SVEUČILIŠTE U MOSTARU
FAKULTET STROJARSTVA, RAČUNARSTVA I ELEKTROTEHNIKE
PREDDIPLOMSKI STUDIJ RAČUNARSTVA

OPERACIJSKI SUSTAVI

VJEŽBE

Nastavnik: prof.dr.sc. Sven Gotovac
gotovac@fesb.hr

Asistent: Željko Šeremet
zeljko.seremet@fsre.sum.ba

MOSTAR, SVIBANJ 2024.

Sinkronizacija dretvi

Sadržaj

- Načini sinkronizacije među dretvama
- Uvjetne varijable
- Monitor
- Semafori

Uvod

- 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.

Načini sinkronizacije među dretvama (1)

- UNIX operacijski sustav (*SUN Solaris, a i Linux*) omogućuje sljedeće načine sinkronizacije među dretvama: *međusobno isključivanje, uvjetne varijable* i *semafori*. Kada je potrebno zaštititi neke dijelove programa od istovremenog korištenja više dretvi (kritični odsječci), tada se koriste funkcije međusobnog isključivanja, odnosno, dijelovi programa se zaključuju 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_**")

Načini sinkronizacije među dretvama (2)

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, onda samo jedna dretva ulazi slijedeća u kritični osječak, uz zaključavanje varijable. Zaključavanjem se smanjuje moguća istovremenost u izvođenju skupine dretvi, te je njihova upotreba prihvatljiva (obavezna) samo u kritičnim odsječcima.

Uvjetne varijable (1)

- ***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**.

Uvjetne varijable (2)

```
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** definira karakteristike stvorene varijable (NULL za pretpostavljene vrijednosti). Pozivom **pthread_cond_wait** pozivajuća dretva se postavlja u stanje čekanja tj. premještaju se u red uvjeta i istovremeno se otljučava ključ.

Uvjetne varijable (3)

- 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 nastavak 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.

Uvjetne varijable (4)

- Uvjetnim varijablama moguće je ostvariti sustav monitora - mehanizma kojim je moguće istovremeno provjeriti više uvjeta (zauzeti resurse) potrebnih za napredovanje dretve.
- Poziv **pthread_cond_wait(uvjet, ključ)** može se shvatiti u dva koraka: u prvom se otključava ključ i dretva pomiče u red uvjet; u drugom, nakon primitka signala se mora ponovno zaključati ključ. Tj. mogli bi reći da funkcija izgleda:

Uvjetne varijable (5)

- **pthread_cond_wait(uvjet, ključ) {**
  oslobodi(ključ); //funkcionalno skoro isto što i mutex_unlock()
  uvrsti_u_red_uvjeta(uvjet);

  zauzmi(ključ); //funkcionalno isto što i mutex_lock()
  }

Monitor (1)

- ***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.

Monitor (2)

- 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 prikazan je u nastavku:

Monitor (3)

```
void uđi_u_monitor(cond_t *uvjet){
        pthread_mutex_lock(&monitor);
        while(p==0 || q==0)

        pthread_cond_wait(uvjet,&monitor);
        p=q=0;
        pthread_mutex_unlock(&monitor); }
  void izađi_iz_monitora(cond_t *uvjet){
        pthread_mutex_lock(&monitor);
        p=q=1;
        pthread_cond_broadcast(uvjet);
        pthread_mutex_unlock(&monitor); }
```

Semafori (1)

- ***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);

Semafori (2)

- **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). 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.

POSIX dretve

- 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…

Zadaća 5.

- S pomoću više dretvi riješiti problem pet filozofa koristeći koncept monitora. Pri svakoj promjeni program mora vizualno prikazati za sve filozofe što oni rade. Npr. kada filozof 4 ide jesti, tada treba ispis izgledati otprilike ovako: "Stanje filozofa: X o O X o" (X-jede, O-razmišlja, o-čeka na vilice).
- *Problem pet filozofa*. Filozofi obavljaju samo dvije različite aktivnosti: misle ili jedu. To rade na poseban način. Na jednom okruglom stolu nalazi se pet tanjura te pet štapića (između svaka dva tanjura po jedan). Filozof prilazi stolu, uzima lijevi štapić, pa desni te jede. Zatim vraća štapiće na stol i odlazi misliti.

Upute (1)

- Ako rad filozofa predstavimo jednim zadatkom onda se on može opisati na sljedeći način:
- **filozof** i
  ponavljati
    misliti;
    jesti;
  do zauvijek;

Upute (2)

- Potrebno je pravilno sinkronizirati rad filozofa. Uporabom binarnog semafora za pojedine štapiće može doći do potpunog zastoja (kada svi istovremeno uzmu lijevi štapić). Problem se može riješiti uvođenjem dodatnog općeg semafora koji će ograničavati broj filozofa za stolom.
- Međutim, u općem slučaju kada zadaci dijele više sredstava i koriste ih više istovremeno semafori mogu uzrokovati potpuni zastoj. Upravo zbog takvih problema dijeljenja više sredstava među više zadataka uvodi se novi mehanizam za sinkronizaciju: monitori.

Upute (3)

- Za ostvarenje monitora u višedretvenom programu stoje na raspolaganju funkcije međusobnog isključavanja te funkcije za rukovanje uvjetnim varijablama koje služe za ostvarenje reda uvjeta.
- Pseudokod pretvoriti u dio programa koji radi svaka dretva tako da misliti i jesti postaju funkcije. Funkcija **misliti()** treba simulirati trošenje vremena (npr. sleep(3)). Funkcija **jesti(n)** simulira postupak hranjenja filozofa s uključivo potrebnim kodom sinkronizacije koji je skiciran u nastavku.

Upute (4)

```
jesti(n){
        /* n - redni broj filozofa */
   uđi_u_kritični_odsječak;
     filozof[n] = 'o';
     dok (vilica[n] == 0 || vilica[(n + 1) % 5] == 0)
       čekaj_u_redu_uvjeta(red[n]);
     vilica[n] = vilica[(n + 1) % 5] = 0;
     filozof[n] = 'X';
     ispiši_stanje(n);
   izađi_iz_kritičnog_odsječka;

   njam_njam; /* sleep(2); */
```

Upute (5)

```
uđi_u_kritični_odsječak;
      filozof[n] = 'O';
      vilica[n] = vilica[(n + 1) % 5] = 1;
      oslobodi_dretvu_iz_reda(red[(n - 1) % 5]);
      oslobodi_dretvu_iz_reda(red[(n + 1) % 5]);
      ispiši_stanje(n);
   izađi_iz_kritičnog_odsječka;
}
```

Literatura

- Korisni linkovi:
- http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html
- http://docs.oracle.com/cd/E19683-01/806-6867/sthreads-84932/index.html
- http://faq.programmerworld.net/programming/win32-multithreading-and-synchronization.html

Kraj