Operacinės sistemos P175B304

07T

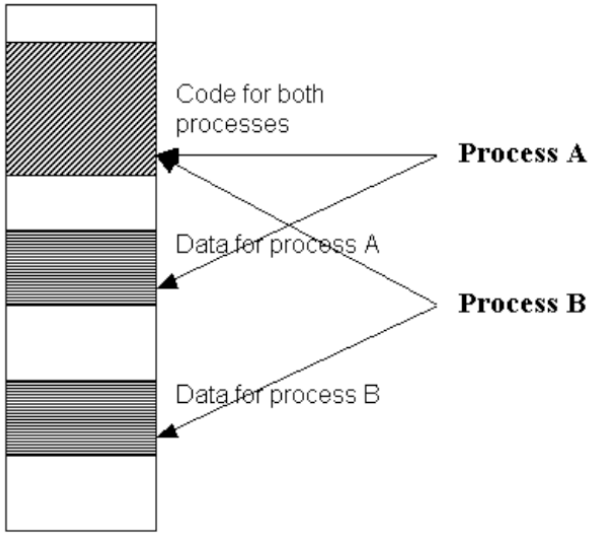
2018-02-19
2018-02-20

Procesų API

Operacija	API funkcijos (UNIX)
Kurti procesą	`fork (), execve(name, argv, envp), ...`
Naikinti procesą	`exit(status)`
Laukti proceso pabaigos	`wait(&status), waitpid(pid, &statloc, opts), ...`
Skaityti proceso info	`getpid(), getpgrp(), ...`
Kitos procesų valdymui skirtos operacijos	`kill(pid, sig), sigaction(sig, &act, &oldact), pipe (fd)...`

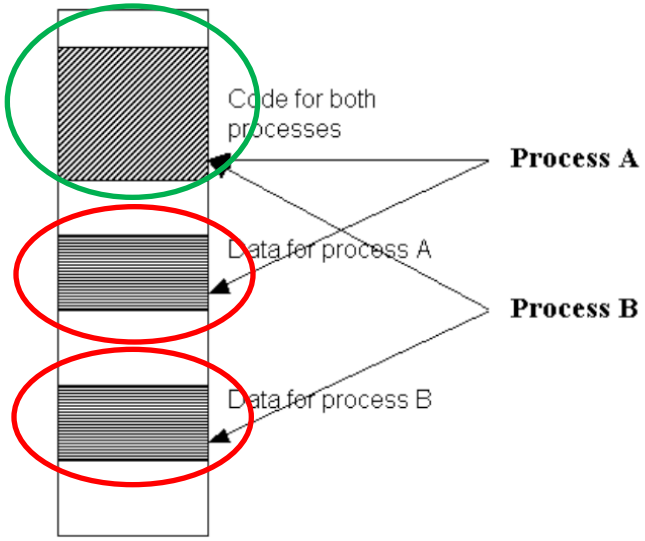
Proceso sukūrimas

- Procesai:
  - Tėvinis procesas (angl. parent process)
  - Vaiko procesas (angl. child process)
- Procesų hierarchijos
- Bendri resursai:
  - Tėvo ir vaiko procesai dalinasi visais resursais
  - Vaiko procesas prieina prie dalies tėvo proceso resursų
  - Tėvo ir vaiko procesai resursais nesidalina



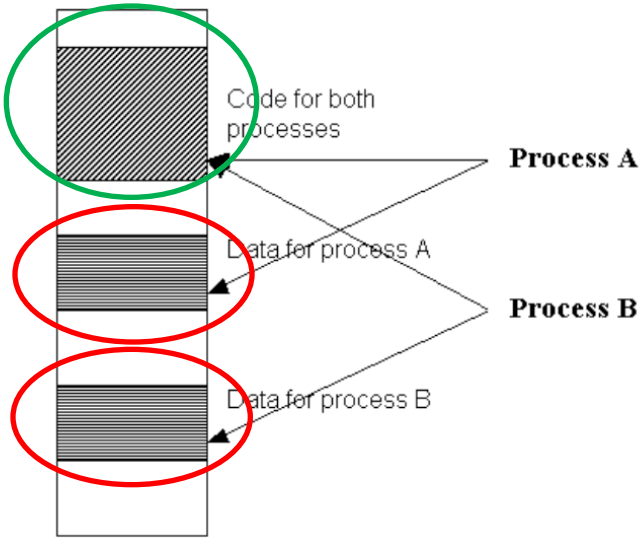
Proceso sukūrimas

- Procesai:
  - Tėvinis procesas (angl. parent process)
  - Vaiko procesas (angl. child process)
- Procesų hierarchijos
- Bendri resursai:
  - Tėvo ir vaiko procesai dalinasi visais resursais
  - Vaiko procesas prieina prie dalies tėvo proceso resursų
  - Tėvo ir vaiko procesai resursais nesidalina



Proceso sukūrimas

- Procesų vykdymas:
  - Tėvo ir vaiko procesai vykdomi lygiagrečiai
  - Tėvo procesas laukia kol vaiko procesas baigs vykdyti veiksmus



- Adresų erdvė:
  - Vaiko adresų erdvė yra tėvo adresų erdvės kopija (tėvo programos kopija)
  - Vaiko procesas vykdo kitą programą (nėra tėvo programos kopija)

Proceso sukūrimas: `fork()`

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 int
6 main(int argc, char *argv[])
7 {
8 printf("hello world (pid:%d)\n", (int) getpid());
9 int rc = fork();
10 if (rc < 0) { // fork failed; exit
11 fprintf(stderr, "fork failed\n");
12 exit(1);
13 } else if (rc == 0) { // child (new process)
14 printf("hello, I am child (pid:%d)\n", (int) getpid());
15 } else { // parent goes down this path (main)
16 printf("hello, I am parent of %d (pid:%d)\n", rc, (int) getpid());
17 }
18 return 0;
19 }
```

```
prompt> ./p1
hello world (pid:29146)
hello, I am parent of 29147 (pid:29146)
hello, I am child (pid:29147)
prompt>
```

Proceso pabaigos laukimas: `wait()`

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 int
6 main(int argc, char *argv[])
7 {
8 printf("hello world (pid:%d)\n", (int) getpid());
9 int rc = fork();
10 if (rc < 0) { // fork failed; exit
11 fprintf(stderr, "fork failed\n");
12 exit(1);
13 } else if (rc == 0) { // child (new process)
14 printf("hello, I am child (pid:%d)\n", (int) getpid());
15 } else { // parent goes down this path (main)
16 int wc = wait(NULL);
17 printf("hello, I am parent of %d (pid:%d)\n", rc, (int) getpid());
18 }
19 return 0;
20 }
```

```
prompt> ./p2
hello world (pid:29266)
hello, I am child (pid:29267)
hello, I am parent of 29267 (wc:29267) (pid:29266)
prompt>
```

Proceso keitimas kita programa: `exec*()`

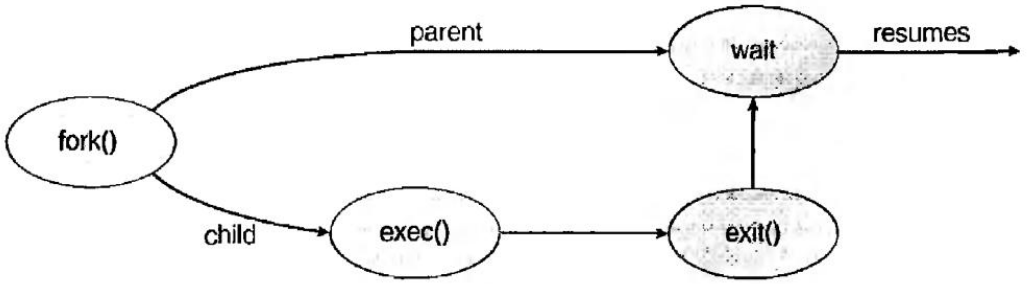
(supaprastintas shell variantas)

```
...
while (TRUE) {                          /* repeat forever */
   type_prompt ();                      /* display prompt on the screen */
   read_command (command, params);      /* read input line from keyboard */

/* fork a child process */

pid = fork () ;
if (pid < 0) {                          /* error occurred */
fprintf(stderr, "Fork Failed");
continue;                               /* repeat the loop */
             }

else if (pid == 0) {                    /* child process */
execve(command, params, NULL);          /* child does the work */
                    }

else {                                  /* parent process */
wait(NULL);                             /* parent will wait */
                                        /* for the child to complete */
     }
}
```

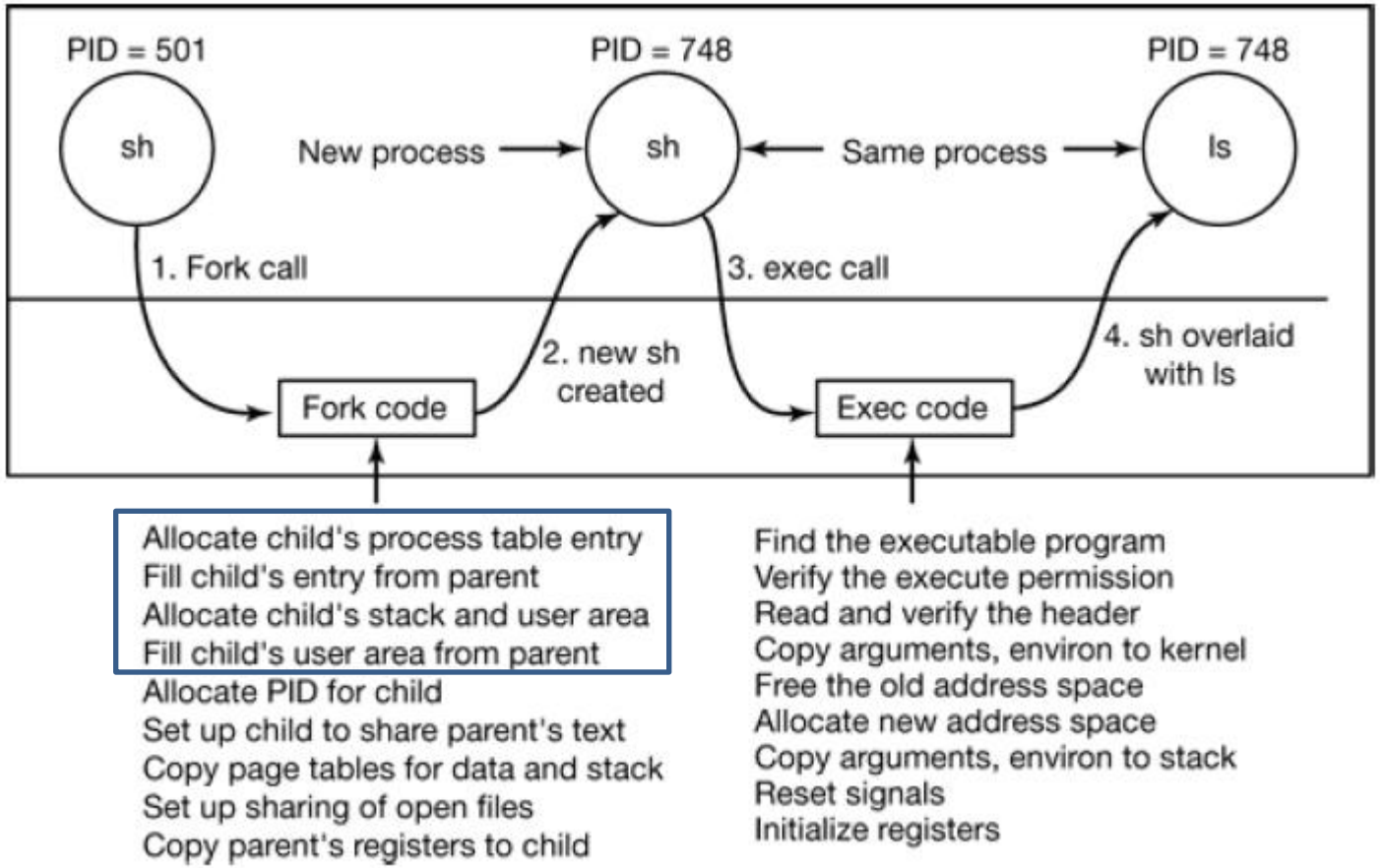
Proceso keitimas kita programa: `exec*()`

(pavyzdys su komanda ls)

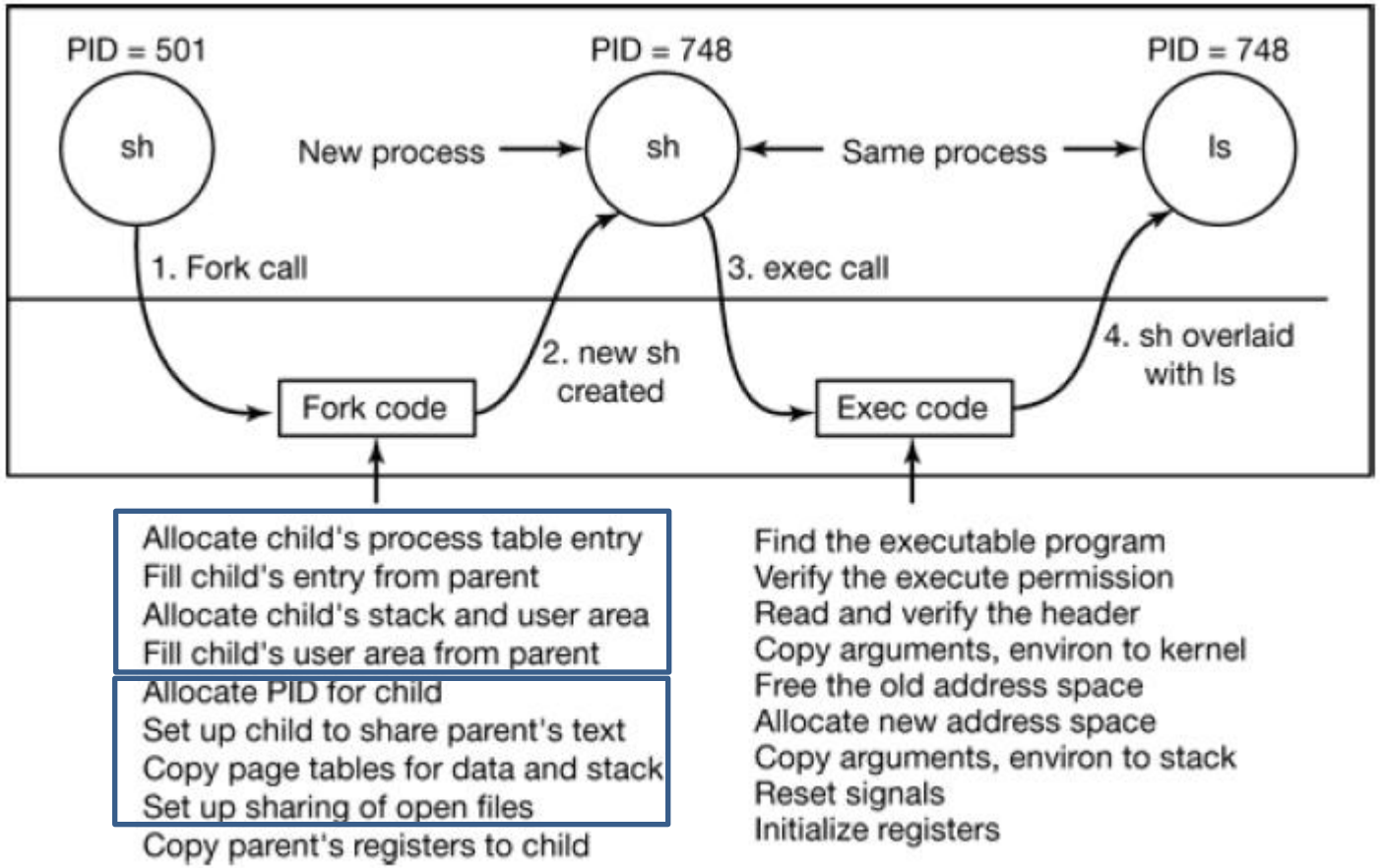
```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main ()
{
pid_t pid;
/* fork a child process */

pid = fork () ;
if (pid < 0) {/* error occurred */
fprintf(stderr, "Fork Failed");
exit (-1) ;
}

else if (pid == 0) {/* child process */
execlp("/bin/ls","ls", NULL);
}

else {/* parent process */
/* parent will wait for the child to complete */
wait(NULL);
printf("Child Complete");
exit (0) ;
}
}
```



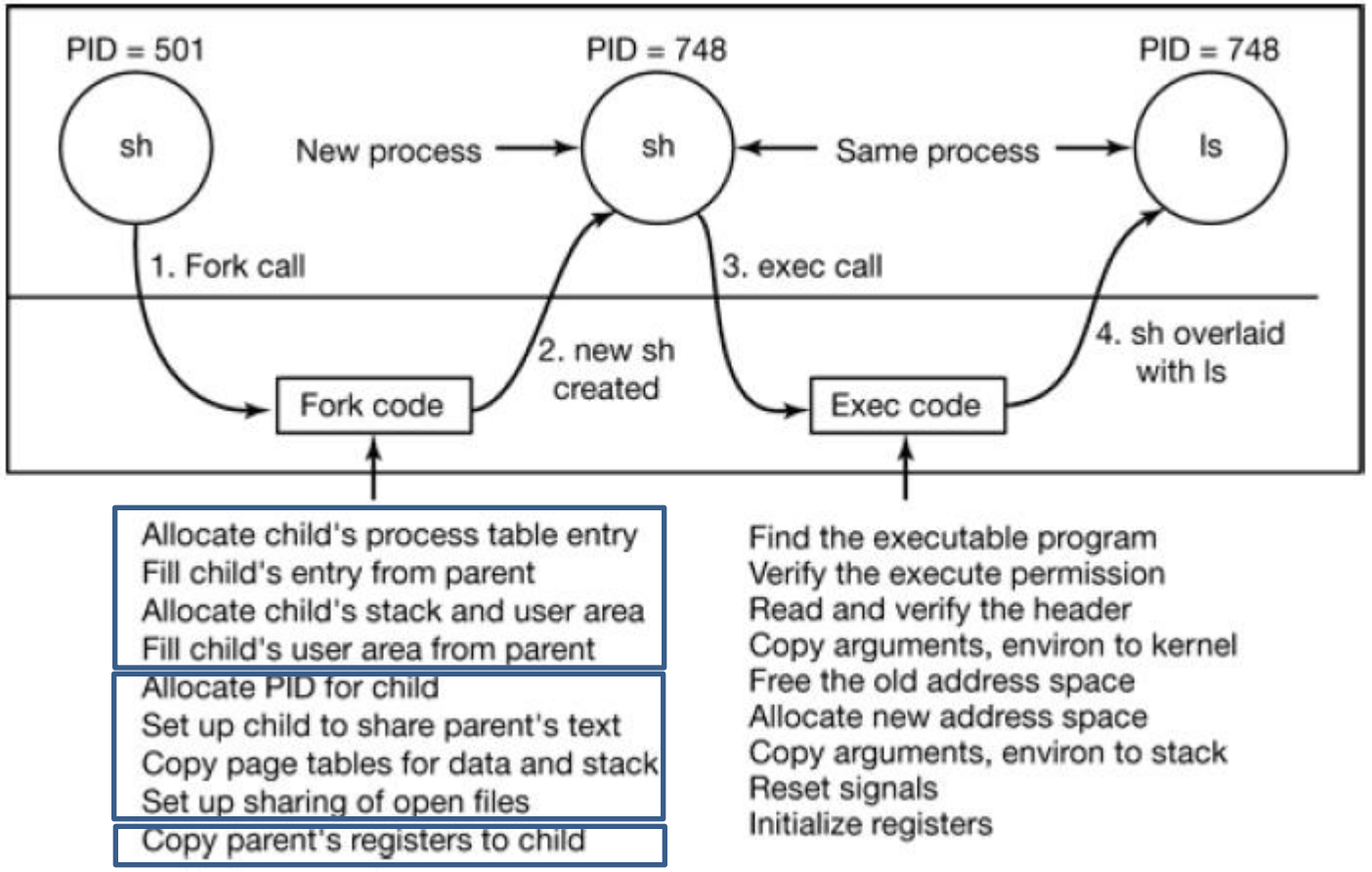
Proceso keitimas kita programa: `exec*()`



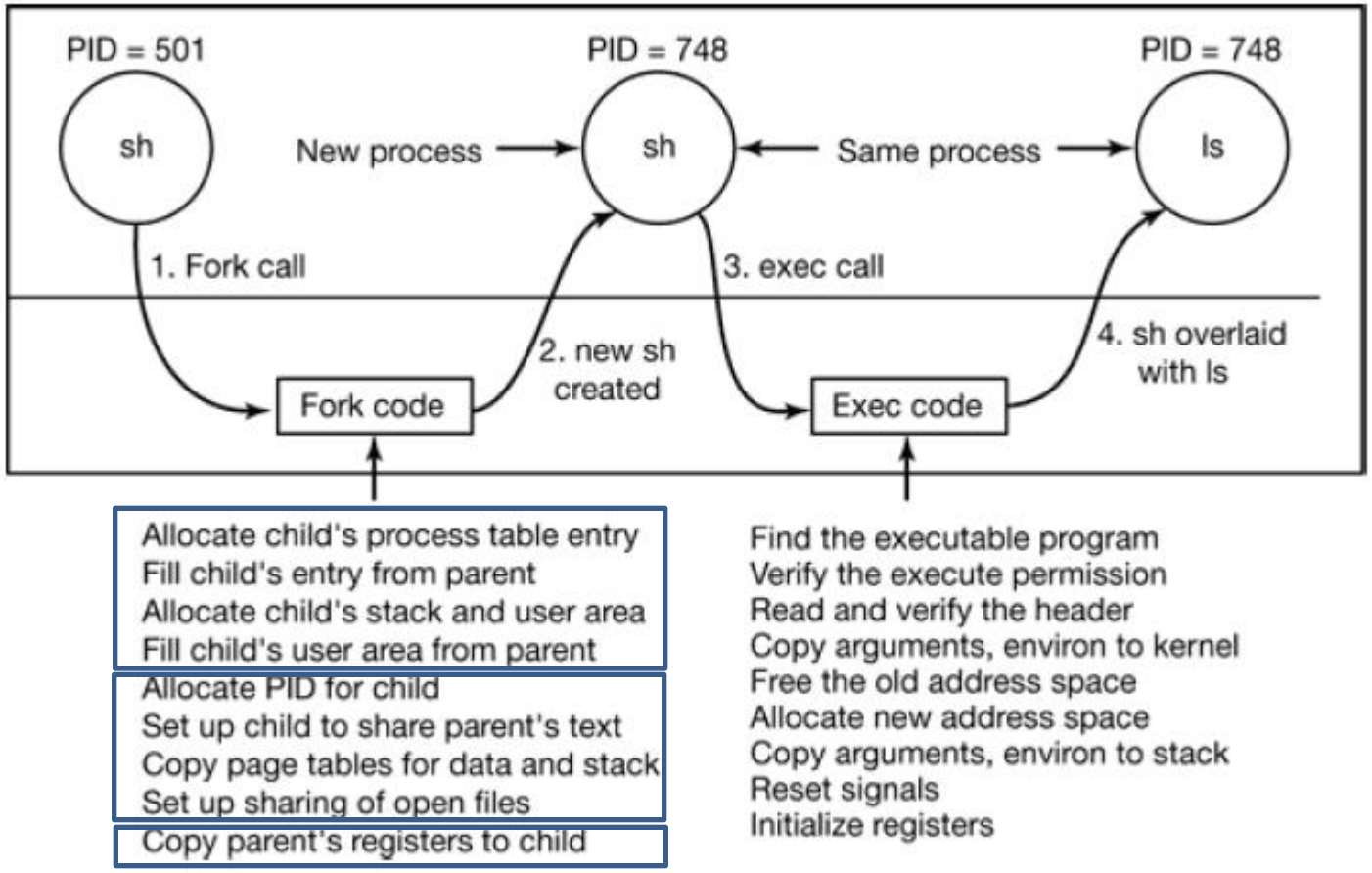
Proceso keitimas kita programa: `exec*()`



Proceso keitimas kita programa: `exec*()`

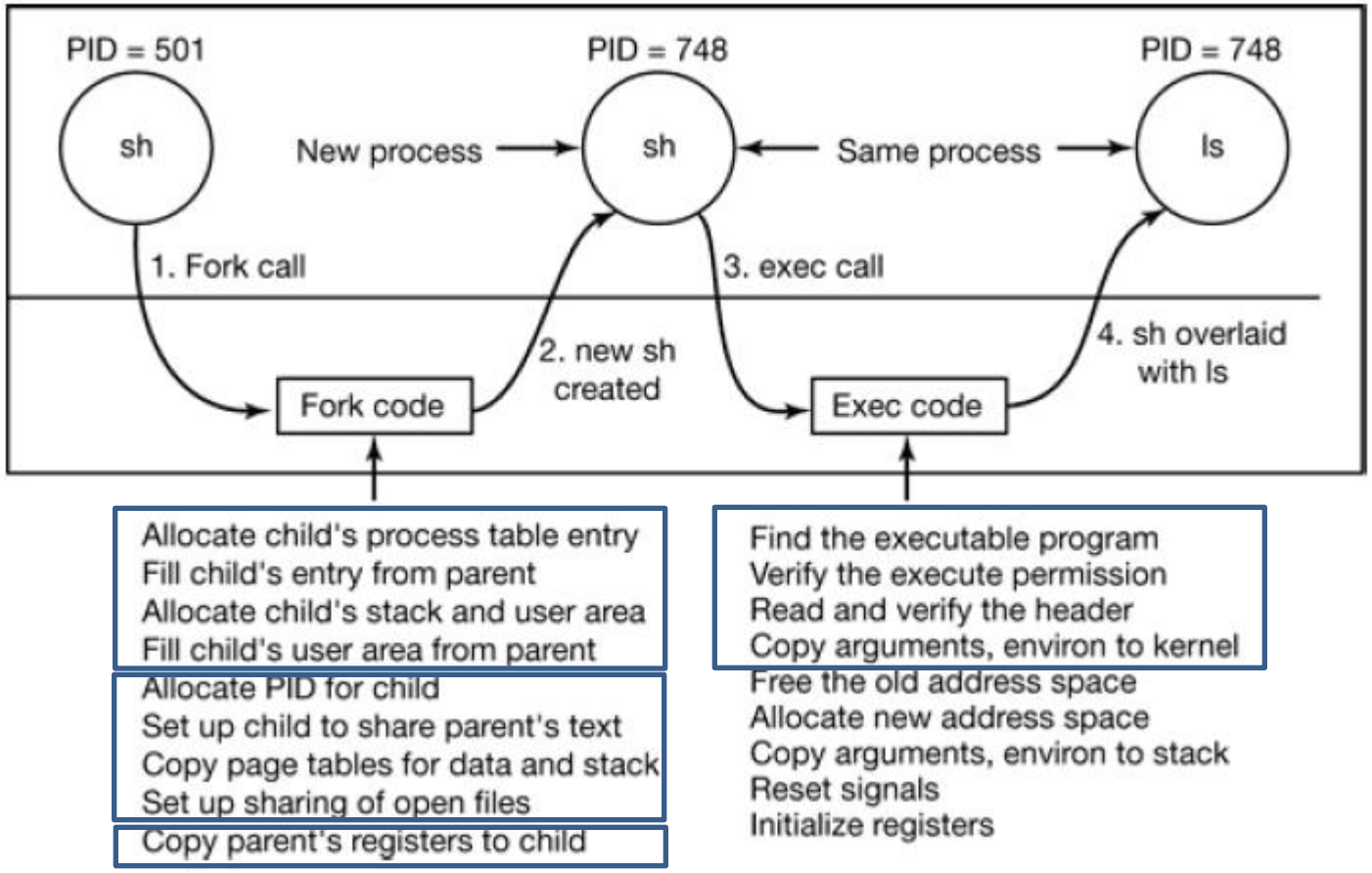


Proceso keitimas kita programa: `exec*()`



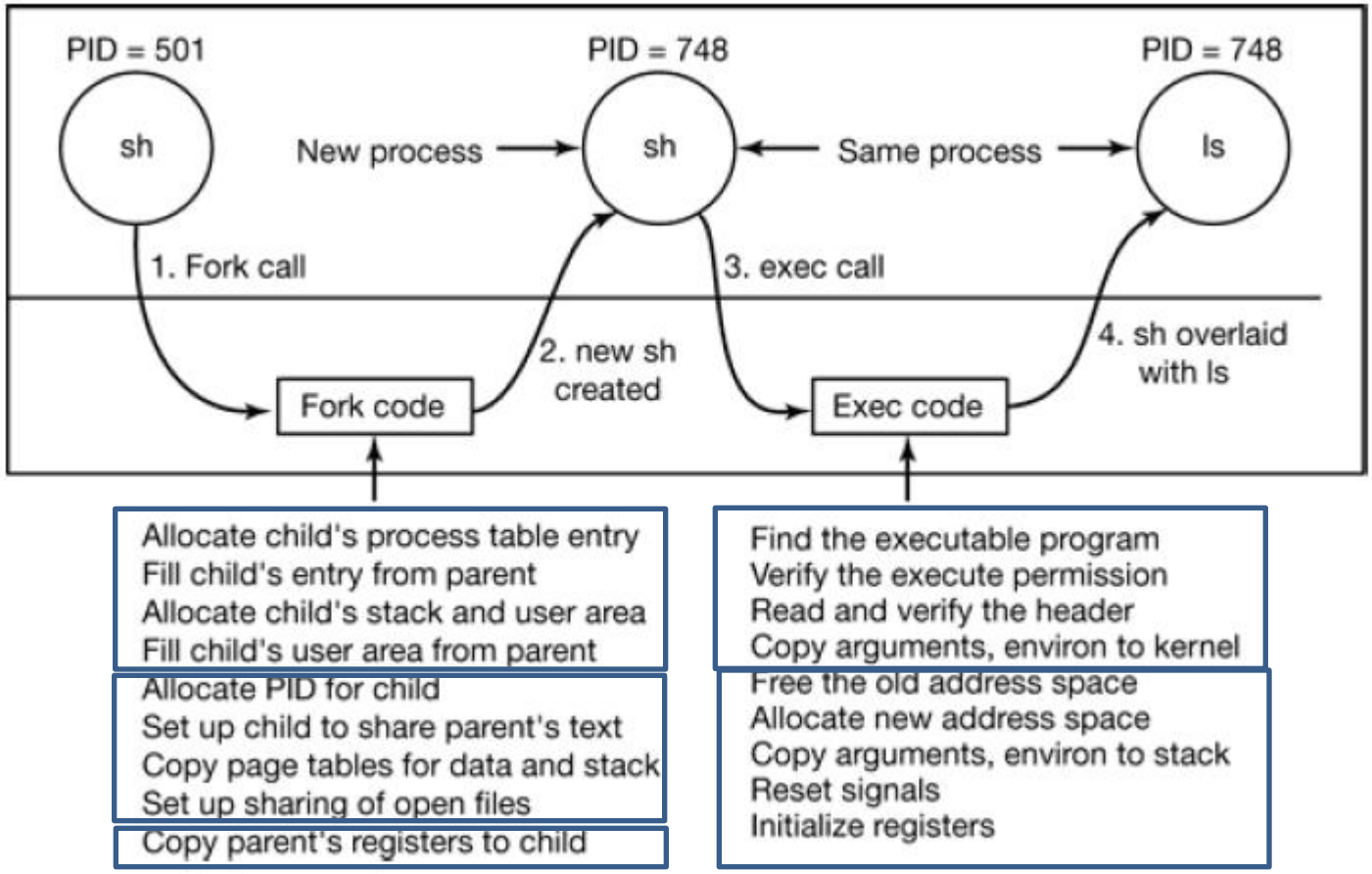
copy-on-write

Proceso keitimas kita programa: `exec*()`



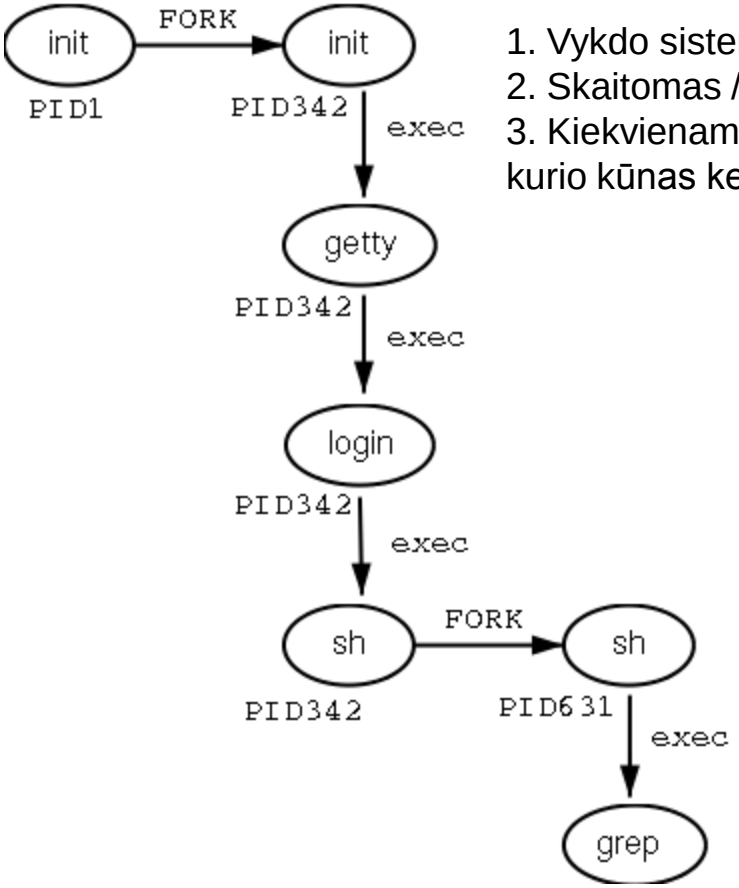
copy-on-write

Proceso keitimas kita programa: `exec*()`



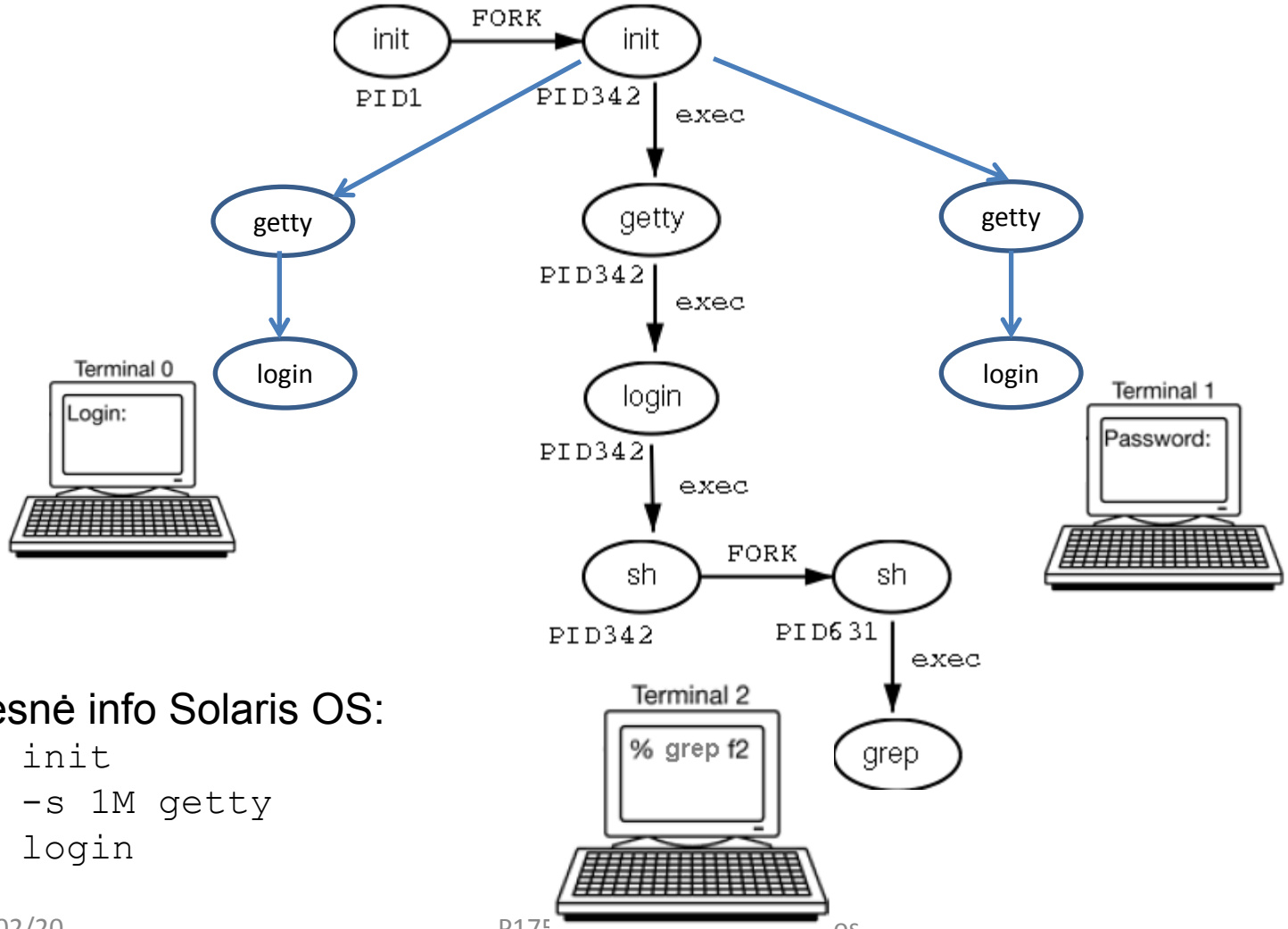
copy-on-write

`fork()` ir `exec()` mechanizmų iliustracija Linux OS



1. Vykdo sistemos inicializacijos skriptą /etc/rc
2. Skaitomas /etc/ttys failas
3. Kiekvienam term kuriamas naujas proc, kurio kūnas keičiamas programa getty

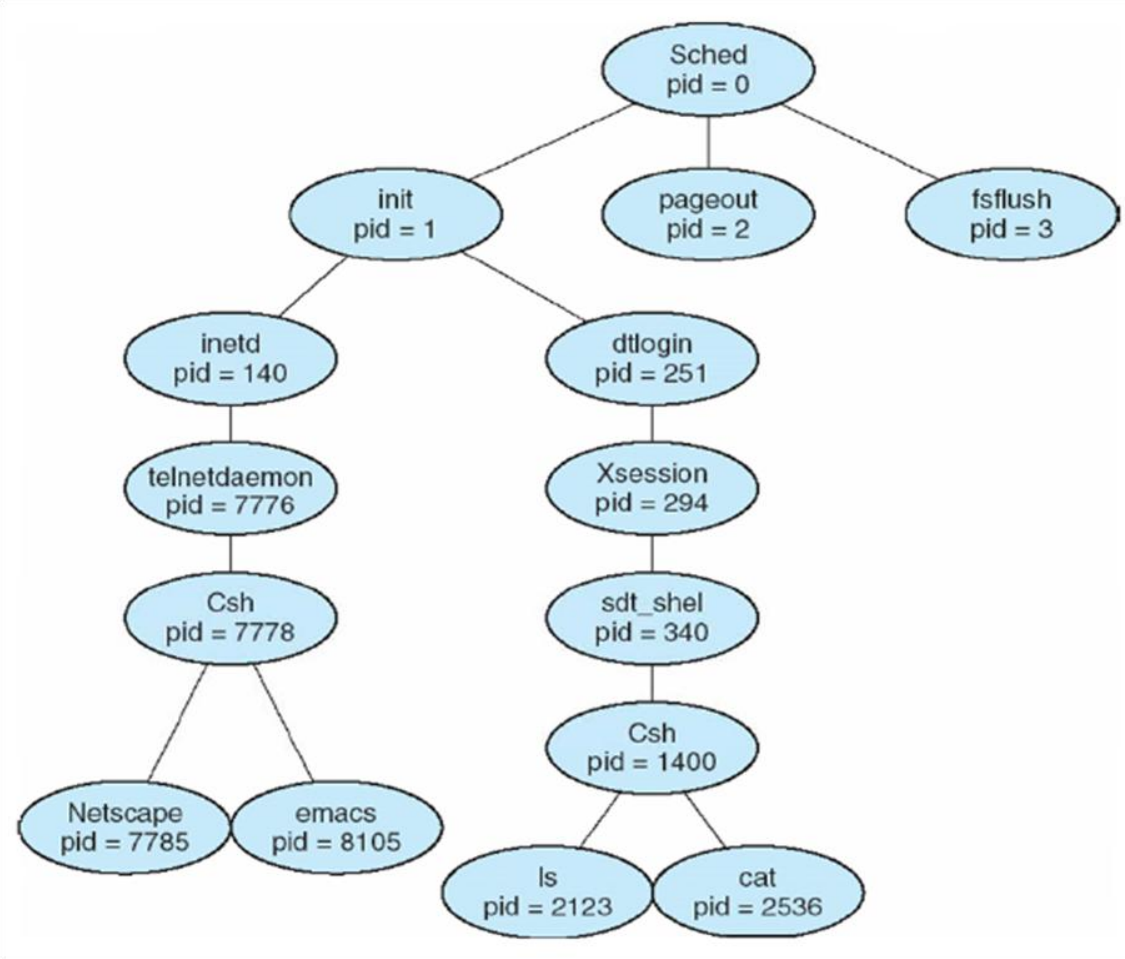
`fork()` ir `exec()` mechanizmų iliustracija Linux OS



Detalesnė info Solaris OS:

```
$ man init
$ man -s 1M getty
$ man login
```

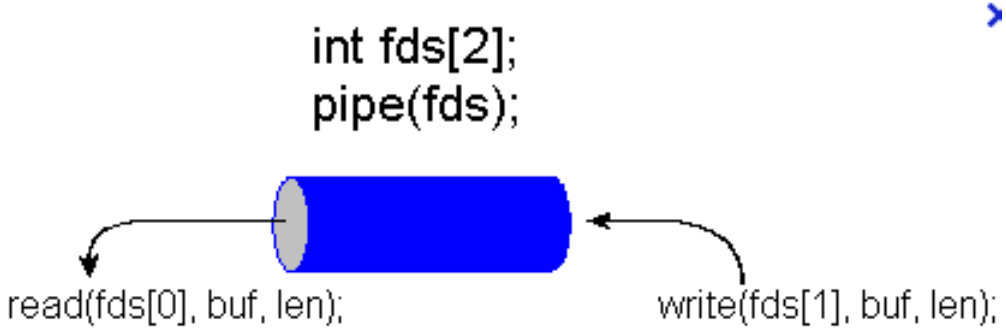
Procesų medis Solaris OS



Komunikacija tarp procesų

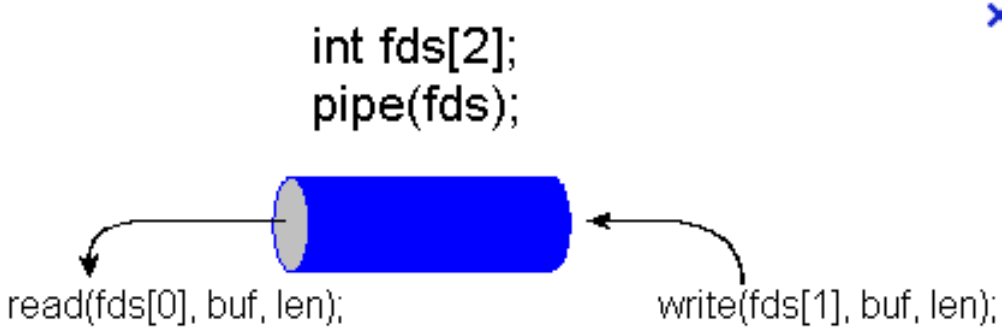
- naudojami šie mechanizmai:
  - Programiniai kanalai (angl. pipe),
  - Pranešimai,
  - Bendrai naudojama atmintis.
- Procesų tarpusavio veiksmų sinchronizacija gali būti iššaukiama naudojant šias priemones:
  - Signalus
  - Semaforus.

UNIX programiniai kanalai



- Informacijos mainai vienos kompiuterinės sistemos rėmuose
- Realizacija remiasi vartotojo-gamintojo modeliu
- Procesų tarpusavio išskirtinumą garantuoja operacinė sistema

UNIX programiniai kanalai



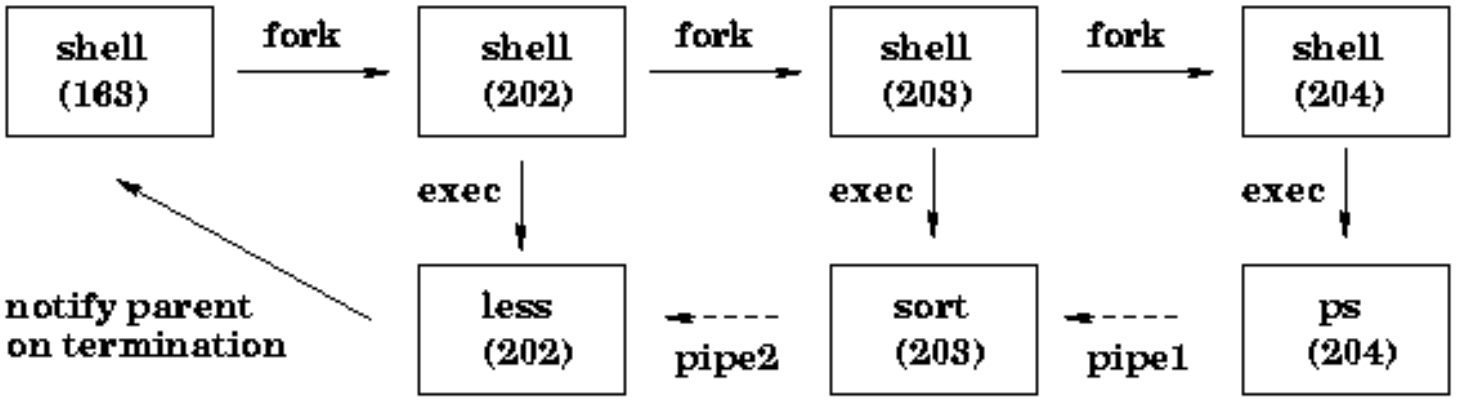
- Programinio kanalo sukūrimas: `pipe(fd)`
- Naudojimo pavyzdys:

```
int pfd[2];
pipe(pfd);
write(pfd[1], buf, len);
read(pfd[0], buf, len);
```

UNIX programiniai kanalai

Pavyzdys su shell komandomis

ps | sort | less



UNIX signalai

- Signalai:
  - yra panašūs savo prigimtim į techninės įrangos generuojamus pertraukimus.
  - signalams nėra taikomas prioritetas.
  - Kiekvienam signalui atitinka tam tikra skaitmeninė reikšmė.
  - Detalūs aprašymai – `signal()` f-jos aprašymuose `(man -s 3HEAD signal)`
- Signalai yra skiriami procesams.
  - Jiems pasirodžius yra įjungiamas atitinkamas bitas procesų lentelės įraše, susijusiame su gavėjo procesu.
  - Kai tik procesorius pradeda vykdyt procesą, kuris yra gavęs signalą, atitinkamas signalas yra apdorojamas.

UNIX signalai

- Paprastai signalą apdorojanti funkcija yra numatyta:
  - Ctrl-C klavišų paspaudimas priverčia sistemą pasiųsti vykdomam procesui INT tipo signalą (SIGINT). Pagal nutylėjimą šis signalas priverčia nutraukti procesą.
  - Ctrl-Z klavišų paspaudimas priverčia sistemą pasiųsti vykdomam procesui TSTP signalą (SIGTSTP). Pagal nutylėjimą šis signalas priverčia suspenduoti procesą vykdymą.
  - pasiunčiami iš komandinės eilutės, naudojant įvairias komandas – tai dažniausiai shell'o komandos (viena tokių - kill komanda, su kuria susipažinote LD14 lab. darbo metu).
- ...tačiau procesas gali turėti savo specialų signalo apdorojimą

UNIX signalai

- Signalo apdorojimas realizuojamas naudojant kreipinį `signal()`:

```
void (*signal(int sig, void (*func)(int)))(int);
```

- **sig** - Signalo ID (vardas)
- **func** - Rodyklė į f-ją, kuri iškviečiama signalo pasirodymo metu. Šioje f-joje aprašomi veiksmai susiję su nurodyto signalo apdorojimu.

UNIX signalai

INT signalo specialaus (ne pagal nutylėjimą) apdorojimo pavyzdys:

```c
/* Ingrida Lagzdinyte-Budnike KTK inglagz */
/* Failas: inglagz_signal00.c */

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>

void il_catch_INT(int);

void il_catch_INT(int snum) {
   printf("Caught signal %d, coming out...\n", snum);
   exit(1);
}

int main(int argc, char **argv) {
   printf( "(C) 2013 Ingrida Lagzdinyte-Budnike, %s\n", __FILE__ );
   signal(SIGINT, il_catch_INT);

   while(1)
   {
      printf("Going to sleep for a second...\n");
      sleep(1);
   }

   return(0);
}
```

```
$ ./inglagz_signal00
(C) 2013 Ingrida Lagzdinyte-Budnike, inglagz_signal00.c
Going to sleep for a second...
Going to sleep for a second...
Going to sleep for a second...
^CCaught signal 2, coming out...
```

UNIX signalų siuntimas

- Signalai gali būti persiunčiami kitam procesui naudojant sisteminį kreipinį `kill()`:

```
int kill(pid_t pid, int sig);
```

- **pid** - Proceso ID
- **sig** - Signalo ID (vardas)

Signalo SIGUSR1 siuntimo ir apdorojimo pavyzdys, kai tėvo procesas pasiunčia signalą vaiko procesui

- https://os.info.tm:3000/projects/p175b304/wiki/LD4_2#Signalai

Paskaitos pabaiga