Biblioteka std::thread w C++

Wyżej wymieniona biblioteka, jest jedną z najprostszych metod pracy wielowątkowej. Opera się na podejściu stwórz wątek wykonujący daną funkcje z danymi parametrami.

Link do dokumentacji:

https://en.cppreference.com/w/cpp/thread/thread

Jak widać biblioteka nie ma jakoś bardzo dużo elementów, dlatego jest bardzo przyjemna do nauki.

Uruchomienie wątku:

KOD1:

```cpp
#include <cstdio>
#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 wątku
    std::thread t1(action, 1); //konstruktor klasy t1 przyjmuje minimum wskaźnik na funckje do wykonania
    std::thread t2(action, 2); //funckja ta może coś zwracać, ale może zwaracać też void
    std::thread t3(action, 3); //dalsze parametry zostaną przekazane do podanej funckji

    t1.join();  //synchronizacja
    t2.join();  //wątek 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;
}
```

Jak widać oprócz uruchomienia dokonano synchronizacji. Wątek główny poczeka na wątki utworzone. Same wątki nie robią nic ciekawego. Można sklasyfikować je jako IO bound.

Zadania do KOD1:

1. Uruchom program wiele razy i porównaj wyjście.
2. Usuń wszystkie 3 joiny, skompiluj i sprawdź wyjście.
3. Usuń tylko jeden join, skompiluj i sprawdź co zostanie wyświetlone (najlepiej więcej niż raz).
4. Zmodyfikuj program tak by wypisywane było id uzyskane z funkcji get_id()
   https://en.cppreference.com/w/cpp/thread/get_id
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

Otwieranie bardzo dużej ilości wątków:

Poniższy przykład pokazuje jak otworzyć i kontrolować większą ilość wątków np. pobraną od użytkownika.

KOD2:

```cpp
#include <cstdio>
#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, która będzie przechowywać wskaźniki na wątki
    std::thread** threads = new std::thread*[thread_count];

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

    //watki pracują, ale trzeba je zsynchronizować
    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;
}
```

Zadania do KOD2:

1. Uruchom program wiele razy i porównaj wyjścia.
2. Zmodyfikuj program tak by ilość wątków była pobierana od użytkownika.
3. Zmodyfikuj program tak by zamiast tablicy przechowującej wskaźniki na wątki użyć vectora przechowującego wskaźniki na wątki.

Pomiar czasu za pomocą biblioteki chrono:

https://en.cppreference.com/w/cpp/chrono

Bibioteka ta umożliwia wiele operacji na czasie. My wykorzystamy ją tylko do pomiaru czasu trwania operacji.

KOD3:

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

int main(){
    auto start = std::chrono::steady_clock::now();
    //dtugie 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;
}
```

Zadania do KOD3:

1. Sprawdź ile trwają na Twoim komputerze operacje: otwarcia i zamknięcia pliku (https://www.cplusplus.com/reference/fstream/fstream/fstream)
2. Sprawdź ile trwa na Twoim komputerze wygenerowanie jakieś większej (np. 40) ilości elementów ciągu fibonacciego.