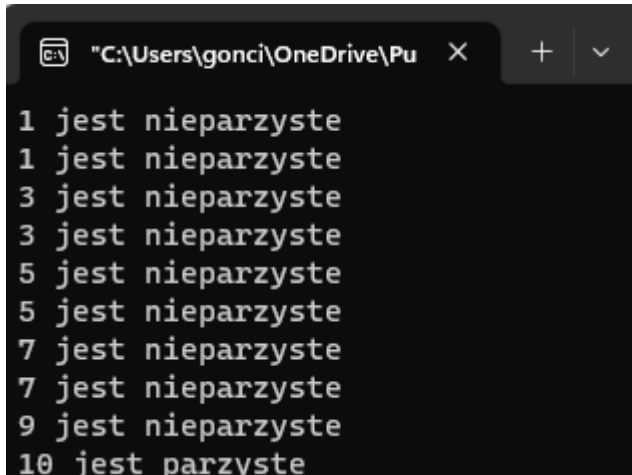


Programowanie współbieżne i rozproszone
Projekt
Marcin Gonciarz
P1

Zadanie 1.

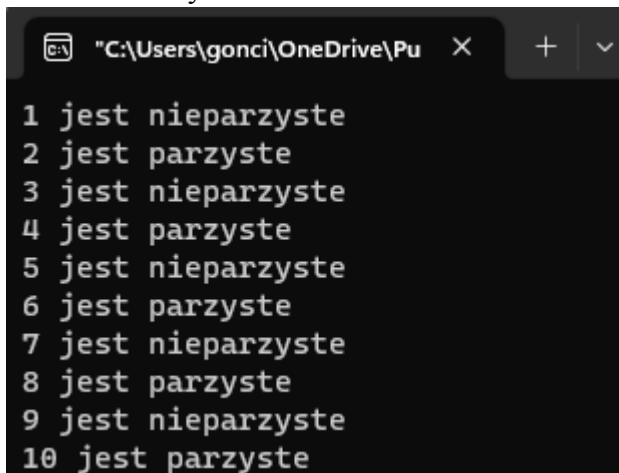
1. Przetestuj co stanie się gdy w jednym z wątków usunie się blokowanie i odblokowanie.

Z odblokowaniem i blokowaniem



```
"C:\Users\gonci\OneDrive\Pu" X + v
1 jest nieparzyste
1 jest nieparzyste
3 jest nieparzyste
3 jest nieparzyste
5 jest nieparzyste
5 jest nieparzyste
7 jest nieparzyste
7 jest nieparzyste
9 jest nieparzyste
10 jest parzyste
```

Bez odblokowywania i blokowania



```
"C:\Users\gonci\OneDrive\Pu" X + v
1 jest nieparzyste
2 jest parzyste
3 jest nieparzyste
4 jest parzyste
5 jest nieparzyste
6 jest parzyste
7 jest nieparzyste
8 jest parzyste
9 jest nieparzyste
10 jest parzyste
```

Jeśli usuniemy blokowanie i odblokowanie za pomocą mutex-a w jednym z wątków, może wystąpić wyścig (race condition) między wątkami, co prowadzi do nieprzewidywalnych wyników.

W szczególności, wątek parity może odczytać wartość counter w momencie, gdy jest tylko częściowo zaktualizowana przez wątek increment. Może to prowadzić do wyświetlania nieprawidłowych informacji o parzystości/nieparzystości liczby. Innymi słowy, wynik działania programu stanie się nieprzewidywalny i nieodpowiedni.

Dlatego ważne jest stosowanie synchronizacji, takiej jak mutexy, aby uniknąć wyścigów wątków.

2. Dodaj w obu pętlach break po pewnej ilości iteracji po czym porównaj czasy wykonania.

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

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

void increment(){
    for(;;){
        //counter_mutex.lock();
        counter++;
        //counter_mutex.unlock();
        Sleep(2000);
        if (counter == 10) {break;}
    }
}

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);
        if (counter == 10) {break;}
    }
}

int main(){
    auto start = std::chrono::steady_clock::now();
    std::thread inc(increment);
    std::thread par(parity);

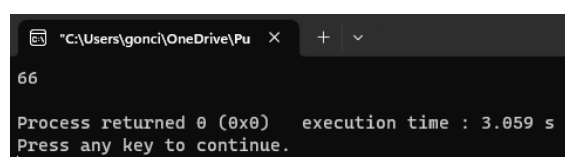
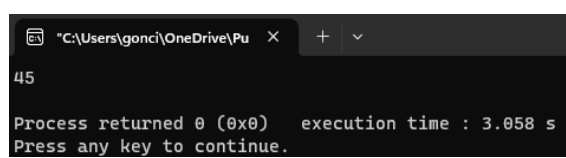
    inc.join();
    par.join();

    printf("Done\r\n");
    auto end = std::chrono::steady_clock::now();
    printf("\nCzas twania programu: %llu\r\n",
std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count());
    return 0;
}
```

Zadanie 2.

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

9_1_nolocal	9_2_local
<pre>#include <thread> #include <stdio> #include <windows.h> #include <stdlib> #include <ctime> thread_local unsigned int counter = 0; void increment(int id, int* arr){ for(int i = 0;i<10;i++){ counter += arr[i]; Sleep(300); } if(id == 1){ printf("%u\n", counter); } } int main(){ int arr[100]; srand(time(NULL)); for(int i=0; i<100; i++){ arr[i] = rand()%10 + 1; } std::thread t1(increment,1, arr); std::thread t2(increment,2, arr); t1.join(); t2.join(); return 0; }</pre>	<pre>#include <thread> #include <stdio> #include <windows.h> #include <stdlib> #include <ctime> thread_local unsigned int counter = 0; void increment(int id, int* arr){ for(int i = 0;i<10;i++){ counter += arr[i]; Sleep(300); } if(id == 1){ printf("%u\n", counter); } } int main(){ int arr[100]; srand(time(NULL)); for(int i=0; i<100; i++){ arr[i] = rand()%10 + 1; } std::thread t1(increment,1, arr); std::thread t2(increment,2, arr); t1.join(); t2.join(); return 0; }</pre>

 <p>Wnioski: Wynik działania programu z kodem 1 nie jest przewidywalny ze względu na brak synchronizacji w dostępie do zmiennej counter przez oba wątki. Wartość tej zmiennej może być nieprawidłowa, ponieważ wątki wykonują się równolegle i mogą jednocześnie zwiększać wartość counter.</p> <p>Po dodaniu tablicy losowych liczb do funkcji increment, wartość zmiennej counter będzie zależna od sumy elementów w tablicy. Ponieważ tablica jest generowana losowo, wartość ta będzie również losowa. Z tego powodu, wynik działania programu będzie również losowy i nieprzewidywalny.</p>	 <p>Wnioski: W przypadku kodu 2, zmienna counter jest zadeklarowana jako thread_local, co oznacza, że każdy wątek ma swoją własną kopię tej zmiennej. Dlatego, każdy wątek zwiększa wartość swojej własnej kopii zmiennej counter, co zapobiega nieprawidłowemu zwiększaniu tej zmiennej przez wiele wątków naraz.</p> <p>Podobnie jak w przypadku kodu 1, wartość zmiennej counter zależy od sumy elementów w tablicy. Jednak w tym przypadku każdy wątek zwiększa wartość swojej własnej kopii zmiennej counter, co skutkuje dokładnym i poprawnym wynikiem działania programu.</p>
---	---

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.

9_1_nolocal	9_2_local
<pre>#include <thread> #include <stdio> #include <windows.h> #include <stdlib> #include <ctime> unsigned int counter = 0; void increment(int id, int* arr){</pre>	<pre>#include <thread> #include <stdio> #include <windows.h> #include <stdlib> #include <ctime> thread_local unsigned int counter = 0; void increment(int id, int* arr){</pre>

```

int start = id * 10;
int end = (id + 1) * 10;
for(int i = start; i < end;
i++){
    counter += arr[i];
    Sleep(300);
}
if(id == 0){
printf("%u\n", counter);
}
}

int main(){
int arr[100];
srand(time(NULL));
for(int i=0; i<100; i++){
    arr[i] = rand()%10 + 1;
}

std::thread t[10];
for(int i = 0; i < 10; i++){
    t[i] =
std::thread(increment, i, arr);
}

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

return 0;
}

```

```

555
Process returned 0 (0x0) execution time : 3.085 s
Press any key to continue.

```

Wnioski:

W kodzie 1 zastosowano tablicę liczb losowych o rozmiarze 100, z której każdy wątek sumuje

```

int start = id * 10;
int end = (id + 1) * 10;
for(int i = start; i < end;
i++){
    counter += arr[i];
    Sleep(300);
}
if(id == 0){
printf("%u\n", counter);
}
}

int main(){
int arr[100];
srand(time(NULL));
for(int i=0; i<100; i++){
    arr[i] = rand()%10 + 1;
}

std::thread t[10];
for(int i = 0; i < 10; i++){
    t[i] =
std::thread(increment, i, arr);
}

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

return 0;
}

```

```

47
Process returned 0 (0x0) execution time : 3.067 s
Press any key to continue.

```

Wnioski:

W kodzie 2 zastosowano zmienną thread_local "counter", co oznacza, że każdy wątek ma

<p>kolejne 10 elementów. Zmienna "counter" jest globalna i współdzielona przez wszystkie wątki. Problem z tym rozwiązaniem polega na tym, że wiele wątków próbuje równocześnie modyfikować tę samą zmienną, co może prowadzić do nieprzewidywalnych wyników.</p>	<p>swoją własną kopię tej zmiennej. Dzięki temu uniknięto problemu współdzielenia zmiennej przez wiele wątków. W tym przypadku każdy wątek sumuje kolejne 10 elementów z tablicy liczb losowych i dodaje je do swojej kopii zmiennej "counter". Wypisanie wyniku odbywa się tylko dla wątku o id = 0.</p>
--	---

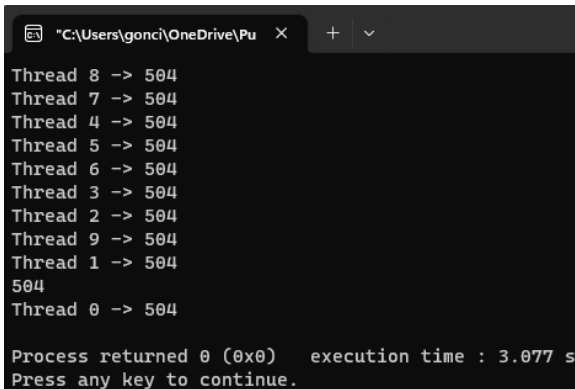
3. Na końcu funkcji wątku wypisz: id -> wartość

9_1_nolocal	9_2_local
<pre>#include <thread> #include <stdio> #include <windows.h> #include <stdlib> #include <ctime> unsigned int counter = 0; void increment(int id, int* arr){ int start = id * 10; int end = (id + 1) * 10; for(int i = start; i < end; i++){ counter += arr[i]; Sleep(300); } if(id == 0){ printf("%u\n", counter); } printf("Thread %d -> %u\n", id, counter); } int main(){ int arr[100];</pre>	<pre>#include <thread> #include <stdio> #include <windows.h> #include <stdlib> #include <ctime> thread_local unsigned int counter = 0; void increment(int id, int* arr){ int start = id * 10; int end = (id + 1) * 10; for(int i = start; i < end; i++){ counter += arr[i]; Sleep(300);} if(id == 0){ printf("%u\n", counter); } printf("Thread %d -> %u\n", id, counter); } int main(){ int arr[100];</pre>

```

srand(time(NULL));
for(int i=0; i<100; i++){
    arr[i] = rand()%10 + 1;
}
std::thread t[10];
for(int i = 0; i < 10; i++){
    t[i] =
std::thread(increment, i, arr);
}
for(int i = 0; i < 10; i++){
    t[i].join();
}
return 0;
}

```



```

"C:\Users\gonci\OneDrive\Pu... X + v
Thread 8 -> 504
Thread 7 -> 504
Thread 4 -> 504
Thread 5 -> 504
Thread 6 -> 504
Thread 3 -> 504
Thread 2 -> 504
Thread 9 -> 504
Thread 1 -> 504
504
Thread 0 -> 504

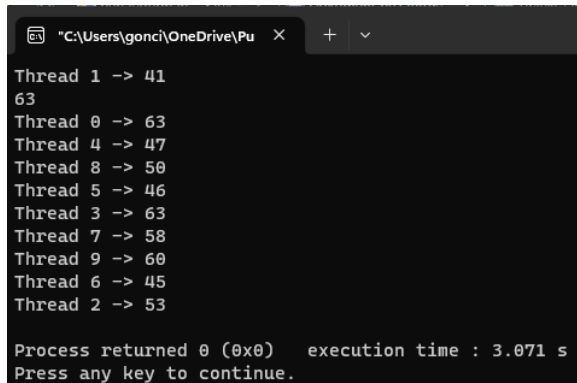
Process returned 0 (0x0)   execution time : 3.077 s
Press any key to continue.

```

```

srand(time(NULL));
for(int i=0; i<100; i++){
    arr[i] = rand()%10 + 1;
}
std::thread t[10];
for(int i = 0; i < 10; i++){
    t[i] =
std::thread(increment, i, arr);
}
for(int i = 0; i < 10; i++){
    t[i].join();
}
return 0;
}

```



```

"C:\Users\gonci\OneDrive\Pu... X + v
Thread 1 -> 41
63
Thread 0 -> 63
Thread 4 -> 47
Thread 8 -> 50
Thread 5 -> 46
Thread 3 -> 63
Thread 7 -> 58
Thread 9 -> 60
Thread 6 -> 45
Thread 2 -> 53

Process returned 0 (0x0)   execution time : 3.071 s
Press any key to continue.

```

Wnioski:

W kodzie 1 i 2 użyto tablicy 100 losowych liczb z przedziału 1-10, a następnie 10 wątków, aby zsumować 10 liczb w każdym wątku. W obu kodach ustawiono zakres sumowania dla każdego wątku i dodano wypisanie wartości zmiennej counter w każdym wątku.

W kodzie 1 użyto globalnej zmiennej counter, która jest wspólna dla wszystkich wątków, co oznacza, że wątki mogą wpływać na wartość zmiennej. Wynik tego kodu nie jest deterministyczny, a wynik sumowania zależy od tego, w jakiej kolejności wątki wykonywały operacje sumowania.

W kodzie 2 zdefiniowano counter jako zmienną thread_local, co oznacza, że każdy wątek ma swoją kopię zmiennej. W tym kodzie każdy wątek sumuje swoje liczby w swojej własnej instancji zmiennej counter, co skutkuje deterministycznym wynikiem sumowania.