PRODUCER-CONSUMER

-arhitektura racunara I operativni sistemi – projekat

Student:

Stefan Sedlarević 333/2020

Beograd, septembar 2022.

Sadržaj

Producer – consumer problem

Problem je uzet iz knjige “The Little book of Semaphores”.
U problemu postoje dva učesnika, proizvođači (producer) i korisnici (consumers). Srž ovog problema je nepraviti bafer, tj zajedničku memoriju proivođača i korisnika, u koju proizvođači ubacuju podatke, i iz koje ih korisnici uzimaju i obrađuju. Stoga, imamo tri problema koja moramo da rešimo:

1. Omogućiti pristup deljenoj memoriji
2. Provera da li je bafer pun
3. Provera da li je bafer prazan

Rešenje problema u kodu:

Pravimo bafer u koji će kasnije proizvođači da proizvode nasumičan broj, a iz kog će korisnici da ga uzimaju, i obrade. Definisaćemo bafer sa 10 mesta za upis podataka, i brojač podataka u baferu, koji je na početku jednak nuli, zato što je buffer prazan na početku.

```
int buffer[10];
int count = 0;
```

Takođe, definišemo dve niti, jedna koja će biti vezana za proizvođače, a druga za korisnike.

```
#define THREAD_NUM 2;
```

Main metoda:

U main metodi ćemo prvenstveno krenuti sa kreiranjem niti.

```
pthread_t th[THREAD_NUM];
int i;
for (i = 0; i < THREAD_NUM; i++) {
    if (i % 2 == 0) {
        if (pthread_create(&th[i], NULL, &producer, NULL) != 0) {
            perror("Failed to create thread");
        }
```

```
} else {
    if (pthread_create(&th[i], NULL, &consumer, NULL) != 0) {
        perror("Failed to create thread");
    }
}
}
```

Ovo smo uradili kroz if, da bi podelili na jednak broj proizvođača i korisnika.
Nakon toga ćemo spojiti niti sa glavnim programom.

```
for (i = 0; i < THREAD_NUM; i++) {
    if (pthread_join(th[i], NULL) != 0) {
        perror("Failed to join thread");
    }
}
```

Zato što će jedna nit da pristupi baferu dok je druga nit trenutno vrši operaciju nad baferom, moramo da iskoristimo mutex, kako bismo osigurali da samo jedna nit u trenutku ima pristup baferu. Tako da ćemo van main metode da definišemo mutex pthread_mutex_t mutexBuffer; u samoj main metodi ćemo inicijalizovati mutex, pre kreiranja niti, a osloboditi prostor nakon spajanja niti sa glavnim programom.

```
pthread_mutex_init(&mutexBuffer,NULL);
.
.
.
pthread_mutex_destroy(&mutexBuffer);
```

Ovim rešavamo prvi problem, a to je omogućavanje pristupa deljenoj memoriji, zato što ne želimo da dozvolimo da obe niti vrše operacije nad baferom istovremeno.

Da bismo osigurali da proizvođač ne može da ubaci podatke u pun bafer, kao i da korisnik ne može da uzima podatke iz praznog bafera, van main funkcije ćemo definisati 2 semafora, prvi koji se odnosi na broj praznih mesta u baferu, a drugi za broj punih mesta u baferu; sem_t semEmpty; sem_t semFull;.
Inicijalizovaćemo ih u main funkciji, odmah nakon inicijalizacije mutexa, i osloboditi tu memoriju pre oslobađanje memorije na kojoj je inicijalizovan mutex.

```
sem_init(&semEmpty, 0, 10);
sem_init(&semFull, 0, 0);
.
```

.
.
sem_destroy(&semEmpty);
sem_destroy(&semFull);

Poćetna vrednost semafora semEmpty je 10, zato što na početku imamo 10 slobodnih mesta, a početna vrednost semFull je 0, zato što nemamo zauzeta mesta na samom početku.

Funkcija void* producer(void* args)

U producer-consumer problemu, proizvođač, tj. producer, stvara neki podatak, i skladišti ga na bafer. To samo stvaranje podataka ćemo simulirati stvaranjem nasumičnog broja. I smatraćemo da je potrebno jedna sekunda za stvaranje datog podatka.

int x = rand() % 100;
sleep(1);

Takođe ćemo u mainu napisati srand(time(null)); na samom početku.
Sledeći zadatak koju obavlja proizvođač, nakon proizvodnje jeste dodavanje tog podatka na bafer. Kako smo došli do operacije koja pristupa baferu, mi ćemo tu samu operaciju obgrliti mutexom, zato što smo rekli da u bilo kom trenutku ne mogu dve operacije da imaju omogućen pristup baferu. To izgleda ovako:

pthread_mutex_lock(&mutexBuffer);
.
.
.
pthread_mutex_unlock(&mutexBuffer);

Takođe, kako bismo rešili dva velika problema, ubacivanje podataka kada je bafer pun, i izbacivanja kada je bafer prazan, mi ćemo se koristiti gorepomenutim semaforima. U ovoj funkciji, prvo ćemo naterati semafor semEmpty da se dekrementuje, i da čeka ukoliko je on jednak nuli, a na kraju same operacije želimo da se semafor semFull inkrementuje, zato što je proizvođač dodao podatak na slobodno mesto, tako da ono postaje zauzeto. To izgleda ovako:

sem_wait(&semEmpty);
.

.
.
sem_post(&semFull);

Sama operacija, koja je obrgljena semaforima i mutexom izgleda ovako:

```
while (1) {
int x = rand() % 100;
sleep(1);
sem_wait(&semEmpty);
pthread_mutex_lock(&mutexBuffer);
buffer[count] = x;
count++;
pthread_mutex_unlock(&mutexBuffer);
sem_post(&semFull);
}
```

Znači dodaje se podatak u buffer, i povećava se broj podataka u bufferu.

Funkcija void* consumer(void* args)

U producer-consumer problemu, korisnik, tj. consumer uzima podatak iz bafera, i obrađuje ga. Obradu podatka ćemo simulirati ispisom podatka, tj. funkcijom printf. Takođe, smatramo da je jedna sekunda potrebna za samu obradu podatka. Pošto korisnik preuzima podatak iz bafera, pristupa mu, mi tu operaciju moramo obezbediti mutexom, kako ne bi obe operacije pristupile baferu u nekom trenutku. To izgleda ovako:

```
pthread_mutex_lock(&mutexBuffer);
.
.
.
pthread_mutex_unlock(&mutexBuffer);
```

Takođe, pošto ne želimo da korisnik uzima podatak iz praznog bafera, želimo da semaforima rešimo taj problem, kao i u prethodnoj funkciji, tako da ćemo dekrementovati pre svega semafor semFull nakon kog će nit da čeka izvršavanje ako je semafor jednak 0, a nakon izvršavanja operacije će se inkrementovati semafor semEmpty, zato što se oslobađa broj slobodnih mesta. To izgleda ovako:

```
sem_wait(&semFull);
.
.
.
sem_post(&semEmpty);
```

Operacija uzmanja podatka iz bafera koja je obrgljena mutexom i semaforima, i operacija obrade podatka koja se naravno, izvršava nakon prošle, izgleda ovako:

```
while (1) {
int y;
sem_wait(&semFull);
pthread_mutex_lock(&mutexBuffer);
y = buffer[count - 1];
count--;
pthread_mutex_unlock(&mutexBuffer);
sem_post(&semEmpty);
printf("Got %d\n", y);
sleep(1);
}
```

Testiranje programa

Uzeto 87
Uzeto 68
Uzeto 17
Uzeto 96
Uzeto 86
Uzeto 2
Uzeto 13
Uzeto 68
Uzeto 4
Uzeto 75
Uzeto 54

Literatura

1. “The Little Book of Semaphores” - Allen B. Downey
2. http://greenteapress.com/semaphores/LittleBookOfSemaphores.pdf
3. http://www.mmklab.fon.bg.ac.rs/wp-content/uploads/2016/01/CSkripta.pdf