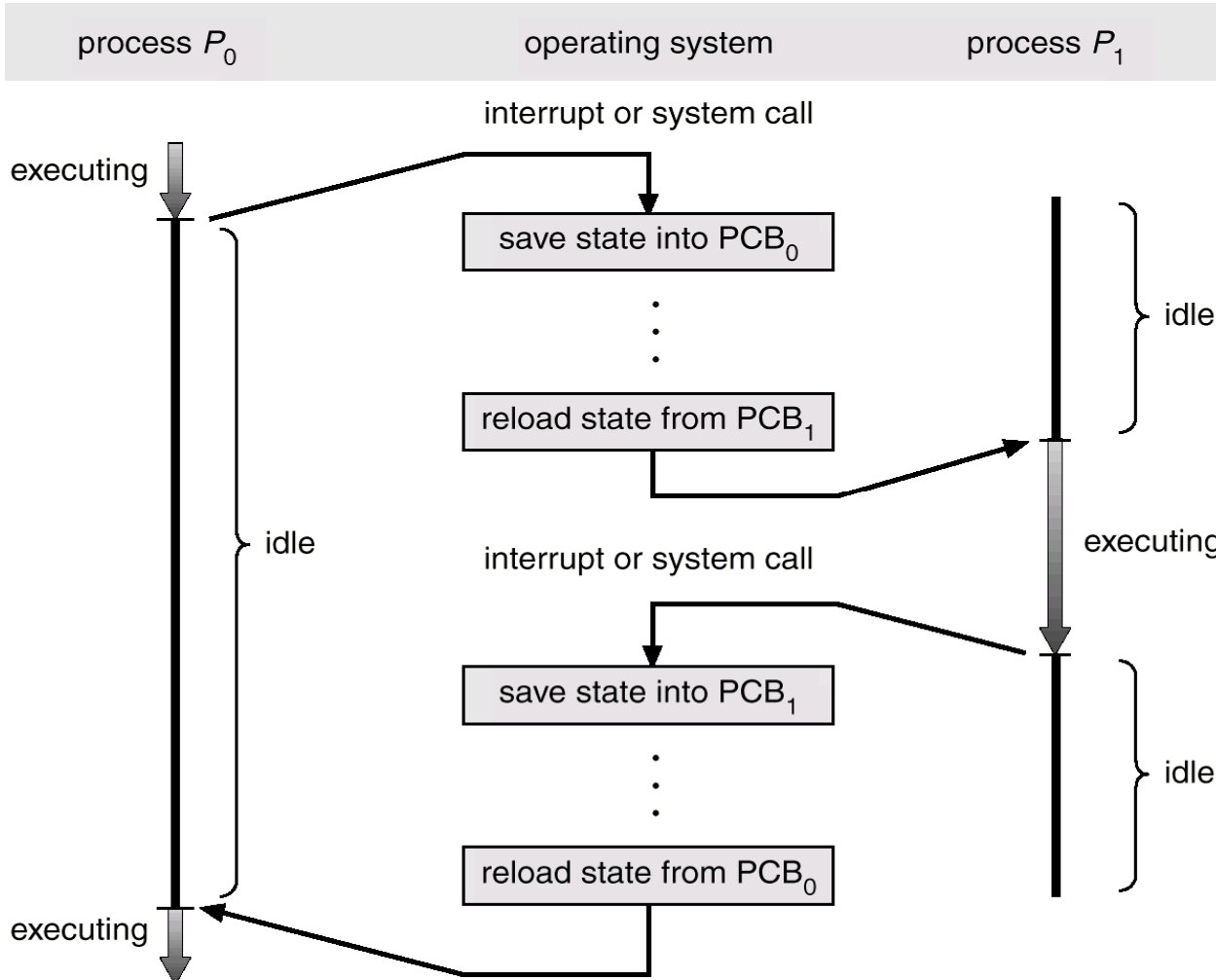


# Informace OS o procesu

- Process Control Block -- tabulka obsahující informace potřebné pro definici a správu procesu
  - stav procesu (běžící, připravený, ...)
  - čítač instrukcí
  - registry procesoru
  - informace potřebné pro správu paměti
  - informace potřebné pro správu I/O
  - účtovací informace

|                    |               |
|--------------------|---------------|
| pointer            | process state |
| process number     |               |
| program counter    |               |
| registers          |               |
| memory limits      |               |
| list of open files |               |
| ⋮                  |               |

# Přepnutí procesu





# Přepnutí kontextu

- Vyžádá se služba, akceptuje se některé asynchronní přerušení, obslouží se a nově se vybere jako běžící proces
- Když OS přepojuje CPU z procesu X na proces Y, musí:
  - uchovat (uložit v PCB procesu X) stav původně běžícího procesu
  - zavést stav nově běžícího procesu (z PCB procesu Y)
- Přepnutí kontextu představuje režijní ztrátu (zátěž)
  - během přepínání systém nedělá nic efektivního
- Doba přepnutí závisí na konkrétní HW platformě
  - Počet registrů procesoru, speciální instrukce pro uložení/načtení všech registrů procesoru apod.
- Při přerušení musí procesor
  - uchovat čítač instrukcí
  - zavést do čítače instrukcí hodnotou adresy vstupního bodu ovladače přerušení z vektoru přerušení

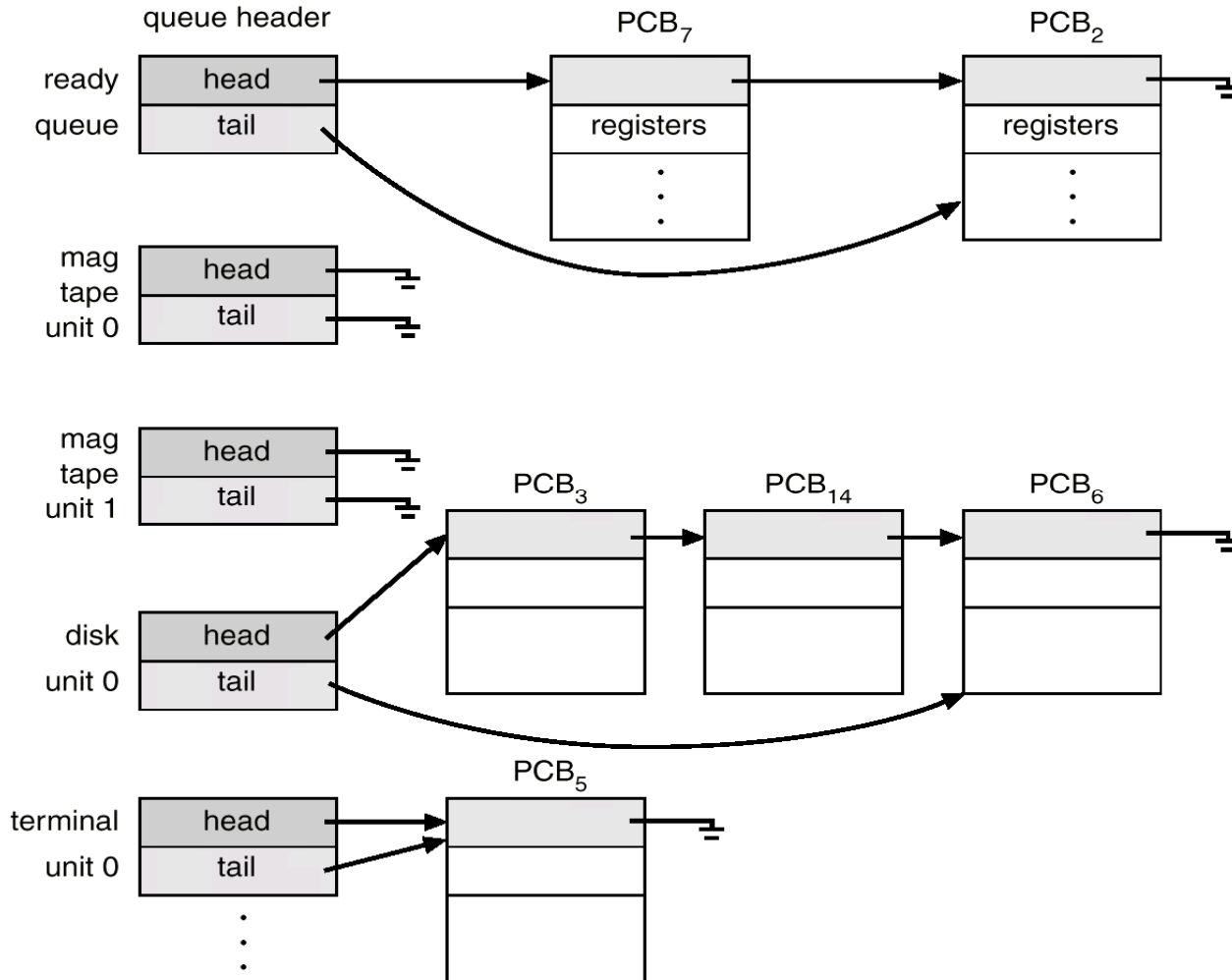




# Fronty plánování procesů

- Fronta úloha
  - seznam všech procesů v systému
- Fronta připravených procesů
  - seznam procesů uložených v hlavní paměti a připravených k běhu
- Fronta na zařízení
  - seznam procesů čekajících na I/O operaci
- Fronta odložených procesů
  - seznam procesů čekajících na přidělení místa v hlavní paměti
- Fronta na semafor
  - seznam procesů čekajících na synchronizační událost
- ...

# Příklad: fronty procesů

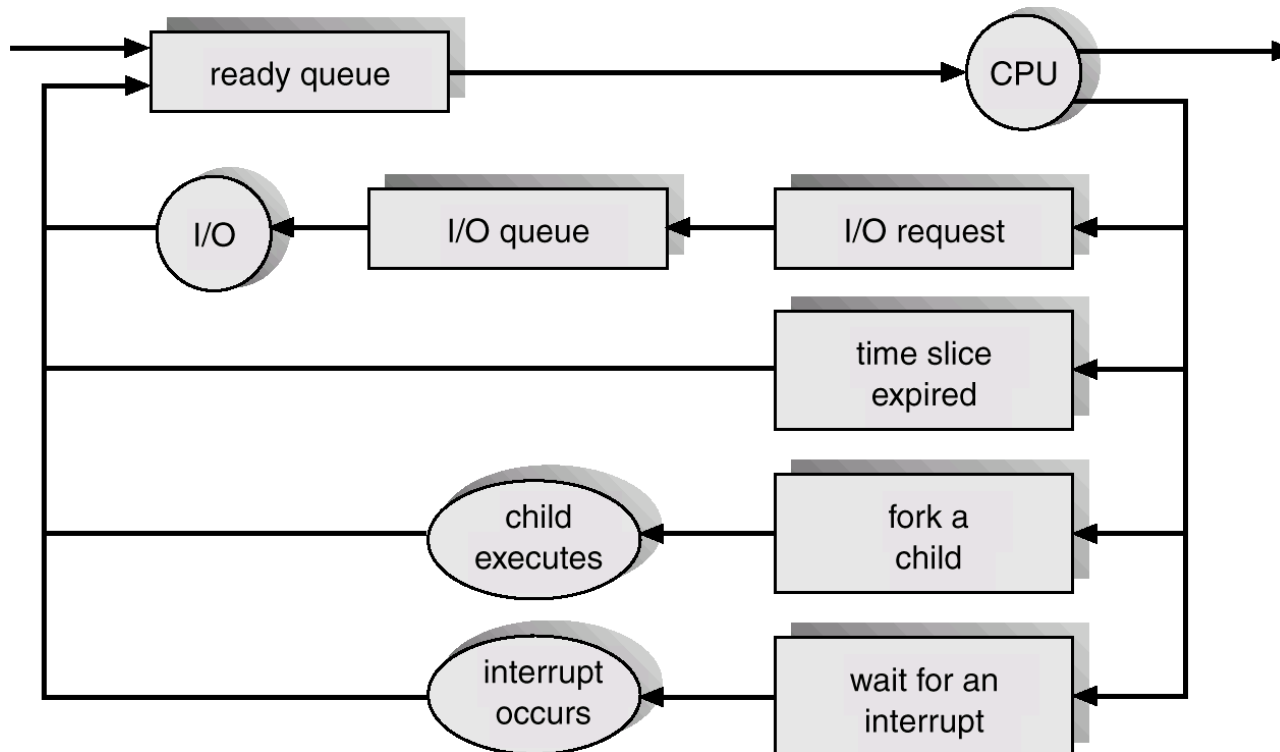


# Strategický plánovač

- Obecně nemusím mít všechny úlohy, které chci spustit, v operační paměti
- Fronta všech úloh může být značně dlouhá a plánovač musí rozhodnout, které úlohy zavést do paměti a spustit
- Toto je úkol dlouhodobého (strategického) plánovače
  - vybírá který proces lze zařadit mezi připravené procesy
  - plánovač je spouštěn je relativně málo často – typicky při ukončení jednoho procesu rozhodne, kterou úlohu dále vybrat k zavedení do paměti a spuštění
  - nemusí být super rychlý
  - určuje stupeň multiprogramování

# Krátkodobý plánovač

- Přiděluje procesor připraveným procesům
- Je spouštěn často (např. každých 10ms)
- Proto musí být rychlý

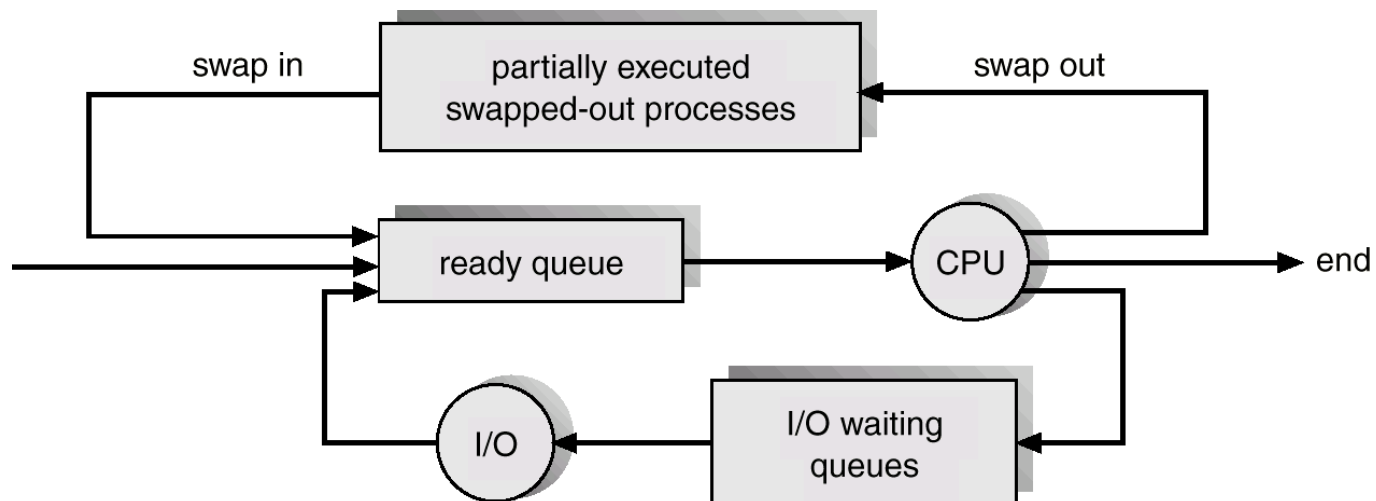


# Odkládání procesů

- Každý proces musíme jednou zavést do RAM
  - alespoň částečně (část může zůstat na disku a dohrána on-demand – viz některé optimalizace)
- Příliš mnoho procesů v RAM však snižuje výkonnost i při využití virtuální paměti
- OS musí provádění některých procesů odložit -  
- „vrátit“ na disk
- Dva nové stavy
  - odložený čekající
  - odložený připravený

# Střednědobý plánovač

- Střednědobý (taktický) plánovač
  - vybírá který proces lze zařadit mezi odložené procesy
  - vybírá který odložený proces lze zařadit mezi připravené procesy
- Náleží částečně i do správy operační paměti
  - kapacita FAP je omezená a odložené procesy uvolňují paměť





# Vytvoření procesu

- Rodič vytváří potomky (další procesy)
- Potomci mohou vytvářet další potomky ...
- Vzniká strom procesů
- Sdílení zdrojů – varianty při vytváření potomků
  - rodič a potomek sdílejí zdroje původně vlastněné rodičem
  - potomek sdílí rodičem vyčleněnou podmnožinu zdrojů s rodičem
  - potomek a rodič jsou plně samostatné procesy, nesdílí žádný zdroj
- Běh
  - rodič a potomek mohou běžet souběžně
  - rodič čeká na ukončení potomka



# Ukončení procesu

- Proces provede poslední příkaz a sám požádá OS o ukončení
  - výstupní data procesu se předají rodiči (pokud o to má zájem – např. čeká na ukončení potomka voláním wait)
  - zdroje končícího procesu se uvolňují operačním systémem
- O ukončení procesu žádá jeho rodič (nebo jiný proces s dostatečnými právy nebo OS), protože např.:
  - potomek překročil stanovenou kvótu přidělených zdrojů
  - úkol přidělený potomkovi rodič již dále nepotřebuje
  - rodič končí svoji existenci a nebylo povoleno, aby potomek přežil svého rodiče
    - může docházet ke kaskádnímu ukončování (ukončí se celá větev stromu procesů)





# Kooperující procesy

- Nezávislé procesy
  - nemohou se vzájemně ovlivňovat
- Kooperující procesy
  - mohou ovlivňovat běh jiných procesů nebo jiné procesy mohou ovlivňovat jejich běh
- Přínosy kooperace procesů
  - sdílení informací
  - urychlení výpočtů
  - modularita
  - pohodlí při programování
- Příklady typových úloh kooperace
  - producent – konzument
  - klient – server

# ● ● ● | Příklad: MS-DOS

- Neumožňuje spouštět procesy paralelně (single-taskingový systém)
- Služby OS
  - spust' proces a čekej na ukončení potomka (služba 4bh int21h)
  - ukonči proces (služba 00 int21h nebo služba 4ch int21h nebo int20h)
  - zjistí návratovou hodnotu ukončeného procesu (služba 4dh int21h)
  - ukonči proces, ale neuvolňuj paměť – (terminate, but stay resident) – TSR (služba 31h int21h nebo int 27h)



# Příklad: MS-DOS (2)

- Popisovač procesu (Program Segment Prefix)

| Offset | Size | Contents   |
|--------|------|--|
| +0     | 2    | EXE programs may JMP or RET here (PSP:0) to exit                     |
| +2     | 2    | top of available system memory in paragraphs                         |
| +4     | 1    | (reserved)   |
| +5     | 5    | FAR CALL to DOS function dispatcher                                  |
| +6     | (2)  | bytes available in Program Segment (for COM file only)               |
| +0aH   | 4    | Terminate address. See INT 22H                                       |
| +0eH   | 4    | Ctrl-Break handler address INT 23H                                   |
| +12H   | 4    | Critical Error handler addr INT 24H                                  |
| +16H   | 16H  | DOS reserved area  |
| +2cH   | 2    | Segment address of DOS environment                                   |
| +2eH   | 2eH  | DOS reserved area  |
| +5cH   | 10H  | setup as an FCB for 1st cmd parameter                                |
| +6cH   | 14H  | setup as an FCB for 2nd cmd parameter                                |
| +80H   | 1    | count of characters in UPA at 81H also offset of default DTA         |
| +81H   | 7fH  | characters from DOS command line (except any redirection directives) |
| 100H   |      | Program Segment Prefix size  |

# ● ● ● | Příklad: UNIX

- OS s preemptivním multitaskingem
- Služby OS
  - fork – vytvoří nový proces jako kopii rodiče
  - execve – nahradí současný proces spuštěním jiného programu
  - exit (resp. `_exit`) – ukončí proces
  - wait, waitpid – čeká na ukončení potomka

## ● ● ● | Příklad: UNIX (2)

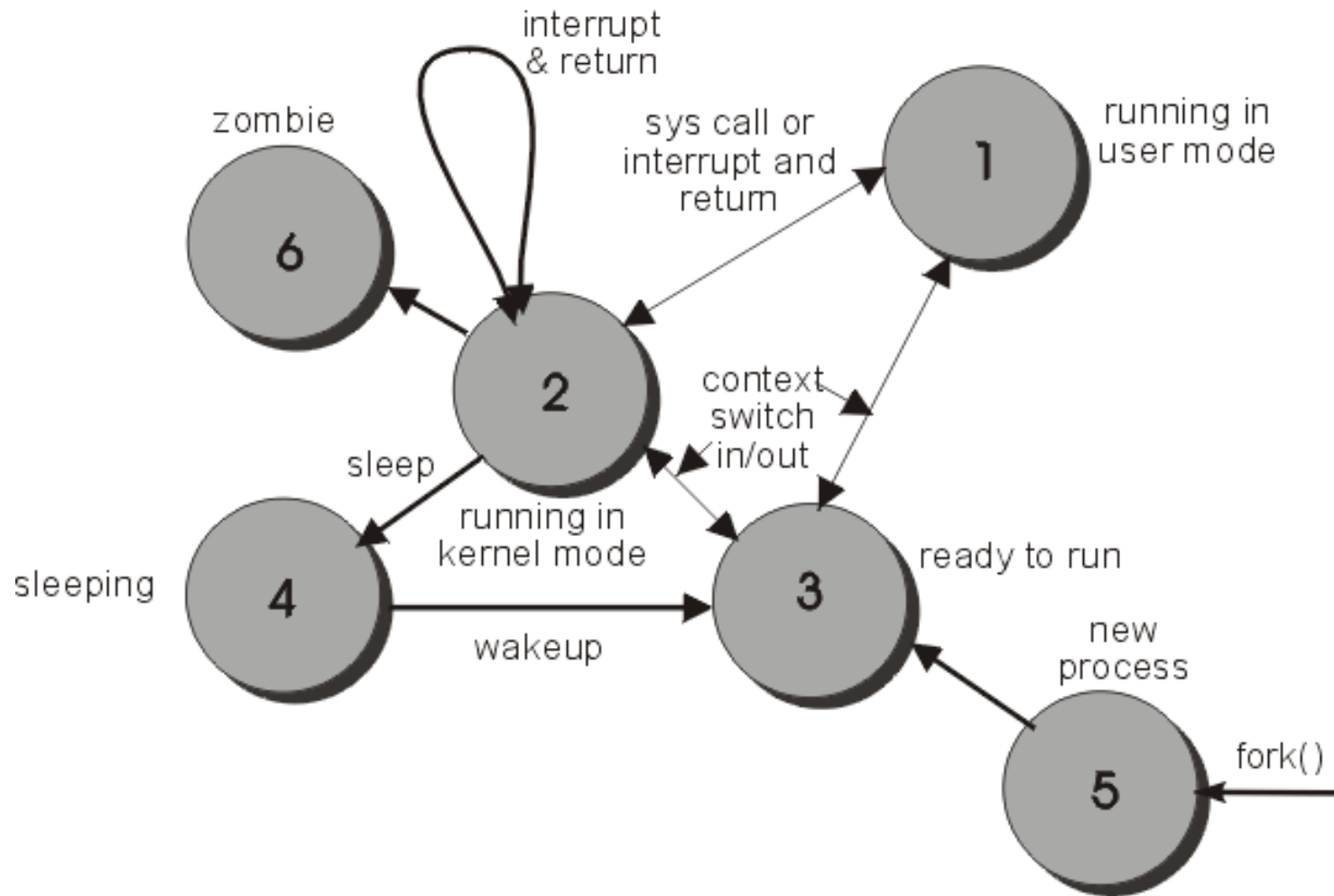
- Spuštění procesu a vyčkání na jeho ukončení

```
pid=fork();
if(pid<0) { perror("fork failed");
           exit(1); }
else if(pid==0) { /* child */
                 execlp("/bin/ls","ls",NULL); }
else { /* parent */
      wait(NULL);
      printf("child completed");
      exit(0);
    }
```

# ● ● ● | Příklad: Linux

- volání `fork()` implementováno jako copy-on-write (tj. dokud paměť není měněna je sdílena a až při pokusu o modifikaci je vytvořena kopie)
- `vfork` – upravené `fork`, které nekopíruje stránky paměti rodičovského procesu
  - rychlejší
  - vhodné pro okamžité spuštění execve
- `clone` – upravené `fork`, které umožňuje sdílet některé zdroje (například paměť, deskriptory souborů, ovladače signálů) mezi rodičovským a nově vytvořeným procesem.
- Informace o procesu jsou uloženy ve struktuře `task_struct` (viz `usr/include/sched.h`)

# Příklad: Linux (2)





# Příklad: Win32

## ○ Služby OS

- CreateProcess – vytvoří nový proces (spustí specifikovaný program)
- OpenProcess – pro získání přístupu k existujícímu procesu
- ExitProcess – ukončí tento proces
- TerminateProcess – ukončí nějaký (např. synovský) proces
- GetExitCodeProcess – zjistí návratovou hodnotu ukončeného procesu (nebo fakt, že proces ještě neskončil)





# Příklad: Win32 (CreateProcess)

```
STARTUPINFO si;  
PROCESS_INFORMATION pi;  
ZeroMemory( &si, sizeof(si) );  
si.cb = sizeof(si);  
ZeroMemory( &pi, sizeof(pi) );  
  
if(!CreateProcess(L"c:\\windows\\system32\\sol.exe", NULL, NULL, NULL, FALSE,  
    0, NULL, NULL, &si, &pi))  
{  
    printf( "CreateProcess failed (%d)\n", GetLastError() );  
    getch();  
    return;  
}  
  
WaitForSingleObject( pi.hProcess, INFINITE );  
CloseHandle( pi.hProcess );  
CloseHandle( pi.hThread );
```

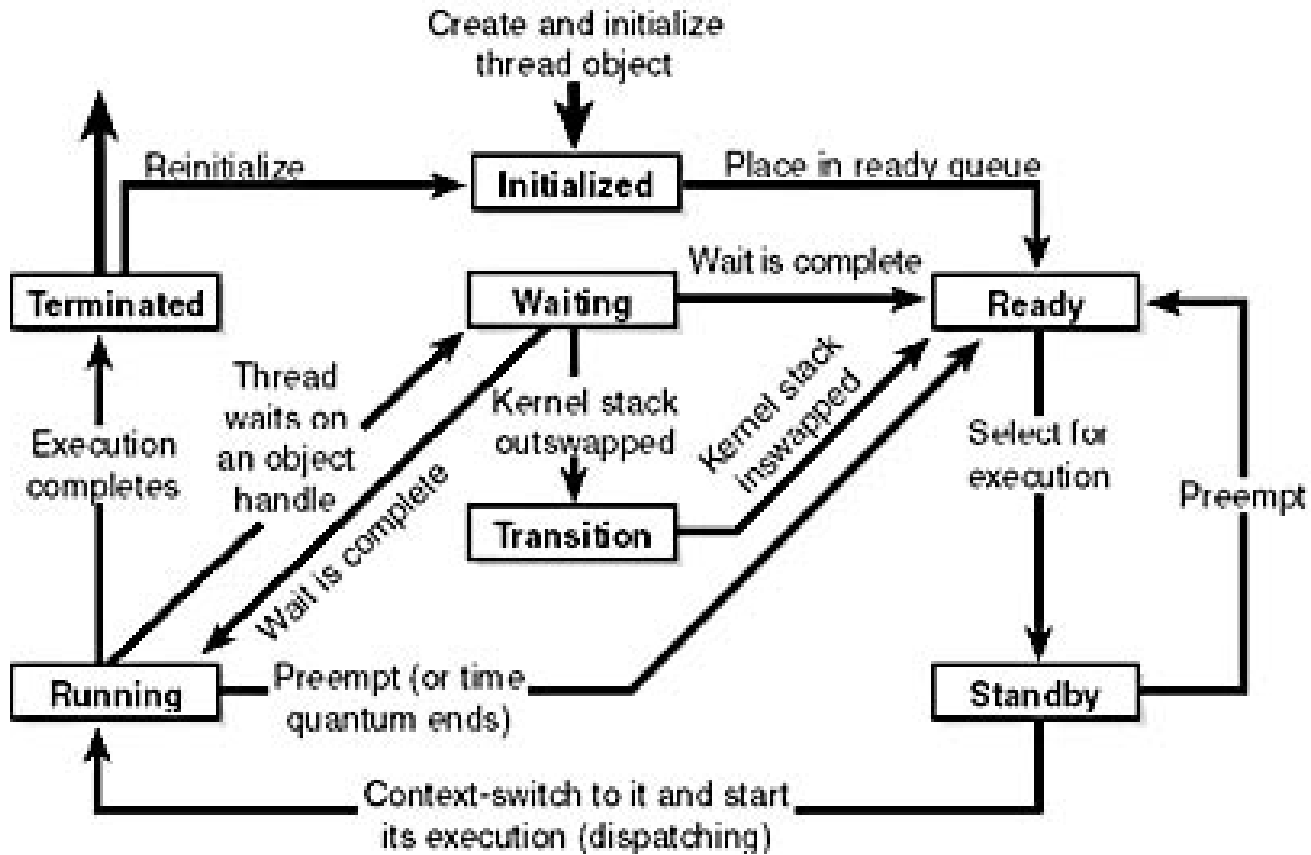


# Příklad: Win32 (EnumProcesses)

```
DWORD aProcesses[1024], cbNeeded, cProcesses;
```

```
if ( EnumProcesses( aProcesses, sizeof(aProcesses),  
    &cbNeeded ) )  
{  
    cProcesses = cbNeeded / sizeof(DWORD);  
    for (int i = 0; i < cProcesses; i++ )  
        if( aProcesses[i] != 0 )  
            VypisProces( aProcesses[i] );  
}
```

# Příklad: Win32





# Příklad: Win32

## Values for the Thread Wait Reason counter

- 0 Waiting for a component of the Windows NT Executive
- 1 Waiting for a page to be freed
- 2 Waiting for a page to be mapped or copied
- 3 Waiting for space to be allocated in the page or nonpaged pool
- 4 Waiting for an Execution Delay to be resolved
- 5 Suspended
- 6 Waiting for a user request
- 7 Waiting for a component of the Windows NT Executive
- 8 Waiting for a page to be freed
- 9 Waiting for a page to be mapped or copied
- 10 Waiting for space to be allocated in the page or nonpaged pool
- 11 Waiting for Execution Delay to be resolved
- 12 Suspended
- 13 Waiting for a user request
- 14 Waiting for an event pair high
- 15 Waiting for an event pair low
- 16 Waiting for an LPC Receive notice
- 17 Waiting for an LPC Reply notice
- 18 Waiting for virtual memory to be allocated
- 19 Waiting for a page to be written to disk
- 20+ Reserved