**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\_postisem\_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\_signali 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](http://www.zemris.fer.hr/predmeti/os/pripreme/upute/pthread/pthread.htm), [pthread\_create](http://www.zemris.fer.hr/predmeti/os/pripreme/upute/pthread/pthread_create.htm), [pthread\_exit](http://www.zemris.fer.hr/predmeti/os/pripreme/upute/pthread/pthread_exit.htm), [pthread\_detach](http://www.zemris.fer.hr/predmeti/os/pripreme/upute/pthread/pthread_detach.htm), [pthread\_join](http://www.zemris.fer.hr/predmeti/os/pripreme/upute/pthread/pthread_join.htm), [pthread\_mutex\_init](http://www.zemris.fer.hr/predmeti/os/pripreme/upute/pthread/pthread_mutex_init.htm), [pthread\_mutex\_lock](http://www.zemris.fer.hr/predmeti/os/pripreme/upute/pthread/pthread_mutex_lock.htm), [pthread\_mutex\_unlock](http://www.zemris.fer.hr/predmeti/os/pripreme/upute/pthread/pthread_mutex_unlock.htm), [pthread\_mutex\_destroy](http://www.zemris.fer.hr/predmeti/os/pripreme/upute/pthread/pthread_mutex_destroy.htm), [pthread\_cond\_init](http://www.zemris.fer.hr/predmeti/os/pripreme/upute/pthread/pthread_cond_init.htm), [pthread\_cond\_wait](http://www.zemris.fer.hr/predmeti/os/pripreme/upute/pthread/pthread_cond_wait.htm), [pthread\_cond\_signal](http://www.zemris.fer.hr/predmeti/os/pripreme/upute/pthread/pthread_cond_signal.htm), [sem\_init](http://www.zemris.fer.hr/predmeti/os/pripreme/upute/pthread/sem_init.htm), [sem\_wait](http://www.zemris.fer.hr/predmeti/os/pripreme/upute/pthread/sem_wait.htm), [sem\_post](http://www.zemris.fer.hr/predmeti/os/pripreme/upute/pthread/sem_post.htm), [sem\_destroy](http://www.zemris.fer.hr/predmeti/os/pripreme/upute/pthread/sem_destroy.htm)...