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. 4: **KOMUNIKACIJA MEĐU PROCESIMA 2**

UVOD

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

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

U ovoj će se vježbi raditi sa semaforima i dijeljenom memorijom. Primjeri su izrađeni u C-u korištenjem osnovnih poziva sustava (*system calls*).

ZADACI ZA VJEŽBU

1. DIJELJENA MEMORIJA

Novi proces koji je nastao sistemskim pozivom `fork()` dobiva svoje lokalne kopije svih podataka. To vrijedi i za globalne varijable tako da pomoću njih nije moguće prenositi podatke između procesa. Zato postoji poseban mehanizam kojim se kreira dijeljena memorija koja je zajednička za više procesa te je na taj način moguće razmijeniti podatke među njima.
Jednom kada je dijeljena memorija kreirana, pristup njoj (čitanje i pisanje) je vrlo brzo. Njoj procesi pristupaju direktno, kao bilo kojem drugom memorijskom prostoru, bez potrebe za sistemskim pozivima i prenošenjem podataka u jezgru operacijskog sustava (kao npr. za cjevovod).
Po dijeljenoj memoriji se može pisati i čitati međutim ona ne osigurava sinkronizaciju među procesima. Tako je moguće da dođe do tzv. *race conditiona* kada jedan proces prebriše zapis od drugog procesa ili pročita zapis prije nego drugi završio sa pisanjem i sl.

Sistemski pozivi za rad sa dijeljenim segmentom memorije:

Prvo je potrebno da jedan proces alocira dijeljenu memoriju. To se ostvaruje sistemskim pozivom:

```
int shmget(key_t key, int size, int flags);
```

`key` predstavlja jedinstveni broj dijeljene memorije i ako želimo da se kreira novi segment stavit ćemo IPC_PRIVATE. `size` je željena veličina dijeljenog segmenta u byte-ovima. Stvarna veličina će zapravo biti višekratnik broja veličine stranice u memoriji (tzv. *page size*) koja je obično velika 4k. Sa `flags` se podešavaju prava pristupa tako ako želimo pisati i čitati, stavit ćemo zastavice S_IRUSR (user read), S_IWUSR (user write) pomoću OR operacije (`S_IRUSR | S_IWUSR`). Rezultat je segment ID koji se koristi u drugim pozivima za rad sa tim dijeljenim segmentom memorije.

Svaki od procesa koji žele koristiti dijeljeni segment moraju ga priključiti (*attach*) svom adresnom prostoru. Svaki proces sadrži tablicu mapiranja između svojih adresa i virtualnih memorijskih adresa u kojima se nalaze stvarni podaci. Zato svaki proces mora u tu tablicu pridodati mapiranje na virtualne stranice u kojima je dijeljeni segment, pomoću:

```
char *shmat(int segment_id, char *addr, int flags);
```

`flags` je obično NULL, pod `addr` mi možemo sami specificirati na koju adresu da priključi dijeljenu memoriju ili možemo prepustiti OSu da to napravi ako se stavi NULL. Rezultat je pokazivač na dijeljeni segment.

Otpuštanje dijeljene memorije iz svog adresnog prostora se obavlja sa:

```
int shmdt(char *addr);
```

gdje je `addr` adresa koju je vratila funkcija `shmat`.

Na kraju jedan proces mora i dealocirati dijeljeni segment ali tek kad svi procesi otpuste svoje priključke. Dealociranje se izvodi sa:

```
shmctl(int segment_id, IPC_RMID, NULL);
```

Naredbom `shmctl` (SHared Memory ConTroL) se mogu dohvaćati i neki parametri kao npr. konačna veličina dijeljene memorije i sl. Ukoliko se ovaj poziv izostavi, resurs dijeljene memorije će ostat zauzet.

Koji resursi su u upotrebi se može saznati iz ljuske naredbom `ipcs`. Ako treba obrisati određeni zaostali segment to se radi sa: `ipcrm shm shmid`, gdje umjesto `shmid` ide konkretan broj koji javi `ipcs`.

Primjer programa koji kreira dijeljenu memoriju te dva procesa u kojem jedan piše a drugi čita je dan ovdje:

```
#include <stdio.h>
#include <sys/shm.h>
#include <sys/stat.h>

int segment_id;

int main() {

      int *shared_memory;
      int segment_size, real_segment_size;
      struct shmid_ds shmbuffer;
      int pid,broj;
```

```
        //Alociranje zajednicke memorije
        segment_id = shmget(IPC_PRIVATE, sizeof(int), S_IRUSR | S_IWUSR);

        shmctl(segment_id, IPC_STAT, &shmbuffer);
        real_segment_size = shmbuffer.shm_segsz;
        printf("Velicina segmenta je: %d\n", real_segment_size);


        shared_memory = (int *) shmat(segment_id,0,0); // attach rutina
        printf("1.D: shared memory se vidi na adresi %p\n", shared_memory);

        broj = 5;                           // to zapisi u memoriju
        *shared_memory = broj;              // vrsi zapis

        shmdt(shared_memory);               // detach


        shared_memory = (int*) shmat(segment_id,(void*)0x5000000,0);//re-attach
        printf("2.D: shared memory se sada vidi na adresi %p\n", shared_memory);

        broj = *shared_memory;
        printf("U memoriji pise %d\n",broj);

        shmdt(shared_memory);


        // dealociraj dijeljeni segment
        shmctl(segment_id, IPC_RMID, 0);

        return 0;
}
```

Proučite program da vam bude jasno kako funkcionira. Prevedite i pokrenite program. Zakomentirajte predzadnju liniju u kojoj je `shmctl` naredba. Ponovno pokrenite program i počistite zauzete resurse koristeći `ipcs` i `ipcrm` naredbe.

Konkretan zadatak je sljedeći:

- Treba prepraviti gornji program tako da se pokaže problem nesinkroniziranog pristupa dijeljenoj memoriji. Dva procesa čitaju sadržaj dijeljene memorije, uvećavaju taj broj za jedan te zapisuju nazad. Problem nastaje kada dođe do prekida nakon što jedan proces pročita sadržaj a još nije zapisao uvećani iznos. Kada se vrati na njega red za izvođenje prebrisat će u međuvremenu zapisan podatak. Prekid simulirajte sistemskim pozivom `sleep(br_sec)`. Ideja je da, nakon što se oba procesa izvrše, rezultat neće biti za dva veći nego samo za jedan jer je jednom procesu prebrisan njegov zapis.

2. SEMAFORI

Jedan mehanizam kako osigurati sinkronizaciju procese je semafor. Pristup dijeljenoj memoriji se može predstaviti kao *kritični odsječak*. Znači da u tom odsječku se u jednom trenutku smije nalaziti samo jedan proces. Da bi se kritični odsječak zaštitio postavi se binarni semafor (dva stanja). Kad je svijetlo na semaforu zeleno znači da je ulaz u kritični odsječak slobodan a kada je crveno znači da je već netko u kritičnom odsječku i potrebno se stati u red i čekati da se ulaz oslobodi.

Proces koji prolazi kroz zeleno svjetlo iza sebe postavlja semafor na crveno tako da drugi ne mogu ući za njim. Kada izlazi, postavlja semafor na zeleno.

U realnosti semafor ima puno stanja (predstavljena brojem) ali sva moraju biti veća ili jednaka nuli. Za binarni semafor vrijednost 1 znači zeleno a vrijednost 0 crveno. Ulazak u kritični odsječak se izvodi tako da se želi smanjiti vrijednost semafora za 1. Ako je vrijednost bila 1 (zeleno), proces će uči u kritični odsječak i iza sebe će ostaviti semafor na 0 (crveno). Ako je semafor bio na 0, pošto ga ne može smanjiti na -1, čekat će u redu sve dok ne bude 1 (čeka na zeleno). Kada izlazi onda samo uvećava vrijednost semafora za 1 (postavlja na zeleno, jer je bio 0) da drugi procesi mogu ući.

Pristup semaforu je osiguran tako da samo jedan proces u jednom trenutku može mijenjati njegovu vrijednost (ne može doći do prekida za vrijeme te akcije). Znači ne može doći do *race conditiona* za korištenje semafora.

Sistemski pozivi za rad sa semaforima:

Semafor se prvo mora zauzeti. Zapravo se zauzima skup semafora koji u našem slučaju se sastoji samo od jednog semafora.

```
int semget(key_t key, int broj_semafora, int flags) ;
```

Slično kao i za dijeljenu memoriju, `key` se postavlja na IPC_PRIVATE, `broj_semafora` je 1 a `flags` je `S_IRUSR | S_IWUSR`. Rezultat je semafor ID koji se koristi u daljnjim sistemskim pozivima nad tim semaforom.

Na početku su vrijednosti svih semafora u skupu jednake 0. Promjena toga se izvodi sa:

```
int semctl(int semid, int semnum, int cmd, union semun *arg)
```

`semnum` je redni broj semafora u skupu, `cmd` je naredba čiji parametri su definirani zadnjim argumentom. Primjer korištenja je prikazan u kodu naknadno.

Brisanje semafora se odvija isto pomoću `semctl` sa (bitno je samo da je `cmd` IPC_RMID):

```
semctl(sem_id, 1, IPC_RMID,0);
```

Postavljanje semafora se odvija pomoću funkcije:

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

Sama operacija je definirana pokazivačem na strukturu `sembuf`. Primjer korištenja za postavljanje u semafora u dva stanja je dan u kodu koji slijedi.

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

int semafor_id;

union semun {     // struktura (union) potrebno za inicijalizaciju
      int val;
      struct semid_ds *buf;
      unsigned short int *array;
      struct seminfo *__buf;
```

```
};

int sem_init(int semid) {
      union semun argument;
      argument.val = 1;
      return semctl(semid, 0, SETVAL, argument);
}

int sem_down(int semid) { // postavi ili cekaj na crveno.
      struct sembuf operations[1];
      operations[0].sem_num = 0;
      operations[0].sem_op = -1;
      operations[0].sem_flg = SEM_UNDO;

      return semop(semid, operations, 1);
}

int sem_up(int semid) { // postavi na zeleno
      struct sembuf operations[1];
      operations[0].sem_num = 0;
      operations[0].sem_op = 1;
      operations[0].sem_flg = SEM_UNDO;

      return semop(semid, operations, 1);
}

int main() {

      int sem_stanje;

      // kreiranje skupa od 1 semafora.
      semafor_id = semget(IPC_PRIVATE, 1, S_IRUSR | S_IWUSR); // ili 0600

      sem_init(semafor_id);   // inicijaliziram na vrijednost 1
      sem_stanje = semctl(semafor_id, 0, GETVAL); // dohvati vrijednost semafora
      printf("Pocetna vrijednost semafora je: %d\n",sem_stanje);

      sem_down(semafor_id); // pokusaj postaviti na crveno
      printf("Ulazim u kriticni odsjecak - ");
      sem_stanje = semctl(semafor_id, 0, GETVAL); // dohvati vrijednost semafora
      printf("Vrijednost semafora je: %d\n",sem_stanje);

      // KRITICNI ODSJECAK

      sem_up(semafor_id); // postavi na zeleno
      printf("Izasao iz kriticnog odsjecka - ");
      sem_stanje = semctl(semafor_id, 0, GETVAL); // dohvati vrijednost semafora
      printf("Vrijednost semafora je: %d\n",sem_stanje);

      semctl(semafor_id,0 , IPC_RMID, 0); // brisem skup semafora

      return 0;
}
```

Proučite program da vam bude jasno kako funkcionira. Prevedite i pokrenite program.

Zadatak:

- Korištenjem semafora riješite problem pristupa dijeljenoj memoriji iz prethodnog zadatka. Dakle da nakon što oba procesa povećaju vrijednost dijeljenog segmenta rezultat uvijek (bez obzira na prekide) bude za 2 veći nego na početku.