# Seminar 4: KORIŠTENJE SEMAFORA

## SEMAFORI

Semafor je mehanizam koji sprečava da dva ili procesa pristupaju zajedničkom sredstvu istovremeno. Binarni semafor ima dva stanja: propusno i nepropusno. Opći semafor ima beskonačan broj stanja (ili barem vrlo velik). To je brojač koji se smanjuje za jedan kada se zahtijeva semafor, a povećava se za jedan kada se oslobađa. Ako je na nuli, a proces zahtijeva semafor, onda taj proces mora čekati dok drugi proces ne poveća vrijednost semafora. Semafor u UNIX-u (System V) ne može imati negativnu vrijednost iako su teoretski ostvarivi i semafori kod kojih bi bile dozvoljene i negativne vrijednosti. Upotreba semafora se obično razmatra kroz dvije jednostavne operacije: "dohvati" i "otpusti" (*acquire* i *release*) ili "čekaj" i "postavi" (*wait* - *signal*, odnosno, izvorno iz nizozemskog: P i V).

## SEMAFORI U UNIX-u

Pozivi za korištenje semafora u UNIX-u su daleko od jednostavnosti operacija dohvati i otpusti. Dapače, izuzetno teško ih je objasniti ili shvatiti u potpunosti (vidi: `man semget`, `man semop`, `man semctl`). Zbog toga ćemo se ograničiti samo na dio funkcionalnosti poziva za rad sa semaforima koji je dovoljan za većinu primjena.

### Dobavljanje semafora

Semafori se dobavljaju sistemskim pozivom *semget* slično kao i segment zajedničke memorije:

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

Ovaj sistemski poziv dohvaća skup semafora koji se mogu kontrolirati svi od jednom ili kreira novi skup semafora. Novi skup sa ukupno *nsem* semafora će biti kreiran ako se kao ključ upotrijebi `IPC_PRIVATE`. U devet najnižih bitova *flags* se stavljaju dozvole pristupa. *semget* vraća identifikacijski broj skupa semafora ili -1 u slučaju greške.

Redni brojevi semafora u skupu počinju od nule. Zbog jednostavnosti bilo bi dobro imati uvijek samo po jedan semafor u skupu. Međutim, kao i kod zajedničke memorije, ukupan broj skupova semafora u sustavu je ograničen. Zbog velikog broja korisnika taj je broj lako premašiti, pa je poželjno sve potrebne semafore uvijek dobavljati odjednom, tj. u jednom skupu.

### Postavljanje početne vrijednosti semafora

Kada se kreira novi skup semafora, vrijednosti svih semafora u skupu se postavljaju na 0. Ova vrijednost se može promijeniti sistemskim pozivom *semctl*, ali to je samo jedna od brojnih operacija koje on obavlja:

```
union semun {
   int val;
   struct semid_ds *buf;
   unsigned short *array;
};
int semctl(int semid, int semnum, int cmd, union semun *arg) ;
```

*semid* je uvijek identifikacijski broj skupa semafora. Vrijednost semafora se može postaviti na dva načina:

1. Ako je *semnum* redni broj semafora, *cmd* `SETVAL`, a *arg* nenegativan cijeli broj (*arg.val*), postavlja se vrijednost određenog semafora.

2. Ako je *cmd* `SETALL`, a *arg* kazaljka na niz kratkih cijelih brojeva bez predznaka (*arg.array*), postavljaju se svi semafori u skupu na vrijednosti iz danog niza.

*semctl* pozvan na ovaj način vraća 0 ako je sve u redu ili -1 u slučaju greške (na primjer, vrijednost na koju treba postaviti semafor je prevelika ili negativna).

## Uništavanje semafora

Skup semafora treba uništiti nakon upotrebe jer u protivnom ostaje kao trajno zauzeto sredstvo u sustavu. Uništavanje semafora također se provodi sistemskim pozivom *semctl*. *semid* je identifikacijski broj skupa semafora, a *cmd* treba biti `IPC_RMID`. Ostali argumenti nisu bitni.

## Operacije sa semaforima

```
struct sembuf {
   short sem_num;
   short sem_op;
   short sem_flg;
};
int semop(int semid, struct sembuf (*sops)[], int nsops) ;
```

*semop* se može upotrijebiti za nedjeljivo izvođenje niza operacija na skupu semafora. Međutim, najčešće je sasvim dovoljno izvesti jednu operaciju u jednom pozivu. U tom slučaju, *semid* je identifikacijski broj skupa semafora, *nsops* je 1, a *sops* je kazaljka na strukturu koja opisuje traženu operaciju. Unutar te strukture, *sem_num* je redni broj semafora u skupu, *sem_op* je tražena operacija koju dodatno opisuje i *sem_flg*.

Za uobičajeno korištenje semafora dovoljne su dvije vrste operacija: povećavanje vrijednosti semafora (*sem_op* je pozitivan) i smanjivanje vrijednosti semafora (*sem_op* je negativan). U oba slučaja, *sem_flg* treba biti 0. *semop* vraća rezultat 0 ako je sve u redu, ili -1 u slučaju greške.

Smanjivanje vrijednosti semafora odgovara operaciji "čekaj". Ukoliko je vrijednost semafora moguće umanjiti za dani broj, a da ne postane negativna, vrijednost se umanjuje, a proces nastavlja rad. U suprotnom slučaju, proces čeka dok vrijednost semafora ne postane dovoljno velika da se umanjivanje može provesti.

Povećavanje vrijednosti odgovara operaciji "postavi". Vrijednost semafora se povećava za dani broj, a proces nastavlja rad. Proces zaustavljen u redu semafora se oslobađa ako je ovim povećanjem vrijednost semafora postala dovoljno velika da je on može umanjiti, a da ipak ne postane negativna.

## Pomoćne funkcije za rad sa semaforima

Prethodna objašnjenja rada sa semaforima mogu se upotrijebiti za izradu pomoćnih funkcija za rad sa semaforima koje su mnogo jednostavnije za upotrebu od izravnih sistemskih poziva.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int SemId;   /* identifikacijski broj skupa semafora */

void SemGet(int n)
{  /* dobavi skup semafora sa ukupno n semafora */
   SemId = semget(IPC_PRIVATE, n, 0600);
   if (SemId == -1) {
      printf("Nema semafora!\n");
      exit(1);
   }
}

int SemSetVal(int SemNum, int SemVal)
{  /* postavi vrijednost semafora SemNum na SemVal */
   return semctl(SemId, SemNum, SETVAL, &SemVal);
```

```
}

int SemOp(int SemNum, int SemOp)
{  /* obavi operaciju SemOp sa semaforom SemNum */
   struct sembuf SemBuf;

   SemBuf.sem_num = SemNum;
   SemBuf.sem_op  = SemOp;
   SemBuf.sem_flg = 0;
   return semop(SemId, & SemBuf, 1);
}

void SemRemove(void)
{  /* uništi skup semafora */
   (void) semctl(SemId, 0, IPC_RMID, 0);
}
```

# ZADATAK

**1.** Ako je poznato da je najveća dozvoljena vrijednost semafora `MAXSHORT` (definicija u `<values.h>`), napisati operacije za rad s binarnim semaforima:

`CekajBSem(SemNum)` i `PostaviBSem(SemNum)`

koristeći pozive za rad sa semaforima u UNIX-u.

Ostvariti međusobno isključivanje dva procesa korištenjem binarnog semafora. Struktura procesa dana je slijedećim pseudokodom:

```
proces proc(i)          /* i  [0..1] */
   za k = 1 do 5 čini
      čekaj binarni semafor
      za m = 1 do 10 čini
         ispiši (i, k, m)
      postavi binarni semafor
kraj.
```

Može li isti postupak služiti za međusobno isključivanje većeg broja procesa ?

**2.** Napisati operacije za rad s općim semaforima:

`CekajOpciSem(SemNum)` i `PostaviOpciSem(SemNum)`

koristeći pozive za rad sa semaforima u UNIX-u.

Najjednostavnije ostvarenje općih semafora će zadržati ograničenje semafora u UNIX-u koji se ne mogu postaviti na negativnu vrijednst. Međutim, to ograničenje se može zaobići nešto složenijim ostvarenjem.

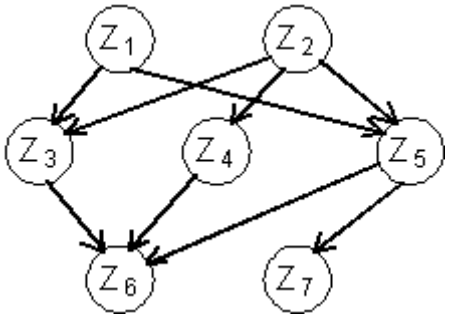
Napisati operacije za rad s općim semaforima koji mogu poprimiti i negativne vrijednosti: `CekajOSem(BrojSemafora)`, `PostaviOSem(BrojSemafora)`, i `PostaviVrijednostOSem(BrojSemafora, Vrijednost)` koristeći pozive za rad sa semaforima u UNIX-u.

Interval dozvoljenih vrijednosti za semafore u UNIX-u treba za OSem podijeliti u dvije grupe: prolazne vrijednosti i neprolazne vrijednosti. Na primjer, može se uzeti da vrijednost semafora 16384 odgovara nuli za `OSem`. Manje vrijednosti odgovaraju negativnim brojevima za `OSem`, a veće pozitivnim.

Operacija `CekajOSem` započinje umanjivanjem vrijednosti semafora za 1, a zatim treba provjeriti je li dobivena vrijednost veća ili jednaka nuli. Provjeru, također, treba obaviti UNIX-ovom operacijom sa semaforom i to operacijom oduzimanja vrijednosti koja odgovara nuli. Kada ta operacija uspješno završi, semaforu odmah treba vratiti očekivanu vrijednost, i to opet korištenjm UNIX-ove operacije sa semaforom. Prema tome, `CekajOSem` ce se sastojati od tri poziva operacija sa semaforom zaredom. Za razliku od nje, operacije `PostaviOSem` i `PostaviVrijednostOSem` su mnogo jednostavnije.

Korištenjem tako dobivenih općih semafora ili operacija za rad sa semaforima u UNIX-u, ostvariti sinkronizaciju sedam procesa prema zadanom grafu (na primjer, $Z_3$ smije početi tek kada završe $Z_1$ i $Z_2$). Zadatak koji obavlja svaki od procesa dan je pseudokodom:

```
proces Z(i)  /* i  [1..7] */
   za j = 1 do 20 čini
   ispiši (i, j)
kraj.
```




**3.** Ostvariti komunikaciju među procesima, koji se paralelno izvode, preko međuspremnika ograničene duljine. Dva procesa proizvođača trebaju na početku pročitati niz znakova s tastature i nakon toga poslati znak po znak preko međuspremnika procesu tipa potrošač. Proces potrošač treba primiti znak po znak od proizvođača i nakon što je primio sve znakove od oba proizvođača, treba ih ispisati na zaslonu.

Struktura podataka zajedničkih za procese proizvođače i potrošače:

```
int ULAZ = 0
int IZLAZ = 0
char međuspremnik[5]
binarni semafor BSEM
opći semafor OSEM_PUN
opći semafor OSEM_PRAZAN
```

Struktura procesa proizvođač:

```
proces proizvođač
   pročitaj niz znakova sa tastature u polje s
   i = 0
   čini
      čekaj_OSEM_PUN
      čekaj_BSEM
      međuspremnik[ULAZ] = s[i]
      ULAZ = (ULAZ+1) mod 5
      postavi_BSEM
      postavi_OSEM_PRAZAN
      i = i+1
   do kraja niza
kraj.
```

Struktura procesa potrošač:

```
proces potrošač
   i = 0
   čini
      čekaj_OSEM_PRAZAN
      s[i] = međuspremnik[IZLAZ]
      IZLAZ = (IZLAZ+1) mod 5
      postavi_OSEM_PUN
      i = i+1
   do kraja oba niza
   ispiši niz znakova iz polja s na zaslon
kraj.
```

# UPUTA

**1.** Pripaziti na ispravnu inicijalizaciju binarnih i općih semafora.

**2.** Oznaku kraja niza je potrebno prenositi kroz međuspremnik kako bi proces potrošač mogao znati kada nastupa kraj niza. Proces potrošač treba ispisati primljene znakove i završiti tek kada primi dvije oznake kraja niza.

**3.** Početak glavnih petlji proizvođača treba uskladiti korištenjem semafora ili na neki drugi način kako se ne bi desilo da jedan od procesa proizvođača već završi dok drugi još prikuplja niz znakova s tastature.