

Semafori

UDŽBENIK STRANICE 50-63

Sinhronizacija pomoću semafora

- Sinhronizacija niti može da se zasnije i na ideji saobraćajnog **semafora** koji reguliše ulazak vozova u stanicu sa **jednim** kolosekom.
- Kada se **jedan** voz nalazi na staničnom koloseku, pred semaforom se moraju zaustaviti **svi** vozovi koji treba da dođu na stanični kolosek.
- Po analogiji sa saobraćajnim semaforom prolaz niti kroz (softversku) kritičnu sekciju bi regulisao (softverski) semafor.

Sinhronizacija pomoću semafora

- Sinhronizacija niti, koju omogućuje semafor, se zasniva na **zaustavljanju aktivnosti niti**, kao i na **omogućavanju nastavljanja njihove** aktivnosti.
- Ulazak niti u kritičnu sekciju zavisi od **stanja semafora**.
- Kada stanje semafora dozvoli ulazak niti u kritičnu sekciju, pri ulasku se semafor prevodi u stanje koje **onemogućuje** ulazak **druge niti** u kritičnu sekciju.
- Ako se takva nit pojavi, njena aktivnost se **zaustavlja** pred kritičnom sekcijom.
- Pri izlasku niti iz kritične sekcije semafor se prevodi u stanje koje **dozvoljava novi ulazak** u kritičnu sekciju i ujedno omogućuje nastavak aktivnosti niti koja **najduže čeka** na ulaz u kritičnu sekciju (ako takva nit postoji).

Semafori

```
class Semaphore {
    mutex mx;
    int state;
    condition_variable queue;
public:
    Semaphore(int value = 1) : state(value) {};
    void stop();
    void resume();
};

void Semaphore::stop()
{
    unique_lock<mutex> lock(mx);
    while(state < 1)
        queue.wait(lock);
    state--;
}

void Semaphore::resume()
{
    unique_lock<mutex> lock(mx);
    state++;
    queue.notify_one();
}
```

Vrste i upotreba semafora

- Semafor čije stanje ne može preći vrednost **1** se zove **binarni semafor**.
- Ako se njegovo stanje inicijalizuje na vrednost **1**, tada su njegove operacije **stop()** i **resume()** slične operacijama **lock()** i **unlock()** klase **mutex**.
- Ako se njegovo stanje inicijalizuje na vrednost **0**, tada su njegove operacije **stop()** i **resume()** slične operacijama **wait()** i **notify_one()** klase **condition_variable**.
- Operacije klase **condition_variable** su namenjene za ostvarenje uslovne sinhronizacije u okviru kritičnih sekcija u kojima je međusobna isključivost ostvarena pomoću operacija klase **mutex**.

Raspodeljeni binarni semafor

- Međutim, upotreba operacija binarnog semafora (sa stanjem inicijalizovanim na vrednost **0**) po uzoru na operacije klase **condition_variable** u okviru kritičnih sekcija u kojima je međusobna isključivost ostvarena pomoću operacija drugog binarnog semafora (sa stanjem inicijalizovanim na vrednost **1**) izaziva **mrtvu petlju**.
- Zato se uvodi posebna vrsta binarnog semafora, nazvana **raspodeljeni binarni semafor** (split binary semaphore).

Raspodeljeni binarni semafor

- Realizuje se uz pomoć **više** binarnih semafora za koje važi ograničenje da suma njihovih stanja ne može preći vrednost **1**.
- Pomoću raspodeljenog binarnog semafora se ostvaruje uslovna sinhronizacija tako što se na ulazu u svaku kritičnu sekciju poziva operacija **stop()** **jednog** od njegovih binarnih semafora, a na izlazu iz nje operacija **resume()** **tog** ili **nekog** od preostalih binarnih semafora.
- Na taj način najviše **jedna** nit se može nalaziti najviše u **jednoj** od pomenutih kritičnih sekcija, jer su stanja svih semafora manja od vrednosti 1 za vreme njenog boravka u dotičnoj kritičnoj sekciji.

Klasa Message_box sa upotrebom raspodeljenog binarnog semafora

```
#include "sem.hh"

template<class MESSAGE>
class Message_box {
    MESSAGE content;
    Semaphore empty;
    Semaphore full;
public:
    Message_box() : full(0) {};
    void send(const MESSAGE* message);
    MESSAGE receive();
};

template<class MESSAGE>
void Message_box<MESSAGE>::send(const MESSAGE* message)
{
    empty.stop();
    content = *message;
    full.resume();
}
```

Klasa Message_box sa upotrebom raspodeljenog binarnog semafora

```
template<class MESSAGE>
MESSAGE Message_box<MESSAGE>::receive()
{
    MESSAGE message;
    full.stop();
    message = content;
    empty.resume();
    return message;
}
```

Generalni semafor – primer sa slobodnim baferima

- Semafor, čije stanje može sadržati vrednost **veću od 1**, se naziva generalni semafor (**general semaphore**).
- On omogućuje ostvarenje **uslovne sinhronizacije** prilikom rukovanja resursima.
- Pozitivno stanje generalnog semafora može predstavljati broj slobodnih primeraka nekog resursa.
- Zahvaljujući tome, zauzimanje primerka resursa se može opisati pomoću operacije **stop()**, a njegovo oslobađanje pomoću operacije **resume()** generalnog semafora.

Generalni semafor – primer sa slobodnim baferima

- Ova klasa sadrži **binarni semafor mex** i **generalni semafor list_member_count**.
- **Binarni semafor** omogućuje **međusobnu isključivost** prilikom uvezivanja i izvezivanja slobodnog bafera.
- **Generalni semafor** omogućuje **uslovnu sinhronizaciju**, jer njegovo stanje pokazuje **broj slobodnih bafera** (ono je na početku inicijalizovano na 0).

Generalni semafor – primer sa slobodnim baferima

```
#include "sem.hh"

struct List_member {
    List_member* next;
    char buffer[512];
};

class List {
    List_member* first;
    Semaphore list_member_count;
    Semaphore mex;
public:
    List () : first(0), list_member_count(0) {};
    void link(List_member* member);
    List_member* unlink();
};
```

Generalni semafor – primer sa slobodnim baferima

```
void List::link(List_member* member)
{
    mex.stop();
    member->next=first;
    first=member;
    mex.resume();
    list_member_count.resume();
}
```

```
List_member* List::unlink()
{
    List_member* unlinked;
    list_member_count.stop();
    mex.stop();
    unlinked=first;
    first=first->next;
    mex.resume();
    return unlinked;
}
```

Rešenje problema pet filozofa pomoću semafora

- Rešenje problema **pet filozofa** pomoću semafora sprečava pojavu mrtve petlje, jer za **parne** filozofe zauzima prvo **levu**, a za **neparne** filozofe zauzima prvo **desnu** viljušku.
- Viljuške predstavljaju **binarni semafori forks[5]**, koji omogućuju **uslovnu sinhronizaciju** prilikom zauzimanja viljuški.
- Međusobnu isključivost omogućuje **binarni semafor mex**.
- **Stanja** filozofa su **promenjena**, jer nije moguća mrtva petlja, pa nije bitno koju viljušku filozof čeka.

Rešenje problema pet filozofa pomoću semafora

```
#include<thread>
#include<iostream>

using namespace std;
using namespace chrono;
using namespace this_thread;

#include "sem.hh"

int mod5(int a)
{
    return (a > 4 ? 0 : a);
}

enum Philosopher_state { THINKING = 'T', HUNGRY = 'H', EATING = 'E' };
Philosopher_state philosopher_state[5];
Semaphore forks[5];
Semaphore mex;
```

Rešenje problema pet filozofa pomoću semafora

```
void show()
{
    for(int j = 0; j < 5; j++) {
        cout << '(' << (char)(j+'0') << ':'
            << (char)(philosopher_state[j]) << ") ";
    }
    cout << endl;
}

int philosopher_identity();
const milliseconds THINKING_PERIOD(10);
const milliseconds EATING_PERIOD(10);
```

Rešenje problema pet filozofa pomoću semafora

```
void thread_philosopher()
{
    mex.stop();
    int pi = philosopher_identity++;
    philosopher_state[pi] = THINKING;
    mex.resume();
    for(;;) {
        sleep_for(THINKING_PERIOD);
        mex.stop();
        philosopher_state[pi] = HUNGRY;
        show();
        mex.resume();
        forks[pi%2 == 0 ? pi : mod5(pi+1)].stop();
        forks[pi%2 == 0 ? mod5(pi+1) : pi].stop();
        mex.stop();
        philosopher_state[pi] = EATING;
        show();
        mex.resume();
        sleep_for(EATING_PERIOD);
        mex.stop();
        philosopher_state[pi] = THINKING;
        show();
        mex.resume();
        forks[pi].resume();
        forks[mod5(pi+1)].resume();
    }
}
```

Rešenje problema pet filozofa pomoću semafora

```
int main()
{
    cout << endl << "DINING PHILOSOPHERS" << endl;
    thread philosopher0(thread_philosopher);
    thread philosopher1(thread_philosopher);
    thread philosopher2(thread_philosopher);
    thread philosopher3(thread_philosopher);
    thread philosopher4(thread_philosopher);
    philosopher0.join();
    philosopher1.join();
    philosopher2.join();
    philosopher3.join();
    philosopher4.join();
}
```

Rešenje problema čitanja i pisanja pomoću semafora

- Rešenje problema **čitanja** i **pisanja** pomoću semafora se oslanja na **raspodeljeni binarni semafor** koga obrazuju semafori **readers**, **writers** i **mex**.

Rešenje problema čitanja i pisanja pomoću semafora

```
#include<thread>
#include<iostream>

using namespace std;
using namespace chrono;
using namespace this_thread;

#include "sem.hh"

const int ACCOUNTS_NUMBER = 10;
const int INITIAL_AMOUNT = 100;
```

Rešenje problema čitanja i pisanja pomoću semafora

```
class Bank {
    Semaphore mex;
    int accounts[ACCOUNTS_NUMBER];
    short readers_number;
    short writers_number;
    short readers_delayed_number;
    short writers_delayed_number;
    Semaphore readers;
    Semaphore writers;
    void show();
    void reader_begin();
    void reader_end();
    void writer_begin();
    void writer_end();
public:
    Bank();
    void audit();
    void transaction(unsigned source, unsigned destination);
};
```

Rešenje problema čitanja i pisanja pomoću semafora

```
Bank::Bank() : mex(1), readers(0), writers(0)
{
    for(int i = 0; i < ACCOUNTS_NUMBER; i++)
        accounts[i] = INITIAL_AMOUNT;
    readers_number = 0;
    writers_number = 0;
    readers_delayed_number = 0;
    writers_delayed_number = 0;
}

void Bank::show()
{
    cout << "RN: " << readers_number << " RDN: "
         << readers_delayed_number
         << " WN: " << writers_number << " WDN: "
         << writers_delayed_number << endl;
}
```

Rešenje problema čitanja i pisanja pomoću semafora

```
void Bank::reader_begin()
{
    mex.stop();
    if((writers_number > 0) || (writers_delayed_number > 0)) {
        readers_delayed_number++;
        show();
        mex.resume();
        readers.stop();
    }
    readers_number++;
    show();
    if(readers_delayed_number > 0) {
        readers_delayed_number--;
        show();
        readers.resume();
    } else
        mex.resume();
}
```

Rešenje problema čitanja i pisanja pomoću semafora

```
void Bank::reader_end()
{
    mex.stop();
    readers_number--;
    show();
    if((readers_number == 0) && (writers_delayed_number > 0)) {
        writers_delayed_number--;
        show();
        writers.resume();
    } else
        mex.resume();
}
```

Rešenje problema čitanja i pisanja pomoću semafora

```
void Bank::writer_begin()
{
    mex.stop();
    if((readers_number > 0) || (writers_number > 0)) {
        writers_delayed_number++;
        show();
        mex.resume();
        writers.stop();
    }
    writers_number++;
    show();
    mex.resume();
}
```

Rešenje problema čitanja i pisanja pomoću semafora

```
void Bank::writer_end()
{
    mex.stop();
    writers_number--;
    show();
    if(writers_delayed_number > 0) {
        writers_delayed_number--;
        show();
        writers.resume();
    } else {
        if(readers_delayed_number > 0) {
            readers_delayed_number--;
            show();
            readers.resume();
        } else
            mex.resume();
    }
}
```

Rešenje problema čitanja i pisanja pomoću semafora

```
const milliseconds READING_PERIOD(1);

void Bank::audit()
{
    int sum = 0;
    reader_begin();
    sleep_for(READING_PERIOD);
    for(unsigned i = 0; i < ACCOUNTS_NUMBER; i++)
        sum += accounts[i];
    reader_end();
    if(sum != ACCOUNTS_NUMBER*INITIAL_AMOUNT) {
        mex.stop();
        cout << " audit error " << endl;
        mex.resume();
    }
}
```

Rešenje problema čitanja i pisanja pomoću semafora

```
const milliseconds WRITING_PERIOD(1);

void Bank::transaction(unsigned source, unsigned destination)
{
    int amount;
    writer_begin();
    sleep_for(WRITING_PERIOD);
    amount = accounts[source];
    accounts[source] -= amount;
    accounts[destination] += amount;
    writer_end();
}

Bank bank;

void thread_reader()
{
    bank.audit();
}
```

Rešenje problema čitanja i pisanja pomoću semafora

```
void thread_writer0to1()
{
    bank.transaction(0, 1);
}

void thread_writer1to0()
{
    bank.transaction(1, 0);
}

int main()
{
    cout << endl << "READERS AND WRITERS" << endl;
    thread reader0(thread_reader);
    thread reader1(thread_reader);
    thread writer0(thread_writer0to1);
    thread reader2(thread_reader);
    thread writer1(thread_writer1to0);
    reader0.join();
    reader1.join();
    writer0.join();
    reader2.join();
    writer1.join();
}
```