Operativni sistemi
2012

Sistemsko programiranje

Sinhronizacija niti

Katedra za računarstvo
Elektronski fakultet u Nišu

Prof. dr Dragan Stojanović
mr Aleksandar Stanimirović
mr Bratislav Predić

Sadržaj

- Semafori
- Sinhronizacija niti
- Mutex
- Uslovne promenljive
- POSIX semafori

Sadržaj

- Semafori
- Sinhronizacija niti
- Mutex
- Uslovne promenljive
- POSIX semafori

Semafori

Pojam

- Primitiva za sinhronizaciju koju je predložio Dijkstra 1965 godine.
- Semafor se u suštini ponaša kao celobrojnu promenljivu.
- Nad semaforom se mogu izvršiti samo dve operacije Pi V (WAIT i SIGNAL, DOWN i UP).
- Ove operacije su nedeljive (atomične).
- Sistem garantuje da se ove dve operacije izvršavaju bez prekida odnosno kada započne njihovo izvršavanje nijedna druga instrukcija ih ne može prekinuti.

Semafori

Operacije

S – semaforska promenljiva

P(S): if S > 0
then S = S -1
else Blokirati proces na semaforu S

V(S): if Postoji proces koji čeka na semaforu S
then Aktivirati proces iz liste čekanja
else S = S + 1

Semafori

Karakteristike

- Tipovi semafora
  - binarni semafori (mutex) – mogu uzeti samo vrednosti 0 i 1
  - semafori opšteg tipa – mogu uzimati vrednosti iz opsega 0..N gde je N maksimalna vrednost koju semafor može da ima
- Nedostaci semafora
  - dualizam P i V operacija – strogo se mora poštovati da svaka P operacija ima svoju V operaciju i obratno
  - redosled operacija – pogrešan redosled operacija može da dovede do nepravilnog funkcionisanja aplikacije pa čak i potpunog blokiranja aplikacije

Sadržaj

- Semafori
- Sinhronizacija niti
- Mutex
- Uslovne promenljive
- POSIX semafori

Sinhronizacija niti

Karakteristike

- Komunikacija između različitih niti je relativno jednostavan problem.
- Niti koje pripadaju istom procesu dele zajednički adresni prostor pa mogu pristupati zajedničkim resursima (promenljivama).
- Niti mogu istovremeno pristupati deljivom resursu bez konflikata ukoliko ne menjaju taj resurs.
- Problemi se javljaju kada veći broj niti istovremeno menja neki resurs – problem trke.
- U tom slučaju stanje resursa zavisi od redosleda izvršavanja niti.
- Greške se veoma teško otkrivaju jer se krajnji rezultat često menja od slučaja do slučaja.
- Neke od metoda za sinhronizaciju niti su:
  - Mutex
  - Uslovne promenljive
  - POSIX semafori

Sadržaj

- Semafori
- Sinhronizacija niti
- Mutex
- Uslovne promenljive
- POSIX semafori

Mutex

Pojam

- Mutex predstavlja implementaciju binarnog semafora.
- Koristi se za međusobno isključenje niti prilikom pristupanja deljivom resursu.
- Međusobno isključenje niti obezbeđuje da se pristupanje deljivom resursu vrši strogo sekvencijalno.
- Sve neophodne strukture podataka i funkcije su definisane u zaglavlju **<pthread.h>**
- Mutex se definiše kao promenljiva tipa **pthread_mutex_t**.

Mutex

Kreiranje mutex-a

```
#include <pthread.h>
int pthread_mutex_init ( pthread_mutex_t * mutex, pthread_mutexattr_t   * attr );
```

Semantika

- Sistemski poziv koji kreira novi objekat tipa mutex.
- Prvi argument je pokazivač na promenljivu tipa mutex koja je prethodno deklarisana.
- Drugi argument definiše atribute mutex-a koji se kreira. NULL vrednost ovog argumenta obezbeđuje korišćenje podrazumevanih vrednosti.

Mutex

Brisanje mutex-a

```
#include <pthread.h>
int pthread_mutex_destroy ( pthread_mutex_t * mutex);
```

Semantika

- Sistemski poziv koji se koristi za brisanje objekta tipa mutex nakon završetka rada sa njim.
- Argument je pokazivač na objekat tipa mutex koji želimo da obrišemo.

Mutex

Zaključavanje mutex-a

```
#include <pthread.h>
int pthread_mutex_lock ( pthread_mutex_t * mutex);
```

Semantika

- Niti koja zaključa mutex postaje njegov vlasnik i samo ga ona može otključati.
- U jednom trenutku samo jedna nit može biti vlasnik mutex-a (može zaključati mutex).
- Niti koje pokušaju da preuzmu vlasništvo nad zaključanim mutex-om se blokiraju sve dok se mutex ne otključa.
- Odključavanjem mutex-a jedna od niti koje čekaju postaje vlasnik tog mutex-a a ostale niti nastavljaju da čekaju.

Mutex

Odključavanje mutex-a

```
#include <pthread.h>
int pthread_mutex_unlock ( pthread_mutex_t * mutex);
```

Semantika

- Niti koja je vlasnik mutex-a može ga osloboditi odnosno otključati.
- Funkcija vrća grešku ukoliko se pokuša otključavanje mutex-a koji nije zaključan ili je mutex zaključala neka druga nit (nit koja poziva funkciju nije vlasnik niti)

Mutex

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

pthread_mutex_t lock;
void * prva_nit(void *args);
void * druga_nit(void *args);

int main()
{
   pthread_t t1, t2;

   pthread_mutex_init(&lock, NULL);

   pthread_create(&t1, NULL, (void *)&prva_nit, NULL);
   sleep(3);
   pthread_create(&t2, NULL, (void *)&druga_nit, NULL);

   pthread_join(t1, NULL);
   pthread_join(t2, NULL);
   pthread_mutex_destroy(&lock);
   return 0;
}

void * prva_nit(void *args)
{
   pthread_mutex_lock(&lock);
   sleep(20);
   pthread_mutex_unlock(&lock);
}

void * druga_nit(void *args)
{
   pthread_mutex_lock(&lock);
   printf("Poruka\n");
   pthread_mutex_unlock(&lock);
}
```

Mutex deklarisan kao globalni objekat da bi bio vidljiv svim nitima.

Kreiranje mutex-a.

Kreiranje niti.

Čeka se da se niti završe.

Brisanje mutex-a.

Kritična sekcija.

Kritična sekcija.

Sadržaj

- Semafori
- Sinhronizacija niti
- Mutex
- Uslovne promenljive
- POSIX semafori

Uslovne promenljive

Pojam

- Uslovna promenljiva je primitiva koja se koristi za sinhronizaciju niti u sadejstvu sa objektima tipa mutex.
- Uslovna promenljiva omogućava sinhronizaciju niti na osnovu vrednosti nekog zajedničkog resursa (obično programske promenljive).
- Sve neophodne strukture podataka i funkcije su definisane u zaglavlju **<pthread.h>.**
- Uslovna promenljiva se definiše kao promenljiva tipa **pthread_cond_t**.

Uslovne promenljive

Korišćenje uslovnih promenljivih

Prva nit:

1. nit zaključava mutex
2. nit se blokira na uslovnoj promenljivoj i čeka ispunjenje uslova
3. nit otključava mutex

Druga nit:

1. nit zaključava mutex
2. ispunjava se uslov
3. signalizira se blokiranoj niti da je uslov ispunjen
4. nit otključava mutex

Uslovne promenljive

Kreiranje uslovne promenljive

```
#include <pthread.h>
int pthread_cond_init ( pthread_cond_t * condition, pthread_attr_t   * attr );
```

Semantika

- Sistemski poziv koji kreira novi objekat tipa uslovna promenljiva.
- Prvi argument je pokazivač na uslovnu promenljivu koja je prethodno deklarisana.
- Drugi argument definiše atribute uslovne promenljive koja se kreira. NULL vrednost ovog argumenta obezbeđuje korišćenje podrazumevanih vrednosti.

Uslovne promenljive

Brisanje uslovne promenljive

```
#include <pthread.h>
int pthread_cond_destroy ( pthread_cond_t * condition );
```

Semantika

- Sistemski poziv koji se koristi za brisanje uslovne promenljive nakon završetka rada sa njom.
- Argument je pokazivač na uslovnu promenljivu koji želimo da obrišemo.

Uslovne promenljive

Blokiranje niti

```
#include <pthread.h>
int pthread_cond_wait ( pthread_cond_t * condition, pthread_mutex_t   * mutex );
```

Semantika

- Sistemski poziv blokira izvršavanje niti na određenoj uslovnoj promenljivoj.
- Nit se blokira na uslovnoj promenljivoj sve dok neka druga nit ne signalizira da je uslov koji se čeka ispunjen.
- Funkcija se poziva samo ukoliko je objekat tipa mutex zaključan.
- Mutex se automatski otključava tokom čekanja niti.

Uslovne promenljive

Signalizacija uslovne promenljive

```
#include <pthread.h>
int pthread_cond_signal ( pthread_cond_t * condition );
int pthread_cond_broadcast ( pthread_cond_t * condition );
```

Semantika

- Sistemski poziv signalizira (budi) nit koja čeka na uslovnoj promenljivoj.
- Funkcija se poziva u okviru kritične sekcije odnosno mutex mora biti zaključan.
- Ukoliko više niti čeka na istu uslovnu promenljivu onda se koristi broadcast funkcija.

Uslovne promenljive

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

pthread_mutex_t lock;
pthread_cond_t cond;
int x = 0;
void * prva_nit(void *args);
void * druga_nit(void *args);

int main()
{
   pthread_t t1, t2;

   pthread_mutex_init(&lock, NULL);
   pthread_cond_init(&cond, NULL);

   pthread_create(&t1, NULL, (void *)&prva_nit, NULL);
   sleep(3);
   pthread_create(&t2, NULL, (void *)&druga_nit, NULL);

   pthread_join(t1, NULL);
   pthread_join(t2, NULL);

   pthread_mutex_destroy(&lock);
   pthread_cond_destroy(&cond);

   return 0;
}

void * prva_nit(void *args)
{
   pthread_mutex_lock(&lock);
   while(x < 100)
      pthread_cond_wait(&cond, &lock);
   pthread_mutex_unlock(&lock);
}

void * druga_nit(void *args)
{
   pthread_mutex_lock(&lock);
   while (x < 200)
   {
      x++;
      if (x == 100) pthread_cond_signal(&cond);
    }
   pthread_mutex_unlock(&lock);
}
```

Nit čeka na uslovnoj promenljivoj da uslov bude ispunjen.

Kada se ispuni uslov signalizira se blokiranoj niti koja je čekala taj uslov..

Sadržaj

- Semafori
- Sinhronizacija niti
- Mutex
- Uslovne promenljive
- POSIX semafori

POSIX semafor

Pojam

- POSIX semafori predstavljaju implementaciju semafora opšteg tipa.
- Mogu biti binarni i n-arni.
- POSIX semafori su uopšteniji mehanizam za sinhronizaciju procesa i niti.
- Koristićemo ih isključivo za sinhronizaciju niti.
- POSIX semafori nemaju vlasnika pa ne postoji nikakvo ograničenja u odnosu na to koja nit može koju operaciju da izvrši nad semaforom.
- Sve neophodne funkcije i strukture podataka su definisane u zaglavlju **<semaphore.h>.**
- Semafor se definiše kao promenljiva tipa **sem_t**.

POSIX semafori

Kreiranje semafora

```
#include <semaphore.h>
int sem_init ( sem_t * sem, int pshared, unsigned int value );
```

Semantika

- Sistemski poziv koji kreira novi POSIX semafor.
- Prvi argument je pokazivač na promenljivu tipa POSIX semafor koja je prethodno deklarisana.
- Drugi argument je uvek 0 (zato što POSIX semafore koristimo samo za sinhronizaciju niti)
- Treći argument predstavlja vrednost na koju se kreirani semafor inicijalizuje.

POSIX semafori

Brisanje semafora

```
#include <semaphore.h>
int sem_destroy ( sem_t * sem );
```

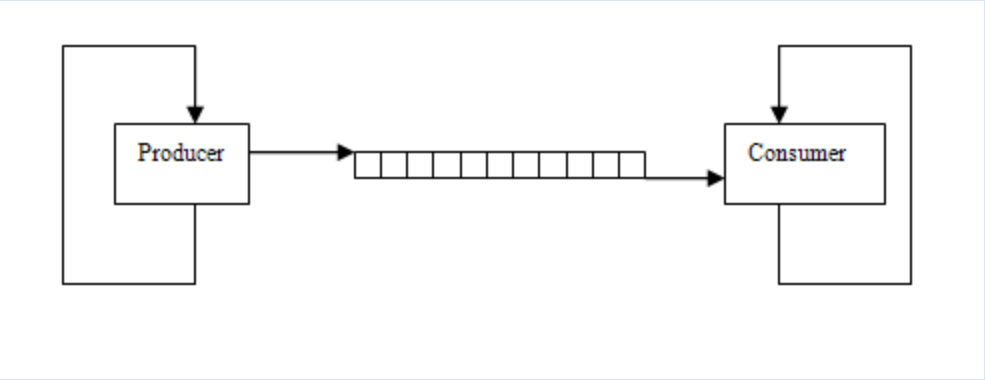
P i V funkcija semafora

```
#include <semaphore.h>
int sem_wait ( sem_t * sem );
int sem_post ( sem_t * sem );
```

POSIX semafori

Proizvođač - potrošač

- Klasičan problem sinhronizacije i međusobnog isključenja.
- Dve niti (ili dva procesa) dele zajednički kružni bafer fiksne veličine. Jedan od njih, proizvođač, smešta podatke u bafer a drugi, potrošač, uzima podatke iz bafera. Proizvošač i potrošač se izvršavaju paralelno (ili kvaziparalelno) i asinhrono.



POSIX semafori

Proizvođač - potrošač

- Problemi koje treba rešiti:
  - bafer pun – proizvošač nema slobodnih lokacija u koje bi upisao podatak. Treba ga uspavati sve dok potrošač ne oslobodi bar jednu lokaciju.
  - bafer prazan – ne postoji podataka koji potrošač može pročitati. Treba ga uspavati sve dok proizvođač ne proizvede bar jedan podatak.

Rešenje

- Rešenje se bazira na postojanju tri semafora:
  - full – koristi se za sinhronizaciju i pamti broj zauzetih pozicija u baferu (inicijalno je 0).
  - empty – koristi se za sinhronizaciju i pamti broj slobodnih pozicija u baferu (inicijalno je N)
  - mutex – koristi se za međusobno isključenje prilikom pristupanja baferu

POSIX semafori

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>

#define N 100

void *  producer(void * args);
void * consumer(void * args);
int buf[N];
sem_t mutex, empty, full;

int main()
{
   pthread_t proizvodjac, potrosac;

   sem_init(&mutex, 0, 1);
   sem_init(&full, 0, 0);
   sem_init(&empty, 0, N);

   pthread_create(&proizvodjac, NULL, (void *)&producer,
NULL);
   pthread_create(&potrosac, NULL, (void *)&consumer,
NULL);

   pthread_join(proizvodjac, NULL);
   pthread_join(potrosac, NULL);

   sem_destroy(&mutex);
   sem_destroy(&full);
   sem_destroy(&empty);

   return 0;
}
```

Kreiranje i inicijalizacija semafora.

Kreiranje i startovanje niti.

Čeka se da se niti završe.

Brisanje semafora.

POSIX semafori

```
void * producer(void * args)
{
   int i;

   for(i = 0; i < 10000; i++)
   {
      //generisanje podataka koji se upisuju u bafer

      sem_wait(&empty);

      sem_wait(&mutex);
      //upisivanje podataka u bafer
      sem_post(&mutex);

      sem_post(&full);
   }
}
```

Sinhronizacija.

Kritična sekcija.

Sinhronizacija.

```
void * consumer(void * args)
{
   int i;

   for(i = 0; i < 10000; i++)
   {
      sem_wait(&full);

      sem_wait(&mutex);
      //očitavanje podataka iz bafera
      sem_post(&mutex);

      sem_post(&empty);

      //korišćenje podataka

   }
}
```

Sinhronizacija.

Kritična sekcija.

Sinhronizacija.