Operativni sistemi 2

Neimenovane cevi (unnamed pipes)

- Objekti za komunikaciju između procesa.
- Kapacitet je ograničen, što je zavisno od implementacije.
  - Od jezgra 2.6.11 kapacitet je ograničen na 65536B.
- Procesi cev vide kroz dva deskriptora, jedan kao ulazni kraj cevi i drugi kao izlazni kraj cevi.
- Cev se pre korišćenja mora kreirati i po završenom korišćenju zatvoriti (svaki proces zatvara svoju kopiju deskriptora).
- Potrebne definicije:

```
#define INPUT 0
#define OUTPUT 1
int file_descriptors[2];
file_descriptors[INPUT] – deskriptor kraja
                          cevi za citanje
file_descriptors[OUTPUT] – deskriptor kraja
                           cevi za upis
```

- Kreiranje:

```
int pipe(int *file_descriptors);
```

- u nizu vraća par deskriptora za ulazni i izlazni kraj cevi;
- vraća 0 u slučaju uspeha i -1 u slučaju greške.

- Čitanje (blokirajuća operacija):

```
ssize_t read(int fd, void *buf, size_t count);
```

- prosleđuju se redom deskriptor izlaznog kraja cevi, bafer u koji će biti učitani podaci i maksimalan broj bajtova koji mogu biti smešteni u bafer;
- podaci se posmatraju kao prost tok bajtova (ne postoji podela na poruke);
- rezultat funkcije je broj pročitanih bajtova (može biti i manji od traženog u slučaju zatvaranja svih krajeva cevi za upis);
- u slučaju da je traženi broj bajtova 0, operacija nema efekta.

- Upis (neblokirajuća operacija):

```
ssize_t write(int fd, const void *buf, size_t count);
```

- upisuje count znakova iz buf u cev zadatu sa fd;

-blokira se samo ako u cevi nema dovoljno mesta da se smeste svi bajtovi (deblokira se čim se oslobodi dovoljno prostora);
- vraća broj upisanih bajtova ili -1 u slučaju greške.

- Zatvaranje cevi:

```
close(file_descriptors[INPUT]);
close(file_descriptors[OUTPUT]);
```

- svaki process mora zatvoriti svoju kopiju deskriptora;
- neki kraj cevi je zatvoren tek kada svi procesi zatvore svoju kopiju odgovarajućeg deskriptora.

Primer korišćenja:

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

//        child     parent
// pipe1  [0,0]     [0,1]
// pipe2  [1,1]     [1,0]
int main(){
     int f[2][2];
     int a = 0x4141;
     pipe(f[0]);
     pipe(f[1]);
     if (fork() == 0) {       //proces potomak
          close(f[0][1]);
          close(f[1][0]);     //ZASTO?
          write(f[1][1],&a,sizeof(int));
          read(f[0][0],&a, sizeof(int));
          printf("potomak primio: %d\n",a);
          close(f[1][1]);
          close(f[0][0]);
          exit(0);
     }else {                  //roditelj
          char c;
          close(f[1][1]);
          close(f[0][0]);
          a=1;
          write(f[0][1],&a,sizeof(int));
          read(f[1][0],&c, sizeof(char));
          printf("roditelj primio prvi bajt: %d\n",c);
          read(f[1][0],&c, sizeof(char));
          printf("roditelj primio drugi bajt: %d\n",c);
          close(f[0][1]);
          close(f[1][0]);
          exit(0);
     }
```

```
    return 0;
}
```

Imenovane cevi (FIFO)

- Objekti koji u sistemu fajlova imaju naziv i ponašaju se ekvivalentno neimenovanim cevima.
- Iako postoje u fajl sistemu, na uređaju ne postoje istoimeni fajlovi.
- Pristupa im se kao običnim fajlovima.
- Prilikom otvaranja fajla, ako drugi kraj nije otvoren, pozivajući process će biti blokiran.
- Po zatvaranju oba kraja, sadržaj koji se eventualno zatekne u cevi se gubi.

- Kreiranje:

```
int mkfifo( const char *pathname, mode_t mode );
```

- vraća 0 u slučaju uspeha i -1 u slučaju greške;
- postoji i istoimena komanda za korišćenje iz komandne linije:

```
mkfifo –m prava_pristupa naziv
```

- Primer koda koji čita iz cevi (pod pretpostavkom da je kreirana cev `a_pipe`):

```
FILE * in_file;
char buf[80];
in_file = fopen("a_pipe", "r");
if (in_file == NULL) {
     perror("Error in fopen");
     exit(1);
}
fread(buf, 1, 80, in_file);
printf("received from pipe: %s\n", buf);
fclose(in_file);
```

- Primer koda koji upisuje u cev (pod istom pretpostavkom):

```
FILE * out_file;
char buf[80];
out_file = fopen("a_pipe", "w");
if (out_file == NULL){
     perror("Error opening pipe");
     exit(1);
}
sprintf(buf,"this is test data for the named pipe
example\n");
```

```
fwrite(buf, 1, 80, out_file);
fclose(out_file);
```

Prosleđivanje poruka

- Mehanizam koji procesima omogućava komunikaciju indirektnim imenovanjem sa i bez poštovanja redosleda pristizanja poruka.

- Kreiranje i/ili otvaranje sandučeta:

```
int msgget(key_t key, int msgflg);
```

- `key` – broj koji predstavlja ključ koji identifikuje sanduče;
- `msgflg` – definiše operaciju i prava pristupa sandučetu (niz binarnih flegova):
  `IPC_CREAT` – kreira novo sanduče ako ne postoji ili vraća identifikator postojećeg sandučeta.
  `IPC_EXCL` – u kombinaciji sa predhodnim kreira sanduče samo ako ne postoji, ako postoji vraća -1 i u errno pamti grešku EEXIST.

`0666` – prava pristupa, samo u slučaju kreiranja (oktalne vrednosti).

```
 |||
 |||_  ostali
 ||_   članovi grupe
 |_    vlasnik
```

```
0400 Dozvola za čitanje za kreatora
0200 Dozvola za upis za kreatora
0040 Dozvola za čitanje za grupu kreatora
0020 Dozvola za upis za grupu kreatora
0004 Dozvola za čitanje za ostale
0002 Dozvola za upis za ostale
```

- izgled strukture koja sadrži podatke:

```
struct msgbuf {
     long mtype; // tip poruke – obavezno polje
     char mtext[1]; // sadržaj poruke
};
```

- Polje mtype mora biti strogo pozitivno i ne računa se u veličinu poruke.
- Sadržaj poruke može biti proizvoljne fiksne veličine.

Primer:

```
struct my_msgbuf {
     long mtype;
     char mtext[80];
};
```

- slanje poruke:

```
int msgsnd(int msqid, const void *msgp, size_t msgsz,
     int msgflg);
```

- `msqid` – identifikacija sandučeta;
- `msgp` – pokazivač na poruku;
- `msgsz` – veličina poruke (polje mtype se ne računa);
- `msgflg` – polje za preciznije određivanje operacije:
  - `IPC_NOWAIT` – čak i kada nema dovoljno mesta, msgsnd treba da se završi prijavljujući grešku (podrazumevano čeka da se pojavi slobodan prostor).

U slučaju greške, rezultat je -1. U suprotnom 0.

Primer:

```
msgsnd(id, (struct msgbuf *)&mq_test_buf,
sizeof("test message") + 1, 0);
     // +1 zbog \0 znaka na kraju stringa.
```

- prijem poruke:

```
ssize_t msgrcv(int msqid, void *msgp, size_t msgsz,
     long msgtyp, int msgflg);
```

Parametri imaju isto značenje kao pri slanju osim:
- `msgtyp` – polje koje preciznije određuje način izbora poruke:
  - 0 – primiti prvu poruku iz reda;
  - `msgtyp`>0 – primiti prvu poruku iz reda čiji je tip `msgtyp`, osim ako je zadat fleg `MSG_EXCEPT`, kada se traži prva poruka čiji tip nije `msgtyp`
  - `msgtyp`<0 – primiti prvu poruku sa najnižom vrednošću tipa koja je manja ili jednaka apsolutnoj vrednosti `msgtyp`;
- Pored flegova `IPC_NOWAIT` i `MSG_EXCEPT`, postoji i `MSG_NOERROR`.
- Ako se ovaj fleg ne navede, u slučaju da treba primiti poruku koja ne može stati u prihvatni bafer, prijem je neuspešan i prijavljuje se greška.
- U slučaju navođenja `MSG_NOERROR` operacija će biti uspešna, biće prihvaćen samo deo poruke koji može da stane u bafer, dok će ostatak biti izgubljen.
- U slučaju greške, rezultat je -1, u suprotnom, rezultat je veličina primljene poruke, ne računajući veličinu polja mtype.

Primer:

```
int status = msgrcv(id, (struct msgbuf*)&mq_test_buf, 80,
     msgtype, 0);
```

- uništavanje sandučeta:
  `msgctl(id, IPC_RMID, 0);`

`IPC_RMID` – označava operaciju brisanja

Deljena memorija

- Za potrebe deljenja memorije, potrebno je prvo kreirati (ili dohvatiti id kreiranog) segmenta memorije koji će biti deljen. Za to se koristi funkcija:

```
int shmget(key_t key, size_t size, int shmflg);
```

- `key` – ključ za dohvatanje odgovarajućeg segmenta mem.
- `size` – veličina segmenta;
- `shmflg` – flegovi za kontrolu operacije i definisanje prava pristupa;
  - Ako se za ključ navede IPC_PRIVATE, biće kreiran novi segment.
  - Isto važi i ako se u flegovima navede IPC_CREAT i zadati ključ ne postoji.
  - Ako se u flegovima zada IPC_EXCL zajedno sa IPC_CREAT, u slučaju postojanja segmenta sa zadatim ključem, biće vraćena greška.
  - Najnižih 9 bita označavaju prava pristupa deljenoj memoriji
    - **O_RDONLY**, **O_WRONLY**, **O_RDWR**.
  - Kada se kreira i/ili dohvati id segmenta deljene momorije, potrebno je deo virtuelnog adresnog prostora preslikati u dati segment deljene memorije:

```
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

- `shmid` – id dobijen prethodnom operacijom;
- `shmaddr` – željena virtuelna adresa početka segmenta deljene memorije;
- `shmflg` – flegovi za kontrolu operacije;

- `shmaddr` treba da bude poravnata na granicu stranica ili će biti prijavljena greška.
- Ako se navede fleg SHM_RND u prethodnom slučaju neće biti prijavljena greška već će doći do zaokruživanja adrese.
- Ako se kao željena adresa zada NULL, biće pronađena odgovarajuća slobodna adresa u adresnom prostoru pozivaoca.
- U flegovima je moguće navesti i SHM_RDONLY ukoliko procesu segment deljene memorije treba samo za potrebe čitanja.
- Funkcija vraća konačnu adresu na koju je deljena memorija zakačena ili -1 u slučaju greške.
- Vraćena adresa se koristi za dalji pristup deljenoj memoriji.

Po završenom korišćenju deljene memorije, potrebno je otkačiti iz adresnog prostora:

```
int shmdt(const void *shmaddr);
```

- Kao jedini argument zadaje se adresa na koju je segment deljene memorije zakačen.
- Poslednji ko završi sa korišćenjem segmenta, pod uslovom da taj segment deljene memorije više neće biti potreban, treba da oslobodi zauzetu memoriju:

```
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

- `shmid` – id segmenta deljene memorije na koji se poziv odnosi;
- `cmd` – komanda koja se zadaje – IPC_RMID;
- `buf` – pokazivač na odgovarajuću strukturu, u slučaju brisanja 0.

Primer:

```
#include <stdio.h>
#include <sys/shm.h>
#include <sys/stat.h>
int main ()
{
  int segment_id;
  char* shared_memory;
  struct shmid_ds shmbuffer;
  int segment_size;
  const int shared_segment_size = 0x6400;
  segment_id = shmget (IPC_PRIVATE, shared_segment_size,
     IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
  shared_memory = (char*) shmat (segment_id, 0, 0);
  printf ("VA deljene mem: %p\n", shared_memory);
/* Dohvatanje velicine deljenog segmenta */
  shmctl (segment_id, IPC_STAT, &shmbuffer);
  segment_size = shmbuffer.shm_segsz;
  printf ("Velicina deljene mem: %d\n", segment_size);
/* Upis u deljenu memoriju. */
  sprintf (shared_memory, "Hello world!");
/* Po koriscenju raskinuti vezu sa deljenom memorijom */
  shmdt (shared_memory);
/* Zakaciti isti segment deljene memorije, ali na drugu
adresu. */
  shared_memory = (char*) shmat (segment_id, (void*)
     0x5000000, 0);
  printf ("Nova VA deljene mem: %p\n", shared_memory);
/* Ispisati deo sadržaja deljene memorije */
  printf ("%s\n", shared_memory);
/* Otkaciti deljenu memoriju . */
```

```
  shmdt (shared_memory);
/* Oslobadjanje zauzete memorije.*/
  shmctl (segment_id, IPC_RMID, 0);
  return 0;
}
```

Memorijsko mapiranje fajla

- Sistemske usluge za mapiranje proizvoljnog fajla (ili uređaja) u virtuelni adresni prostor pozivajućeg procesa je:

```
#include <sys/mman.h>

void *mmap(void *start, size_t length, int prot, int flags,
int fd, off_t offset);
```

- `start` – adresa u virtuelnom adresnom prostoru pozivajućeg procesa na koju treba preslikati fajl;
- `length` – broj bajtova koje treba preslikati, zaokružuje se na ceo broj stranica;
- `prot` – definišu se prava pristupa mapiranom prostoru; ne sme postojati konflikt sa modom u kojem je fajl otvoren. Dozvoljeni flegovi:
  - PROT_EXEC – pravo izvršavanja;
  - PROT_READ – pravo čitanja;
  - PROT_WRITE – pravo upisa;
- `flags` – određuju vrstu objekta koji se mapira:
  - MAP_PRIVATE – mapirani fajl je privatan za proces (copy-on-write) nema uticaja na fajl sve dok se u ovaj region ne izvrši upis,
  - MAP_SHARED – mapirani fajl je deljen; upis ekvivalentan upisu u fajl;

Napomena: tačno jedan od prethodna dva flega se mora navesti;

- `fd` – deskriptor fajla koji se preslikava;
- `offset` – pomeraj od kojeg počinje mapiranje.

- Oslobađanje:

```
int munmap(void *start, size_t length);
```

  - Prosleđuje se adresa na koju je mapiran fajl, kao i dužina.

Primer:

```
#include<sys/mman.h>
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include <unistd.h>

int main() {
```

```
  char *mem_pointer;
  int f;
  if ((f=open("test.txt", O_RDWR)) < 0) {
     printf("Error opening test.txt\n");
     exit(1);
  }

  char buff[80];
  read(f,buff,80);
  printf("file content: %s\n", buff);

  mem_pointer = (char *)mmap(0, 8192, PROT_READ |
     PROT_WRITE, MAP_SHARED, f,0);
  printf("first two bytes: %d %d\n", mem_pointer[0],
mem_pointer[1]);
  mem_pointer[0] = '0'+2; // write into file
  mem_pointer[1] = '0'+3; // write into file

  munmap(mem_pointer, 8192);
  close(f);

  return 0;
}
```

Semafori

```
#include<sys/sem.h>
```

- kreiranje semafora:

```
sem_id = semget((key_t)123, 1, 0666 | IPC_CREAT);
```

- Prvi parameter je ključ (način imenovanja semafora).
- Drugi parametar je broj semafora u setu koji će biti kreirani.Ako treba da se koriste semafor(i) kreiran(i) od strane drugog procesa, navodi se 0.
- Treći paremetar su flegovi. Imaju isto značenje kao kod kreiranja reda za prosleđivanja poruka.

- operacije na semaforu:

```
int semop(int semid, struct sembuf *sops, size_t nsops);
```

- Pri pozivanju semop prosleđuje se:
  - id skupa semafora,
  - niz struktura koje predstavljaju niz traženih operacija
  - broj elemenata u prethodnom nizu.

- Sve operacije se izvršavaju atomično, i to tek u trenutku kada je moguće izvršiti sve zadate operacije.
- Svaka operacija se zadaje strukturom sa sledećim poljima:

  `sem_num` – indeks semafora u setu;
  `sem_op` – broj koji se dodaje na vrednost semafora.
  `sem_flg` – flegovi IPC_NOWAIT i SEM_UNDO

Vrednost `sem_op` zadaje jednu od tri moguće operacije:

- sem_op>0 – signal_n, sem_op se dodaje na vrednost zadatog semafora i uvek može da se izvrši;
- sem_op=0 – wait_for_zero, nema efekta na vrednost semafora; ako semafor nije imao vrednost 0, operacija u tom trenutku ne može da se izvrši, već se pozivajući proces blokira; kada vrednost semafora postane 0, ovakav proces se budi i operacija uspeva;
- sem_op<0 – wait_n, ako je vrednost semafora manja od |sem_op|, pozivajući proces se suspenduje do trenutka kada ovaj uslov postane netačan; kada se steknu uslovi da se izvršavanje nastavi, od vrednosti semafora se oduzima |sem_op|.
- IPC_NOWAIT - ukoliko makar jedna operacija ne može odmah da se izvrši, čitav poziv semop je neuspešan (vraća -1).
- SEM_UNDO - efekti operacije se poništavaju kada se proces završi.

Impelementacija standardnih `wait i signal` operacija:

```
static int sem_wait(int sem_id)
{
     struct sembuf sem_b;
     sem_b.sem_num = 0; //index u setu
     sem_b.sem_op = -1; //oduzima 1 od sem
     sem_b.sem_flg = SEM_UNDO;
     if (semop(sem_id, &sem_b, 1) == -1) {
       fprintf(stderr, "neuspesan wait\n");
       return 0;
     }
     return 1;
}

static int sem_signal(int sem_id)
{
     struct sembuf sem_b;
     sem_b.sem_num = 0; //index u setu
     sem_b.sem_op = 1;
     sem_b.sem_flg = SEM_UNDO;
     if (semop(sem_id, &sem_b, 1) == -1) {
       fprintf(stderr, "neuspesan signal\n");
       return 0;
     }
```

```
        return 1;
    }
```

- brisanje semafora:

```
union semun arg; //ostali argumenti
//drugi argument za semctl je redni broj semafora
//u setu
if (semctl(sem_id, 0, IPC_RMID, arg) == -1)
 fprintf(stderr,"Neuspesno brisanje semafora\n");
```

- postavljanje vrednosti semafora:

```
static int set_semvalue(int sem_id, int val)
{
  union semun arg;
  arg.val = val;    // vrednost
  if (semctl(sem_id, 0, SETVAL, arg) == -1)
    return(0);
  return(1);
}
```

Komande za upravljanje objektima komunikacije iz komandne linije

- Komanda za prikaz trenutno kreiranih objekata.

```
% ipcs -m
------ Shared Memory Segments --------
key         shmid  owner perms bytes nattch status
0x00000000 1627649 user  640   25600 0
```

- Opcija –m u ovom slučaju govori da treba da se izlista samo deljena memorija.
- Bez opcija se dobijaju svi kanali za komunikaciju.

- Komanda za brisanje segmenta deljene memorije sa ID 1627649:

```
% ipcrm -m 1627649
Opcije:
-m id_shm
-q id_msq
-s id_sem
-M key_shm
-Q key_msq
-S key_sem
```