Razvoj softvera - Čas 11

Primeri sa predavanja iz predmeta *Razvoj softvera* na Matematičkom fakultetu Univerziteta u Beogradu u školskoj 2022/23. godini.

Prof. dr Saša Malkov

primer.07.future.cpp

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

using namespace std;

int nestoBezveze( int n )
{
   cout << "    " << n << " : pocinje izracunavanje..." << endl;
   this_thread::sleep_for( chrono::seconds(3));
   cout << "    " << n << " : gotovo izracunavanje..." << endl;
   return n+100;
}

int main(int argc, char **argv)
{
   cout << "PODRAZUMEVANO:" << endl;
   cout << "pravimo future objekat..." << endl;
   future<int> rezultat1 = async( nestoBezveze, 1 );
   this_thread::sleep_for( chrono::milliseconds(100));
   cout << "cekamo..." << endl;
   cout << rezultat1.get() << endl << endl;

   cout << "ASYNC:" << endl;
   cout << "pravimo future objekat..." << endl;
   future<int> rezultat2 = async( std::launch::async, nestoBezveze, 2 );
   this_thread::sleep_for( chrono::milliseconds(100));
   cout << "cekamo..." << endl;
   cout << rezultat2.get() << endl << endl;

   cout << "DEFERRED:" << endl;
   cout << "pravimo future objekat..." << endl;
   future<int> rezultat3 = async( std::launch::deferred, nestoBezveze, 3 );
   this_thread::sleep_for( chrono::milliseconds(100));
   cout << "cekamo..." << endl;
   cout << rezultat3.get() << endl;

    return 0;
}
```

primer.08.future.cpp

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

using namespace std;

int nestoBezveze( int n )
{
   cout << "    " << n << " : pocinje izracunavanje..." << endl;
   this_thread::sleep_for( chrono::seconds(3));
   cout << "    " << n << " : gotovo izracunavanje..." << endl;
   return n+100;
}


int main(int argc, char **argv)
{
   // future.get() ne sme da se pozove vise puta na istom objektu
   // ali mozemo da podelimo objekat, nakon cega original vise nije upotrebljiv
   cout << "pravimo future objekat..." << endl;
   //future<int> deljenRezultat = async( nestoBezveze, 5 );
   future<int> rezultat = async( nestoBezveze, 5 );
   shared_future<int> deljenRezultat = rezultat.share();
   cout << "cekamo..." << endl;
   cout << deljenRezultat.get() << endl;
   cout << "cekamo ponovo isti rezultat..." << endl;
   cout << deljenRezultat.get() << endl;
    return 0;
}
```

primer.09.future.cpp

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

using namespace std;

unsigned long suma( int n, int m )
{
   cout << "    " << n << " : Racunanje sume od " << n << " do " << m << "..." <<
endl;
   unsigned long suma = 0;
   for( int i=n; i<=m; i++ ){
      suma += i;
      this_thread::yield();
   }
   cout << "    " << n << " : " << n << " + ... + " << m << " = " << suma << endl;
   return suma;
```

```
}

void test( std::launch tl )
{
   vector<future<unsigned long>> v;
   cout << "* Pravimo future objekte..." << endl;
   for( int i=1; i<=10; i++ )
// umesto v.emplace_back( aFuture )
// moze i v.push_back( move(aFuture) )
      v.emplace_back( async(
         tl,
         suma, i, 10000000 + i
      ));

   this_thread::sleep_for( chrono::milliseconds(100));
   cout << "* Racunamo ukupan zbir..." << endl;
   unsigned long suma = 0;
   for( auto& x: v )
      suma += x.get();
   cout << "* Ukupan zbir = " << suma << endl;
}

int main(int argc, char **argv)
{
   cout << "DEFERRED:" << endl;
   cout << "Izracunavanje pocinje tek kada se zatrazi rezultat:" << endl;
   test( std::launch::deferred );
   cout << endl;
   cout << "ASYNC:" << endl;
   cout << "Izracunavanje pocinje odmah po pravljenju posla:" << endl;
   test( std::launch::async );

    return 0;
}
```

primer.10.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> sumaDrugePolovine = std::async( rezim, sumaNiza<Iterator>, rezim,
mid, end );
    int sumaPrvePolovine = sumaNiza( rezim, beg, mid );
   // spojimo delove i vratimo rezultat
    return sumaPrvePolovine + sumaDrugePolovine.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 << endl << "DEFERRED:" << endl;
   test( std::launch::deferred );

   return 0;
}
```

primer.11.suma.cpp

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

using namespace std;

template <typename Iterator>
int sumaNiza(Iterator beg, Iterator end)
{
    return std::accumulate(beg, end, 0);
}

template <typename Iterator>
```

```
int paralelizovanaSumaNiza(Iterator beg, Iterator end)
{
   int nOfThreads = thread::hardware_concurrency();
    auto len = end - beg;
   if( len > 1000 && nOfThreads > 1 ){
      cout << "Radimo u " << nOfThreads << " niti..." << endl;
      cout << "Pravimo niti..." << endl;
      vector<future<int>> v;
      Iterator blockbeg = beg;
      for( int i=0; i<nOfThreads; i++ ){
         Iterator blockend = blockbeg + len/nOfThreads;
         if( i==nOfThreads-1 )
            blockend = end;
         v.emplace_back(std::async(
            std::launch::async,
            sumaNiza<Iterator>,
            blockbeg, blockend
         ));
         blockbeg = blockend;
      }
      cout << "Sabiramo medjurezultate..." << endl;
      int suma = 0;
      for( auto& q: v )
         suma += q.get();
      return suma;
   }
    else{
        return sumaNiza(beg,end);
   }
}

int main()
{
   // niz od ... elemenata, svi imaju vrednost 1
    std::vector<int> v(50000000, 1);

    auto t0 = chrono::system_clock::now();
   int suma = sumaNiza( v.begin(), v.end() );
    std::cout << "Suma niza je " << suma << endl;
    auto t1 = chrono::system_clock::now();
    auto d = chrono::duration_cast<chrono::milliseconds>( t1 - t0 );
    cout << "Trajanje: " << d.count()/1000.0 << "s" << endl << endl;

    t0 = chrono::system_clock::now();
   suma = paralelizovanaSumaNiza( v.begin(), v.end() );
    std::cout << "Suma niza je " << suma << endl;
    t1 = chrono::system_clock::now();
    d = chrono::duration_cast<chrono::milliseconds>( t1 - t0 );
    cout << "Trajanje: " << d.count()/1000.0 << "s" << endl;

   return 0;
}
```

primer.12.reentrant.cpp

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

using namespace std;

// Primer obezbedjivanja funkcije sa ponovljenim pozivanjima (reentrant)
// izvodjenjem zakljucavanja tela prema razlicitim podacima.

// Pravimo 100 niti i 50 brojaca
// tako da svaka nit po 100 puta povecava svaki brojac.
// Trebalo bi da na kraju svi brojaci imaju vrednost 10000

// Bez zakljucavanja obicno dolazi do odstupanja.
// Sa zakljucavanjem radi osetno sporije (pre svega zbog niske granularnosti
zakljucavanja).
// Sredina je da se zakljucava ne ceo niz nego svaki element posebno.

void reentrant_function( vector<long>& niz, unsigned i )
{
/* v1: bez muteksa, daje neispravan rezultat */

/* v2: sa jednim muteksom, daje dobar rezultat ali veoma neefikasno
   static mutex m;
   lock_guard<mutex> guard( m );
*/

/* v3: sa po jednim muteksom za svaki podatak, daje efikasniji rezultat
   static vector<mutex> mutexes( niz.size() );
   lock_guard<mutex> guard( mutexes[i] );
*/
   static vector<mutex> mutexes( niz.size() );
   lock_guard<mutex> guard( mutexes[i] );

   // telo funkcije...
   long x = niz[i];
   x++;
   niz[i] = x;

   // Bez optimizacije koda je moguce i samo "niz[i]++".
   // Namerno je razlozeno na vise instrukcija, da bi se povecala verovatnoca
sukoba
   // ako se radi bez zakljucavanja.
   // Sa optimizacijom koda se cesto cak i prethodni kod svodi na atomicne
operacije.
}

void thread_fn( vector<long>& niz )
```

```
{
   for( unsigned i=0; i<100; i++ )
      for( unsigned i=0; i<niz.size(); i++ )
         reentrant_function( niz, i );
}

int main()
{
   vector<long> niz(50,0);
   vector<thread> threads;

    auto t0 = chrono::system_clock::now();
   for( unsigned i=0; i<100; i++ )
      threads.emplace_back(
         [&niz]{ thread_fn(niz); }
      );
   for( thread& t: threads )
      t.join();

   cout << endl;
   for( long n: niz )
      cout << n << ", ";
   cout << endl;

    auto t1 = chrono::system_clock::now();
    auto d = chrono::duration_cast<chrono::milliseconds>( t1 - t0 );
    cout << "Trajanje: " << d.count()/1000.0 << "s" << endl << endl;

   return 0;
}
```