

Programowanie współbieżne i rozproszone	PWIR02	
05.04.2022	P2	Szymon Zwoliński

KOD 1:

1_1_cr_new_thread.cpp

```
#include <stdio>
#include <thread>
#include <windows.h>

int action(int id){
    printf("Uruchamiam watek %d\n", id);
    Sleep(10*1000); //10 sekund
    printf("Koncze watek %d\n", id);
    return 0;
}

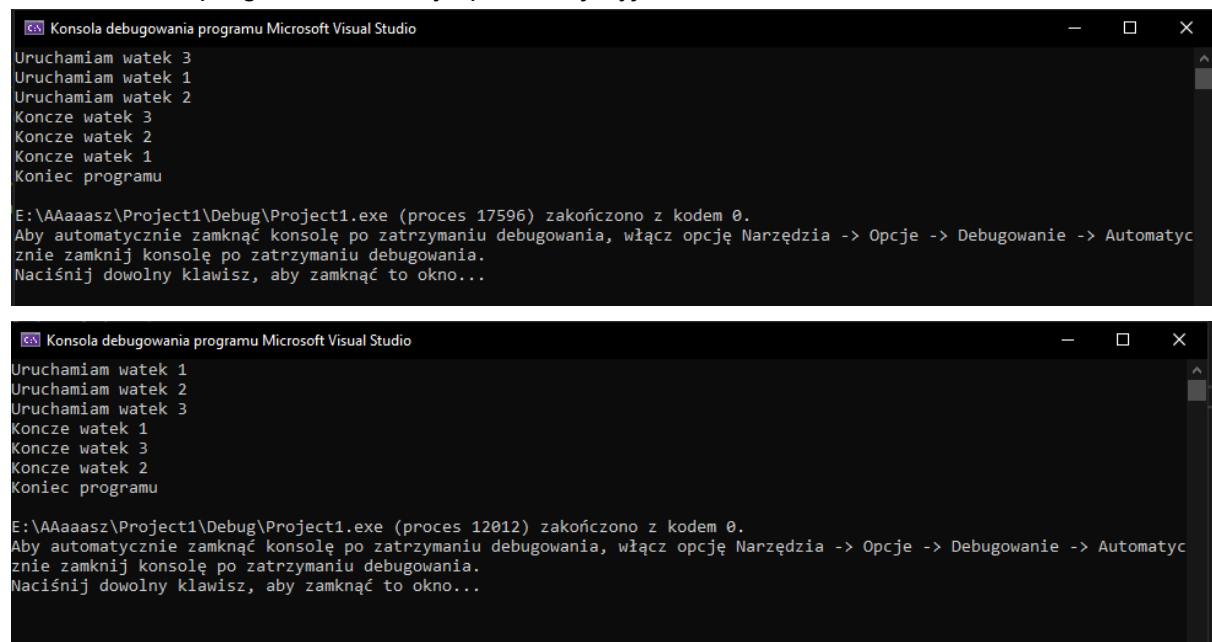
int main(){
    //tworzenie watku
    std::thread t1(action, 1); //konstruktor klasy t1 przyjmuje minimum wskaźnik na funkcje do wykonania
    std::thread t2(action, 2); //funkcja ta może coś zwracać, ale może zwracać też void
    std::thread t3(action, 3); //dalsze parametry zostaną przekazane do podanej funkcji

    t1.join(); //synchronizacja
    t2.join(); //watek główny ma tu poczekać na te 3 wątki
    t3.join(); //inaczej program by się zakończył wcześniej bo wątki trwają minimum 10 sekund

    printf("Koniec programu \r\n");

    return 0;
}
```

1.1. Uruchom program wiele razy i porównaj wyjście.:



```
Konsola debugowania programu Microsoft Visual Studio

Uruchamiam watek 3
Uruchamiam watek 1
Uruchamiam watek 2
Koncze watek 3
Koncze watek 2
Koncze watek 1
Koniec programu

E:\AAAAasz\Project1\Debug\Project1.exe (proces 17596) zakończono z kodem 0.
Aby automatycznie zamknąć konsolę po zatrzymaniu debugowania, włącz opcję Narzędzia -> Opcje -> Debugowanie -> Automatycznie zamknij konsolę po zatrzymaniu debugowania.
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

```
Konsola debugowania programu Microsoft Visual Studio

Uruchamiam watek 1
Uruchamiam watek 2
Uruchamiam watek 3
Koncze watek 1
Koncze watek 3
Koncze watek 2
Koniec programu

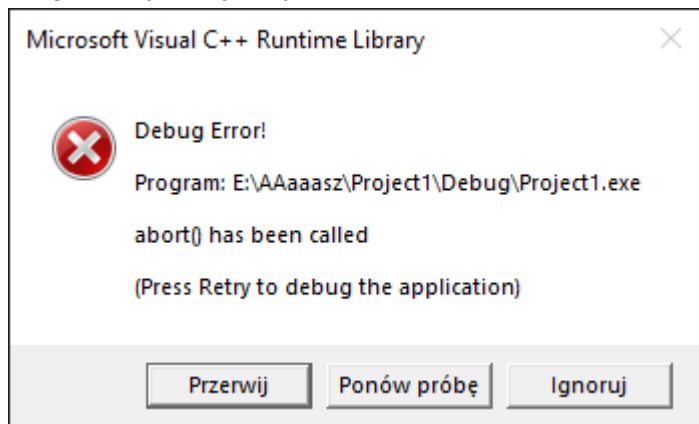
E:\AAAAasz\Project1\Debug\Project1.exe (proces 12012) zakończono z kodem 0.
Aby automatycznie zamknąć konsolę po zatrzymaniu debugowania, włącz opcję Narzędzia -> Opcje -> Debugowanie -> Automatycznie zamknij konsolę po zatrzymaniu debugowania.
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

Wątki nie uruchamiają się po kolei, oraz nie muszą kończyć się w kolejności w jakiej zostają uruchomione.

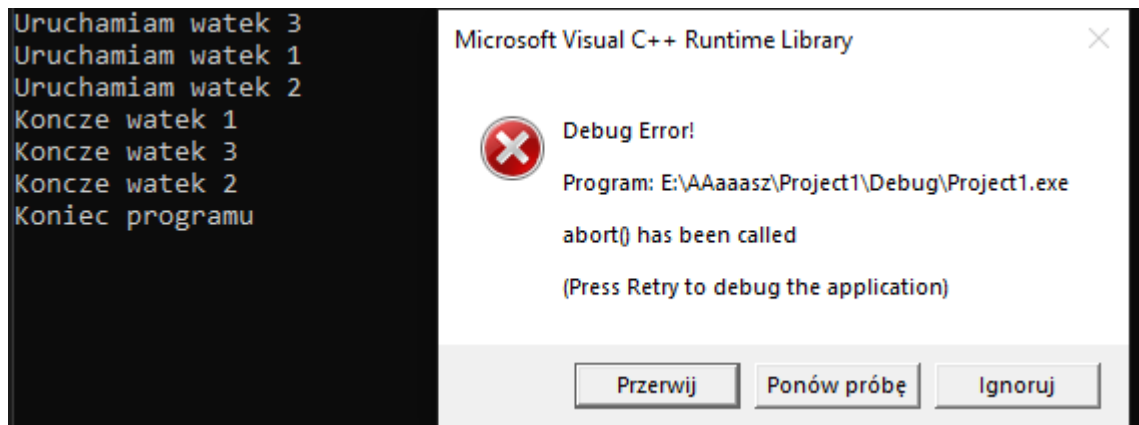
1.2. Usuń wszystkie 3 joiny, skompiluj i sprawdź wyjście:

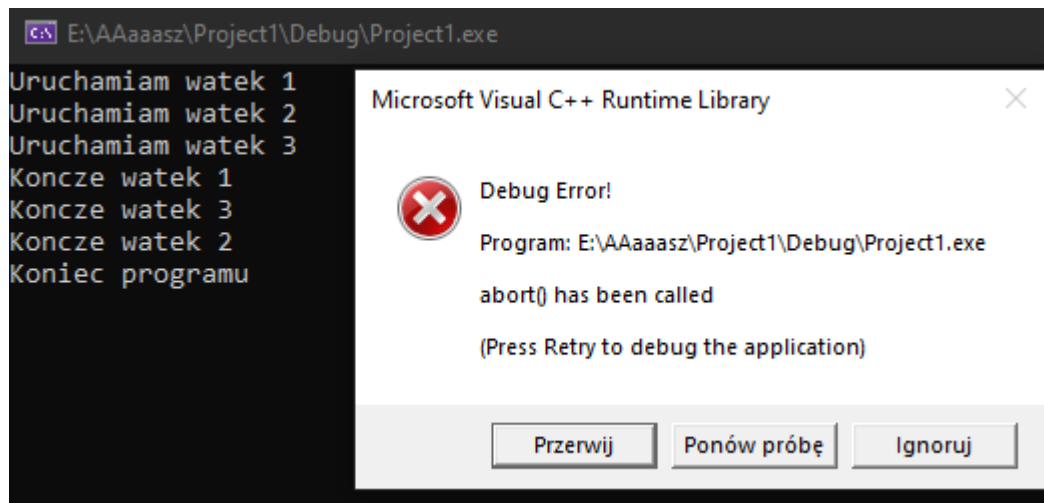
```
Koniec programu
Uruchamiam watek 1
Uruchamiam watek 2
Uruchamiam watek 3
Koncze watek 1
Koncze watek 3
Koncze watek 2
```

Program wykonuje się z ostrzeżeniami, oraz zwraca abort()



1.3 Usuń tylko jeden join, skompiluj i sprawdź co zostanie wyświetlone (najlepiej więcej niż raz).





Program działa tak samo jak w podpunkcie 1.2.

1.4. Zmodyfikuj program tak by wypisywane było id uzyskane z funkcji `get_id()` :

```
C:\> Konsola debugowania program
Uruchamiam watek 3
Uruchamiam watek 1
Uruchamiam watek 2
Koncze watek 3
id watku: 15376
Koncze watek 1
id watku: 10852
Koncze watek 2
id watku: 10548
Koniec programu
```

```
int action(int id) {
    printf("Uruchamiam watek %d\n", id);
    Sleep(10 * 1000); //10 sekund
    printf("Koncze watek %d\n", id);
    std::cout << "id watku: " << std::this_thread::get_id() << std::endl;
    return 0;
}
```

Id wątku uzyskiwane jest poprzez `get_id()`.

1.5. Zmodyfikuj funkcje `action` w taki sposób by czas uśpienia dostawał jako parametr, następnie przetestuj zmiany. Sprawdź:

- w jakiej kolejności zostaną zamknięte wątki względem podanych czasów

```

int action(int id, int czas) {
    printf("Uruchamiam watek %d\n", id);
    std::cout << "spie: " << czas / 1000 << "sekund\n";
    Sleep(czas); //10*1000 = 10 sekund
    printf("Koncze watek %d\n", id);
    //std::cout << "id watku: " << std::this_thread::get_id();
    return 0;
}

int main() {
    //tworzenie wÅtku
    int podCzas;
    std::cout << "Podaj czas uspienia watku 1:\n";
    std::cin >> podCzas;

    std::thread t1(action, 1, podCzas); //konstruktor klasy thread
    std::cout << "Podaj czas uspienia watku 2:\n";
    std::cin >> podCzas;
}

```

```

Podaj czas uspienia watku 1:
50000
Podaj czas uspienia watku 2:
Uruchamiam watek 1
spie: 50sekund
10000
Podaj czas uspienia watku 3:
Uruchamiam watek 2
spie: 10sekund
12222
Uruchamiam watek 3
spie: 12sekund
Koncze watek 2
Koncze watek 3
Koncze watek 1
Koniec programu

```

Można utworzyć wątki podczas spania poprzednich wątków. Wątek o najkrótszym podanym czasie lub utworzony odpowiednio wcześniej zakończy się pierwszy.

Kod 2

obsługa bardzo dużej ilości wątków:

```
#include <stdio>
#include <thread>
#include <windows.h>

void action(int id) {
    printf("Uruchamiam watek %d\n", id);
    Sleep(5 * 1000); //5 sekund
    printf("Koncze watek %d\n", id);
}

int main() {
    int thread_count = 8;

    //alokacja tablicy, ktora bÄ™dzie przechowywaÄ™ wskaÅ™niki na wÄ™tki
    std::thread** threads = new std::thread * [thread_count];

    //otwieranie wÄ™tków
    for (int i = 0; i < thread_count; i++) {
        threads[i] = new std::thread(action, i); //wykorzystuje i jako id danego wÄ™tku
    }

    //wÄ™tki pracujÄ™, ale trzeba je zsynchronizowaÄ™
    for (int i = 0; i < thread_count; i++) {
        threads[i]->join();
    }

    //alokowaliśmy pamięć więc pasuje ją zwolnić
    for (int i = 0; i < thread_count; i++) {
        delete threads[i];
    }
    delete[] threads;

    printf("Koniec programu \r\n");

    return 0;
}
```

2.1. Uruchom program wiele razy i porównaj wyjścia.

```
Konsola debugowania programu Microsoft Visual Studio
Uruchamiam watek 0
Uruchamiam watek 1
Uruchamiam watek 3
Uruchamiam watek 2
Uruchamiam watek 4
Uruchamiam watek 5
Uruchamiam watek 6
Uruchamiam watek 7
Koncze watek 6
Koncze watek 3
Koncze watek 5
Koncze watek 7
Koncze watek 2
Koncze watek 0
Koncze watek 1
Koncze watek 4
Koniec programu
```

```
Konsola debugowania programu Microsoft Visual Studio
Uruchamiam watek 0
Uruchamiam watek 1
Uruchamiam watek 2
Uruchamiam watek 3
Uruchamiam watek 4
Uruchamiam watek 5
Uruchamiam watek 6
Uruchamiam watek 7
Koncze watek 7
Koncze watek 1
Koncze watek 6
Koncze watek 0
Koncze watek 3
Koncze watek 5
Koncze watek 2
Koncze watek 4
Koniec programu
```

Wątki tworzone są z niewielką różnicą czasową, dlatego wątki wychodzą ze stanu uśpienia o podobnych czasach, przez co wątek utworzony później może zakończyć się przed wątkiem pierwszym.

2.2. Zmodyfikuj program tak by ilość wątków była pobierana od użytkownika:

```
int thread_count;
std::cout << "Podaj ilosc watkow: ";
std::cin >> thread_count;
```

```
Podaj ilosc watkow: 10
Uruchamiam watek 8
Uruchamiam watek 3
Uruchamiam watek 4
Uruchamiam watek 2
Uruchamiam watek 0
Uruchamiam watek 9
Uruchamiam watek 5
Uruchamiam watek 6
Uruchamiam watek 7
Uruchamiam watek 1
Koncze watek 1
Koncze watek 7
Koncze watek 6
Koncze watek 9
Koncze watek 2
Koncze watek 5
Koncze watek 0
Koncze watek 3
Koncze watek 4
Koncze watek 8
Koniec programu
```

2.3. Zmodyfikuj program tak by zamiast tablicy przechowującej wskaźniki na wątki użyć wektora przechowującego wskaźniki na wątki

```

#include <stdio>
#include <thread>
#include <windows.h>
#include <iostream>
#include <vector>

void action(int id) {
    printf("Uruchamiam watek %d\n", id);
    Sleep(5 * 1000); //5 sekund
    printf("Koncze watek %d\n", id);
}

int main() {
    int thread_count;
    std::cout << "Podaj ilosc watkow: ";
    std::cin >> thread_count;

    //alokacja tablicy, ktora bÄdzie przechowywaÄ wskaŹniki na wÄtki
    std::vector<std::thread*>threads; // = new std::thread * [thread_count];
    threads.resize(thread_count);

    //otwieranie wÄtków
    for (int i = 0; i < thread_count; i++) {
        threads.at(i) = new std::thread(action, i); //wykorzystuje i jako id danego wÄtku
    }


    //watki pracujÄ, ale trzeba je zsynchronizowaÄ
    for (int i = 0; i < thread_count; i++) {
        threads.at(i)->join();
    }

    //alokowaliŹmy pamieÄ wiÄc pasuje jÄ zwolniÄ
    for (int i = 0; i < thread_count; i++) {
        delete threads.at(i);
    }
    threads.~vector();

    printf("Koniec programu \r\n");

    return 0;
}

```

 Konsola debugowania programu

```

Podaj ilosc watkow: 5
Uruchamiam watek 0
Uruchamiam watek 1
Uruchamiam watek 3
Uruchamiam watek 4
Uruchamiam watek 2
Koncze watek 4
Koncze watek 2
Koncze watek 3
Koncze watek 0
Koncze watek 1
Koniec programu

```


Kod 3:

Pomiar czasu za pomocą biblioteki chrono

```
#include <chrono>
#include <cstdio>
#include <windows.h>

int main() {
    auto start = std::chrono::steady_clock::now();
    //długość operacje
    Sleep(2000);
    auto end = std::chrono::steady_clock::now();

    printf("Czas trwania: %llu\n", std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count());

    return 0;
}
```

3.1. Sprawdź ile trwają na Twoim komputerze operacje: otwarcia i zamknięcia pliku :

```
#include <chrono>
#include <cstdio>
#include <windows.h>
#include <fstream>

int main() {
    auto start = std::chrono::steady_clock::now();
    //długość operacje
    //Sleep(2000);
    std::fstream fs;
    fs.open ("test.txt");
    if (fs.is_open())
    {
        printf("Otworzono plik\n");
        fs.close();
    }

    auto end = std::chrono::steady_clock::now();

    printf("Czas trwania: %llu\n", std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count());

    return 0;
}
```

```
C:\> Konsola debugowania programu Microsoft Visual Studio
Otworzono plik
Czas trwania: 0
```

3.2 Sprawdź ile trwa na Twoim komputerze wygenerowanie jakiejś większej (np. 40) ilości elementów ciągu fibonacciego

```

#include <chrono>
#include <cstdio>
#include <windows.h>
#include <iostream>
void fibonacci(int n)
{
    long long a = 0, b = 1;

    for (int i = 0; i < n; i++)
    {
        std::cout << b << std::endl;
        b += a; //pod zmienną b przypisujemy wyraz następny czyli a+b
        a = b - a; //pod zmienną a przypisujemy wartość zmiennej b
    }
}

int main() {
    auto start = std::chrono::steady_clock::now();
    //długość operacje
    //Sleep(2000);
    fibonacci(40);

    auto end = std::chrono::steady_clock::now();

    printf("Czas trwania: %llu\n", std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count());

    return 0;
}

```

```

C:\> WybierzKonsola debugowania pro
2584
4181
6765
10946
17711
28657
46368
75025
121393
196418
317811
514229
832040
1346269
2178309
3524578
5702887
9227465
14930352
24157817
39088169
63245986
102334155
Czas trwania: 11

```