C++
Threads

Barbara Borowy

Jolanta Lachman

`thread/jthread`

- Klasa `std::thread` jest odpowiedzialna za tworzenie obiektów, które uruchamiają wątki i zarządzają nimi
- W konstruktorze podawana jest wykonywana przez dany wątek funkcja i jej parametry
- Przekazyjąc jako parametr referencję należy użyć std::ref
- Obiekty wątków nie są kopiowalne. Wątki mogą być jednak przenoszone między obiektami (fukcja `std::move()`)

mutex

- Umożliwia ochronę współdzielonych zmiennych lub miejsc w pamięci przed równoczesnym dostępem przez kilka wątków
- Wątek pozyskuje muteks wywołując metodę `lock()`.
- Wątek zwalnia muteks wywołując metodę `unlock()`

`mutex` - przykład użycia

```cpp
std::map<std::string, std::string> g_pages;
std::mutex g_pages_mutex;

void save_page(const std::string &url)
{
    // simulate a long page fetch
    std::this_thread::sleep_for(std::chrono::seconds(2));
    std::string result = "fake content";

    std::lock_guard<std::mutex> guard(g_pages_mutex);
    g_pages[url] = result;
}

int main()
{
    std::thread t1(save_page, "http://foo");
    std::thread t2(save_page, "http://bar");
    t1.join();
    t2.join();

    // safe to access g_pages without lock now, as the threads are joined
    for (const auto &pair : g_pages) {
        std::cout << pair.first << " => " << pair.second << '\n';
    }
}
```

atomic

- Typ danych enkapsulujący pewną wartość, a dostęp do nich nie będzie powodował wyścigu.
- Operacje wykonywane na zmiennych atomowych:
  - są niepodzielne - żaden inny wątek nie może zobaczyć pośredniego stanu operacji atomowej,
  - wprowadzają mechanizm synchronizujący - nie powodują wyścigu,
  - ostrzegają kompilator przed potencjalnym wyścigiem - w rezutacie kompilator rezygnuje z niebezpiecznych w takim kontekście optymalizacji.

`atomic` - przykład użycia

```
std::atomic_int foo (0);

void set_foo(int x) {
  foo.store(x,std::memory_order_relaxed);     // set value atomically
}

void print_foo() {
  int x;
  do {
    x = foo.load(std::memory_order_relaxed);  // get value atomically
  } while (x==0);
  std::cout << "foo: " << x << '\n';
}

int main ()
{
  std::thread first (print_foo);
  std::thread second (set_foo,10);
  first.join();
  //second.join();
  return 0;
}
//output: foo: 10
```

condition_variable

- Zmienne warunkowe używane są do powiadamiania wątków o zaistnieniu określonego warunku
- Są powiązane z muteksem, który jest ponownie pozyskiwany w momencie wybudzenia z blokady
- Warunek wybudzenia z blokady jest zdefiniowany przy pomocy predykatu, który jest parametrem blokady

condition_variable
- przykład użycia

```cpp
std::mutex m;
std::condition_variable cv;
std::string data;
bool ready = false;
bool processed = false;

void worker_thread()
{
    // Wait until main() sends data
    std::unique_lock lk(m);
    cv.wait(lk, []{return ready;});

    // after the wait, we own the lock.
    std::cout << "Worker thread is processing data\n";
    data += " after processing";

    // Send data back to main()
    processed = true;
    std::cout << "Worker thread signals data processing completed\n";

    // Manual unlocking is done before notifying, to avoid waking up
    // the waiting thread only to block again (see notify_one for details)
    lk.unlock();
    cv.notify_one();
}

int main()
{
    std::thread worker(worker_thread);

    data = "Example data";
    // send data to the worker thread
    {
        std::lock_guard lk(m);
        ready = true;
        std::cout << "main() signals data ready for processing\n";
    }
    cv.notify_one();

    // wait for the worker
    {
        std::unique_lock lk(m);
        cv.wait(lk, []{return processed;});
    }
    std::cout << "Back in main(), data = " << data << '\n';

    worker.join();
}
```

std::this_thread::yield

- Rezygnuje z czasu procesora na rzecz innych wątków.

```
#include <iostream>       // std::cout
#include <thread>         // std::thread, std::this_thread::yield
#include <atomic>         // std::atomic

std::atomic<bool> ready (false);

void count1m(int id) {
  while (!ready) {              // wait until main() sets ready...
    std::this_thread::yield();
  }
  for (volatile int i=0; i<1000000; ++i) {}
  std::cout << id;
}

int main ()
{
  std::thread threads[10];
  std::cout << "race of 10 threads that count to 1 million:\n";
  for (int i=0; i<10; ++i) threads[i]=std::thread(count1m,i);
  ready = true;                 // go!
  for (auto& th : threads) th.join();
  std::cout << '\n';

  return 0;
}
```

Źródła

- https://infotraining.bitbucket.io/cppthd/synchronizacja_watkow.html
- https://en.cppreference.com/w/cpp/thread
- https://en.cppreference.com/w/cpp/thread/mutex
- https://en.cppreference.com/w/cpp/atomic/atomic
- https://en.cppreference.com/w/cpp/thread/condition_variable