Sinkronizacijski mehanizmi

Istovremeni pristup podacima od strane više dretvi se ponekad ne smije dopustiti. To su situacije u kojima dretve koriste zajedničke varijable, a čija bi istovremena uporaba izazvala greške. Takvi se kritični odsječci programa zaštićuju međusobnim isključivanjem. Kod korištenja zajedničkih podataka treba također uzeti u obzir da promjena koju uzrokuje jedna dretva ne mora baš istog trenutka biti vidljiva svim ostalim dretvama. Odnosno, zbog toga što procesori koriste vlastite priručne spremnike, promjena podatka se privremeno može obaviti samo lokalno, a ažuriranje glavnog spremnika može se obaviti tek kasnije. Takvi se dijelovi programa također zaštićuju.

UNIX operacijski sustav (*SUN Solaris, a i Linux*) omogućuje sljedeće načine sinkronizacije među dretvama: *semafori*, *međusobno isključivanje, uvjetne varijable*.

Semafori

Semafori se obično upotrebljavaju za pristup *ograničenim* sredstvima od strane većeg broja korisnika (dretvi), tj. služe kao brojači događaja. Osnovne funkcije za rad sa semaforima u višedretvenom programu su: `sem_init`, `sem_post` i `sem_wait`.

```
int sem_init(sem_t *sem, int mprocesi, unsigned int koliko);
int sem_post(sem_t *sem);
int sem_wait(sem_t *sem);
```

`sem` je pokazivač na varijablu semafora, `koliko` je broj na koji se postavi semafor, `mprocesi` označava je li semafor predviđen za sinkronizaciju dretvi istog procesa (0) ili dretvi različitih procesa (nešto različito od nule).

Ukoliko se ove funkcije koriste za sinkronizaciju procesa, objekt semafora (ono na što pokazuje sem) mora biti u zajedničkoj memoriji (shmget+shmat+...)!

Funkcija `sem_post` jedinično povećava semafor.

Funkcija `sem_wait` smanjuje vrijednost semafora za jedan, ako je vrijednost semafora veća od nule, u protivnom čeka dok se vrijednost ne poveća.

Primjer sa semaforima i procesima (isječci koda)

```
sem_t *sem; //globalna varijabla = za kazaljku na objekt u zajedničkoj
memoriji

void proces (int id)
{       ...
```

```
        sem_wait (sem); i/ili sem_post (sem);
        ...
}
int main()
{
        ...
        ID = shmget (IPC_PRIVATE, sizeof(sem_t), 0600);
        sem = shmat (ID, NULL, 0);
        shmctl (ID, IPC_RMID, NULL); //moze odmah ovdje, nakon shmat, ili
na kraju nakon shmdt jer IPC_RMID oznacava da segment treba izbrisati nakon
sto se zadnji proces odijeli od tog segmenta (detach)
        sem_init (sem, 1, 5); //početna vrijednost = 5, 1=>za procese

        ... fork () ...
        ...
        ... wait (NULL) ...

        sem_destroy (sem);
        shmdt (sem);

        return 0;
}
```

Kada bi se semafor koristio za dretve unutar istog procesa, onda bi `sem` mogao biti obična globalna varijable (`sem_t sem;`), ne bi bilo potrebe rezervirati memoriju za njega, a su svim ostalim funkcijama bi se pozivao preko adrese (npr. `sem_init(&sem, 1, 5);`).

Monitori

Kada je potrebno zaštititi neke dijelove programa od istovremenog korištenja više dretvi (kritični odsječci), tada se koriste funkcije međusobnog isključivanja, odnosno, dijelovi programa se zaključavaju pomoću kontrolnih varijabli - ključeva. Inicijalizacija ključeva se obavlja funkcijom `pthread_mutex_init`, zaključavanje funkcijom `pthread_mutex_lock`, a otključavanje funkcijom `pthread_mutex_unlock` (*Solaris* ima i svoje vlastite funkcije za te operacije koje nemaju prefix "pthread_")

```
int pthread_mutex_init(pthread_mutex_t *ključ, const pthread_mutexattr_t
*attr);
int pthread_mutex_lock(pthread_mutex_t *ključ);
int pthread_mutex_unlock(pthread_mutex_t *ključ);
```

`ključ` pokazuje na varijablu zaključavanja, `attr` definira karakteristike stvorene varijable (NULL za pretpostavljene vrijednosti). Sve dretve koje pokušaju zaključati već zaključanu varijablu ostaju blokirane na pozivu sve dok varijabla ostaje zaključana. Kada se varijabla otključa, tada samo jedna dretva ulazi u kritični osječak, uz ponovno zaključavanje varijable. Zaključavanjem se smanjuje moguća istovremenost u izvođenju skupine dretvi, te je njihova upotreba prihvatljiva (obavezna) samo u kritičnim odsječcima.

Uvjetne varijable se koriste kada želimo da neka dretva zaustavi svoje izvođenje te čeka da se određeni uvjet ispuni, tj. da ga ispuni neka druga dretva. Funkcijama za

korištenje uvjetnih varijabli obavezno prethodi zaključavanje, budući su uvjetne varijable globalne, tj. zajedničke cijelom procesu.

Osnovne funkcije za rukovanje uvjetnim varijablama su `pthread_cond_init`, `pthread_cond_wait`, `pthread_cond_signal` i `pthread_cond_broadcast`.

```
int pthread_cond_init(pthread_cond_t *uvjet, const pthread_condattr_t
*attr);
int pthread_cond_wait(pthread_cond_t *uvjet, pthread_mutex_t *ključ);
int pthread_cond_signal(pthread_cond_t *uvjet);
int pthread_cond_broadcast(pthread_cond_t *uvjet);
```

`uvjet` je kazaljka na uvjetnu varijablu, `ključ` jest kazaljka na varijablu zaključavanja, `attr` odefinira karakteristike stvorene varijable (NULL za pretpostavljene vrijednosti). Pozivom `pthread_cond_wait` pozivajuća dretva se postavlja u stanje čekanja tj. premješta ju se u red `uvjet` i istovremeno se otljučava `ključ`. Čekanje te dretve završava neka druga dretva pozivima `pthread_cond_signal` ili `pthread_cond_broadcast`. Po primitku "signala" dretva prije nastavka rada najprije mora ponovno zaključati `ključ`.
Poziv `pthread_cond_signal` ispunjuje uvjet za nastavaka samo jedne dretve iz reda čekanja na istu uvjetnu varijablu, dok poziv `pthread_cond_broadcast` omogućuje svim takvim dretvama nastavak izvođenja. Ako niti jedna dretva nije bila blokirana na uvjetnoj varijabli prilikom poziva `pthread_cond_signal` i `pthread_cond_broadcast`, onda ovi pozivi nemaju nikakav učinak, tj. ako već u slijedećem trenutku neka dretva pozove `pthread_cond_wait` ona ostaje blokirana. Uvjetnim varijablama moguće je ostvariti sustav monitora - mehanizma kojim je moguće istovremeno provjeriti više uvjeta (zauzeti resurse) potrebnih za napredovanje dretve.

Monitor se sastoji od skupa procedura i strukture podataka nad kojima procedure djeluju i koje nisu vidljive izvan monitora. Procedure se ne smiju izvoditi paralelno. Unutar monitora ispituje se uvjet koji utječe na odvijanje zadatka. Ukoliko je uvjet ispunjen, zadatak zauzima sredstva i obavlja potrebne akcije nad njima unutar monitora. Po završetku oslobađa sredstva te dozvoljava drugom zadatku ulazak u monitor te izlazi iz monitora. Ukoliko po ulasku u monitor uvjet nije ispunjen tada zadatak odlazi u red čekanja na taj uvjet, tj. prividno napušta monitor.

Ostvarenje monitora u višedretvenom programu prilično je jednostavna budući na raspolaganju stoje funkcije međusobnog isključavanja čime se jednostavno postiže da samo jedna dretva ulazi u monitor, te funkcije za rukovanje uvjetnim varijablama koje služe za ostvarenje reda uvjeta. Ostvarenje monitora koji zauzima sredstva `p` i `q` (npr. lijevi i desni štapić kod problema pet filozofa) prikazan je u nastavku:

```
void uzmi_sredstva (int p, int q)
{
```

```
        pthread_mutex_lock (&monitor);
        while ( p == 0 || q == 0 )
                pthread_cond_wait (&uvjet,&monitor);
        p = q = 0;
        pthread_mutex_unlock (&monitor);
}
void vrati_sredstva (int p, int q)
{
        pthread_mutex_lock (&monitor);
        p = q = 1;
        pthread_cond_broadcast (&uvjet);
        pthread_mutex_unlock (&monitor);
}
```

Primjer s monitorom

```
...
pthread_mutex_t m;
pthread_cond_t red;
...
void *dretva (void *p)
{       ...
        pthread_mutex_lock (&m);
        ...
        while ( uvjet-blokiranja )
                pthread_cond_wait (&red, &m);
        ...
        pthread_mutex_unlock (&m);
        ...
        pthread_mutex_lock (&m);
        ...
        if ( uvjet-otpuštanja )
                pthread_cond_signal (&red); ili pthread_cond_broadcast
(&red);
        ...
        pthread_mutex_unlock (&m);
        ...
}
int main ()
{       ...
        pthread_mutex_init (&m, NULL);
        pthread_cond_init (&red, NULL);
        ...
        pthread_create (..., ..., dretva, ...);
        ...
        pthread_join (...);
        ...
        return 0;
}
```

Stranice (manual) POSIX dretvi u kojima su detaljno opisane funkcije za rad s dretvama *pthread*: pthread, pthread create, pthread exit, pthread detach, pthread join, pthread mutex init, pthread mutex lock, pthread mutex unlock, pthread mutex destroy, pthread cond init, pthread cond wait, pthread cond signal, sem init, sem wait, sem post, sem destroy...

Vrtuljak

Zadatak

Modelirati vrtuljak (ringišpil) s dva tipa dretvi/procesa:
dretvama/procesima *posjetitelj* (koje predstavljaju posjetitelje koji žele na vožnju) te dretvom/procesom *vrtuljak*. Dretvama/procesima *posjetitelj* se ne smije dozvoliti ukrcati na vrtuljak kada više nema praznih mjesta (kojih je ukupno *N*) te prije nego li svi prethodni posjetitelji siđu. Vrtuljak se može pokrenuti tek kada je pun. Za sinkronizaciju koristiti opće semafore i dodatne varijable.

```
Dretva posjetitelj() {
   ...
   sjedi;
   ...
   ustani; // ili sidji
   ...
}


Dretva vrtuljak() {
   dok je(1) {
      ...
      pokreni vrtuljak;
      zaustavi vrtuljak;
      ...
   }
}
```

MS & Linux programeri

Zadatak

U istoj zgradi rade Microsoftovi programeri i Linux programeri. Zgrada ima jedan restoran kojega programeri moraju dijeliti. U istom trenutku u restoranu smije biti samo jedna vrsta programera (ili je restoran prazan). Sinkronizirati programere monitorom (ispravna sinkronizacija podrazumijeva i sprječavanje izgladnjivanja).

Svaki programer ima sljedeći oblik:

```
Programer(vrsta){
     udji(vrsta);
     obavi;
     izadji(vrsta);
}
```