Fakultet elektrotehnike i računarstva Sveučilišta u Zagrebu
Zavod za elektroničke sustave i obradbu informacija
Laboratorij za sustave i signale

Programska podrška mjernih i procesnih sustava

vježba br. 3: **KOMUNIKACIJA MEĐU PROCESIMA**

UVOD

Komunikaciju među procesima moguće je ostvariti na više načina. Neke od mogućnosti su:

- **cjevovod (*pipe*)**
- **imenovani cjevovod (*named pipe*) - FIFO**
- **signali**
- poruke
- semafori
- dijeljena memorija

U ovoj će se vježbi raditi sa signalima, cjevovodima i imenovanim cjevovodima. Većina će primjera biti izrađena u C-u korištenjem nekih osnovnih poziva sustava (*system calls*).

ZADACI ZA VJEŽBU

1. SIGNALI

Signali su kratke poruke kojima je omogućeno slanje zahtjeva za prekid određenom procesu. Signali se mogu razmjenjivati između dvaju procesa (jedan šalje signal drugom) ili sama jezgra operacijskog sustava (*kernel*) može slati signal određenom procesu. Popis signala može se naći u `/usr/include/bits/signum.h` *header* datoteci sustava. Proučite njezin sadržaj programom `less`. Vidjet ćete među ostalim i nešto slično ovom popisu:

```
#define SIGHUP          1
#define SIGINT          2
#define SIGQUIT         3
#define SIGILL          4
#define SIGTRAP         5
#define SIGABRT         6
#define SIGIOT          6
#define SIGBUS          7
#define SIGFPE          8
#define SIGKILL         9
#define SIGUSR1        10
#define SIGSEGV        11
#define SIGUSR2        12
#define SIGPIPE        13
#define SIGALRM        14
#define SIGTERM        15
```

```
#define SIGSTKFLT       16
#define SIGCHLD         17
#define SIGCONT         18
#define SIGSTOP         19
#define SIGTSTP         20
#define SIGTTIN         21
#define SIGTTOU         22
#define SIGURG          23
#define SIGXCPU         24
#define SIGXFSZ         25
#define SIGVTALRM       26
#define SIGPROF         27
#define SIGWINCH        28
#define SIGIO           29
#define SIGPOLL         SIGIO
```

Vidi se da svaki signal ima svoje ime i broj koji ga predstavlja. Funkcije većine navedenih signala su unaprijed dodijeljene. Većina signala se može i redefinirati ili ignorirati osim signala KILL (broj 9) koji uzrokuje trenutni završetak procesa kojemu je namijenjen. Kažemo da smo proces 'ubili' signalom KILL. Signalima STOP i CONT može se zaustaviti te nastaviti izvođenje procesa. No, ima i signala kojima funkciju određuje korisnik (programer). To su signali USR1 i USR2. Ponekad se može dogoditi da je isti signal predstavljen drugim brojem na drugom UNIX sustavu. Zbog toga je preporučljivo koristiti imena signala, a ne njihove brojeve. Isto tako, nisu baš svi signali definirani na svim UNIX sustavima.
Slanje signala vrši se pozivom sustava `kill`:

```
int kill(int PID, int SIGNAL)
```

gdje je PID cjelobrojna varijabla koja predstavlja *Process ID* procesa kojem se šalje signal, a SIGNAL je cjelobrojna varijabla koja predstavlja broj signala koji se šalje. No, postoji i u ljusku ugrađena naredba istog naziva (koja se i zasniva na istoimenom pozivu sustava): `kill`. Ona služi za slanje signala određenom procesu iz komandne linije.
Naredbom `kill –l` dobiva se popis svih signala. Usporedite ga sa sadržajem `signal.h` datoteke.

Pokušajmo ubiti neki proces:

- sa drugog virutalnog terminala naredbom `ps l` doznati PID procesa ljuske (vjerojatno će to biti `bash`) od prvog virtualnog terminala (ima manji PID i TTY broj)
- izdati naredbu za ubijanje procesa ljuske u prvoj virtualnoj konzoli:
  `$ kill –kill 380` (broj 380 zamijeniti sa pravim PID-om)

Uočite da je ljuska na prvoj konzoli ubijena. Pošto znamo da je broj signala KILL jednak 9, mogli smo istu naredbu zadati i ovako:

```
$ kill –9 380
```

Ukoliko se kao PID pošalje broj –1 (`kill –9 –1`), svi će procesi kojima je vlasnik korisnik koji je izdao ovu naredbu biti ubijeni. Razmislite, što će se dogoditi ako tu naredbu izvršite kao suprekorisnik (*root*).

Isprobati korištenje signala STOP i CONT:

- pokrenuti program `tockice` iz prosle vjezbe
- logirati se u drugoj konzoli
- naredbom `ps l` doznati PID procesa `tockice`
- slanjem signala STOP (korištenjem naredbe `kill`) zaustaviti program `tockice`

- slanjem signala CONT nastaviti izvođenje programa `tockice`
- slanjem signala KILL ubiti proces `tockice`

U svjetlu gornjih saznanja možemo objasniti što je zapravo Ctrl+C zadana od strane korisnika koji želi prekinuti izvođenje neredbe koju je izdao. To je zapravo signal INT koji korisnik šalje procesu koji upravlja terminalom.

2. REAKCIJA NA SIGNALE

U nastavku je dan kod C programa koji u petlji izvršava pisanje iste poruke na standardni izlaz sve dok ne primi signal INT. Kada primi signal INT, skače u prekidnu rutinu `prekid()` u kojoj ispisuje poruku o primitku signala. Definiranje kako će na koji signal proces reagirati omogućuje poziv `signal()`. Ako želimo da proces stane i čeka signal onda se to ostvaruje pozivom `pause()`. Nakon povratka iz prekidne rutine, proces se zaustavlja sistemskim pozivom `exit()`.

moj_signal.c

```
#include <signal.h>

void prekid()
{
        printf("Primio sam signal SIGINT i usao sam u funkciju prekid!!!\n");
        exit(1);
}

main()
{
        signal (SIGINT, prekid);        //ukoliko ovaj proces primi signal
                                        //SIGINT, izvrsit ce se funkcija
                                        //prekid
        while(1)
        {
                int i;
                printf("Vrtim se u petlji i cekam signal SIGINT\n");
                for(i=0;i<10000000;i++);
        }
        exit(0);
}
```

Zadatci:

- Program iskompajlirati i isprobajte kako reagira na Ctrl+C tipku.
- Ako je drugi parametar funkcije `signal`, `SIG_DFL` obavit će se defaultna rutina. Ako je pak taj parametar `SIG_IGN` onda se taj signal ignorira. Probajte oba dva slučaja.

Program `moj_signal.c` preraditi na slijedeći način:

- korištenjem poziva `fork()` stvoriti dva procesa: roditelj i dijete
- proces dijete čeka, pomoću `pause()`, signal USR1 (SIGUSR1)
- kada proces dijete primi signal SIGUSR1, izvršava se prekidna rutina u kojoj se na standardni izlaz ispisuje poruka: «Primio sam signal SIGUSR1»
- proces roditelj vrti se u `for` petlji 15 puta
- svaki puta u petlji čeka dvije sekunde (`sleep(2)`)...
- ...i nakon toga šalje procesu dijete signal USR1 (`kill(pid, SIGUSR1)` gdje `pid` predstavlja *Process ID* djeteta, a to je upravo ono što vrati funkcija `fork()` u procesu roditelj)

3. CJEVOVOD

Cjevovod ili *pipe* prestavlja komunikacijski kanal između dvaju procesa. Njime se između ta dva procesa mogu izmjenjivati podaci. Cjevovod se koristi kao jednosmjeran kanal dakle na jedan kraj jedan proces šalje a na drugom kraju jedan proces prima podatke. Cjevovod se stvara pozivom `pipe()`.

```
int pipe(int fildes[2])
```

Poziv sustava `pipe` vraća dva *file descriptora,* `fildes[0]` i `fildes[1]`. Za pisanje u cjevovod iz procesa koristi se poziv `write()` u *file descriptor* `fildes[0]`. Za čitanje iz cjevovoda koristi se poziv `read()` uz *file descriptor* `fildes[1]`. Zatvaranje cjevovoda vrši se pozivom `close()`.

Cjevovod se ne može kreirati nakon što su procesi već stvoreni, zato što proces koji stvara cjevovod ne može prenijeti *file descriptor* drugom procesu. Oni se prenose samo kod kreiranja novog procesa. Zato se prvo stvori cjevovod, a zatim kreira dijete koje će naslijediti *file descriptor* cjevovoda.

Da bismo dva procesa povezali cjevovodom potrebno je napraviti slijedeće:

- napraviti cjevovod pozivom `pipe()`
- pozivom `fork()` napraviti dva procesa
- u procesu dijete zatvoriti pisanje u cjevovod i aktivirati primanje poruke
- u procesu roditelj zatvoriti čitanje iz cjevovoda i aktivirati slanje poruke

Ovime smo postigli da proces roditelj šalje podatke kroz cjevovod procesu dijete.

cjevovod.c

```
#define duljina_poruke  21
char *poruka = "PORUKA ZA DIJETE!!!";

main()
{
        int pid;
        char primljena_poruka[100];
        int file_descriptor[2];

        pipe(file_descriptor);
        pid = fork();
        if (pid > 0)
        //ovo je proces roditelj
        {
                close(file_descriptor[0]);                                  //zatvori citanje
                printf("Ja sam roditelj i saljem poruku djetetu.\n");
                write(file_descriptor[1], poruka, duljina_poruke);          //pisi u cjevovod
                wait(0);                                                    //cekaj da zavrsi
                                                                            //proces dijete
        }
        else
        //ovo je proces dijete
        {
                close(file_descriptor[1]);                                  //zatvori pisanje
                printf("\nJa sam dijete i cekam  poruku od roditelja.\n");
                read(file_descriptor[0], primljena_poruka, duljina_poruke);     //citaj iz
                                                                            //cjevovoda
                printf("Evo poruke koju sam primio: %s\n", primljena_poruka);
                exit(0);                                                    //zavrsio proces
                                                                            //dijete
        }

}
```

Proučite primjer, iskompajlirajte i pokrenite.

Podsjetite se operatora | (*pipe*) unutar ljuske (npr. `du | sort -rn | less`). Njegova je funkcionalnost također omogućena pozivom sustava `pipe()`. Ljuska prvo stvori dva cjevovoda a zatim forka tri procesa koji ih koriste. `stdout` i `stdin` file *descriptori* tih procesa su spojeni na ulaz/izlaz tih cjevovoda.

Potrebno proširiti postojeći program tako da:

- omogući dvostranu komunikaciju (uz pomoć dva cjevovoda) između dva procesa. Želimo da kad roditelj pita dijete "Kak si?" ono odgovori sa "Pa tak." a ako pita "Kako je bilo?" odgovara sa "Ludilo brale.". Sva pitanja i odgovore ispisuje roditelj.

4. IMENOVANI CJEVOVOD - FIFO

Običan cjevovod često nije pogodan za komunikaciju među procesima. Naime, da bi radio potrebno je da procesi budu u 'rodbinskim' odnosima. Imenovani cjevovod (*named pipe*) - često može biti puno pogodniji. On postoji kao posebna datoteka unutar datotečnog sustava. Prenosi podatke na principu prvi nutra – prvi van pa se često naziva i FIFO (*First In First Out*). Za razliku od običnog cjevovoda on omogućuje povezivanje procesa različitih 'predaka'.
Imenovani cjevovod stvara se naredbom `mknod`:

```
$ mknod moj_fifo p
```

Naredbom `ls -la` provjerite što ste dobili. Dobit ćete nešto slično ovome:

```
prw-------   1 student  users           0 Oct 22 20:26 moj_fifo
```

Uočite da na početku niza od devet znakova koja definiraju ovlasti nad datotekom stoji znak `p`. To je oznaka da se radi o imenovanom cjevovodu.
Evo jednog tipičnog primjerera rada imenovanog cjevovoda. U pozadini (znak `&` na kraju naredbe) pokrenimo naredbu `cat` koja čita sa imenovanog cjevovoda `moj_fifo` i ispisuje na standardni izlaz (ekran).

```
$ cat < moj_fifo &
```

Nakon toga pokrenimo još jednu naredbu `cat` kojom pišemo u imenovani cjevovod `moj_fifo`.

```
$ cat > moj_fifo
pisem a dolazi do jeke
pisem a dolazi do jeke
```

Rezultat je dvostruki ispis. Dakle, sve ono što smo napisali ušlo je u FIFO i ponovo se ispisalo na monitoru.
Poziv sustava koji omogućuje stvaranje imenovanog cjevovoda je `mknod()`.