

Akademia Nauk Stosowanych w Nowym Sączu Programowanie Współbieżne i Rozproszone		
Temat: Przykład z parzystością.		
Nazwisko i imię: Ciapała Tadeusz	Ocena sprawozdania	Zaliczenie:
Data wykonania ćwiczenia: 21.03.2023	Grupa: P3	

### 1) KOD 7 i 8:

Na początku zadanych do wykonania zadań należało zapoznać się z kodem numer 7 i 8. Przedstawiał on przykład ze sprawdzaniem parzystości danej liczby, używając do tego 2 wątków. Wątek nr 1 miał za zadanie inkrementację zmiennej globalnej, natomiast wątek nr 2 oceniał parzystość inkrementowanej zmiennej.

```
#include <thread>
#include <stdio>
#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;
}
```

Listing nr 1: dostarczy do zadań kod nr 7 i 8

```
12 Counter44;  
C:\Users\TadeK\Documents\PROJECTS\pwir05\x64\Debug\pwir05.exe  
0 jest parzyste  
12 jest parzyste  
3 jest nieparzyste  
4 jest parzyste  
78 4 jest parzyste  
5 jest nieparzyste  
7 jest nieparzyste  
8 jest parzyste  
9 jest nieparzyste  
9 jest nieparzyste  
11 jest nieparzyste  
12 jest parzyste  
12 jest parzyste  
nr
```

Zrzut nr 1: pierwsze uruchomienie kodu 7/8.

Pierwszą rzeczą którą widzimy jest wartość licznika, która wydaje się być błędnie obliczana. Kolejną rzeczą jest sama parzystość. Jeśli byśmy poczekali dłużej okazałoby się, że parzystość liczby nie zawsze jest dobrze wskazywana. Jest to spowodowane błędnym rozwiązaniem przedstawionego problemu ( undefined behaviour ), dlatego też należy użyć biblioteki cpp - mutex:

```
+;  
r mutex  
00);  
C:\Users\TadeK\Documents\PROJECTS\pwir05\x64\Debug\pwir05.exe  
0 jest parzyste  
2 jest parzyste  
4 jest parzyste  
6 jest parzyste  
8 jest parzyste  
mutex.lock 10 jest parzyste  
ter % 2 12 jest parzyste  
tf("%u 14 jest parzyste  
16 jest parzyste  
18 jest parzyste  
20 jest parzyste  
tf("%u 22 jest parzyste  
24 jest parzyste  
mutex.unlock  
00);
```

Zrzut nr 2: kod 7/8 po usunięcie blokowania i odblokowania w jednym z wątków

W pierwszym zadaniu należało usunąć mechanizm blokady licznika w jednym z wątków. Po zakomentowaniu odpowiednich linii, zaobserwować można było zjawisko pokazane na rzucie nr 2. Wypisywane były tylko liczby parzyste.

```
rzyste\r\n", counter);
```

Konsola debugowania programu Microsoft Visual Studio

```
1 jest nieparzyste
2 jest parzyste
3 jest nieparzyste
4 jest parzyste
5 jest nieparzyste
6 jest parzyste
6 jest parzyste
8 jest parzyste
8 jest parzyste
9 jest nieparzyste
11 jest nieparzyste
Czas trwania 2: 11073
Czas trwania 1: 11073
Done
C:\Users\TadeK\Documents\PROJECTS\pwir05\x64\Debug\pwir05.exe (
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

Zrzut nr 3: wykonanie zadania 2 do kodu 7 i 8

```
#pragma warning( disable : 4473 )
#include <thread>
#include <cstdio>
#include <windows.h>
#include <mutex>
#include <chrono>
#include <iostream>

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

void increment() {
    auto start = std::chrono::steady_clock::now();
    for (;;) {
        if (counter == 11)
            break;
        counter_mutex.lock();
        counter++;
        counter_mutex.unlock();
        Sleep(1000);
    }
    auto end = std::chrono::steady_clock::now();
    printf("Czas trwania 1: %llu\n",
        std::chrono::duration_cast<std::chrono::milliseconds>(end -
start).count()
    );
}
```

```

void parity() {
    auto start = std::chrono::steady_clock::now();
    for (;;) {
        if (counter == 11)
            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(1000);
    }
    auto end = std::chrono::steady_clock::now();
    printf("Czas trwania 2: %llu\n",
        std::chrono::duration_cast<std::chrono::milliseconds>(end -
start).count()
    );
}

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

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

    printf("Done\r\n");

    return 0;
}

```

*Listing nr 2: rozwiązanie do zadania nr 2 kodu 7 i 8*

Zmierzone zostały dokładnie czasy wykonania pętli dla obu wątków. Ustawiłem punkt zakończenia pętli na moment w którym zmienna "counter" osiąga wartość 11. Na zrzucie nr 3 widzimy, że zmierzone czasy dla obu pętli były identyczne.

## 2) KOD 9:

Kod ten dotyczył zmiennych lokalnych w danym wątku. Przedstawiał on typ zmiennej `thread_local`, którego zastosowanie powodowało, że każdy z wątków otrzymywał kopię zmiennej "counter".

```

#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 się tylko raz mimo, że wątków jest więcej
    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;
}

```

*Listing nr 3: kod nr 9, wersja bez thread\_local*

```

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

thread_local unsigned int counter = 0;

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

    //ten blok wykona się tylko raz mimo, że wątków jest więcej
    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;
}

```

*Listing nr 4: kod nr 9, wersja z thread\_local*

The screenshot shows a Visual Studio IDE with a C++ source file on the left and a debug console on the right. The source file contains the code from Listing 4, with a green cursor on the line "raz mimo, że wątków jest więcej". The debug console has a yellow title bar that reads "Konsola debugowania programu Microsoft Visual Studio". It shows the output "20" and a message: "C:\Users\TadeK\Documents\PROJECTS\pwir05\x64\Debug\pwir05.exe (proces 14696) zakończono. Naciśnij dowolny klawisz, aby zamknąć to okno...".

*Zrzut nr 4: pierwsze uruchomienie pierwszej wersji kodu nr 9*

```
= 0;

{
    10
    C:\Users\TadeK\Documents\PROJECTS\pwir05\x64\Debug\pwir05.exe (proces 17768) zako
    Naciśnij dowolny klawisz, aby zamknąć to okno...

az mimo, że
```

Zrzut nr 5: pierwsze uruchomienie drugiej wersji kodu nr 9

Jak widać wyniki różnią się, ponieważ w drugim przypadku jako licznik do wątku przekazywana jest kopia.

```
#include <thread>
#include <cstdlib>
#include <windows.h>
#include <time.h>

thread_local unsigned int counter = 0;

int* generowanie()
{
    srand(time(NULL));
    int* my_array = new int[100];
    for (size_t i = 0; i < 100; i++)
        my_array[i] = rand() % 10 + 1;
    return my_array;
}

void wypisywanie(int* my_array)
{
    for (size_t i = 0; i < 100; i++)
        printf("tablica %d -> %d\n", i, my_array[i]);
}

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

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

int main() {
    int* tab = generowanie();
    wypisywanie(tab);

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

    t1.join();
```

```

    t2.join();

    return 0;
}

```

Listing nr 5: rozwiązanie zadania nr 1

Na listingu nr 5 widać, że do kodu zostały dodane dwie funkcje. Pierwsza z nich ma za zadanie zaalokować pamięć dla tablicy 100-elementowej oraz wypełnić ją losowymi wartościami od 1 do 10. Druga funkcja wypisuje jej zawartość.

```

Konsola debugowania programu Microsoft Visual Studio
tablica 74 -> 7
tablica 75 -> 2
tablica 76 -> 3
tablica 77 -> 9
tablica 78 -> 6
tablica 79 -> 4
tablica 80 -> 2
tablica 81 -> 5
tablica 82 -> 2
tablica 83 -> 2
tablica 84 -> 9
tablica 85 -> 7
tablica 86 -> 10
tablica 87 -> 5
tablica 88 -> 8
tablica 89 -> 3
tablica 90 -> 9
tablica 91 -> 9
tablica 92 -> 1
tablica 93 -> 3
tablica 94 -> 7
tablica 95 -> 10
tablica 96 -> 4
tablica 97 -> 9
tablica 98 -> 7
tablica 99 -> 7
10
C:\Users\TadeK\Documents\PROJECTS\pwir05\x64\Debug\pwir05.exe (proces 19264) zakończono z kodem 0.
Naciśnij dowolny klawisz, aby zamknąć to okno...

```

Zrzut nr 6: rozwiązanie zadania nr 1 do kodu nr 9

```

#include <thread>
#include <cstdio>
#include <windows.h>
#include <time.h>

thread_local unsigned int counter = 0;
//unsigned int counter = 0;

int* generowanie()
{
    srand(time(NULL));
    int* my_array = new int[100];
    for (size_t i = 0; i < 100; i++)
        my_array[i] = rand() % 10 + 1;
    return my_array;
}

void wypisywanie(int* my_array)
{
    for (size_t i = 0; i < 100; i++)
        printf("tablica %l -> %d\n", i, my_array[i]);
}

void increment(int id, int *tab)

```

```

{
    int start = id * 10;
    int end = 10*(id + 1);
    int help = 0;
    for (size_t i = start; i < end; i++)
    {
        help = help + tab[i];
        Sleep(100);
    }

    counter += help;
    //ten blok wykona się tylko raz mimo, że wątków jest więcej
    if (id == 1) {
        printf("%u\n", counter);
    }
}

int main() {
    int* tab = generowanie();
    //wypisywanie(tab);

    std::thread threads[10];
    for (size_t i = 0; i < 10; i++) {
        threads[i] = std::thread(increment, i, tab);
    }

    for (size_t i = 0; i < 10; i++) {
        threads[i].join();
    }

    delete[] tab;
    return 0;
}

```

Listing nr 6: rozwiązanie zadania nr 2

The screenshot shows the Visual Studio IDE with a C++ source file on the left and the 'Konsola debugowania programu Microsoft Visual Studio' (Visual Studio Program Debug Console) on the right. The source code on the left includes a function `generowanie()` that creates an array of 42 random integers, and a `main` function that creates 10 threads to process this array. The debug console on the right shows the output of the program, including the path to the executable and a message indicating that the process has finished.

```

erowanie()
d(time(NULL));
my_array = new int[42];
(size_t i = 0; i < 42; i++)
my_array[i] = rand();
return my_array;

wypisywanie(int* my_a
(size_t i = 0; i < 42; i++)
printf("tablica %d\n", my_array[i]);

increment(int id, int
start = id * 10;
end = 10*(id + 1);
help = 0;

```

Konsola debugowania programu Microsoft Visual Studio

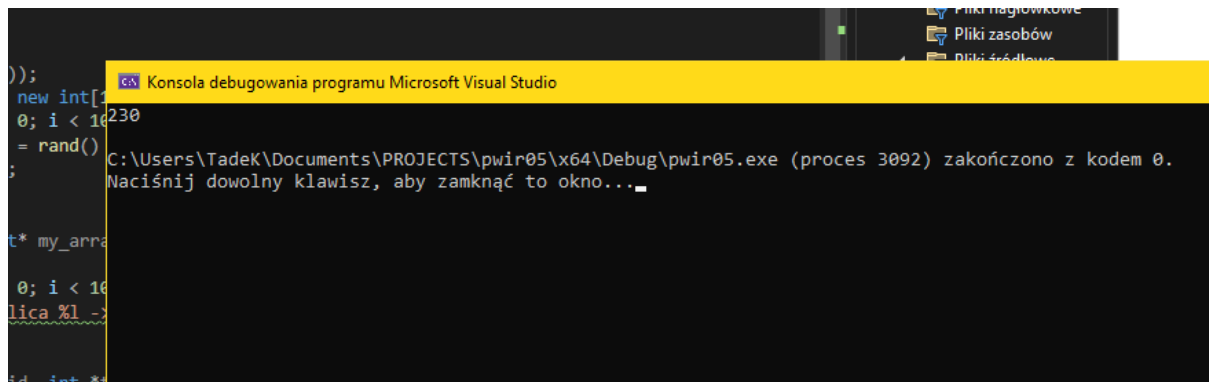
```

C:\Users\Tadek\Documents\PROJECTS\pwir05\x64\Debug\pwir05.exe (proces 5168) zakończono z
Naciśnij dowolny klawisz, aby zamknąć to okno...

```

Zrzut nr 7: wykonanie programu dla zadania nr 2 - wersja z `thread_local`





Zrzut nr 8: wykonanie programu dla zadania nr 2 - wersja bez `thread_local`

Na zrzutach nr 7 i 8 widzimy, że dwa różne typy zmiennych dają dwa różne wyniki. Dla rozwiązania `thread_local`, jak wiemy każdy wątek posiada swoją własną kopię, natomiast drugie rozwiązanie sumuje licznik globalnie.

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

//thread_local unsigned int counter = 0;
unsigned int counter = 0;

int* generowanie()
{
    srand(time(NULL));
    int* my_array = new int[100];
    for (size_t i = 0; i < 100; i++)
        my_array[i] = rand() % 10 + 1;
    return my_array;
}

void wypisywanie(int* my_array)
{
    for (size_t i = 0; i < 100; i++)
        printf("tablica %l -> %d\n", i, my_array[i]);
}

void increment(int id, int *tab)
{
    int save_id = id;
    int start = id * 10;
    int end = 10*(id + 1);
    int help = 0;
    for (size_t i = start; i < end; i++)
    {
        help = help + tab[i];
        Sleep(100);
    }

    counter += help;
    printf("ID: %u -> %d\n", save_id, counter);
}

int main() {
    int* tab = generowanie();
    //wypisywanie(tab);
}
```

```

std::thread threads[10];
for (size_t i = 0; i < 10; i++) {
    threads[i] = std::thread(increment, i, tab);
}

for (size_t i = 0; i < 10; i++) {
    threads[i].join();
}

delete[] tab;
return 0;
}

```

Listing nr 7: rozwiązanie zadania nr 3

The screenshot shows the Visual Studio debug console with a yellow header. It displays a list of thread IDs and their corresponding values, followed by a message indicating the program has finished execution.

```

ID: 8 -> 57
ID: 0 -> 251
ID: 2 -> 218
ID: 5 -> 207
ID: 6 -> 363
ID: 1 -> 154
ID: 7 -> 306
ID: 9 -> 260
ID: 4 -> 107
ID: 3 -> 415

C:\Users\TadeK\Documents\PROJECTS\pwir05\x64\Debug\pwir05.exe (proces 16448) zakończono z kodem 0.
Naciśnij dowolny klawisz, aby zamknąć to okno...

```

Zrzut nr 9: uruchomienie rozwiązania do zadania 3 - bez `thread_local`

The screenshot shows the Visual Studio debug console with a yellow header. It displays a list of thread IDs and their corresponding values, followed by a message indicating the program has finished execution.

```

ID: 5 -> 49
ID: 6 -> 67
ID: 9 -> 62
ID: 7 -> 64
ID: 3 -> 37
ID: 4 -> 52
ID: 1 -> 64
ID: 8 -> 46
ID: 0 -> 53
ID: 2 -> 61

C:\Users\TadeK\Documents\PROJECTS\pwir05\x64\Debug\pwir05.exe (proces 11916) zakończono
Naciśnij dowolny klawisz, aby zamknąć to okno...

```

Zrzut nr 10: uruchomienie rozwiązania do zadania 3 - wraz z użyciem `thread_local`

Teraz, gdy nasz program wypisuje id wątku oraz obliczane wartości, dokładniej możemy obserwować jak zachowują się dwa przedstawione podejścia do dzielenia się zmienną globalną.