Razvoj softvera - Čas 10

Primeri sa predavanja iz predmeta *Razvoj softvera* na Matematičkom fakultetu Univerziteta u Beogradu u školskoj 2021/22. godini.

Prof. dr Saša Malkov

primer.01.cpp

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

using namespace std;

void myThreadFn( int tId )
{
    for( int i=0; i<10; i++ ){
        cout << "Nit " << tId << " : " << i << " : ";
      for( unsigned i=0; i<40000; i++ )
         if( i%1000 == 0 )
            cout << tId;
      cout << "\n";
   }
}

int main(int argc, char **argv)
{
    cout << "Pripremi i kreni..." << endl;
   thread t1( myThreadFn, 1 );
   thread t2( myThreadFn, 2 );

   cout << "...sacekaj..." << endl;
    t1.join();
    t2.join();

    cout << "Kraj." << endl;
    return 0;
}
```

primer.02.mutex.cpp

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

using namespace std;

mutex ekran;
```

```
void myThreadFn( int tId )
{
    for( int i=0; i<10; i++ ){
      ekran.lock();
        cout << "Nit " << tId << " : " << i << " : ";
      for( unsigned i=0; i<40000; i++ )
         if( i%1000 == 0 )
            cout << tId;
      cout << "\n";
      ekran.unlock();
   }
}

int main(int argc, char **argv)
{
    cout << "Pripremi i kreni..." << endl;
   thread t1( myThreadFn, 1 );
   thread t2( myThreadFn, 2 );

   ekran.lock();
   cout << "...sacekaj..." << endl;
   ekran.unlock();

    t1.join();
    t2.join();

    cout << "Kraj." << endl;
    return 0;
}
```

primer.03.mutex.lock.guard.cpp

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

using namespace std;

mutex ekran;

void myThreadFn( int tId )
{
    for( int i=0; i<10; i++ ){
      lock_guard<mutex> katanac(ekran);
        cout << "Nit " << tId << " : " << i << " : ";
      for( unsigned i=0; i<40000; i++ )
         if( i%1000 == 0 )
            cout << tId;
      cout << "\n";
   }
}
```

```
int main(int argc, char **argv)
{
    cout << "Pripremi i kreni..." << endl;
   thread t1( myThreadFn, 1 );
   thread t2( myThreadFn, 2 );

   {
   lock_guard<mutex> katanac(ekran);
   cout << "...sacekaj..." << endl;
   }

    t1.join();
    t2.join();

    cout << "Kraj." << endl;

    return 0;
}
```

primer.04.globalData.cpp

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

using namespace std;

int globalData = 0;

void myThreadFn( int tId )
{
    for( int i=0; i<10; i++ ){
      int x = globalData;
        cout << "Nit " << tId << " : " << i << endl;
        x++;
      globalData = x;
    }
}

int main(int argc, char **argv)
{
    int n = 100;
    vector<thread> threads;

    cout << "Pripremi i kreni..." << endl;
    for( int i=0; i<n; i++ )
        threads.emplace_back( myThreadFn, i );
//        threads.push_back( thread( myThreadFn, i ));

   cout << "...sacekaj..." << endl;
    for( thread& t: threads )
        t.join();
    cout << "Kraj." << endl;
    cout << "globalData = " << globalData << endl;

    return 0;
}
```

primer.04.globalData.p.cpp

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

using namespace std;

int globalData = 0;

void myThreadFn( int tId )
{
    for( int i=0; i<10; i++ ){
      int x = globalData;
        cout << "Nit " << tId << " : " << i << endl;
        x++;
      globalData = x;
    }
}

int main(int argc, char **argv)
{
    int n = 100;
    vector<thread*> threads;

    cout << "Pripremi i kreni..." << endl;
    for( int i=0; i<n; i++ )
        threads.push_back( new thread( myThreadFn, i ));

   cout << "...sacekaj..." << endl;
    for( thread* t: threads )
        t->join();

    cout << "Kraj." << endl;
    for( thread* t: threads )
        delete t;
    threads.clear();

    cout << "globalData = " << globalData << endl;
    return 0;
}
```

primer.05.globalData.mutex.cpp

```cpp
#include <iostream>
#include <thread>
#include <vector>
#include <mutex>

using namespace std;

int globalData = 0;
mutex globalDataMutex;

void myThreadFn( int tId )
{
    for( int i=0; i<10; i++ ){
      lock_guard<mutex> globalDataLock(globalDataMutex);
      int x = globalData;
        cout << "Nit " << tId << " : " << i << endl;
        x++;
      globalData = x;
    }
}

int main(int argc, char **argv)
{
    int n = 100;
    vector<thread> threads;

    cout << "Pripremi i kreni..." << endl;
    for( int i=0; i<n; i++ )
        threads.emplace_back( myThreadFn, i );
//        threads.push_back( thread( myThreadFn, i ));

   cout << "...sacekaj..." << endl;
    for( thread& t: threads )
        t.join();

    cout << "Kraj." << endl;
    cout << "globalData = " << globalData << endl;

    return 0;
}
```

primer.05.globalData.mutex.p.cpp

```cpp
#include <iostream>
#include <thread>
#include <vector>
#include <mutex>

using namespace std;

int globalData = 0;
```

```
mutex globalDataMutex;

void myThreadFn( int tId )
{
    for( int i=0; i<10; i++ ){
      lock_guard<mutex> globalDataLock(globalDataMutex);
      int x = globalData;
        cout << "Nit " << tId << " : " << i << endl;
        x++;
      globalData = x;
    }
}

int main(int argc, char **argv)
{
    int n = 100;
    vector<thread*> threads;

    cout << "Pripremi i kreni..." << endl;
    for( int i=0; i<n; i++ )
        threads.push_back( new thread( myThreadFn, i ));

   cout << "...sacekaj..." << endl;
    for( thread* t: threads )
        t->join();

    cout << "Kraj." << endl;
    for( thread* t: threads )
        delete t;
    threads.clear();

    cout << "globalData = " << globalData << endl;
    return 0;
}
```

primer.06.zadaci.cpp

```
#include <iostream>
#include <vector>
#include <queue>
#include <thread>
#include <mutex>

using namespace std;

class Zadaci
{
public:
    Zadaci()
    {}

    void dodajZadatak( int n )
    {
        lock_guard<mutex> lock(zadaciMutex);
        zadaci.push( n );
    }

    int uzmiZadatak()
    {
        lock_guard<mutex> lock(zadaciMutex);

        if( zadaci.empty() )
            return -1;

        // 0 koristimo kao oznaku za kraj,
        // ostavljamo je da bi i druge niti zavrsile sa radom
        int n = zadaci.front();
        if( n>0 )
            zadaci.pop();

        return n;
    }

private:
    mutex         zadaciMutex;
    std::queue<int> zadaci;
};


void izvrsavanjeZadataka( Zadaci& zadaci, int _ID )
{
    while(true){
        int zadatak = zadaci.uzmiZadatak();
        if( !zadatak )
            break;
        if( zadatak > 0 ){
            for( int i=1; i<=zadatak; i++ ){
                cout << "Nit " << _ID << " : " << i << endl;
            this_thread::sleep_for( chrono::milliseconds( 200 ));
            }
        }else
         this_thread::sleep_for( chrono::milliseconds( 200 ));
    }
}

void postavljanjeZadataka( Zadaci& zadaci )
{
    while(true){
        cout << "Unesi pozitivan ceo broj: ";
        int n;
        cin >> n;
        if( n<0 )
            cout << "Greska!" << endl;
        else{
            zadaci.dodajZadatak(n);
            if( n==0 )
```

```
                break;
            }
        }
}

int main(int argc, char **argv)
{
    int n = 5;
    vector<thread> threads;
   Zadaci zadaci;

    cout << "Pokretanje niti..." << endl;
    for( int i=0; i<n; i++ )
        threads.emplace_back( izvrsavanjeZadataka, ref(zadaci), i );

   postavljanjeZadataka(zadaci);

    cout << "wait..." << endl;
    for( thread& t: threads )
        t.join();

    cout << "Kraj." << endl;
    return 0;
}
```

primer.06.zadaci.p.cpp

```
#include <iostream>
#include <vector>
#include <queue>
#include <thread>
#include <mutex>

using namespace std;

class Zadaci
{
public:
    Zadaci()
    {}

    void dodajZadatak( int n )
    {
        lock_guard<mutex> lock(zadaciMutex);
        zadaci.push( n );
    }

    int uzmiZadatak()
    {
        lock_guard<mutex> lock(zadaciMutex);

        if( zadaci.empty() )
```

```
            return -1;

        // 0 koristimo kao oznaku za kraj,
        // ostavljamo je da bi i druge niti zavrsile sa radom
        int n = zadaci.front();
        if( n>0 )
            zadaci.pop();

        return n;
    }

private:
    mutex         zadaciMutex;
    std::queue<int> zadaci;
};


void izvrsavanjeZadataka( Zadaci& zadaci, int _ID )
{
    while(true){
        int zadatak = zadaci.uzmiZadatak();
        if( !zadatak )
            break;
        if( zadatak > 0 ){
            for( int i=1; i<=zadatak; i++ ){
                cout << "Nit " << _ID << " : " << i << endl;
            this_thread::sleep_for( chrono::milliseconds( 200 ));
            }
        }else
         this_thread::sleep_for( chrono::milliseconds( 200 ));
    }
}

void postavljanjeZadataka( Zadaci& zadaci )
{
    while(true){
        cout << "Unesi pozitivan ceo broj: ";
        int n;
        cin >> n;
        if( n<0 )
            cout << "Greska!" << endl;
        else{
            zadaci.dodajZadatak(n);
            if( n==0 )
                break;
        }
    }
}

int main(int argc, char **argv)
{
    int n = 10;
    vector<thread*> threads;
   Zadaci zadaci;
```

```
    cout << "pokretanje niti..." << endl;
    for( int i=0; i<n; i++ )
        threads.push_back(new thread(izvrsavanjeZadataka,ref(zadaci),i));

   postavljanjeZadataka(zadaci);

    cout << "wait..." << endl;
    for( int i=0; i<n; i++ ){
        threads[i]->join();
        delete threads[i];
   }

    cout << "end..." << endl;
    threads.clear();

    return 0;
}
```

primer.08.suma.cpp

```
#include <numeric>
#include <iostream>
#include <vector>
#include <future>

using namespace std;

template <typename Iterator>
int sumaNiza(std::launch rezim, Iterator beg, Iterator end)
{
    auto len = end - beg;
   // ako je niz kratak, racunamo odjednom
    if(len < 1000000)
        return std::accumulate(beg, end, 0);

   // inace delimo racunanje na dva jednaka dela
    Iterator mid = beg + len/2;
    future<int> medjuzbir = std::async( rezim, sumaNiza<Iterator>, rezim, mid, end
);
    int sum = sumaNiza( rezim, beg, mid );
   // spojimo delove i vratimo rezultat
    return sum + medjuzbir.get();
}

void test( std::launch rezim )
{
   // niz od n elemenata, svi imaju vrednost 1
    std::vector<int> v(50000000, 1);

    auto t0 = chrono::system_clock::now();
   int suma = sumaNiza(
                rezim,
                v.begin(), v.end()
             );
    auto t1 = chrono::system_clock::now();
    auto d = chrono::duration_cast<chrono::milliseconds>( t1 - t0 );

    std::cout << "Suma niza je " << suma << endl;
    cout << "Trajanje: " << d.count()/1000.0 << "s" << endl;
}

int main()
{
   cout << "ASYNC:" << endl;
   test( std::launch::async );
   cout << "DEFERRED:" << endl;
   test( std::launch::deferred );

   return 0;
}
```

primer.09.zadaci2.cpp

```
#include <iostream>
#include <vector>
#include <queue>
#include <thread>
#include <mutex>
#include <condition_variable>

using namespace std;

class Zadaci
{
public:
    Zadaci()
    {}

    void dodajZadatak( int n )
    {
        lock_guard<mutex> lock(zadaciMutex);
        zadaci.push( n );
        //  ako neko ceka, bice obavesten, ako ne ceka, nista se nece desiti
        if( n>0 )
            //  o obicnom poslu obavestavamo jednu nit
            cvar.notify_one();
        else
            //  o zavrsetku obavestavamo sve niti
            cvar.notify_all();
    }

    //  ako nema posla, ceka dok se ne pojavi
    int uzmiZadatak_cekaj()
    {
        unique_lock<mutex> lock(zadaciMutex);
        //  Iako izgleda da je dovoljno `if`, moze da se desi da neka druga nit
prodje
        //  i uzme prvi posao pre nego sto ona koja ceka obavestenje nastavi,
        //  pa da red u medjuvremenu postane ponovo prazan.
        //  Ako ima vise `consumera`, onda je potrebno `while`.
        while( zadaci.empty() )
            cvar.wait(lock);

        // 0 koristimo kao oznaku za kraj,
        // ostavljamo je u redu da bi i druge niti zavrsile sa radom
        int n = zadaci.front();
        if( n>0 )
            zadaci.pop();
        return n;
    }

private:
    mutex              zadaciMutex;
    condition_variable  cvar;
    std::queue<int>     zadaci;
};


void izvrsavanjeZadataka( Zadaci& zadaci, int id_ )
{
    while(true){
        int zadatak = zadaci.uzmiZadatak_cekaj();
        if( !zadatak )
            break;
        cout << "Nit " << id_ << " : Pocinje zadatak : " << zadatak << endl;
        for( int i=1; i<=zadatak; i++ ){
            cout << "    Nit " << id_ << " : " << i << endl;
            this_thread::sleep_for( chrono::milliseconds( 200 ));
        }
    }
}

void postavljanjeZadataka( Zadaci& zadaci )
{
    while(true){
        cout << "Unesi pozitivan ceo broj: ";
        int n;
        cin >> n;
        if( n<0 )
            cout << "Greska!" << endl;
        else{
            zadaci.dodajZadatak(n);
            if( n==0 )
                break;
        }
    }
}
```

```
int main(int argc, char **argv)
{
    int brojNiti = 5;
    vector<thread> threads;
   Zadaci zadaci;

    cout << "Pokretanje niti..." << endl;
    for( int i=0; i<brojNiti; i++ )
        threads.emplace_back( izvrsavanjeZadataka, ref(zadaci), i );

   postavljanjeZadataka(zadaci);

    cout << "wait..." << endl;
    for( thread& t: threads )
        t.join();

    cout << "Kraj." << endl;
    return 0;
}
```