
# Praktikum iz operativnih sistema

## Vežbe 6

# Oblasti

- POSIX
  - rad sa nitima
  - sinhronizacija pomoću semafora

# POSIX standard

- Ranije je svaki proizvođač hardvera imao svoj API
- Problem: Portabilnost
- Rešenje: POSIX standard (Portable Operating System Interface)
- Nama će od interesa biti:
  - pthreads.h
    - pthread_create, pthread_join, pthread_exit
  - semaphore.h
    - sem_init, sem_wait, sem_post, sem_destroy
  - unistd.h
    - sleep

# pthread_create

```
int pthread_create(pthread_t * thread,
                   const pthread_attr_t * attr,
                   void * (*start_routine)(void *),
                   void *arg);
```

- `thread` – pokazivač na `pthread_t` gde se smešta ID kreirane niti
- `attr` – pokazivač na strukturu koja definiše atribute niti (prioritet, adresa steka, itd.), za podrazumevane atribute prosleđuje se `NULL(0)`
- `start_routine` – funkcija nad kojom se kreira nit (ista namena kao namena `run` metode u OO jezicima)
- `arg` – parametar koji se proleđuje `start_routine`
- povratna vrednost je nula ukoliko je uspešno izvršena funkcija, u suprotnom se vraća kod greške

# pthread_join

```
int pthread_join(pthread_t th, void **thread_return);
```

- `th` – ID niti koju pozivaoc čeka da završi
- `thread_return` – pokazivač na lokaciju na kojoj će se naći *exit status* niti čiji završetak se čeka
- povratna vrednost je nula ukoliko je uspešno izvršena funkcija, u suprotnom se vraća kod greške

# pthread_exit

```
void pthread_exit(void *retval);
```

- `retval` – pokazivač na lokaciju na kojoj se čuva exit stutus, pokazivač mora ukazivati na globalnu promenljivu kako bi *exit status* bio vidljiv svima koji su pozvali `pthread_join` za posmatranu nit
- Funkcija `exit()` prekida kompletan proces, dakle ukoliko neka nit pozove `exit()` operativni sistem prekidan proces i sve niti tog procesa koje su pokrenute, čak i one koje nisu završile izvršavanje

# sem_init

```
int sem_init(sem_t *sem, int pshared, unsigned int value);
```

- `sem` – pokazivač na `sem_t` gde se smešta ID kreiranog semafora
- `pshared` – određuje da li semafor koriste niti u okviru jednog procesa (vrednost nula) ili se semafor koristi između više procesa (vrednost različita od nule)
- `value` – početna vrednost semafora
- povratna vrednost je nula ukoliko je uspešno izvršena funkcija, u suprotnom se vraća -1

# sem_wait - wait
# sem_post - signal

```
int sem_wait(sem_t *sem);
int sem_post(sem_t *sem);
```

- `sem` – pokazivač na ID semafora nad kojim se poziva operacija
- povratna vrednost je nula ukoliko je uspešno izvršena funkcija, u suprotnom se vraća -1

# sem_destroy

```
sem_destroy(sem_t *sem);
```

- `sem` – pokazivač na ID semafora koji se uništava

# sleep

```
unsigned sleep(unsigned seconds);
```

- `seconds` – nit koja pozove ovu funkciju uspavljuje se na `seconds` sekundi
- povratna vrednost je nula ako je isteklo vreme spavanja, a ako se nit ranije probudi onda vrednost koja nije nula (nekim signalom je moguće probuditi ranije nit, izlazi iz okvira ovog kursa)

# Zadatak 1

- Korišćenjem POSIX funkcija na jeziku C napisati globalne deklaracije i inicijalizacije, kao i kod tela dve uporedne niti A i B koje ciklično rade sledeće:
- A: upisuje vrednost u deljene promenljive x i y, a zatim čeka da proces B upiše zbir x i y u promenljivu z čiju vrednost ispisuje na standardni izlaz;
- B: čeka da nit A upiše vrednosti u deljene promenljive x i y, zatim ove dve vrednosti sabira i zbir upisuje u deljenu promenljivu z.

# Rešenje

Opcija `-pthread` je obavezna kada se prevodi kod koji koristi `pthread` funkcije

```c
// zad1.c
// prevodjenje: gcc -o prog zad1.c -pthread
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <time.h>
// otkomentarisati za sleep ili usleep
//#include <unistd.h>

// deljeni podaci za niti A i B
int x, y, z;
sem_t semXY;
sem_t semZ;

// potpis funkcije niti je uvek void* (f*)(void*)
void* a(void* param){
     int i = 0;
     srand(time(0));
     while(1){
          x = rand()%10;
          y = rand()%10;
          // signal semXY
          sem_post(&semXY);
          // wait semZ
          sem_wait(&semZ);
          // uspavljivanje tekuce niti, u sekundama
          // sleep(1);
          printf("%d + %d = %d\n", x, y, z);
          if(++i == 10){
              break;
          }
     }
     printf("a() - finished\n");
     return 0;
}
```

```c
void* b(void* param){
    int i = 0;
    while(1){
        // wait semXY
        sem_wait(&semXY);
        z = x + y;
        // semZ signal
        sem_post(&semZ);
        if(++i == 10){
            break;
        }
    }
    printf("b() - finished\n");
    return 0;
}
int main(int argc, char* argv[]){
    pthread_t nitA, nitB;
    // pravljenje semafora, inicijalizacija na 0
    if(sem_init(&semXY, 0, 0) || sem_init(&semZ, 0, 0)){
        perror(NULL); exit(1);
    }
    // pravljenje niti
    if(pthread_create(&nitA, 0, a, 0) || pthread_create(&nitB, 0, b, 0)){
        perror(NULL); exit(1);
    }
    // main nit ceka da niti A i B zavrse
    // (u slucaju da vise niti radi join za istu nit, ponasanje je nedefinisano)
    pthread_join(nitA, 0);
    pthread_join(nitB, 0);
    // obavezno unistavanje semafora
    // (u slucaju da je neka nit blokirana na semaforu, ponasanje je nedefinisano)
    sem_destroy(&semXY);
    sem_destroy(&semZ);
    printf("main() - finished\n");
    return 0;
}
```

# Rešenje – verzija 2

```c
// zad1_verzija2.c
// prevodjenje: gcc -o prog zad1.c -pthread
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <time.h>
// otkomentarisati za sleep ili usleep
//#include <unistd.h>

// definisana je nova struktura podataka
typedef struct shared_data{
        int x, y, z;
        sem_t semXY;
        sem_t semZ;
}  shared_data;

// potpis funkcije niti je uvek void* (f*)(void*)
void* a(void* param){
      shared_data* data = (shared_data*)param;
        int i = 0;
        srand(time(0));
        while(1){
              data->x = rand()%10;
              data->y = rand()%10;
              // signal semXY
              sem_post(&(data->semXY));
              // wait semZ
              sem_wait(&(data->semZ));
              // uspavljivanje tekuce niti, u sekundama
              // sleep(1);
              printf("%d + %d = %d\n", data->x, data->y,
      data->z);
              if(++i == 10){
                      break;
              }
        }
        printf("a() - finished\n");
        return 0;
}
```

```c
void* b(void* param){
      shared_data* data = (shared_data*)param;
      int i = 0;
      while(1){
          // wait semXY
          sem_wait(&(data->semXY));
          data->z = data->x + data->y;
          // semZ signal
          sem_post(&(data->semZ));
          if(++i == 10){
             break;
          }
       }
       printf("b() - finished\n");
       return 0;
}
int main(int argc, char* argv[]){
       shared_data data;
       pthread_t nitA, nitB;
       // pravljenje semafora, inicijalizacija na 0
       if(sem_init(&data.semXY, 0, 0) || sem_init(&data.semZ, 0, 0)){
          perror(NULL); exit(1);
       }
       // pravljenje niti
       if(pthread_create(&nitA, 0, a, &data) || pthread_create(&nitB, 0, b, &data)){
          perror(NULL); exit(1);
       }
       // main nit ceka da niti A i B zavrse
       // (u slucaju da vise niti radi join za istu nit, ponasanje ne nedefinisano)
       pthread_join(nitA, 0);
       pthread_join(nitA, 0);
       // unistavanje semafora
       // (u slucaju da je neka nit blokirana na semaforu, ponasanje je nedefinisano)
       sem_destroy(&(data.semXY));
       sem_destroy(&(data.semZ));
       printf("main() - finished\n");
       return 0;
}
```

# Zadatak 2

- Nit `P` treba da sačeka da sve tri niti `X`, `Y` i `Z` ispune neki svoj uslov, u bilo kom redosledu.
- Napisati kod niti `P` i niti `X`, `Y` i `Z` (koji je približno isti), uz potrebne deklaracije, koji obezbeđuju ovu uslovnu sinhronizaciju pomoću jednog standardnog brojačkog semafora.

# Rešenje

- Rešenje je dato u `zad2.c`
  - Niti `X`, `Y` i `Z` postavljaju redom polja `a`, `b` i `c` deljenog podatka tipa `shared_data`
  - Nit `P` čeka da dobije signal od niti `X`, `Y` i `Z` (u bilo kom redosledu) i sabira vrednosti polja `a`, `b` i `c`
- Još jedno rešenje je dato u `zad2_verzija2.c`
  - U ovom rešenju se niti `X`, `Y` i `Z` izvršavaju nad istom funkcijom

# Zadatak 3

- Dve niti $X$ i $Y$ "proizvode" cele brojeve uporedo, nezavisnim i promenljivim brzinama.
- Nit $Z$ uzima po dva proizvedena broja, bez obzira koja nit je proizvela te brojeve, i njihov zbir ispisuje na standardni izlaz.
- Važno je obezbediti da nit $Z$ uvek uzima samo "sveže" proizvedene brojeve, tj. nikada ne uzme više puta isti proizvedeni broj.
- Nije važno koja nit je proizvela brojeve – niti $X$ i $Y$ ne treba nepotrebno sinhronizovati niti uslovljavati njihovu naizmeničnost: ako je npr. nit $X$ spremna da proizvede još jedan broj, a nit $Y$ nije, onda će nit $X$ proizvesti dva uzastopna broja koja $Z$ sabira, i obratno.
- Korišćenjem deljenih promenljivih i klasičnih brojačkih semafora, napisati sve potrebne deklaracije i kod niti $X$, $Y$ i $Z$.

# Rešenje

```c
// zad3.c
// prevodjenje: gcc -o prog zad3.c -pthread -std=gnu99

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

typedef struct sharedData
{
     int a[2];
     int i;
     sem_t readyToRead;
     sem_t readyToWrite;
     sem_t mutex;
} sharedData;

typedef void* (*threadFunc)(void*);

#define THREAD_CNT 3

void* xy(void* args){
     sharedData* data = (sharedData*)args;
     int iter = 10;
     int i;
     while(iter-- > 0){
          sem_wait(&data->readyToWrite);
          sem_wait(&data->mutex);
          i = data->i;
          data->a[i] = rand()%10;
          data->i = (i+1)%2;
          sem_post(&data->mutex);
          sem_post(&data->readyToRead);
     }
     return NULL;
}
```

```c
void* z(void* args){
    sharedData* data = (sharedData*)args;
    int iter = 10;
    int res;
    while(iter-- > 0){
        sem_wait(&data->readyToRead);
        sem_wait(&data->readyToRead);
        res = data->a[0] + data->a[1];
        printf("%d + %d = %d\n", data->a[0], data->a[1], res);
        sem_post(&data->readyToWrite);
        sem_post(&data->readyToWrite);
    }
    return 0;
}

int main(int argc, char* argv[]){
    sharedData data;
    data.i = 0;
    sem_init(&data.readyToRead, 0, 0);      // readyToRead inicijalizovan na 0
    sem_init(&data.readyToWrite, 0, 2);     // readyToWrite inicijalizovan na 2
    sem_init(&data.mutex, 0, 1);            // mutex inicijalizovan na 0

    pthread_t threads[THREAD_CNT];
    threadFunc funcs[THREAD_CNT] = {xy, xy, z};
    int i;

    srand(time(0));

    for(i=0; i<THREAD_CNT; i++){
        pthread_create(&threads[i], 0, funcs[i], &data);
    }

    for(i=0; i<THREAD_CNT; i++){
        pthread_join(threads[i], 0);
    }

    // unistavanje semafora
    sem_destroy(&data.readyToRead);
    sem_destroy(&data.readyToWrite);
    sem_destroy(&data.mutex);

    printf("%s\n", "Happy ending!\n");

    return 0;

}
```

# Zadatak 4

- Kreirano je više niti istog tipa *P* koji imaju istu sledeću strukturu:
  - u kritičnoj sekciji *A* nalaze se ugnežđene dve kritične sekcije *B* i *C*, s tim da se sekcije *B* i *C* izvršavaju jedna posle druge (nisu ugnežđene).
- Potrebno je obezbediti sledeću sinhronizaciju:
  - u kritičnoj sekciji *A* može se u jednom trenutku nalaziti najviše *N* ovih niti tipa *P*, dok se u sekciji *B*, odnosno *C* može nalaziti najviše jedna nit. (Svaka od sekcija *B* i *C* je međusobno isključiva, ali se *B* i *C* ne isključuju međusobno – jedna nit može biti u *B* dok je druga u *C*.)
- Korišćenjem standardnih brojačkih semafora obezbediti ovu sinhronizaciju:
  - prikazati strukturu tipa niti *P*, uz odgovarajuće definicije i inicijalizacije potrebnih semafora.

# Rešenje

- Rešenje je dato u `zad4.c`
  - Proizvoljan broj niti se sinhroniziuju pomoću 3 semafora koji kontrolišu ulazak u A, B i C kritične sekcije.
  - Rad niti unutar kritičnih sekcija se simulira ispisom identifikatora i čekanjem slučajnog broja sekundi

# Zadatak 5

- Implementirati kružni, ograničeni bafer, kapaciteta `N`, koristeći brojačke semafore.

# Literatura

- David R. Butenhof, "Programming with POSIX Threads", Addison-Wesley, 2004.
- https://computing.llnl.gov/tutorials/pthreads/
  - Uputstvo i dokumentacija o korišćenju POSIX niti
- https://pubs.opengroup.org/onlinepubs/9699919799/
  - Dokumentacija za POSIX niti
- https://pubs.opengroup.org/onlinepubs/9699919799/
  - Dokumentacija za POSIX semafore