

# Podstawy programownia (w języku C++)

## Wątki

Marek Marecki

Polsko-Japońska Akademia Technik Komputerowych

13 grudnia 2020

# OVERVIEW

Wstęp

Wątki w C++

Komunikacja między wątkami

Podsumowanie

# PO CO WĄTKI W C++?

Wątki w C++ są narzędziem, które można wykorzystać do kilku celów. Nie znaczy to, że są najlepszym możliwym rozwiązaniem! Są jedynie podstawą, która umożliwia stworzenie i wykorzystanie bardziej eleganckich mechanizmów.

Wątki są również sposobem na wprowadzenie do programów **współbieżności** (ang. *concurrency*) i **równoległości** (ang. *parallelism*).

## WSPÓŁBIEŻNOŚĆ (ANG. CONCURRENCY)

Wykonywanie przez program kilku zadań naraz (ale niekoniecznie *jednocześnie*).

Iluzja wykonywania przez program kilku zadań naraz może być zapewniona, dzięki szybkiemu przełączaniu się między częściowo wykonanymi zadaniami.

Przykład: program kopiujący  $k$  plików. Program może skopiować plik  $n$  w całości, po czym przejść do kopiowania pliku  $n + 1$  i działać sekwencyjnie. Może też kopiować fragmenty plików od  $n$  do  $n + k$  i działać współbieżnie (dzięki temu małe pliki mogą być skopiowane w całości szybciej).

## RÓWNOLEGŁOŚĆ (ANG. PARALLELISM)

Wykonywanie przez program kilku zadań *jednocześnie* (w tym samym kwancie czasu).

Równoległość możliwa jest do osiągnięcia dzięki wykorzystaniu w programie wielu procesorów (na tym samym fizycznym komputerze, lub w systemie rozproszonym).

Przykład: program kopiujący  $k$  plików. Program może podzielić listę plików do skopiowania na  $n$  procesorów, i każdemu przekazać do obsłużenia  $k/n$  plików.

# WSPÓŁBIEŻNOŚĆ VS RÓWNOLEGŁOŚĆ

Współbieżność *nie zawsze* oznacza równoległość.

Równoległość *zawsze* oznacza współbieżność.

Współbieżność jest sposobem na modelowanie zachowania i podział zadań wewnątrz programu. Równoległość jest sposobem na pełniejsze wykorzystanie zasobów systemu.

# OVERVIEW

Wstęp

Wątki w C++

Komunikacja między wątkami

Podsumowanie

## STD::THREAD

### WĄTEK CZYLI STD::THREAD

Wątki tworzy się za pomocą typu `std::thread` z nagłówka `<thread>`.

Utworzenie wątku oznacza rozpoczęcie współbieżnego wykonywania funkcji, która została przekazana do konstruktora typu `std::thread`

```
auto a_thread = std::thread{
    some_function
    , 42
    , "Hello, World!"
};
```



# „DOŁĄCZANIE” WĄTKÓW

STD::THREAD

Za pomocą funkcji składowej `std::thread::join` można „dołączyć” wątek reprezentowany jakąś zmienną do wątku, z którego ta funkcja jest wywołana.

```
auto errand_boy = std::thread{run_some_errands, std::move(errands)};  
errand_boy.join(); // wait until the errand boy returns...
```

Wątek wywołujący funkcję `std::thread::join` jest zablokowany do momentu aż wątek „dołączany” nie zakończy działania.

# „ODCINANIE” WĄTKÓW

STD::THREAD

Za pomocą funkcji składowej `std::thread::detach` można „odciąć” wątek i uniemożliwić jego dołączenie do innego wątku. Pozwala to odciętemu wątkowi działać nawet jeśli żaden inny wątek nie posiada jego „uchwyty”<sup>1</sup>:

```
auto fire_and_forget = std::thread{launch_missile, std::move(target)};
fire_and_forget.detach(); // too late to change our mind now...
```

---

<sup>1</sup>*thread handle*

# ZADANIE: DRUKOWANIE NAPISÓW

STD: : THREAD

Napisz program, który uruchomi 42 wątki drukujące napis „Hello, X!”.  
X musi być różne dla każdego wątku (mogą to być różne imiona, różne liczby, itp.).  
Wykorzystaj funkcję `std::thread::detach`.

Kod w pliku `src/s05-print-thread.cpp`

## ZADANIE: DRUKOWANIE GRUP NAPISÓW

STD : : THREAD

Napisz program, który uruchomi 42 wątki drukujące napis „Hello, X!”.  
X musi być różne dla każdego wątku (mogą to być różne imiona, różne liczby, itp.).  
Wątki mają być uruchamiane grupami po 6 naraz. Wykorzystaj funkcję  
`std::thread::join` do czekania na zakończenie wątku.

Kod w pliku `src/s05-print-thread-group.cpp`

## SEKCJE KRYTYCZNE

Sekcja krytyczna to fragment kodu, który może być wykonywany przez jeden wątek naraz. Takie fragmenty kodu powinny być zabezpieczone przed współbieżnym wykonaniem.

### **Jak rozpoznać, że coś jest sekcją krytyczną?**

Sekcja krytyczna dotyczy (manipuluje, odczytuje, itd.) danych, które widoczne są z potencjalnie wielu wątków.

# ZABEZPIECZENIE SEKCJI KRYTYCZNYCH

## SEKCJE KRYTYCZNE

Najprościej jest wykorzystać typy `std::mutex2` do zabezpieczenia danych, oraz `std::lock_guard` lub `std::unique_lock` do zablokowania wątku w oczekiwaniu na dostęp do sekcji krytycznej.

Zabezpieczenie danych:

```
auto foo = some_type{};
std::mutex foo_mtx; // mutex for foo
```

Oczekiwanie na dostęp do sekcji krytycznej:

```
std::lock_guard<std::mutex> lck { foo_mtx };
// or...
std::unique_lock<std::mutex> lck { foo_mtx };
```

---

<sup>2</sup>*mutex* od *mutual exclusion*

# ZADANIE: LISTA ZADAŃ

STD : : THREAD

Napisz program, w którym uruchomisz cztery wątki pobierające ze wspólnej<sup>3</sup> kolejki `std::queue` liczby do wydrukowania. Pobieranie liczb powinno być sekcją krytyczną.

Każdemu z wątków przydziel ID (przekazywane jako argument podczas tworzenia wątku). Drukowane linie powinny mieć następujący format:

```
from thread THREAD-ID: NUMBER-TO-PRINT
```

Niech wątki zakończą działanie kiedy wykryją, że wektor z liczbami do wydrukowania jest pusty.

Kod w pliku `src/s05-job-queue.cpp`

---

<sup>3</sup>potrzebna będzie tu referencja lub wskaźnik

# OVERVIEW

Wstęp

Wątki w C++

Komunikacja między wątkami

Podsumowanie



# ZADANIE: ŚPIĄCE WĄTKI

STD: : THREAD

Uruchom 4 wątki pobierające rzeczy do wydrukowania ze wspólnej kolejki (jak w poprzednim zadaniu). Niech wątki kończą pracę kiedy dostaną pusty `std::string` i drukują `thread X exiting` kiedy kończą pracę. Kiedy wątek wykryje, że kolejka jest pusta niech zwolni mutex i pójdzie spać na losową wartość milisekund między 10, a 100.

W funkcji `main()` odczytuj napisy ze standardowego strumienia wejścia i odczytane napisy odkładaj na kolejkę, z której czytają wątki drukujące. Zakończ pętlę po otrzymaniu 4 pustych napisów.

Kod w pliku `src/s05-sleeping-threads.cpp`

# OCZEKIWANIE NA, I SYGNALIZACJA WARUNKU

STD::CONDITION\_VARIABLE

Kod taki jak w zadaniu ze śpiącymi wątkami jest nieefektywny, mało czytelny, i - powiedzmy to jasno - kiepski. Jeśli *oczekujemy* na wystąpienie jakiegoś warunku (np. nadejście zadania) to dużo lepszy jest kod, który realizuje to w sposób jawny.

Typ `std::condition_variable` z nagłówka `<condition_variable>` pozwala to zrealizować w prosty sposób.

# PRZYGOTOWANIE

STD::CONDITION\_VARIABLE

Żeby móc oczekiwać na warunek potrzebujemy następujących rzeczy:

1. zmiennej typu `std::condition_variable`, której będziemy używać do oczekiwania i sygnalizacji warunku
2. zmiennej typu `std::mutex`, na której będziemy blokować nasz `condition variable`
3. dodatkowej zmiennej przechowującej dane, które powinniśmy sprawdzić po zejściu warunku (np. kolejki rzeczy do wydrukowania)

# PRZYGOTOWANIE (C.D.)

STD::CONDITION\_VARIABLE

```
// this is our communication channel
std::queue<std::string> items_to_print;

// the mutex protects it from concurrent access
std::mutex mtx;

// the condition_variable is used to signal
// arrival of new data
std::condition_variable cv;
```

# OCZEKIWANIE NA ZAJŚCIE WARUNKU

STD::CONDITION\_VARIABLE

Oczekiwanie na zajęcie warunku jest realizowane funkcjami składowymi  
`std::condition_variable::wait` (czekanie w nieskończoność) i  
`std::condition_variable::wait_for` (czekanie przez określony czas).

Obie funkcje żądają przekazania im jako argumentu wartości typu  
`std::unique_lock` zablokowanej na muteksie, który chroni dane będące kanałem komunikacji:

```
// lock the mutex
std::unique_lock<std::mutex> lck { mtx };
```

## OCZEKIWANIE NA ZAJŚCIE WARUNKU (C.D.)

STD::CONDITION\_VARIABLE

Jeśli jesteśmy w stanie pozwolić sobie na czekanie w nieskończoność możemy użyć prostego wait:

```
// unlock the mutex, block calling thread, and
// wait for a notification (ad inifinitum)
cv.wait(lck);
```

Jeśli jednak w określonym czasie musimy podjąć jakieś działanie niezależnie od tego czy otrzymaliśmy komunikat czy nie, należy użyć wait\_for:

```
// unlock the mutex, block calling thread, and
// wait for a notification for about 1 second
cv.wait(lck, std::chrono::seconds{1});
```

# SYGNALIZACJA WARUNKU

STD::CONDITION\_VARIABLE

```
// lock the mutex and modify communication channel...
{
    std::lock_guard<std::mutex> lck { mtx };
    items_to_print.push("Hello, World!");
}

// notify just one of the waiting threads...
cv.notify_one();

// ...or notify all of them
cv.notify_all();
```

## ZADANIE: ŚPIĄCE WĄTKI 2

STD: : THREAD

Uruchom 4 wątki pobierające rzeczy do wydrukowania ze wspólnej kolejki (jak w poprzednim zadaniu). Niech wątki kończą pracę kiedy dostaną pusty `std::string` i drukują `thread X exiting` kiedy kończą pracę.

Niech wątki oczekują na rzeczy do wydrukowania z wykorzystaniem funkcji `std::condition_variable::wait`.

W funkcji `main()` odczytuj napisy ze standardowego strumienia wejścia i odczytane napisy odkładaj na kolejkę, z której czytają wątki drukujące. Zakończ pętlę po otrzymaniu 4 pustych napisów.

Kod w pliku `src/s05-sleeping-threads-2.cpp`



# ZADANIE: ŚPIĄCE WĄTKI 3

STD: :THREAD

Tak jak w „śpiące wątki 2”, ale z użyciem typu `itp::channel` z repozytorium szablonowego.

Kod w pliku `src/s05-sleeping-threads-3.cpp`

# ZADANIE: PING-PONG

STD : : THREAD

Uruchom dwa wątki - jeden z identyfikatorem „ping”, drugi „pong”. Niech oba wątki wymieniają się liczbą całkowitą „odbijając” ją między sobą. Wątek powinien relizować w pętli następującą logikę:

1. czekać na komunikat o otrzymaniu liczby
2. wydrukować liczbę poprzedzoną swoim identyfikatorem (np. ping 7)
3. zwiększyć liczbę o losową wartość między 1, a 42
4. poinformować drugi wątek o tym, że teraz jego kolej
5. jeśli liczba była większa niż 1024 zakończyć działanie

Pierwszą liczbę wyślij do wątku „ping” z funkcji main().

Kod w pliku src/s05-ping-pong.cpp

# OVERVIEW

Wstęp

Wątki w C++

Komunikacja między wątkami

Podsumowanie

# PODSUMOWANIE

Student powinien umieć:

1. wykorzystać wątki do wprowadzenie współbieżności i równoległości w programie
2. wytłumaczyć do czego służą typy `std::mutex`, `std::unique_lock`, i `std::lock_guard`
3. wytłumaczyć czym jest sekcja krytyczna

# ZADANIA

## PODSUMOWANIE

Zadania znajdują się na slajdach 11, 12, 15, 17, 24, 25, 26.