| | | |
|---|---|---|
| Programowanie współbieżne i rozproszone | PWIR04 | |
| 26.04.2022 | P2 | Szymon Zwoliński |

KOD 7 I 8

Program realizujący odliczanie, a następnie sprawdzanie czy uzyskana liczba jest parzysta lub nieparzysta.

Kod 8 do synchronizacji wątków wykorzystuje mutexy. Wątek nie wykona się do momentu udzielenia zgody do uzyskania dostępu do zasobu chronionego przez mutex

```
#include <thread>
#include <cstdio>
#include <windows.h>
#include <mutex>

std::mutex counter_mutex;
unsigned int counter = 0;

void increment(){
    for(;;){
        counter_mutex.lock();
        counter++;
        counter_mutex.unlock();
        Sleep(2000);
    }
}

void parity(){
    for(;;){
        counter_mutex.lock();
        if (counter % 2){
            printf("%u jest nieparzyste\r\n", counter);
        }
        else{
            printf("%u jest parzyste\r\n", counter);
        }
        counter_mutex.unlock();
        Sleep(2000);
    }
}

int main(){
    std::thread inc(increment);
    std::thread par(parity);

    inc.join();
    par.join();

    printf("Done\r\n");

    return 0;
}
```

7.1-8.1 Przetestuj co stanie się gdy w jednym z wątków usunie się blokowanie i odblokowanie.

```
void increment(){
    for(;;){
        counter_mutex.lock();
        counter++;
        counter_mutex.unlock();
        Sleep(2000);
    }
}

void parity(){
    for(;;){
        //counter_mutex.lock();
        if (counter % 2){
            printf("%u jest nieparzyste\r\n", counter);
        }
        else{
            printf("%u jest parzyste\r\n", counter);
        }
        //counter_mutex.unlock();
        Sleep(2000);
    }
}
```

```
2 jest parzyste
2 jest parzyste
4 jest parzyste
5 jest nieparzyste
6 jest parzyste
7 jest nieparzyste
8 jest parzyste
9 jest nieparzyste
10 jest parzyste
11 jest nieparzyste
12 jest parzyste
13 jest nieparzyste
14 jest parzyste
15 jest nieparzyste
16 jest parzyste
17 jest nieparzyste
18 jest parzyste
19 jest nieparzyste
20 jest parzyste
21 jest nieparzyste
22 jest parzyste
23 jest nieparzyste
24 jest parzyste
25 jest nieparzyste
26 jest parzyste
27 jest nieparzyste
28 jest parzyste
29 jest nieparzyste
30 jest parzyste
```

Program w przypadku wyłączenia mutexa w jednym wątków dalej będzie działał poprawnie, poprzez synchronizację funkcji przyrostowej oraz wywołaniu wątków poprzez thread.join() wymagający poprawnego wykonania wątku przed wywołaniem.

7-8.2 Dodaj w obu pętlach break po pewnej ilości iteracji po czym porównaj czasy wykoniania.

```
void increment(){
    for(;;){
        counter_mutex.lock();
        counter++;
        if(counter==5)
        {
            counter_mutex.unlock();
            break;
        }
        counter_mutex.unlock();
        Sleep(2000);

    }
}

void parity(){
    for(;;){
            if(counter==5)
            {
                break;
            }
        counter_mutex.lock();
        if (counter % 2){
            printf("%u jest nieparzyste\r\n", counter);
        }
        else{
            printf("%u jest parzyste\r\n", counter);
        }
        counter_mutex.unlock();
        Sleep(2000);
    }
}
```

```
1 jest nieparzyste
1 jest nieparzyste
3 jest nieparzyste
4 jest parzyste
4 jest parzyste
Done

Process returned 0 (0x0)   execution time : 10.098 s
Press any key to continue.
```

```
1 jest nieparzyste
1 jest nieparzyste
3 jest nieparzyste
4 jest parzyste
Done

Process returned 0 (0x0)   execution time : 8.072 s
Press any key to continue.
```

Czasy wykonania różnią się w zależności od dokładności obliczeń (w zależności ile wątków zostanie poprawnie przekazanych do wypisania). Błędy dokładności powstają ze względu na brak przechowywania wartości uzyskanych poprzez wątek, przez co może on nie przekazać wartości counter a już zacząć wykonywać się na nowo. W przypadku uzyskania takich samych wyników, czas praktycznie się nie różni.

KOD 9

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

unsigned int counter = 0;

void increment(int id){
    for(int i = 0;i<10;i++){
        counter++;
        Sleep(300);
    }

    //ten blok wykona sie tylko raz mimo, że watków jest wiecej
    if(id == 1){
        printf("%u\n", counter);
    }
}

int main(){

    std::thread t1(increment,1);
    std::thread t2(increment,2);

    t1.join();
    t2.join();

    return 0;
}
```

9_2 local rózni się tylko sposobem implementacji counter

```cpp
thread_local unsigned int counter = 0;
```

9.1 Zaalokuj tablice intów o rozmiarze 100, wypełnij ją losowymi liczbami z zakresu 1-10 i wypisz.

```
1
2
8
5
6
3
1
7
7
10
4
6
9
1
1
6
7
1
4
1
5
7
6
10
9
6
1
7
5
8
```

```
#include <windows.h>
#include <random>
#include <functional>
std::default_random_engine generator;
std::uniform_int_distribution<int> distribution(1,10);
auto los = std::bind(distribution,generator);
thread_local unsigned int counter = 0;

void increment(int id){
    for(int i = 0;i<10;i++){
        counter++;
        Sleep(300);
    }

    //ten blok wykona sie tylko raz mimo, że watków jest wiecej
    if(id == 1){
        printf("%u\n", counter);
    }
}
void _random()
{
    int tab[100];
    for(int i=0;i<100;i++)
    {
        tab[i] = los();
    }
    for(int i =0;i<100;i++)
    {
        printf("%u\n",tab[i]);
    }


}

int main(){

    std::thread t1(increment,1);
    std::thread t2(increment,2);
    std::thread t3(_random);
    t1.join();
    t2.join();
    t3.join();
```

Liczby losowe z zakresu 1-10 zapewniane są poprzez bibliotekę random, a następnie generator oraz dystrybucja zostają przypisane do los, przez co zachowuje ona się jak funkcja (zapisywana jest “procedura” wywołania generatora liczb).

9.2  Zaalokuj 10 wątków i niech każdy z nich zsumuje komórki: [id*10;(id+1)*10] najpierw do zwykłej zmiennej a później do zmiennej thread_local.

```
unsigned int counter_tab = 0;
//thread_local unsigned int counter_tab =0;
void increment(int id){
    for(int i = 0;i<10;i++){
        counter++;
        Sleep(300);
    }

    //ten blok wykona sie tylko raz mimo, że watków jest wiecej
    if(id == 1){
        printf("%u\n", counter);
    }
}
void _random(int id)
{
    int tab[100];
    for(int i=0;i<100;i++)
    {
    /*
    for(int i =0;i<100;i++)
    {
        printf("%u\n",tab[i]);
    }*/
    for(int i =(id)*10;i<(id+1)*10;i++)
    {
        counter_tab+=tab[i];
    }

/*
    if(id==10)
    {

        printf("%u\n",counter_tab);
    }*/
}
```

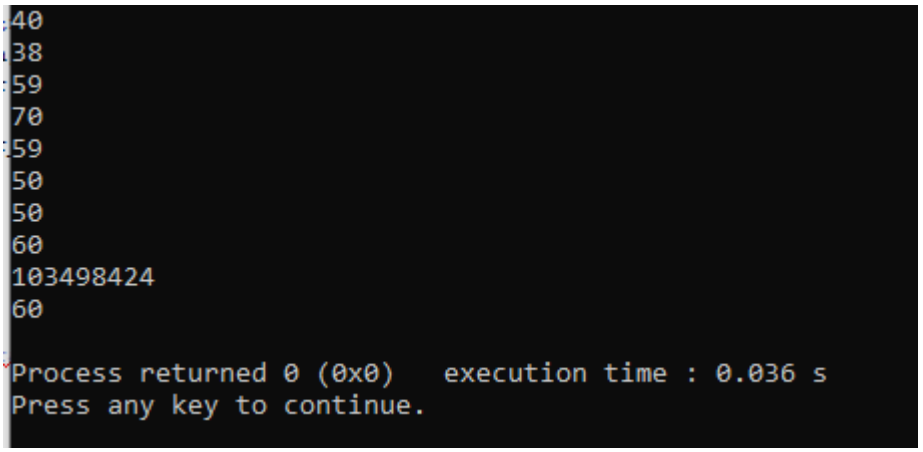
9.3 Na końcu funkcji wątku wypisz: id -> wartość

```cpp
#include <thread>
#include <cstdio>
#include <windows.h>
#include <random>
#include <functional>
std::default_random_engine generator;
std::uniform_int_distribution<int> distribution(1,10);
auto los = std::bind(distribution,generator);
thread_local unsigned int counter = 0;
//unsigned int counter_tab = 0;
thread_local unsigned int counter_tab =0;
void increment(int id){
    for(int i = 0;i<10;i++){
        counter++;
        Sleep(300);
    }

    //ten blok wykona sie tylko raz mimo, że watków jest wiecej
    if(id == 1){
        printf("%u\n", counter);
    }
}
void _random(int id)
{
    int tab[100];
    for(int i=0;i<100;i++)
    {
    /*
    for(int i =0;i<100;i++)
    {
        printf("%u\n",tab[i]);
    }*/
    for(int i =(id)*10;i<(id+1)*10;i++)
    {
        counter_tab+=tab[i];
    }
      if(id==1)
    {

        printf("%u\n",counter_tab);
    }
     if(id==2)
    {
```

Zmienna unsigned int:

```
40
78
137
207
266
316
366
408
463
65441612

Process returned 0 (0x0)   execution time : 0.033 s
Press any key to continue.
```

zmienna thread_local:




zmienna thread_local tworzy dla każdego wątku osobną instancje z wartością przypisaną w procesie deklaracji zmiennej. zliczanie dla każdego wątku wykonuje się osobno, bez sumowania się.