ZOS CV 6/2024

L. Pešička

OBSAH

- Práce s procesy
  - fork()
  - execve()
  - wait()
- Grafy procesů
  - Každý proces samostatný sloupec
  - Znázorněna důležitá volání jako fork(), execve()
  - Znázorněny výstupy printf()
- Grafy paralelních procesů
  - cobegin / coend
  - Synchronizace procesů, max. paralelismus

PROCES

- Tabulka procesů
  - Každý proces v ní má záznam (řádka v tabulce) – PCB
  - PID, získáme funkcí getpid()
    PPID, získáme funkcí getppid()
- Stavy
  - Nový – jde do připravených
  - Připravený – plánovač vybírá, kdo poběží
  - Běžící – max tolik kolik máme k dispozici jader CPU
  - Blokovaný – čeká na událost
  - Ukončený – (mezistav zombie -> ukončený)

PROCESY

Vytvoření nového procesu v Linuxu (Unixu) systémovým voláním **fork()**:

```
int id;
id = fork();
if (id ==0)
   printf(„jsem potomek\n“);
else
  printf(„jsem rodič, potomek má PID %d\n“, id);
```

Zde běží 1 proces

Zde běží 2 procesy liší se návratovou hodnotou forku, tj. proměnnou id

STAŽENÍ PŘÍKLADŮ

Pro usnadnění práce lze stáhnout jednotlivé příklady:

wget http://home.zcu.cz/~pesicka/zos/nazev

Kde nazev je f1.c, zombie.c, f2.c atp.

CELÝ PŘÍKLAD – F1.C

```
#include <stdio.h>
#include <unistd.h>

int main (void) {
int i;
i = fork();

if (i == 0)
  printf ("Jsem potomek s pidem %d, rodic ma %d\n", getpid(),  getppid());

else
  printf ("Jsem rodic s pidem %d, potomek ma %d\n", getpid(), i);
}
```

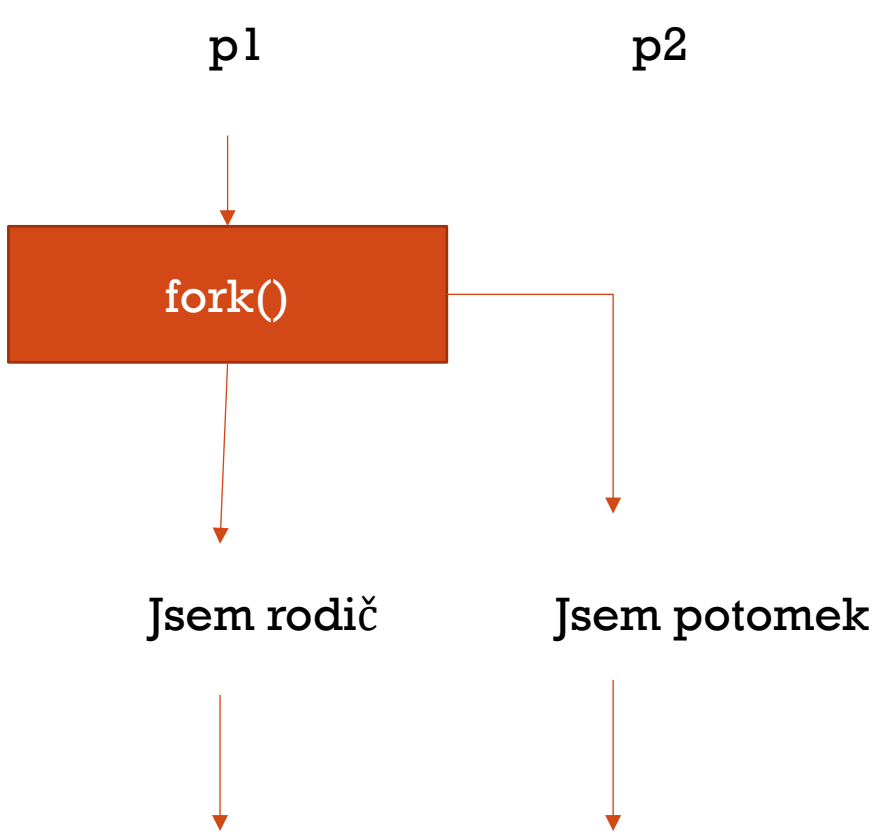
OTESTOVÁNÍ POD LINUXEM

- wget http://home.zcu.cz/~pesicka/zos/f1.c
- přeložte: **gcc –o f1 f1.c**
- spusťte: **./f1**

úkoly:

1. nakreslit graf běhu programu
2. modifikovat příklad na zombie
   (potomek skončí, ale rodič a nepřečte si
   jeho návratový kód – pozorujte z další konzole)

GRAF



každý sloupec zobrazuje běh jednoho procesu

ZOMBIE – ZOMBIE.C

```
#include <stdio.h>
#include <unistd.h>

int main (void) {
int i,j;

i = fork();
if (i == 0)
  printf ("Jsem potomek s pidem %d, rodic ma %d\n", getpid(), getppid());
else {
  printf ("Jsem rodic s pidem %d, potomek ma %d\n", getpid(), i);
  for (j=10; j<100; j++) j=11;
  }
}
```

ÚKOLY S FORKEM() – F2.C

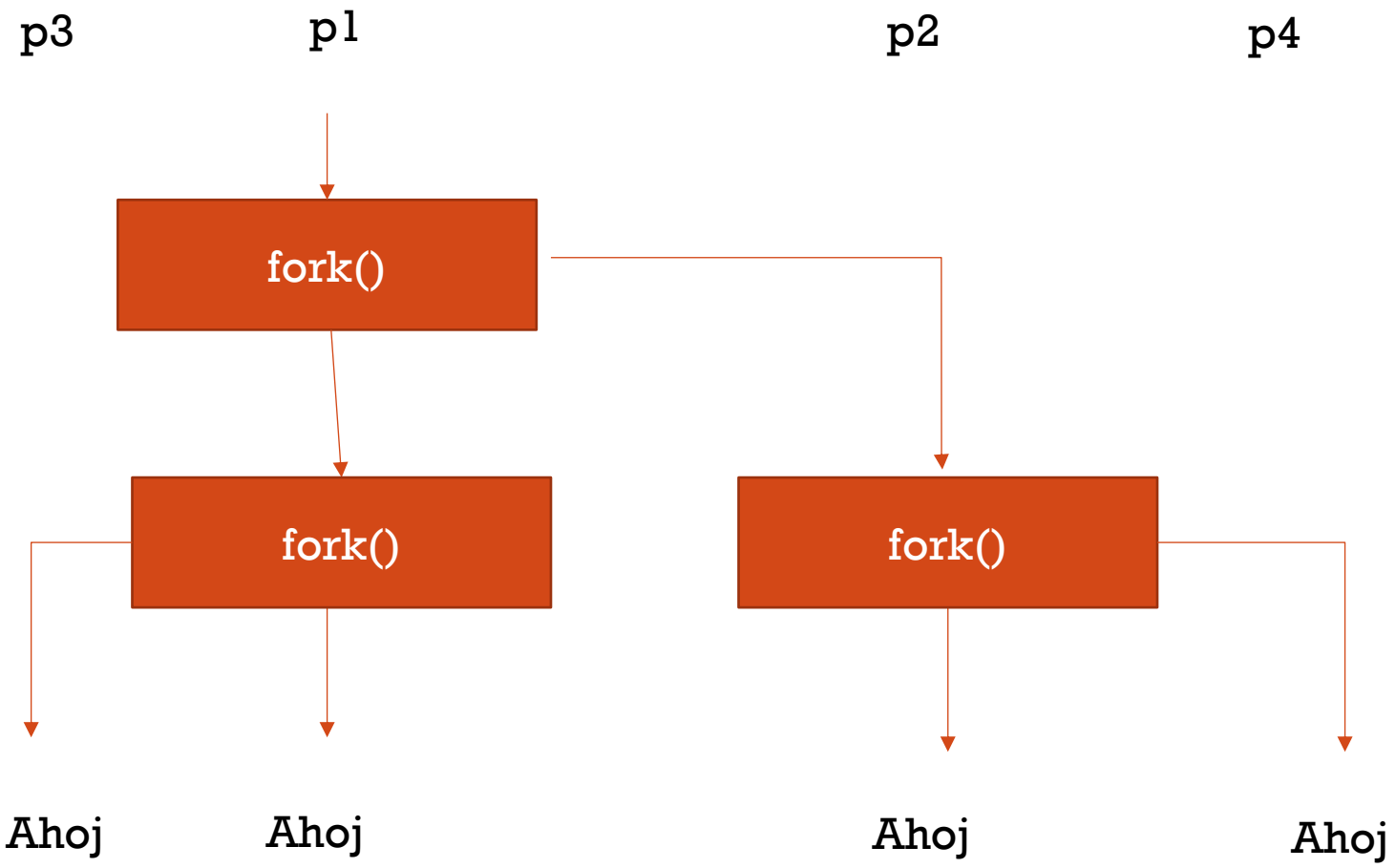
Máme fragment kódu, **nakreslete graf** procesů a určete, kolikrát se vypíše příslušný řetězec:

```
…
fork();
fork();
printf (“Ahoj\n”);
return 0;
```

Zkuste:
./f2 | wc –l

Nakreslete graf procesů

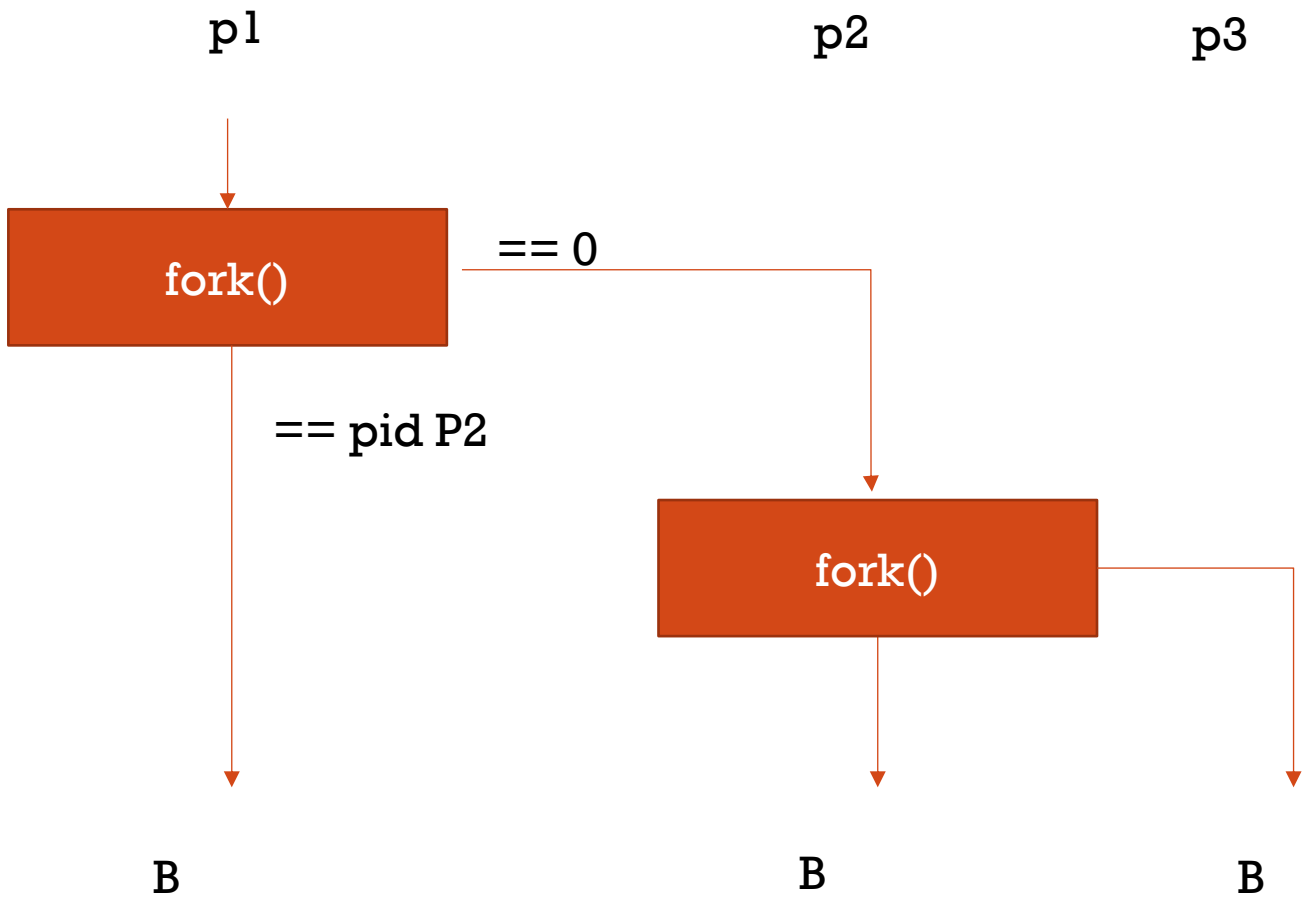
ODPOVÍDAJÍCÍ GRAF



ÚKOLY S FORKEM – F3.C

```
#include <stdio.h>
int main (void) {

if (fork() == 0)
   fork();

printf("B\n");
return 0;
}
```

ODPOVÍDAJÍCÍ GRAF



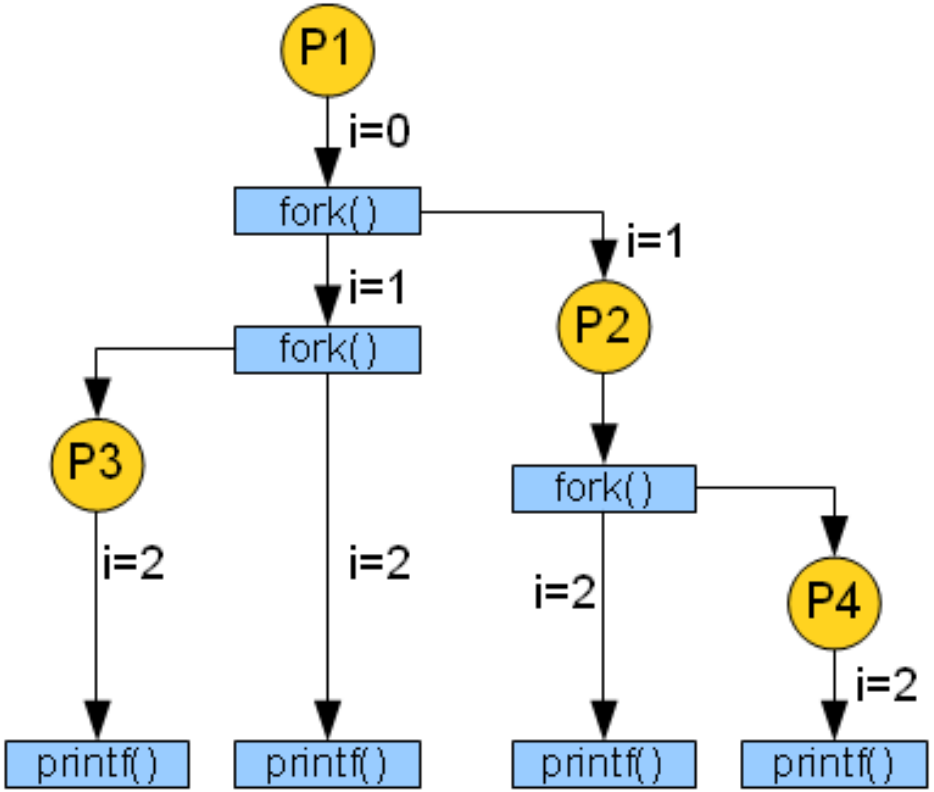
ÚKOLY S FORKEM – F4.C

```
#include <stdio.h>
int main (void) {
int i;

for (i=0; i<2; i++)
   fork();

printf("C\n")
 return 0;
 }
```

POPIS ČINNOSTI

- P1
  - Proměnná i má hodnotu 0
  - Provede fork – vznikne P2
  - Proměnná i se iterací for nastaví na 1
  - Provede fork – vznikne P3
- P2
  - Proměnná i se iterací for nastaví na 1
  - Provede fork – vznikne P4
- P3 a P4
  - Další iterace for už se nevykoná
- Výsledek: 4x se vypíše C

ODPOVÍDAJÍCÍ GRAF



ZÁKLADNÍ OPERACE S PROCESY

Systémové volání	Funkce
fork()	Vytvoření procesu
wait()	Čekání na dokončení procesu
_exit() , exit()	Ukončení procesu (systémové volání, knihovní funkce)
execve(), execl()	Nahradí aktuální kód procesu kódem z daného souboru Celá rodina volání exec* execve –systémové volání execl – knihovní funkce

ÚKOL (!!)

Napište program f5.c:

- Rodič čeká na dokončení potomka wait()
- Potomek vypíše jsem potomek
- Potomek spustí program /bin/date

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main(void) {
 int id, status;

 id = fork();
if (id == 0) {
       printf("jsem potomek\n");
       execl("/bin/date", "date", NULL);
      printf("Sem uz nedojdu");
}
 else {
  printf("jsem rodic, potomek ma PID %d\n",id);
  wait(&status);
} }
```

f5.c
Úkol 1:
dejte za execl
printf("Sem uz nedojdu");

Úkol 2:
dále zkuste spustit execem
neexistující program

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main(void) {

int id, id2, sta, stb;

id = fork();
if (id == 0) {
        printf("jsem potomek\n");
        sleep(5);
} else {
         id2 = fork();
        if (id2 == 0 ) {
                printf("jsem druhy potomek\n");
                sleep(5);
        }
else {
         wait(&sta); wait(&stb); printf("oba potomci skoncili\n");}
}
}
```

f6.c

Vyzkoušejte

PROCESY

- po vytvoření forkem jsou procesy zcela samostatné
- chceme-li aby „něco“ sdíleli (paměť), musíme to naopak explicitně zařídit
- rozdíl proti vláknům - vlákna sdílejí globální proměnné
- řešení souběžného přístupu – kritická sekce

UKÁZKY IPC

InterProcess Communication

- Prostředky pro komunikaci mezi procesy

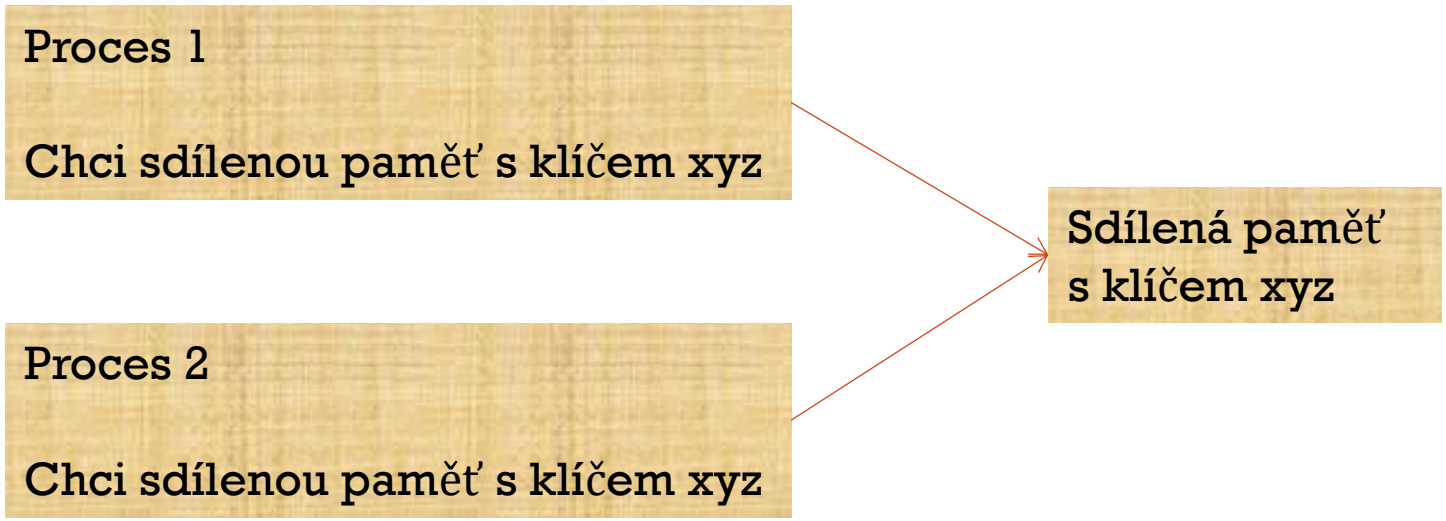
Možnosti

- Posílání zpráv mezi procesy
- Použití nepojmenované roury mezi příbuznými procesy
- Použití sdílené paměti

UKÁZKY IPC

klient.c, server.c

Sdílená paměť



```c
#define SHMSZ     27
main()
{
    char c;
    int shmid;
    key_t key;
    char *shm, *s;

    /*
     * We'll name our shared memory segment
     * "5678".
     */
    key = 5678;

    /*
     * Create the segment.
     */
    if ((shmid = shmget(key, SHMSZ, IPC_CREAT | 0666)) < 0) {
        perror("shmget");
        exit(1);
    }

    /*
     * Now we attach the segment to our data space.
     */
    if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
        perror("shmat");
        exit(1);
    }
```

Server

Klíč 5678

Žádáme o sdílenou paměť shmget()

Když ji dostaneme, připojení do našeho adresního prostoru shmat()

```
/*
 * Now put some things into the memory for the
 * other process to read.
 */
s = shm;

for (c = 'a'; c <= 'z'; c++)
    *s++ = c;
*s = NULL;

/*
 * Finally, we wait until the other process
 * changes the first character of our memory
 * to '*', indicating that it has read what
 * we put there.
 */
while (*shm != '*')
    sleep(1);

exit(0);
```

Server

Když už sdílenou pamět má, zapíše do ní znaky a..z

Potom čeká, až se první znak změní na * a tím se dozví, že si klient data přečetl

```c
key = 5678;

/*
 * Locate the segment.
 */
if ((shmid = shmget(key, SHMSZ, 0666)) < 0) {
    perror("shmget");
    exit(1);
}

/*
 * Now we attach the segment to our data space.
 */
if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
    perror("shmat");
    exit(1);
}

/*
 * Now read what the server put in the memory.
 */
for (s = shm; *s != NULL; s++)
    putchar(*s);
putchar('\n');

/*
 * Finally, change the first character of the
 * segment to '*', indicating we have read
 * the segment.
 */
*shm = '*';

exit(0);
```

Klient

Klíč 5678

shmget()

shmat()

Přečte data a na první pozici uloží znak *

SDÍLENÁ PAMĚŤ

- **shmget()** .. vytvoření sdíleného paměťového segmentu
- **shmat()** .. připojení sdíleného segmentu do paměťového prostoru procesu (shmdt odpoj)

- **shmctl()** .. různá nastavení, viz man

poznámka:

Také se používá mapování souboru do adresního prostoru přes mmap(), munmap()

NEPOJMENOVANÁ ROURA - PIPE

```
#include <unistd.h>

int sharedPipe[2];

pipe(sharedPipe);
```

- vytvoří rouru
  - sharedPipe[0] .. read end – z něj se čte („stdin“)
  - sharedPipe[1] .. write end – do něj se zapisuje („stdout“)

NEPOJMENOVANÁ ROURA

```
#include <unistd.h>

int sharedPipe[2];

pipe(sharedPipe);
if (fork() == 0)
  // jeden konec roury [0] – ctu z nej, potomek
  // druhý nepoužívaný zavřu

else
  // jeden konec roury [1] – zapisuji do něj
  // druhý nepoužívaný zavřu
```

POSÍLÁNÍ ZPRÁV

- http://home.zcu.cz/~pesicka/zos/message.c

- message key
- msgsnd() – poslání zprávy
- msgrcv() – příjem zprávy

ZÁPIS PROCESŮ COBEGIN - COEND

- Nyní nezkoumáme vnitřek jednotlivých procesů, ale jejich vzájemné vztahy
- Využití maximálního paralelismu
  - Spustit vše, co je v daný okamžik možné

- Paralelní činnost p1 a p2 zapíšeme:

```
cobegin
   p1 || p2
coend
```

- pro zpřehlednění můžeme více činností uzavřít mezi begin a end

Příklad s cobegin/coend

- Příklad zadání: Kolejiště

  Stavíme kolejiště.

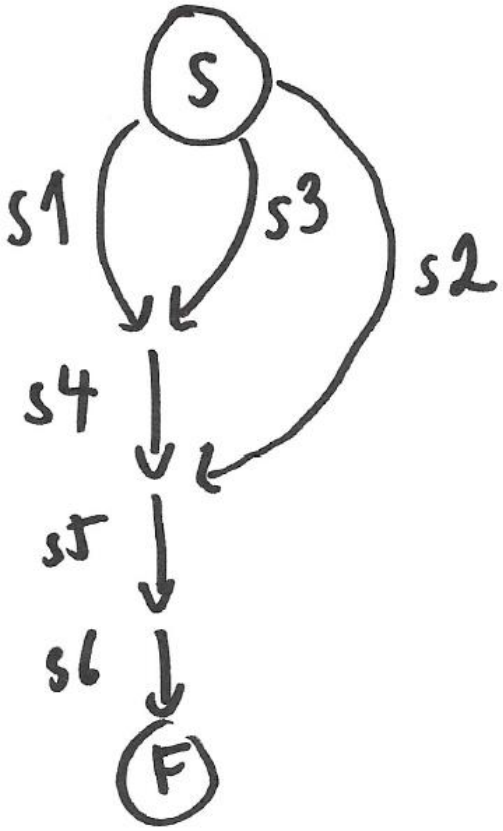
  Nejprve můžeme současně začít kupovat koleje (p1), shánět vlaky (p2) a stavět podkladovou desku (p3).

  Jakmile máme koupené koleje a postavenou podkladovou desku, můžeme začít stavět koleje (p4).

  Když jsou koleje postavené a vlaky koupené, vypravíme vlak (p5).

  Video z fungujícího kolejiště nasdílíme na Instagram(p6).

GRAF COBEGIN/COEND



Popisek s1 odpovídá p1 atd.

Jiné značení je zde jen
z technických důvodů

```
cobegin
begin
  cobegin
     C1 || C3
  coend
  C4
end
||
C2
coend
C5
C6
```

MATERIÁLY NA COURSEWARU

- Cvičení – Materiály ke cvičení – C, Java příklady

Dokument	Popis
cobegin/coend	Zápis paralelních procesů pomocí cobegin / coend včetně grafů
graf_parallel	Využití maximálního paralelismu, precedenční grafy, cobegin / coend
graf_fork	Graf procesů volání fork(), výpisy (podrobný popis průběhu procesu)