



Přednáška 6

Procesy a vlákna (vznik, stavy, atributy). Signály.

Nástroje pro práci s procesy a vlákny.



Procesy I

- Proces je **spuštěný program**.
- Každý proces má v rámci systému přiřazeno **jednoznačné číslo procesu PID**.
- Každý proces **má svého rodiče** a zná ho pod číslem **PPID**.
- Proces vzniká použitím systémových volání:
 - **fork ()**
 - vytvoří nový proces, který je kopií procesu, z kterého byla tato funkce zavolána
 - v rodičovském procesu vrací funkce PID potomka (v případě chyby -1)
 - v potomkovi vrací funkce 0
 - nový proces má jiné PID a PPID, ostatní vlastnosti dědí (např. EUID, EGID, prostředí, ...) nebo sdílí s rodičem (např. soubory, ...)
 - kódový segment sdílí potomek s rodičem
 - datový a zásobníkový segment vznikají kopií dat a zásobníku rodiče



Procesy II

- `exec ()`
 - v procesu, ze kterého je funkce volána, spustí nový program (obsah původního procesu je přepsán novým programem)
 - atributy procesu se nemění (PID, PPID, ...)



Příklad

```
main ()
{ ...
    pid = fork();
    switch (pid) {
    case -1: /* doslo k chybe */
        perror ("chyba ve funkci fork()");
        exit(1);
    case 0: /* program provadeny v potomkovi */
        printf ("PID procesu potomka: %d\n", (int) getpid ());
        execlp("sleep", "sleep", "30", (char *) NULL);
        perror ("chyba ve funkci execlp()");
        exit (1);
    default: /* program provadeny v rodici */
        printf ("PID procesu rodice : %d\n", (int) getpid ());
        wait(&status);
    };
    ... }
```

Vlákno (Thread)

- Vlákno je **spuštěný podprogram** v rámci procesu nebo jádra.
- **Proces tvořený více vlákny může být vykonáván na více procesorech současně.**
- Klasický jednovláknový proces může být vykonáván vždy jen na jednom procesoru.
- **Vytvoření nového vlákna je rychlejší** než vytvoření nového procesu.
- Nové vlákno může vzniknout např. pomocí knihovní funkce `pthread_create()`.

Příklad

```
...  
void *kod_vlakna(void *threadid)  
{ printf("ID vlakna: %d\n", threadid);  
  sleep(60);  
  pthread_exit(NULL);  
}  
  
int main()  
{ pthread_t threads[NUM_THREADS];  
  int rc, i;  
  for(i=0; i<NUM_THREADS; i++){  
    rc = pthread_create(&threads[i], NULL, kod_vlakna, (void *) i);  
    if (rc){ perror("Chyba ve funkci pthread_create()"); exit(1); }  
  }  
  pthread_exit(NULL);  
  ...  
}
```



Přepínání kontextu I

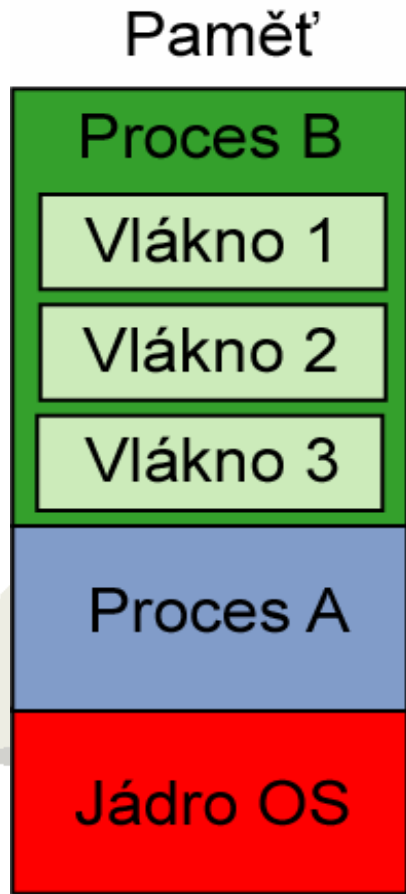
- Přepínáním kontextu se rozumí **střídání vláken na procesoru/ procesorech**.
- **Jádro OS určuje, kdy a které vlákno dostane přidělen procesor.**
- **Jádro se rozhoduje na základě celé řady informací:**
 - priorita procesu (např. 0 – 169 v Solarisu 10)
 - prioritní třída (např. TS, IA, FSS, FX, SYS, RT v Solarisu 10)
 - stav vlákna, chování vlákna v minulosti, ...
- **Vlákno dostává přidělený procesor vždy na určité časové kvantum.**
- Velikost časového kvanta se může lišit pro různé procesy i během času (závisí na implementaci Unixu).



Přepínání kontextu II

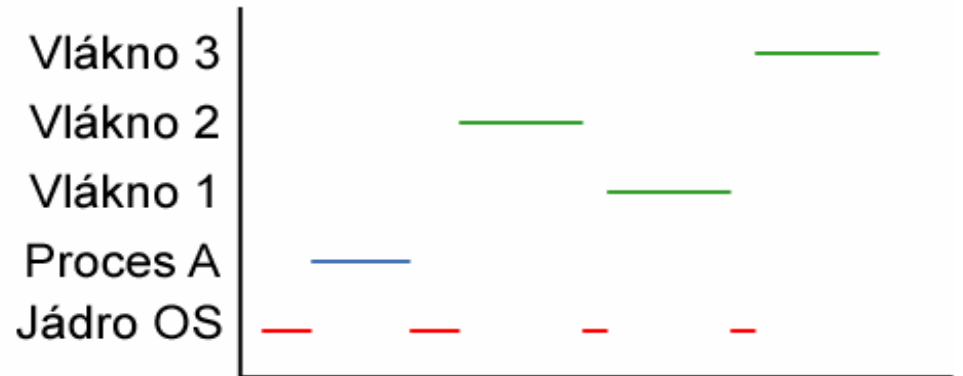
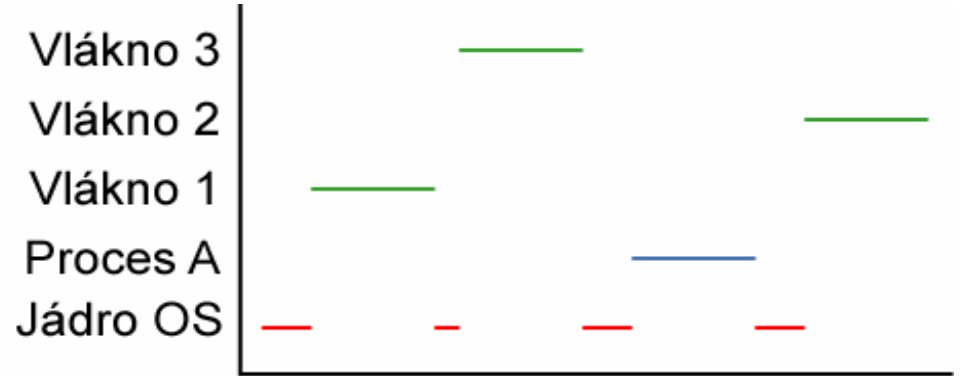
- **Priorita procesu se může dynamicky měnit:**
 - u běžících procesů se snižuje
 - u čekajících procesů se zvyšuje

Přepínání kontextu III

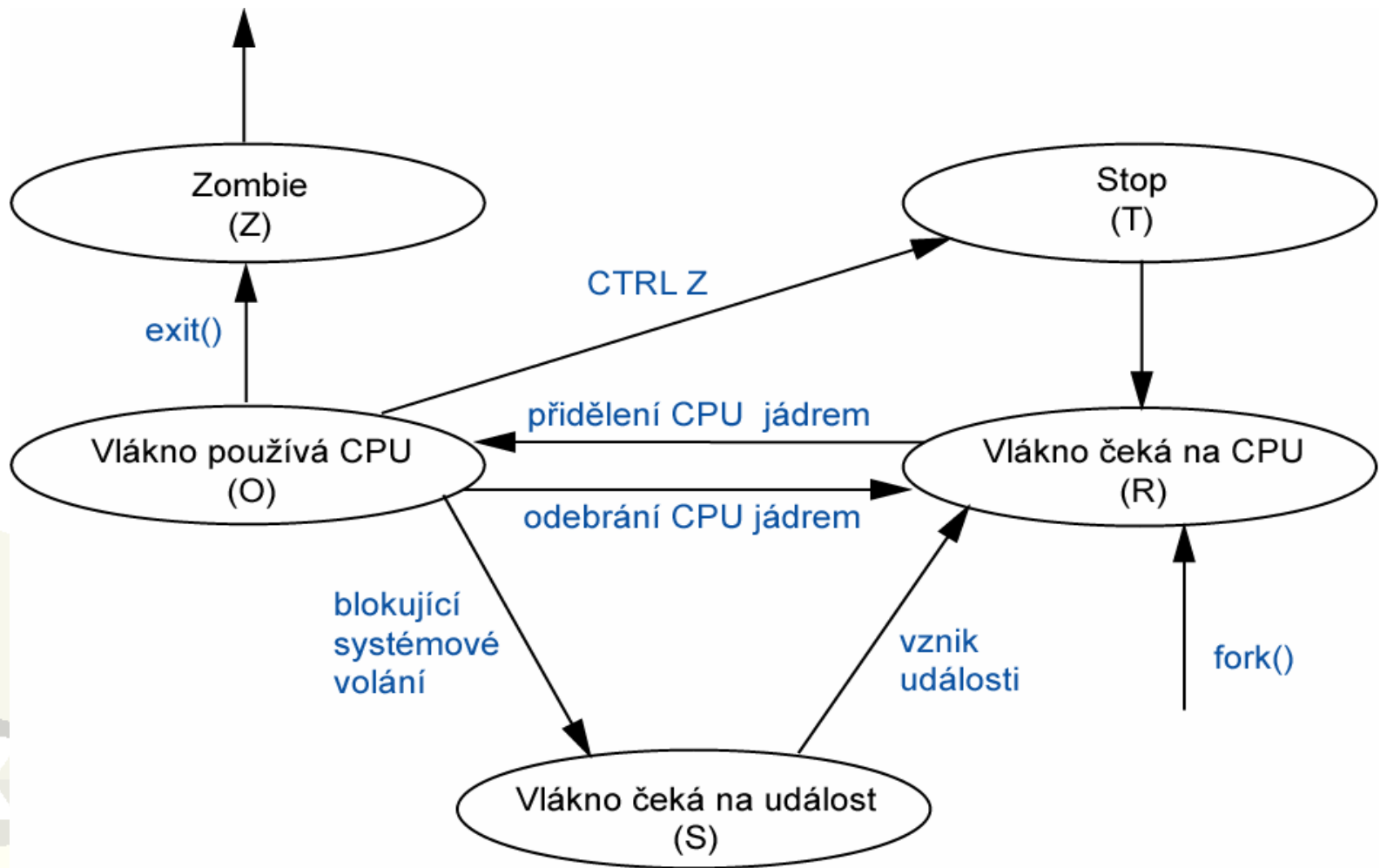


CPU 0

CPU 1



Stavy procesu



Výpis procesů/vláken I

ps

- Vypíše krátkou informaci o procesech odstartovaných z daného terminálu.

ps -e

- Vypíše krátkou informaci o všech procesech v systému.

ps -f nebo **ps -l**

- Vypíše detailnější informaci o procesech:
 - S stavy procesu (O, S, R, Z, T)
 - PID, PPID číslo procesu a číslo rodičovského procesu
 - PRI priorita (v Solarisu při `-c` mění význam a přibývá položka CLS)
 - NI NICE hodnota
 - STIME čas spuštění procesu
 - TIME čas spotřebovaný procesem na procesoru
 - TTY terminál přidělený procesu
 - CMD příkaz, který spustil proces

Výpis procesů/vláken II

`ps -o format`

- Informace jsou vypsané ve formátovaném tvaru.
- **format** může obsahovat následující zkratky:
 - user ruser group rgroup uid ruid gid rgid pid ppid pgid sid pri nice class time etime stime s c lwp ...

`ps -Le`

- Zobrazí informaci i o jednotlivých vláknech běžících v systému.

Výpis procesů/vláken III

`pgrep [-lvx] [vzor -u seznam_uživatelů ...]`

- Vypíše PID (s `-l` navíc jména) procesů, které splňují zadané parametry (např. obsahují ve svém jménu daný vzor, běží pod identitou daných uživatelů,...).
- S parametrem `-v` vypíše procesy, které nesplňují zadané parametry.
- **Vzor** může obsahovat i regulární výraz.

Výpis procesů/vláken IV

`prstat` nebo `top`

- Zobrazí seznam procesů seříděný podle zvoleného kritéria (explicitně podle zátěže CPU).

`ptree [-a] [pid] [uživatel]`

`pstree [-a] [pid] [uživatel]`

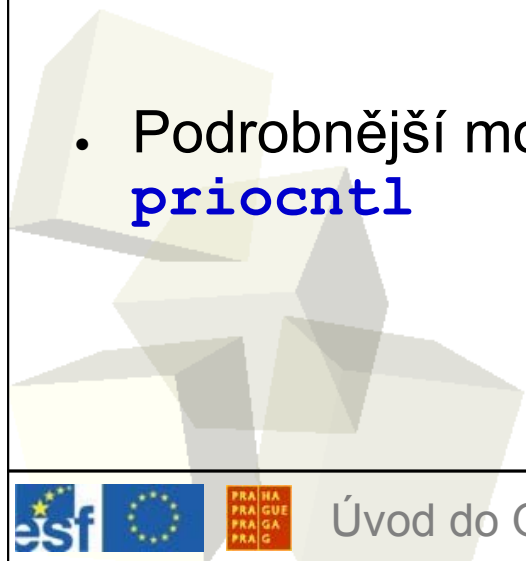
- Zobrazí strom procesů.



Modifikace priority

- **Počáteční prioritu lze snížit (root může i zvýšit) příkazem:**
`nice -priorita příkaz`
`nice -n priorita příkaz`
- kde `priorita` je číslo v rozsahu 1-19
- větší číslo = větší snížení priority
- záporné číslo = zvýšení priority (jen root)

- Podrobnější modifikace priorit lze provést příkazem `prionctl`



Jak správně psát skripty?

- Jaký je rozdíl v chování následujících skriptů?

```
while [ -f $1 ] ; do : ; done; echo " $1 zrusen "
```

```
while [ -f $1 ] ; do sleep 2 ; done; echo " $1 zrusen "
```

- Který z nich bude reagovat dříve na zrušení souboru, bude-li spuštěn najednou 100x (pokaždé s jiným jménem souboru)?

Signály I

- **Signály umožňují přerušování procesu zvenku** (např. jiným procesem, jádrem při špatném přístupu do paměti,...).
- Každý signál je definován jménem a číslem.
- V různých verzích Unixu se signály mohou lišit.
- **Seznam signálů lze vypsat:**
 - příkazem `kill -l`
 - v Solarisu pomocí manuálu `man -s 3HEAD signal`

- **Signál lze zaslat procesu pomocí příkazů:**

```
kill -signál PID
```

```
pkill -signál [ -vx] [ vzor -u seznam_uživatelů ...]
```

Signály II

- **Některé signály lze zaslat procesu pomocí kombinace kláves:**
- (viz. stty –a nebo man stty)

Kombinace kláves	Význam	Jméno signálu
CTRL C	Předčasné ukončení běžícího procesu.	2 SIGINT
CTRL \	Předčasné ukončení běžícího procesu.	3 SIGQUIT
CTRL S	Pozastavení výstupu na obrazovku.	23 SIGSTOP
CTRL Q	Uvolnění pozastaveného výstupu.	25 SIGCONT
CTRL Z	Pozastavení běžícího procesu (ne u sh). Nikoliv ukončení!!!	24 SIGTSTP



Důležité signály I

15 SIGTERM (TERMinate)

- Posílán příkazem `kill PID`.
- Standardní reakce je ukončení procesu. Obvyklý způsob předčasného ukončení procesu.

9 SIGKILL (KILL)

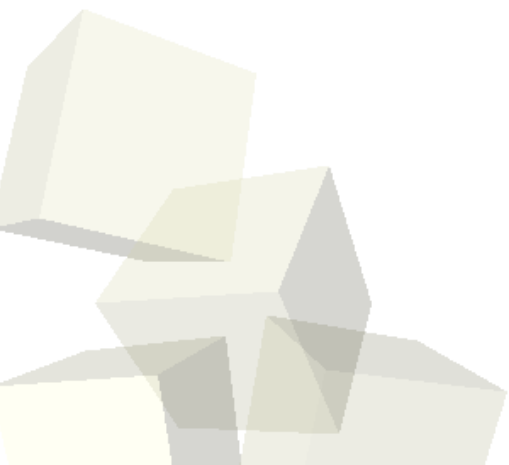
- Standardní reakce (kterou nelze změnit ani ignorovat) je ukončení procesu.



Důležité signály II

2 SIGINT (INTerrupt), 3 SIGQUIT (QUIT)

- Posílány procesům prostřednictvím terminálového ovladače při stisku nadefinovaných znaků.
- Standardní reakce je ukončení (2) a ukončení s vytvořením souboru core (3).
- Nastavení terminálového ovladače lze vypsát /změnit příkazem **stty**.



Důležité signály III

1 **SIGHUP (HangUP)**

- Posílán procesu, když končí jeho rodič (shellu, když zavěsil modem).
- Standardní reakce je ukončení. Končí-li proces, je signál poslán všem jeho potomkům. Proto při odhlášení jsou ukončeny všechny procesy spuštěné v rámci tohoto přihlášení.
- Má-li proces pokračovat po ukončení rodiče, je třeba použít příkaz

nohup příkaz &

- Některé procesy signál odchyťávají a interpretují ho jako žádost o restart (např. init, named, sendmail, inetd, syslogd, automountd).

Reakce na signály

- Každý proces má při startu definovanou reakci na všechny signály.
- Obvykle je to „konec“ nebo „core a konec“.
- **Proces může reakci předefinovat kromě signálů KILL a STOP.**
- Reakci na signály můžeme měnit příkazem `trap`.

- **Nastavení reakce na signál(y):**

```
trap 'příkazy' signály_oddělené_mezerou
```

- **Výpis nastavených reakcí:**

```
trap
```

- **Ignorování signálů:**

```
trap ' ' signály_oddělené_mezerou
```

- **Nastavení výchozí reakce:**

```
trap - signály_oddělené_mezerou
```

Spuštění příkazu I

- **Na popředí**

 - **\$ příkaz**

 - Příkaz (který není zabudovaný) je spuštěn jako nový proces.
 - Proces přebírá vstup i výstup.
 - Shell čeká na jeho dokončení.

- **Na pozadí**

 - **\$ příkaz &**

 - Shell nečeká na jeho dokončení a nepředává mu vstup (pokud nebyl přesměrován).
 - Jestliže se proces pokusí číst z nepředaného vstupu je pozastaven.
 - Na výstup psát může. Pokud výstup nebyl přesměrován, míchá se s výstupem shellu.

Spuštění příkazu II

- **Mimo shell**

`$ nohup příkaz &`

- Proces poběží i po ukončení shellu.
- Výstup procesu je přeměřován do souboru `nohup.out`.

- **V daném čas**

- Pomocí příkazu `at` nebo `crontab` lze naplánovat spuštění procesu v daném čase.
- Spuštění procesu provádí systémový démon `cron`.
- Proces poběží explicitně pod identitou toho, kdo proces naplánoval a výstup bude poslán mailem.

Správa úloh

- Umožňují všechny shelly kromě Bourne shellu (/bin/sh).
- **Každý proces, který je spuštěn v dané instanci shellu, má v této instanci přiřazeno jedinečné číslo úlohy (JID).**
- Pomocí JID se můžeme na proces odkazovat pomocí příkazů:
`jobs` vypíše seznam úloh(procesů) běžících v tomto shellu
`fg [%JID]` přesune úlohu na popředí
`bg [%JID]` přesune úlohu na pozadí
`kill -signál [%JID]` pošle úloze daný signál
- Není-li v příkazu uvedeno JID, uvažuje se poslední úloha.



Užitečné příkazy

- **Doba běhu procesu**

`time` příkaz

- Příkaz vypíše informaci o čase spotřebovaném procesem.

- **Systemová volání**

`truss` příkaz

- Příkaz vypíše systémová volání, které daný proces používá.