Operativni sistemi

Sinhronizacija procesa i niti

(Laboratorijska vežba II-2)

Zadatak 1

Korišćenjem programskog jezika C napisati Linux program koji na standardnom izlazu ispisuje rečenicu "Ovo je test za semafore." Pri čemu svaku reč u rečenici ispisuje posebna nit. Niti sinhronizovati korišćenjem semafora.

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

#define N 5

/* Niz koji sadrzi reci recenice */
char reci[N][20] = {"Ovo ", "je ", "test ", "za ", "semaphore."};
/* Niz od 4 semafora. */
sem_t sems[N-1];

/* Funkcija niti. */
void* stampanje(void* arg)
{
        /* Ulazni argument je struktura argument */
        int p = *((int *)arg);
        /* Samo prva nit odmah stampa svoju rec, ostale moraju da sacekaju
prethodnu */
        if (p > 0)
                sem_wait(&sems[p-1]);

        printf("%s", reci[p]);

        if (p < 4)
                sem_post(&sems[p]);
        sleep(3);
}

int main()
{
        int i;
/* Niz ulaznih argumenata niti */
        int args[N];
        /* Niz identifikatora niti */
        pthread_t niti[N];
        /* Inicijalizacija semafora. */
        for(i=0 ; i<N-1 ; i++)
                sem_init(&sems[i], 0, 0);

        /* Kreiramo niti */
        for(i=0 ; i<N ; i++)
        {
                args[i] = i;
                pthread_create(&niti[i],         NULL,         (void*)stampanje,
(void*)&args[i]);
        }
        /* Main funkcija ceka da se sve niti zavrse */
```

```
        for(i=0 ; i<N ; i++)
                pthread_join(niti[i], NULL);

        return 0;
}
```

Zadatak 2

Korišćenjem programskog jezika C napisati Linux progam koji sadrži bafer u koji se mogu smestiti dva integer broja. Zasebna nit periodično generiše dva broja (u opsegu od 1 do 10) i upisuje ih u ovaj bafer. Kada se u baferu nađe novi par brojeva glavna nit (main funkcija) treba da odredi zbir ta dva broja i odštampa ga na standardnom izlazu. Vremenski razmak između dva uzastopna upisa u bafer je slučajna vrednost između 0 i 5 sekundi.

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

#define TRUE 1
#define FALSE 0

/* Deljeni bafer. */
int buffer[2];

/* Mutex i uslovna promenljiva. */
pthread_mutex_t mutex;
pthread_cond_t condVarFull;
pthread_cond_t condVarEmpty;
int bufferFull = FALSE;

void* threadFunc(void* arg)
{
        /* Pauza izmedju dva upisa u bafer */
        long int rnd;
        float normalisedRnd;
        int sleepTime;
        int i;
        int bufferPointer;

        bufferPointer=0;

        for(i=1 ; i<=10 ; i++)
        {
                rnd = random();
                normalisedRnd = (float)rnd/(float)RAND_MAX;
                sleepTime = (int)(normalisedRnd * 5);
                pthread_mutex_lock(&mutex);
                while (bufferFull)
                        pthread_cond_wait(&condVarEmpty, &mutex);
                buffer[bufferPointer] = i;
                printf("Thread    u    buffer[%d]    upisao    vrednost    %d\n",
bufferPointer, i);
                bufferPointer = (bufferPointer + 1) % 2;
                if (bufferPointer == 0)
                {
                        bufferFull = TRUE;
                }
                pthread_cond_signal(&condVarFull);
                pthread_mutex_unlock(&mutex);
```

```
                sleep(sleepTime);
        }
}

int main()
{
        pthread_t threadID;
        int i;

        pthread_mutex_init(&mutex, NULL);
        pthread_cond_init(&condVarEmpty, NULL);
        pthread_cond_init(&condVarFull, NULL);
        /* Kreira nit koja generise brojeve i upisuje ih u bafer. */
        pthread_create(&threadID, NULL, threadFunc, NULL);

        for(i=0 ; i<5 ; i++)
        {
                pthread_mutex_lock(&mutex);
                /* Nit ceka dok se bafer ne napuni sa dva broja. */
                while(!bufferFull)
                        pthread_cond_wait(&condVarFull, &mutex); /* Ceka signal
od niti. */
                /* Stampa zbir brojeva iz bafera. */
                printf("Zbir brojeva iz bafera %d + %d = %d\n", buffer[0],
buffer[1], buffer[0]+buffer[1]);
                bufferFull = FALSE;
                pthread_cond_signal(&condVarEmpty);
                pthread_mutex_unlock(&mutex);
        }

        /* Brise uslovnu promenljivu i mutex. */
        pthread_mutex_destroy(&mutex);
        pthread_cond_destroy(&condVarFull);
        pthread_cond_destroy(&condVarEmpty);
        return 0;
}
```

Zadatak 3

Korišćenjem programskog jezika C kreirati višenitni Linux proces. U okviru procesa se izvršavaju tri niti (N1, N2 i N3) koje pristupaju deljivom resursu. Deljivi rersurs je predstavljen celobrojnim nizom od 20 elemenata. Niti pristupaju deljivom resursu na sledeći način:

- Nit N1, na svake 2 sekunde, upisuje slučajnu vrednost na slučajnu poziciju u prvoj polovini niza (pozicije 0 do 9).
- Nit N2, na svake 4 sekunde, upisuje slučajnu vrednost na slučajnu poziciju u drugoj polovini niza (pozicije 10 do 19).
- Nit N3, na svakih 8 sekundi, upisuje slučajnu vrednost na slučajnu poziciju u nizu (pozicije 0 do 19).

Korišćenjem nekog od poznatih mehanizama za sinhronizaciju i međusobno isključenje niti, obezbediti da nit N3 može da pristupi deljivom resursu samo ako u datom trenutku deljivom resursu ne pristupa ni nit N1 ni nit N2.

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

```
#define N 20

/*Deljivi resurs koji se definise kao globalna promenljiva*/
int niz[N];

/*Semafori za medjusobno iskljucenje prilikom priatupanja deljivom resursu*/
pthread_mutex_t mutex1, mutex2;


void * nit1(void * arg)
{
   int pos;

   while(1)
   {
         /*Pristupanje deljivom resursu u KS*/
         pthread_mutex_lock(&mutex1);
         /*Vrednost se upisuje u prvu polovinu niza tj. pozicije 0-9*/
         pos = rand() % 10;
         niz[pos] = rand() % 100;
         printf("Prva nit je na poziciju %d upisala vrednost %d\n", pos,
niz[pos]);
         pthread_mutex_unlock(&mutex1);

         /*Pauziramo 2s*/
         sleep(2);
   }
}

void * nit2(void * arg)
{
   int pos;

   while(1)
   {

         /*Pristupanje deljivom resursu u KS*/
         pthread_mutex_lock(&mutex2);
         /*Vrednost se upiosuje u drugu polovinu niza tj. pozicije 10-19;*/
         pos = rand() % 10 + 10;
         niz[pos] = rand() % 100;
         printf("Druga nit je na poziciju %d upisala vrednost %d\n", pos,
niz[pos]);
         pthread_mutex_unlock(&mutex2);

         /*Pauziramo 4s*/
         sleep(4);
   }

}

void * nit3(void * arg)
{
   int pos;

   while(1)
   {

         /*Pristupanje deljivom resursu u moguce samo ako nit1 i nit2 nisu u
```

```
svojim KS.Zbog toga je potrebno zakljucati oba mutex*/
         pthread_mutex_lock(&mutex1);
         pthread_mutex_lock(&mutex2);

         /*Vrednost se upisuje u citac niz tj. pozicije 0-19;*/
         pos = rand() % 20;
         niz[pos] = rand() % 100;
         printf("Treca nit je na poziciju %d upisala vrednost %d\n", pos,
niz[pos]);
         pthread_mutex_unlock(&mutex2);
         pthread_mutex_unlock(&mutex1);

         /*Pauziramo 8s*/
         sleep(8);
   }

}

int main()
{
   pthread_t n1, n2, n3;

   /*Kreiranje mutex-a*/
   pthread_mutex_init(&mutex1, NULL);
   pthread_mutex_init(&mutex2, NULL);

   /*Kreiranej nit*/
   pthread_create(&n1, NULL, (void *)nit1, (void *)NULL);
   pthread_create(&n2, NULL, (void *)nit2, (void *)NULL);
   pthread_create(&n3, NULL, (void *)nit3, (void *)NULL);

   /*Ceka se da se niti n1, n2, n3 zavrse*/
   pthread_join(n1, NULL);
   pthread_join(n2, NULL);
   pthread_join(n3, NULL);

   /*Unistavaju se mutex-i*/
   pthread_mutex_destroy(&mutex1);
   pthread_mutex_destroy(&mutex2);

}
```

Zadatak 4

Korišćenjem programskog jezika C napisati Linux program koji simulira komunikaciju između niti korišćenjem celobrojnih bafera. Nit N0 generiše celobrojne podatke i upisuje ih u bafere B1, B2 i B3. Prilikom upisivanja podataka, nit N0, najpre proverava da li u baferu B3 ima slobodnih pozicija i u tom slučaju novi podatak upisuje u bafer B3. Ako je bafer B3 pun, proverava da li postoje slobodne pozicije u baferu B1. Ako postoje, proizvođač N0 generisani podataka upisuje u bafer B1. Ukoliko je i bafer B1 pun novi podatak se upisuje u bafer B2 ukoliko on poseduje slobodne pozicije. Ukoliko ni u jednom baferu nema slobodnih pozicija, nit N0 čeka, dok se u nekom od bafera ne oslobodi pozicija za upis novog elementa. Niti N1, N2, N3 čitaju podatke iz bafera B1, B2, B3 i prikazje ih na standardnom izlazu. Za sinhronizaciju između niti iskoristiti POSIX semafore.

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

```
/*Deklaracije globalnih promenljivih koje su vidljive u svim funkcijama pa i u
funkcijama niti*/
/*Dimenzije bafera B1, B2, B3*/
#define N1 10
#define N2 5
#define N3 20

/*Celobrojni baferi B1, B2 i B3*/
int B1[N1], B2[N2], B3[N3];

/*Brojaci slobodnih pozicija u baferima B1, B2 i B3*/
int empty1 = N1;
int empty2 = N2;
int empty3 = N3;

/*Indeksi pozicija za citanje i pisanje u bafer. Baferi B1, B2 i B3 se ponasaju
kao FIFO baferi odnosno mora da postoji glava i rep*/
int head1 = 0;
int tail1 = 0;
int head2 = 0;
int tail2 = 0;
int head3 = 0;
int tail3 = 0;

/*Semafori za medjusobno iskljucenje*/
sem_t mutex_B1, mutex_B2, mutex_B3, mutex_empty;



void * nit0(void * arg)
{
   int data;

   while(1)
   {
         /*Pravi pauzu pre generisanja sledeceg podatka*/
         sleep(rand() % 3);

         /*Nit N0 proverava da li bar u jednom baferu postoji prazna pozicija*/
         /*Ukoliko ne postoji nit N0 se blokira dok ne dobija signal da se
oslobodila bar jedna pozicija*/
         sem_wait(&mutex_empty);

         printf("Generisanje podatka\n");

         /*Pristup svakom baferu ide u zasebnoj KS*/


         /*Podatak pokusava najpre da upise u bafer B3*/
         sem_wait(&mutex_B3);
         if (empty3 > 0)
         {
               B3[tail3] = rand() % 1000;
               tail3 = (tail3 + 1) % N3;
               empty3--;
               sem_post(&mutex_B3);
               continue;
```

```
        }
        else
              sem_post(&mutex_B3);

        /*Zatim pokusava da upise u B1 ako ima slobodnih pozicija*/
        sem_wait(&mutex_B1);
        if (empty1 > 0)
        {
              B1[tail1] = rand() % 1000;
              tail1 = (tail1 + 1) % N1;
              empty1--;
              sem_post(&mutex_B1);
              continue;

        }
        else
              sem_post(&mutex_B1);

        /*Na kraju upisuje u B2*/
        sem_wait(&mutex_B2);
        if (empty2 > 0)
        {
              B2[tail2] = rand() % 1000;
              tail2 = (tail2 + 1) % N2;
              empty2--;
              sem_post(&mutex_B2);
        }
        sem_post(&mutex_B2);

   }
}

void * nit1(void * arg)
{
   while(1)
   {
        /*Baferu se pristupa u KS*/
        sem_wait(&mutex_B1);
        /*Podatak se cita samo ukoliko bafer nije prazan*/
        if (empty1 < N1)
        {
              printf("Nit N1 je ocitala podatak %d\n", B1[head1]);;
              head1 = (head1 + 1) % N1;
              empty1++;

              /*Obavestava N0 da s epojavila nova slobodna pozicija*/
              sem_post(&mutex_empty);
        }
        sem_post(&mutex_B1);

        /*Pravi pauzu pre citanja sledeceg podatka*/
        sleep(rand() % 5);

   }
}

void * nit2(void * arg)
{
   while(1)
```

```
   {
         /*Baferu se pristupa u KS*/
         sem_wait(&mutex_B2);
         /*Podatak se cita samo ukoliko bafer nije prazan*/
         if (empty2 < N2)
         {
               printf("Nit N2 je ocitala podatak %d\n", B2[head2]);;
               head2 = (head2 + 1) % N2;
               empty2++;

               /*Obavestava N0 da s epojavila nova slobodna pozicija*/
               sem_post(&mutex_empty);
         }

         sem_post(&mutex_B2);

         /*Pravi pauzu pre citanja sledeceg podatka*/
         sleep(rand() % 7);

   }
}


void * nit3(void * arg)
{
   while(1)
   {
         /*Baferu se pristupa u KS*/
         sem_wait(&mutex_B3);
         /*Podatak se cita samo ukoliko bafer nije prazan*/
         if (empty3 < N3)
         {
               printf("Nit N3 je ocitala podatak %d\n", B3[head3]);;
               head3 = (head3 + 1) % N3;
               empty3++;

               /*Obavestava N0 da s epojavila nova slobodna pozicija*/
               sem_post(&mutex_empty);
         }

         sem post(&mutex B3);

         /*Pravi pauzu pre citanja sledeceg podatka*/
         sleep(rand() % 9);

   }
}


int main()
{
   int i;
   /*Identifikatori nit, */
   pthread_t niti[4];

   srand(3232234);

   /*Kreiranje i inicijalizacija semafora*/
   sem_init(&mutex_B1, 0, 1);
```

```
   sem_init(&mutex_B2, 0, 1);
   sem_init(&mutex_B3, 0, 1);

   /*semafor mutex_empty pamti ukupan broj slobodnih pozicija u sva tri bafera*/

   sem_init(&mutex_empty, 0, N1 + N2 + N3);

   /*Kreiranje niti N0, N1, N3 i N3*/
   pthread_create(&niti[1], NULL, (void *)nit1, (void *)NULL);
   pthread_create(&niti[2], NULL, (void *)nit2, (void *)NULL);
   pthread_create(&niti[3], NULL, (void *)nit3, (void *)NULL);
   pthread_create(&niti[0], NULL, (void *)nit0, (void *)NULL);

   /*Ceka se da se niti zavrse*/
   for(i = 0; i < 4; i++)
         pthread_join(niti[i], NULL);

   /*Brisanje semafora*/
   sem_destroy(&mutex_B1);
   sem_destroy(&mutex_B2);
   sem_destroy(&mutex_B3);
   sem_destroy(&mutex_empty);
}
```

Zadatak 5

Pristup bazi podataka obavlja se radi upisa i čitanja od strane više procesa. U jednom trenutku može postojati više procesa koji čitaju sadržaj iz baze podataka procedurom read_database(), ali ako jedan proces upisuje sadržaj u bazu podataka procedurom write_database(), nijednom drugom procesu nije dozvoljen pristup bazi podataka radi upisa i čitanja. Prednost imaju procesi koji čitaju sadržaj, tako da dok god ima procesa koji čitaju iz baze podataka, proces koji treba da upisuje podatke mora da čeka. Korišćenjem programskog jezika C napisati Linux program koji korišćenjem procesa i poznatih IPC mehanizama simulira prethodno opisani algoritam. (Sinhronizacion problem **Čitaoci – pisci**).

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

int A = 0; /*Globalna promenljiva koja prestavlja bazu podataka*/
int N = 0; /*Globalna promenljiva koja predstavlja brojac citaoca koji u datom
trenutku koriste bazu podataka*/


sem_t mutex_DB, mutex_N; /*Semafori za medjusobno iskljucenje*/

void read_database(int i)
{
   /*Operacija citanja iz baze se simulira stampanjem globalne promenljive A*/
   printf("Nit %d je procitala vrednost %d\n", i, A);
   sleep(rand()%3);
}

void write_database()
{
   /*Operacija upisa u bazu se simulira izmenom vrednosti globalne promenljive
A*/
   printf("Pisac menja vrednost u bazi podataka.\n");
```

```
   A = A + rand()%10;
   sleep(rand()%5);
}

void * pisac(void * arg)
{
   while(1)
   {
         /*Pristup bazi je smesten u KS*/
         sem_wait(&mutex_DB);
         write_database();
         sem_post(&mutex_DB);

         /*Pravi pauzu pre sledeceg pristupa bazi podataka*/
         sleep(rand() % 10);
   }
}

void * citalac(void * arg)
{
   int i;
   i  = *((int *)arg);

   while(1)
   {
         /*Promenljivoj N se pristupa u KS*/
         sem_wait(&mutex_N);
         /*Povecava se broj citalaca*/
         N++;
         /*Prvi citalac zakljucava bazu podataka kako bi sprecio pisca da joj
pristupa*/
         if (N == 1)
               sem_wait(&mutex_DB);
         sem_post(&mutex_N);

         read database(i);

         /*Promenljivoj N se pristupa u KS*/
         sem_wait(&mutex_N);
         /*Smanjuje se broj citalaca*/
         N--;
         /*Poslednji citalac otkljucava bazu podataka*/
         if (N==0)
               sem_post(&mutex_DB);
         sem_post(&mutex_N);

         /*Pravi pauzu pre sledeceg pristupa bazi podataka */
         sleep(rand() % 7);

   }
}

int main()
{
   int i;
   int red_br[4];
   /*Identifikatori nit, 4 citaoca + 1 pisac*/
   pthread_t niti[5];
```

```
   srand(3232234);

   /*Kreiranje i inicijalizacija semafora*/
   sem_init(&mutex_N, 0, 1);
   sem_init(&mutex_DB, 0, 1);

   /*Kreiranje citalaca i pisca*/
   for (i = 0; i < 4; i++)
   {     red_br[i] = i;
         pthread_create(&niti[i], NULL, (void *)citalac, (void *)&red_br[i]);
   }

   pthread_create(&niti[4], NULL, (void *)pisac, (void *)NULL);

   /*Ceka se da se niti zavrse*/
   for(i = 0; i < 5; i++)
         pthread_join(niti[i], NULL);

   /*Brisanje semafora*/
   sem_destroy(&mutex_N);
   sem_destroy(&mutex_DB);
}
```

Zadatak 6

Korišćenjem programskog jezika C napisati Linux progam koji se deli u dva procesa. Jedan proces čita liniju po liniju datoteku prva.txt, a drugi datoteku druga.txt. Ova dva procesa upisuju naizmenično pročitane linije u tekstualnu datoteku zbir.txt tako da su sve neparne linije iz datoteke prva.txt, a parne iz datoteke druga.txt.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/sem.h>
#include <sys/ipc.h>

#define MUTEX_KEY 10101
#define PROC_A_KEY 10102
#define PROC_B_KEY 10103

#define DUZINA 80

union semun{
        int val;
        struct semid_ds *buf;
        ushort *array;
};


int main()
{
        int mutexid, procaid, procbid, retCode;
        union semun semopts;
        char linija[DUZINA];
        FILE* fileA;
        FILE* fileB;
        FILE* fileC;
        struct sembuf sem_lock = {0, -1, 0};
```

```
        struct sembuf sem_unlock = {0, 1, 0};

        /* Kreiranje semafora. */
        mutexid = semget((key_t)MUTEX_KEY, 1, 0666 | IPC_CREAT);
        procaid = semget((key_t)PROC_A_KEY, 1, 0666 | IPC_CREAT);
        procbid = semget((key_t)PROC_B_KEY, 1, 0666 | IPC_CREAT);
        /* Inicijalizacija semafora. */
        semopts.val = 1;
        semctl(mutexid, 0, SETVAL, semopts);
        semopts.val = 1;
        semctl(procaid, 0, SETVAL, semopts);
        semopts.val = 0;
        semctl(procbid, 0, SETVAL, semopts);

        if (fork() == 0)
        {
                mutexid = semget((key_t)MUTEX_KEY, 1, 0666);
                procaid = semget((key_t)PROC_A_KEY, 1, 0666);
                procbid = semget((key_t)PROC_B_KEY, 1, 0666);
                fileA = fopen("prva.txt", "r");
                while (!feof(fileA))
                {
                        fgets(linija, DUZINA, fileA);
                        semop(procaid, &sem_lock, 1);
                        semop(mutexid, &sem_lock, 1);
                        fileC = fopen("zbir.txt", "a");
                        fprintf(fileC, "%s", linija);
                        fclose(fileC);
                        semop(mutexid, &sem_unlock, 1);
                        semop(procbid, &sem_unlock, 1);
                }
                fclose(fileA);
                exit(0);
        }
        else
        {
                mutexid = semget((key_t)MUTEX_KEY, 1, 0666);
                procaid = semget((key_t)PROC_A_KEY, 1, 0666);
                procbid = semget((key_t)PROC_B_KEY, 1, 0666);
                fileB = fopen("druga.txt", "r");
                while (!feof(fileB))
                {
                        fgets(linija, DUZINA, fileB);
                        semop(procbid, &sem_lock, 1);
                        semop(mutexid, &sem_lock, 1);
                        fileC = fopen("zbir.txt", "a");
                        fprintf(fileC, "%s", linija);
                        fclose(fileC);
                        semop(mutexid, &sem_unlock, 1);
                        semop(procaid, &sem_unlock, 1);
                }
                fclose(fileB);

                wait(&retCode);
                semctl(mutexid, 0, IPC_RMID, 0);
                semctl(procaid, 0, IPC_RMID, 0);
                semctl(procbid, 0, IPC_RMID, 0);
        }
}
```