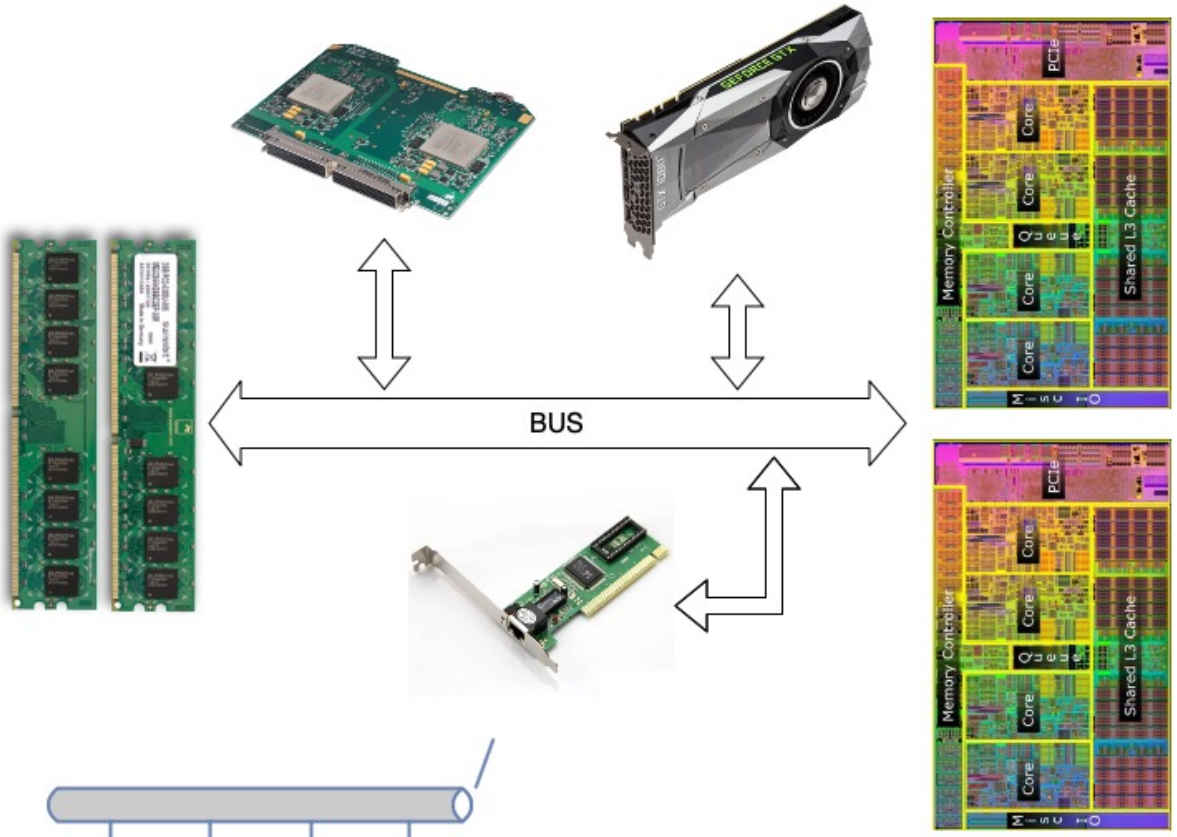
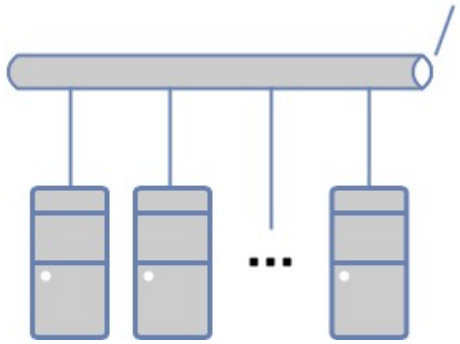
Többszálú programozás, szálkezelés C++ nyelven

C++11 párhuzamosság és többszálúság

Párhuzamos végrehajtás

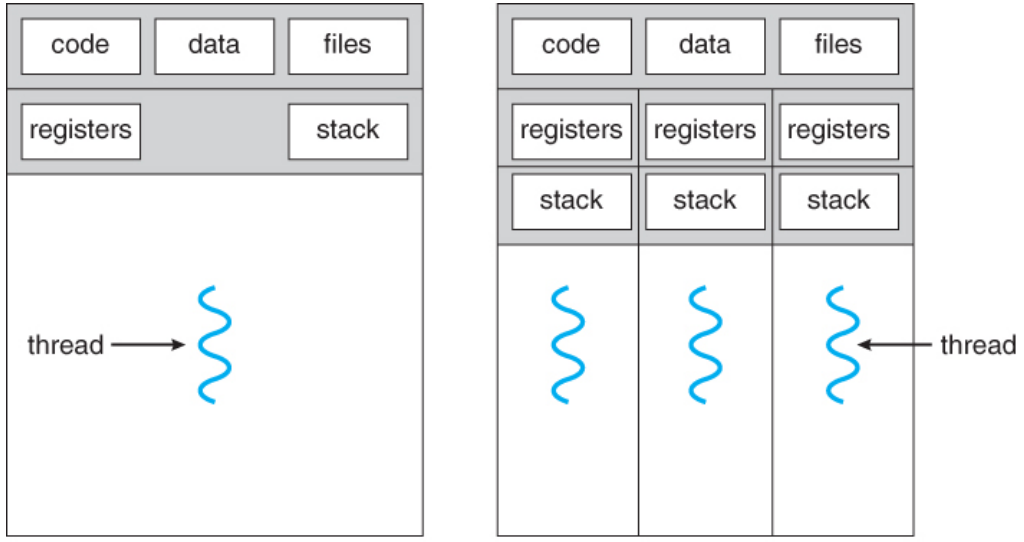
- Bitszintű
- Utasításszintű
- Feladatszintű





Végrehajtási absztrakciók

- Folyamat – futó alkalmazás teljes „infrastruktúrája”
  - saját memóriaterülete és saját címtartománya van
- Szál – folyamaton belüli önálló végrehajtási egység
  - környezet + utasítássorozat
  - könnyűsúlyú folyamat (lightweight process)



Összehasonlítás

- Szálak „könnyűsúlyúak”
  - Gyors felépítés (spawn), váltás (context switching)
- IPC drága
- Közös címtartomány – sok hibalehetőség
  - gondos elemzés és megfontolások
- Igazán skálázható párhuzamos architektúrákat folyamatokkal tudunk csak megvalósítani
- Filozófia
  - „*communicate by sharing memory*” vs. „*share memory by communicating*”

Mire használjuk?

- Konkurens programozás
  - Strukturálás
    - Vonatkozások szétválasztása
    - Modularitás
    - Reszponzívitás
    - Karbantarthatóság
  - Számítás mikor indítható?
  - Közös erőforrások elérése?
- Párhuzamos programozás
  - Teljesítmény-orientált
  - Alg., adatelérés és számítási minták
  - Számítások felosztása?
  - Lokalitás növelése?
  - Redundáns számítások?
  - Hardver optimális kihasználtsága?

Emlékeztető

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 5

void *PrintHello(void *threadid) {
 long tid;
 tid = (long)threadid;
 printf("Hello World! It's me, thread #%ld!\n", tid);
 pthread_exit(NULL);
}

int main(int argc, char *argv[]) {
 pthread_t threads[NUM_THREADS];
 int rc;
 long t;
 for(t=0;t<NUM_THREADS;t++){
  printf("In main: creating thread %ld\n", t);
  rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
 }
 /* Last thing that main() should do */
 pthread_exit(NULL);
}
```

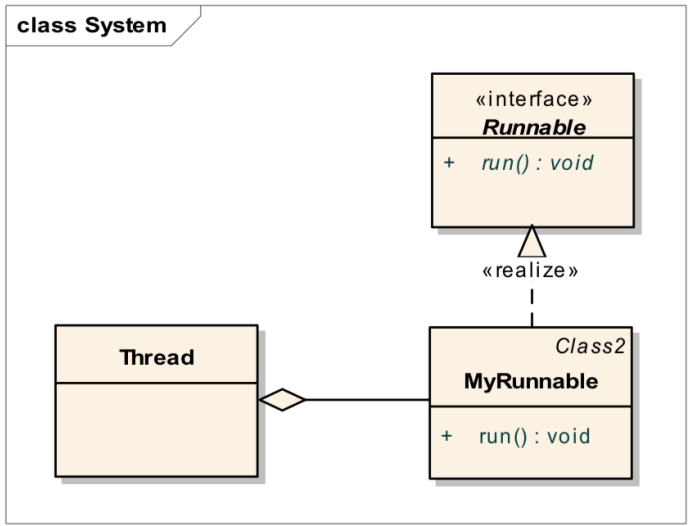
Emlékeztető

```
public class  MyRunnable implements Runnable{
  private int id;

  public MyRunnable(int id ){
    this.id = id;
  }

  public void run(){
    for( int i=0; i<10; ++i){
     System.out.println("Hello"+id+" "+i);
   }
  }
}

…
MyRunnable r = new MyRunnable(1);
Thread t = new Thread( r );
```

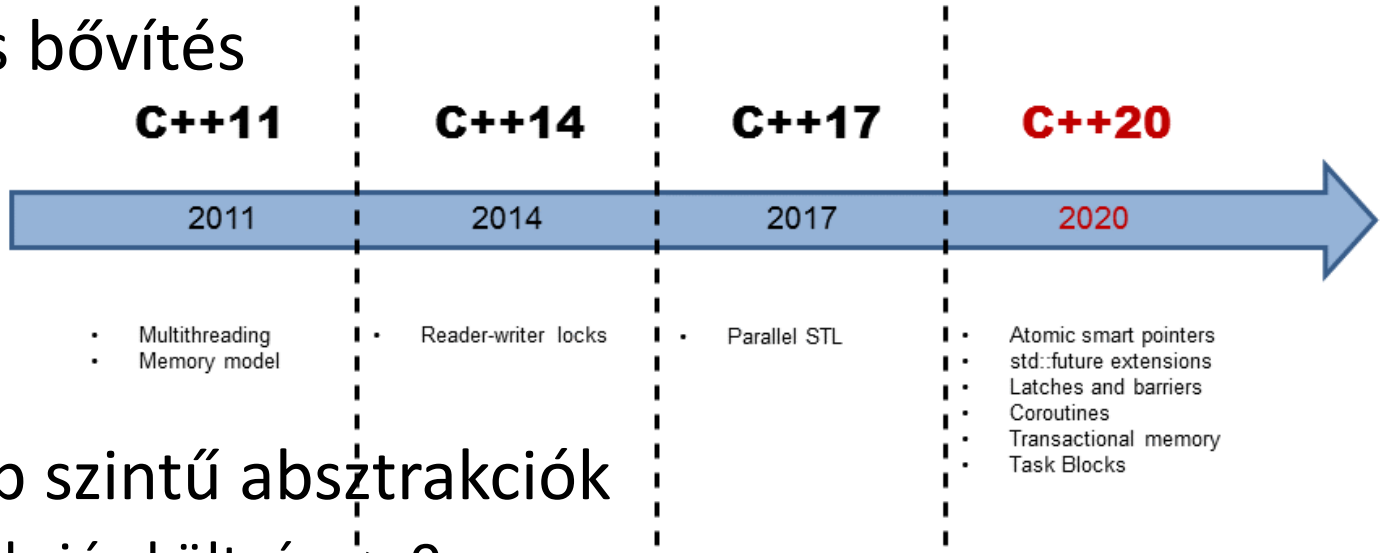


Bevezető

- Többmagos rendszer – many core
  - többszálú programozás
    - imperatív (*std::thread*, *pthreads*)
    - deklaratív (*OpenMP*)
    - könyvtár alapú (*Intel TBB, Microsoft PPL*)

C++ multithreading

- Standard része C++11-től
- Folytonos bővítés



- Magasabb szintű absztrakciók
  - Absztrakciós költség -> 0
- Platformfüggetlen, karbantarthatóság
- Platform-specifikus lehetőségek elérése

C++ multithreading

- Gazdag lehetőségek
- Fejállományok
  - <*thread*>
  - <*future*>
  - <*atomic*>
  - <*mutex*>
  - <*condition_variable*>

Bevezető

- Futtatási mód
  - aszinkron módon (*async*)
  - egymáshoz szinkronizálva (*thread*)
- Minden szálnak rendelkeznie kell egy kezdeti függvénnyel,
  - amivel egy új végrehajtási szekvencia kezdődhet
  - az alkalmazás kezdeti szála esetében ez a main () függvény lesz.
  - minden más szál esetében a konstruktorában meg kell adni a végrehajtandó utasítások belépési pontját
- A futtatni kívánt függvényt (funkcionált) többféleképpen is megadhatjuk:
  - függvénymutatóval (a függvény nevével),
  - tagfüggvény mutatóval
  - lambda kifejezésként
  - funktor - függvény objektummal (amely definiálja az ***operator()*** tagfüggvényt)

Bevezető

- Egy új szál indítása után a kezdeti szál folytatja a végrehajtást.
- Megtörténhet, hogy a main () végére ér, ezáltal befejezve a programot, még mielőtt az új szál lefuthatott volna, vagy befejezhette volna a munkáját.
- Ezért, a szál bevárása érdekében meg kell hívni a szálobjektum join () tagfüggvényét
- Amennyiben nem kívánjuk bevárni a szálat, ez leválasztható a detach() tagfüggvény meghívásával.

Hello World

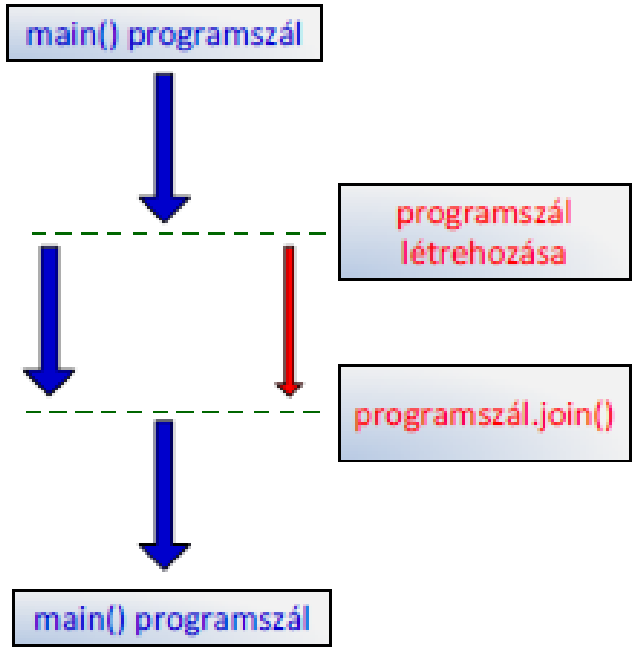
```cpp
#include <iostream>
#include <thread>

int main() {
  std::thread helloThread([]() {
    std::this_thread::sleep_for(std::chrono::milliseconds{500});
    std::cout << "> Hello World! Greetings from thread "
              << std::this_thread::get_id() << '\n';
  });
  std::cout << "Main thread (" << std::this_thread::get_id()
            << ") created another thread with id "
            << helloThread.get_id() << '\n';
  std::cout << "Main thread going to sleep for 2s " << '\n';
  std::this_thread::sleep_for(std::chrono::milliseconds{2000});
  std::cout << "Main thread woke up " << '\n';
  helloThread.join();
  return 0;
}
```

```
Main thread (0x103c285c0) created another thread with
id 0x7000092c3000
Main thread going to sleep for 2s
> Hello World! Greetings from thread 0x7000092c3000
Main thread woke up
```

Főszál (programszál)

- Egy folyamatnak legalább 1 u.n. főszála van
- C++ - main() fgv.
- Új szálakat indíthat
  - Indított szálak, indíthatnak más szálakat
- Ezeket bevárja vagy leválasztja



Új szál indítása

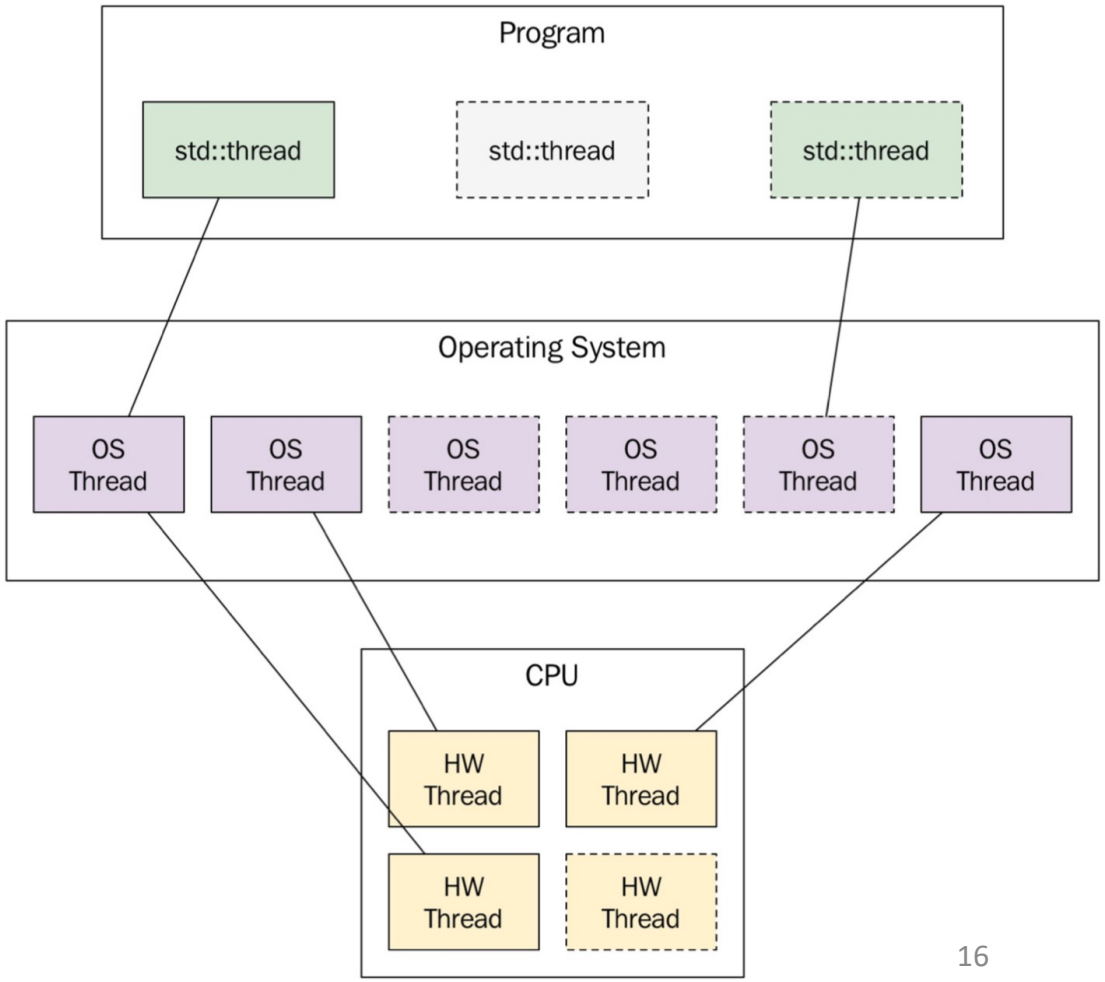
- Végrehajtási szálat a *thread* osztály objektumaként hozzuk létre
  - egy kezdeti végrehajtási pont megadásával a konstruktorban
  - azonnal elindul / ütemezésre vár

- Egy új szál indítása után a kezdeti szál folytatja a végrehajtást

Szál állapota

- *std::thread::joinable*

- Hamis ha
  - Alapértelmezetten épült
  - Áthelyezett
  - Leválasztott
  - Már bevárt



Szál bevárása

- Megtörténhet, hogy a fő szál main () végére ér, még mielőtt az új szál lefuthatott volna, vagy befejezhette volna a munkáját
- Ezért, a szál bevárása érdekében meg kell hívni a szálobjektum *join ()* tagfüggvényét
- Amennyiben nem kívánjuk bevárni a szálat, ez leválasztható a *detach()* tagfüggvény meghívásával.

Szálak azonosítása

- Minden szál rendelkezik egy egyedi azonosítóval
  - *get_id()* tagfüggvényének hívásával kérhetjük le
- Az aktuális szálra könnyedén szerezhetünk referenciát az *std::this_thread* segítségével

- Platform specifikus
  - *std::thread::native_handle*

Argumentumok átadása

- A konstruktorban, a funktor vagy függvény specifikálása után, egyszerűen felsoroljuk az átadásra szánt argumentumokat, parmétereket:

```
template< class Function, class... Args >
explicit thread( Function&& f, Args&&... args );
```

- Átadás érték szerint történik!

Feladatok, példák

- Tömb összege 2 szállal.
- Főátló feletti, alatti prímszámok száma.

Feladatok, példák

- Tömb összege 2 szállal. Modern megoldás. Használjunk:
  - Sablonfüggvényt
  - std::vector<T>
  - std::for_each
  - Lambda függvényeket

Megoldás

```
template <typename T>
T add_with_two_threads(std::vector<T> v){
 T total1 = 0;
 T total2 = 0;

 auto num_elem = std::distance(std::begin(v), std::end(v));

 std::thread t1([&] {
  std::for_each(std::begin(v), std::begin(v) + num_elem/2,
   [&](T x) {total1 += x;});
 });

 std::thread t2([&] {
  std::for_each(std::begin(v) + num_elem/2, std::end(v),
   [&](T x) {total2 += x;});
 });

 t1.join();
 t2.join();
 return total1 + total2;
}
```

Funkcionál

- „*callable object*”
- A szál által futtatni kívánt függvényt többféleképpen is megadhatjuk:
  - függvénymutatóval (a függvény nevével),
  - tagfüggvény mutatóval
  - lambda kifejezésként
  - funktor - függvény objektummal

Példa

```
void msg_print(const std::string &msg){
 std::cout << msg << '\n';
}

class MessagePrint
{
private:
 int n;
public:
 MessagePrint(int _n = 1): n(_n){};
 void operator()(const std::string &msg) const {
  for (auto i=0; i<this->n; i++)
   std::cout << msg << '\n';
 }
 void msg_print(const std::string &msg) {
  std::cout << msg << '\n';
 }
 static void static_msg_print(const std::string &msg) {
  std::cout << msg << '\n';
 }
};
```

```cpp
int main() {
 std::string s{"Hello!"};
 //function name
 std::thread t1 = std::thread(msg_print, s);
 //function pointer
 void (*fnptr)(const std::string &);
 fnptr = &msg_print;
 std::thread t2 = std::thread(fnptr, s);
 //lambda
 auto l = [](const std::string &msg) {
  std::cout << msg << '\n';
 };
 std::thread t3 = std::thread(l, s);
 MessagePrint mp(3);
 //functor
 std::thread t4 = std::thread(mp, s);
 //member function
 std::thread t5 = std::thread(&MessagePrint::msg_print, mp, s);
 //static member function
 std::thread t6 = std::thread(MessagePrint::static_msg_print, s);
 t1.join(); t2.join(); t3.join();t4.join(); t5.join(); t6.join();
 return 0;
}
```

Eredmények visszatérítése

- Nincs mód a szálban futó függvény visszatérési értékének elérésére!
- Szükség esetén a függvényből referencia paraméterek segítségével nyerhetünk ki adatot
- *std::reference_wrapper*

```
void inc(int &n){
  n++;
}

int main() {
  int n = 7;
  std::thread t(inc, std::ref(n));
  t.join();
  std::cout << n << "\n";
  return 0;
}
```

Szálak ütemezésének befolyásolása

- Mind láttuk, az aktuális szálra az *std::this_thread* segítségével kaphatunk referenciát
- Segítségével:
  - javasolhatjuk, hogy vezérlés a száltól adódjon át más szálakhoz *yield()*
  - megadott ideig felfüggeszthetjük a szál futását *sleep_for(időtartam)*
  - adott időpontig is blokkolhatjuk a szál futását *sleep_until(időpont)*

```
#include <chrono>
#include <thread>
using namespace std::chrono;
int main() {
   std::this_thread::sleep_until(system_clock::now() + seconds(10));
}
```

Szálak futtatása

```
#include <iostream>
#include <thread>
void thread_function()
{
    std::cout << "Inside Thread :: ID  = " << std::this_thread::get_id() << std::endl;
}
int main()
{
    std::thread threadObj1(thread_function);
    std::thread threadObj2(thread_function);

    if(threadObj1.get_id() != threadObj2.get_id())
        std::cout << "Both Threads have different IDs" << std::endl;

    std::cout << "From Main Thread :: ID of Thread 1 = " << threadObj1.get_id() << std::endl;
    std::cout << "From Main Thread :: ID of Thread 2 = " << threadObj2.get_id() << std::endl;

    threadObj1.join();
    threadObj2.detach();
    return 0;
}
```

Szálspecifikus tárolás

- Egy adott szálra nézve lehessenek statikus vagy globális változóink
  - Szálanként külön másolat
  - ”Globális” változó név
- C: pthread_key_create
- Java: java.lang.ThreadLocal<T>
- C++: thread_local
- Első használatkor jön létre
  - First touch

```cpp
thread_local int i = 0;
void bar(int id){
 i += id;
}
void foo(int id) {
 bar(id);
 std::cout << std::addressof(i) << "\n";
 std::cout << i << "\n";
}
int main() {
 i = 42;
 std::thread t1(foo, 1);
 std::thread t2(foo, 2);

 t1.join(); t2.join();
 std::cout << std::addressof(i) << "\n";
 std::cout << i << "\n";
 return 0;
}
```

```
0x7fd4ec402b20
1
0x7fd4ec500000
2
0x7fd4ec400710
42
```