

Lock-free programování

PB173 Programování v C++11

Vladimír Štill, Jiří Weiser

Fakulta Informatiky, Masarykova Univerzita

3. listopadu 2014

- Atomické operace obecně
- Paměťové bariéry
- **std::atomic, std::atomic_flag**
 - Synchronizace paměti
- Lock-free programování

Atomické operace obecně

- Někdy je potřeba provést atomicky operaci, která se skládá z více sub-operací
 - `++i`

Atomické operace obecně

- Někdy je potřeba provést atomicky operaci, která se skládá z více sub-operací
 - `++i`
- Lze použít mutex
 - Součinnost s plánovačem (obvykle)
 - Přepínání kontextu
 - Obecně je mutex drahý

Atomické operace obecně

- Někdy je potřeba provést atomicky operaci, která se skládá z více sub-operací
 - ++i
- Lze použít mutex
 - Součinnost s plánovačem (obvykle)
 - Přepínání kontextu
 - Obecně je mutex drahý
- Mnohé architektury umožňují provést některé sekvence operací atomicky
 - C++11 využívá této možnosti → **std::atomic**

Paměťové bariéry

Nevýhody atomických operací

- Atomické operace lze vykonávat pouze nad omezenou množinou typů
 - ukazatele
 - **bool**, (**unsigned**) **int**, **long**, **short**, ...
 - vlastní třídy (za určitých podmínek)

Paměťové bariéry

Nevýhody atomických operací

- Atomické operace lze vykonávat pouze nad omezenou množinou typů
 - ukazatele
 - **bool**, (**unsigned**) **int**, **long**, **short**, ...
 - vlastní třídy (za určitých podmínek)
- Operace jsou časově náročnější
 - Samotný procesorový čas je vyšší
 - Je třeba synchronizovat paměť (včetně cache procesorů)

Paměťové bariéry

Nevýhody atomických operací

- Atomické operace lze vykonávat pouze nad omezenou množinou typů
 - ukazatele
 - **bool**, (**unsigned**) **int**, **long**, **short**, ...
 - vlastní třídy (za určitých podmínek)
- Operace jsou časově náročnější
 - Samotný procesorový čas je vyšší
 - Je třeba synchronizovat paměť (včetně cache procesorů)
- → používají se pouze pro synchronizaci vláken a řízení komunikace

Paměťové bariéry

Nevýhody atomických operací

- Atomické operace lze vykonávat pouze nad omezenou množinou typů
 - ukazatele
 - **bool**, (**unsigned**) **int**, **long**, **short**, ...
 - vlastní třídy (za určitých podmínek)
- Operace jsou časově náročnější
 - Samotný procesorový čas je vyšší
 - Je třeba synchronizovat paměť (včetně cache procesorů)
- → používají se pouze pro synchronizaci vláken a řízení komunikace
- Ale co rozsáhlejší komunikace vláken?

Paměťové bariéry

Nevýhody atomických operací

- Atomické operace lze vykonávat pouze nad omezenou množinou typů
 - ukazatele
 - **bool**, (**unsigned**) **int**, **long**, **short**, ...
 - vlastní třídy (za určitých podmínek)
- Operace jsou časově náročnější
 - Samotný procesorový čas je vyšší
 - Je třeba synchronizovat paměť (včetně cache procesorů)
- → používají se pouze pro synchronizaci vláken a řízení komunikace
- Ale co rozsáhlejší komunikace vláken?
- Jak synchronizovat neatomické proměnné?

Paměťové bariéry

Sekvenční sémantika

- Standard C++ (nejen C++11) dodržuje sekvenční sémantiku.
- Stejně tak i procesory.

Paměťové bariéry

Sekvenční sémantika

- Standard C++ (nejen C++11) dodržuje sekvenční sémantiku.
- Stejně tak i procesory.
- **Ovšem neznamená to, že budou proměnné zapsány do RAM v takovém pořadí, jak jsou zapsány ve zdrojovém kódu!**
 - → sekvenční sémantika je nedostačující pro provoz v paralelním prostředí.

Paměťové bariéry

- Existují instrukce pro uspořádání zápisů a čtení.
- Instruuji překladač a procesor, aby nepřesouvaly čtení/zápis a synchronizovaly paměť
 - Existuje více typů takovýchto instrukcí.

Paměťové bariéry

- Existují instrukce pro uspořádání zápisů a čtení.
- Instruuují překladač a procesor, aby nepřesouvaly čtení/zápis a synchronizovaly paměť
 - Existuje více typů takovýchto instrukcí.
- Tyto instrukce jsou obsaženy při obsluhování mutexů či podmínkových proměnných.

Paměťové bariéry

Seznam přístupů

```
namespace std {  
enum memory_order {  
    memory_order_relaxed, // žádná bariéra  
    // vlákno uvidí aktuální závislé hodnoty  
    memory_order_consume,  
    // vlákno uvidí aktuální hodnoty  
    memory_order_acquire, // načtení  
    // vlákno zpřístupní zapsané hodnoty všem  
    memory_order_release, // zápis  
    memory_order_acq_rel, // acquire+release  
    // acquire+release+totální uspořádání  
    memory_order_seq_cst // default  
}; }
```

`std::memory_order_seq_cst` je defaultně použité dále.

V hlavičkovém souboru `<atomic>` najdete kromě výše uvedeného výčtu též následující nástroje pro atomickou práci s pamětí

- `std::atomic_flag`
 - Garance lock-free funkcionality.
- `template< typename T > std::atomic`
- a pár dalších funkcí...

STL knihovna

std::atomic_flag

// inicializace do nspecifikovaného stavu

```
std::atomic_flag::atomic_flag();
```

// nastaví atomicky na false

```
void std::atomic_flag::clear( memory_order );
```

// nastaví na true a vrátí předchozí hodnotu

```
bool std::atomic_flag::test_and_set(  
    memory_order );
```

// jediná možnost inicializace na false

```
std::atomic_flag flag = ATOMIC_FLAG_INIT;
```

Další metody/operátory/konstruktory to nemá.

STL knihovna

std::atomic

```
T std::atomic< T >::operator=( T t ); // = store
T std::atomic< T >::operator T(); // = load

std::atomic< T >::store( T, memory_order );
T std::atomic< T >::load( memory_order );

T std::atomic< T >::exchange( T, memory_order );

bool std::atomic< T >::compare_exchange_weak(
    T &expected, T desired, memory_order );
bool std::atomic< T >::compare_exchange_strong(
    T &expected, T desired, memory_order );
```

Lock-free programování

- Paralelní programování, kdy se nepoužívají klasické mutexy.
- Veškerá synchronizace se provádí pomocí atomických operací.

Lock-free programování

- Paralelní programování, kdy se nepoužívají klasické mutexy.
- Veškerá synchronizace se provádí pomocí atomických operací.
- V případě krátkých a častých komunikací mezi vlákny vhodnější než mutexy.

Lock-free programování

- Paralelní programování, kdy se nepoužívají klasické mutexy.
- Veškerá synchronizace se provádí pomocí atomických operací.
- V případě krátkých a častých komunikací mezi vlákny vhodnější než mutexy.
- Složitější na programování, ještě složitější na testování
 - Unit testy nemusí odhalit chybu.
 - Formální verifikace je příliš drahá.

Lock-free programování

- Lock-free algoritmy se většinou skládají z
 - cyklů
 - atomických exchange operací
 - atomických CAS operací
- Spin lock
 - Mutex bez využití systémových nástrojů.
 - Čekající vlákno není suspendováno.
 - Nevýhodný, pokud je více vláken než procesorů.

Lock-free programování

- Lock-free algoritmy se většinou skládají z
 - cyklů
 - atomických exchange operací
 - atomických CAS operací
- Spin lock
 - Mutex bez využití systémových nástrojů.
 - Čekající vlákno není suspendováno.
 - Nevýhodný, pokud je více vláken než procesorů.
- Použitím spin locku se již nejedná o lock-free algoritmus
 - Porušuje to jednu z podmínek definice lock-free algoritmu.

Lock-free programování

Spin lock

```
class SpinLock {  
    std::atomic_flag _flag;  
public:  
    SpinLock() { _flag.clear(); }  
    ~SpinLock() {  
        assert( !_flag.test_and_set() );  
    }  
    void lock() {  
        while( _flag.test_and_set() );  
    }  
    void unlock() { _flag.clear(); }  
};
```


Samostatné programování

- Implementujte jednoduchou frontu.
- Fronta musí být funkční v paralelním běhu
- Nesmíte používat mutexy a další synchronizační nástroje
 - Až na nástroje z <atomic>.
- Fronta bude podporovat operace
 - vložení na konec
 - odebrání ze začátku
- Použijte algoritmus z publikace
http://www.cs.rochester.edu/~scott/papers/1996_PODC_queues.pdf.