

Paralelismus v C++11

PV173 Programování v C++11

Vladimír Štill, Jiří Weiser

Fakulta Informatiky, Masarykova Univerzita

27. října 2014

- Paralelizmus obecně
- Paralelizmus v STL
- Paměť a paralelizmus

Poznámka: v kódech na slidech je u konstruktorů a destruktorech vynechán název třídy. Je tu málo místa.

Paralelizmus obecně

“Souběžné provádění více funkcí v daném časovém intervalu”

- Ve sdílené paměti
 - V distribuovaném prostředí
 - Nebudeme se zabývat – C++11 nemá nástroje
-

Paralelizmus obecně

“Souběžné provádění více funkcí v daném časovém intervalu”

- Ve sdílené paměti
- V distribuovaném prostředí
 - Nebudeme se zabývat – C++11 nemá nástroje

-
- Komunikační primitiva
 - Kritická sekce (mutex)
 - Synchronizace vláken (podmínkové proměnné)
 - Asynchronní volání (futures)

Paralelismus v STL

`std::thread`

Třída `std::thread` (hlavička `<thread>`).

`std::thread();` *// nic*

`std::thread(function, args...);` *// spustí funkci*

`void std::thread::join();` *// počká na vlákno*

`void std::thread::detach();` *// odpojí vlákno*

Paralelismus v STL

Mutex

Základní zámek splňující koncept *Lockable* – třída `std::mutex` (hlavička `<mutex>`).

```
void std::mutex::lock(); // zamkne mutex  
bool std::mutex::try_lock(); // zkusí zamknout  
void std::mutex::unlock(); // odemkne
```

Existují i složitější typy zámků – časové, rekurzivní.

Paralelismus v STL

Mutex a RAII

Obecné ovládání mutexu (RAII princip) – třídy `std::lock_guard` a `unique_lock` (hlavička `<mutex>`).

- Třída `std::lock_guard` je jednoduchá.
- Třída `std::unique_lock` umožňuje lepší ovládání (i složitějších) zámků.

```
std::lock_guard( Lockable &m ); // zamkne m  
std::~lock_guard(); // odemkne m
```

Paralelismus v STL

Mutex a deadlock

Zamykání více mutexů bez uváznutí (hlavička <mutex>).

// zamkne všechny zámky

```
void std::lock( Lockable1 &m1, Lockable2 &m2,  
               LockableN... );
```

// zkusí zamknout všechny, pokud se nepodaří,

// vše uvolní

```
int std::try_lock( Lockable &m1, Lockable &m2,  
                  LockableN... );
```


Paralelismus v STL

Podmínkové proměnné

Synchronizace vláken – třída `std::condition_variable` (hlavička `<condition_variable>`).

```
using Lock = std::unique_lock< std::mutex >;
```

```
// umí také pracovat s časem
```

```
void std::condition_variable::wait( Lock &l  
    [, Predicate p ] );
```

```
void std::condition_variable::notify_one();
```

```
void std::condition_variable::notify_all();
```

Paralelismus v STL

Futures

Pro práci s asynchronními běhy funkcí jsou k dispozici následující třídy a funkce (hlavička <future>)

```
// umožní získat hodnotu z asynchronního volání  
// je vytvářeno pomocí níže uvedených tříd / fci  
template< typename T > class std::future;
```

```
// spustí asynchronně funkci  
std::future< ... > std::async( Fce, Args... );
```

```
// asynchronní obal na cokoliv, co lze zavolat  
template< ... > class std::packaged_task;
```

```
// umožní získat parciální výsledek  
template< ... > class std::promise;
```

Paměť a paralelismus

- Přístup ke sdílené paměti ze dvou vláken současně je nedefinované chování.
- Přístupovat ke sdílené paměti lze (zatím pro nás) pouze za použití mutexu.
 - Další možnosti si ukážeme příště.
- Označení paměti jako **volatile** situaci nezachrání.
 - Operace **i++** NENÍ atomická.
 - **volatile** neříká nic o ostatních proměnných.
 - V případě Visual Studia lze **volatile** použít na synchronizaci vláken.
 - Ale jenom někdy...

Samostatné programování

Fronta se zámky

Implementujte sdílenou frontu se zámky.

- Fronta bude schopná uchovat prvky typu **T**.
- Bude schopná vložit prvek na konec fronty.
- Bude schopná odebrat prvek ze začátku fronty.
- Bude bezpečná při paralelním běhu.
- Hint: **std::mutex**.

Samostatné programování

Thread pool

Implementujte množinu vláken, které budou čekat na nově příchozí činnosti. Nové činnosti se budou vkládat do fronty. Volná vlákna si budou činnosti brát z fronty a spouštět je.

- Činnost je cokoliv, co se dá zavolat bez parametrů a nevrací to žádnou hodnotu.
- Nezapoměňte implementovat destruktory.
- Hint: **std::unique_ptr**, **std::thread**, **std::condition_variable**