

Akademia Nauk Stosowanych w Nowym Sączu Programowanie Współbieżne i Rozproszone			
Temat: Operacje atomowe.			spr nr 6
Nazwisko i imię: Ciapała Tadeusz		Ocena sprawozdania	Zaliczenie:
Data wykonania ćwiczenia: 28.03.2023		Grupa: P3	

1) KOD 10:

W tym ćwiczeniu mamy do czynienia z biblioteką `std::atomic`, a w zasadzie szablonem `template <class T> struct atomic;`. Pozwoli on nam na wykonanie operacji atomowych, czyli takich, które są w pewnym sensie niepodzielne. Gdy procesor rozpocznie ich wykonywanie, to nie ma możliwości ich przerwania. Używamy ich w przetwarzaniu równoległym do zmian w obiektach współdzielonych przez różne procesy, a także w systemach baz danych czy niektórych systemach plików.

```
#include <cstdio>
#include <thread>
#include <atomic>

//unsigned long long sum2 = 0;
//std::atomic_uint64_t sum2 = 0;
std::atomic<unsigned long long> sum2(0);

void sum(unsigned char* data, int id, int count) {
    for (unsigned i = id * count; i < (id + 1) * count; i++) {
        sum2 += data[i];
    }
}

int main() {
    unsigned char* data = new unsigned char[10000];

    for (unsigned int i = 0; i < 10000; i++) {
        data[i] = i;
    }

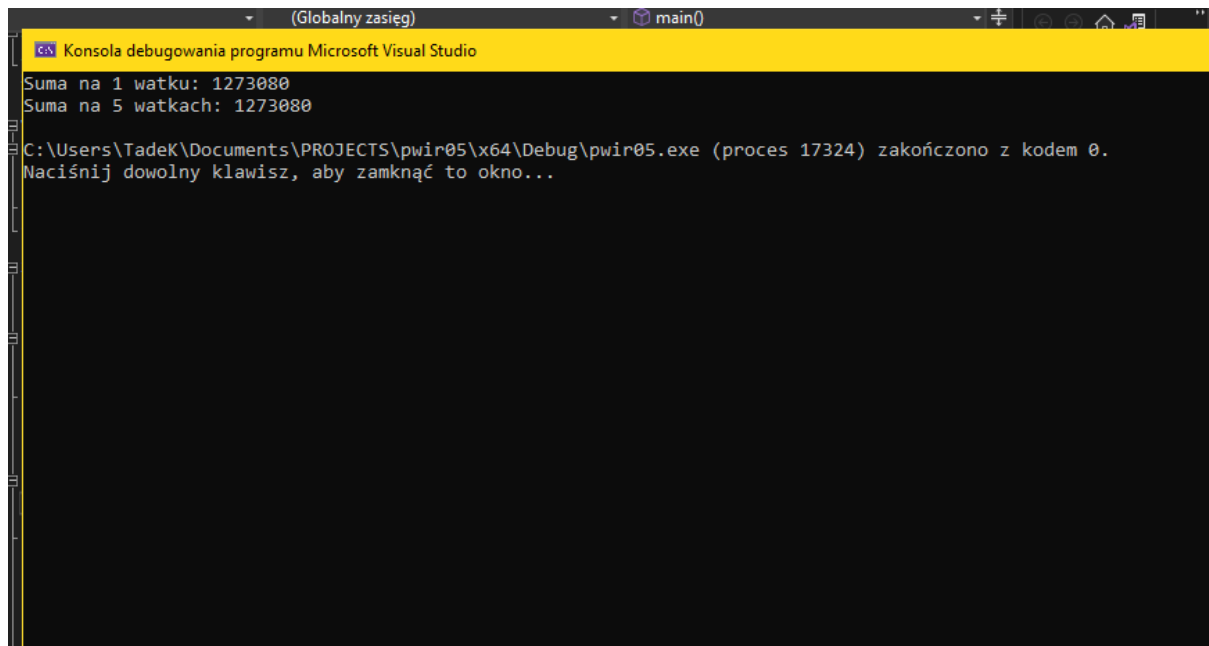
    //suma na jednym wątku
    unsigned long long sum1 = 0;
    for (unsigned i = 0; i < 10000; i++) {
        sum1 += data[i];
    }
    printf("Suma na 1 wątku: %llu\r\n", sum1);

    //suma na wielu wątkach
    std::thread t1(sum, data, 0, 2000);
    std::thread t2(sum, data, 1, 2000);
    std::thread t3(sum, data, 2, 2000);
    std::thread t4(sum, data, 3, 2000);
    std::thread t5(sum, data, 4, 2000);

    t1.join();
    t2.join();
    t3.join();
    t4.join();
    t5.join();

    //printf("Suma na 5 wątkach: %llu\r\n", sum2);
    printf("Suma na 5 wątkach: %llu\r\n", sum2.load());
}
```

Listing nr 1: dostarczony do wykonania zadań kod nr 10



```
(Globalny zasięg) main()
Konsola debugowania programu Microsoft Visual Studio
Suma na 1 watku: 1273080
Suma na 5 watach: 1273080
C:\Users\TadeK\Documents\PROJECTS\pwir05\x64\Debug\pwir05.exe (proces 17324) zakończono z kodem 0.
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

Zrzut nr 1: pierwsze uruchomienie dostarczonego kodu

Zadanie nr 1: Opis operacji atomowych

Jak wspomniałem wcześniej operacje atomowe są nieprzerywalne. Nie oznacza to jednak, że są to elementarne operacje. W rzeczywistości taka operacja pod względem sprzętowym często składa się z kilku kroków.

Zadanie nr 2: Dlaczego są tak ważne dla programowania równoległego

Operacje te są ważne w programowaniu równoległym, gdyż użycie ich zabezpiecza zmienną lub obszar pamięci przed próbą jednoczesnej modyfikacji ze strony procesów lub wątków. Przykładowo mamy daną zmienną i chcemy ją zwiększyć o 1. Jeżeli w danym momencie kilka wątków spróbuje wykonać to dodawanie, to zastosowanie szablonu operacji atomowych zapewni, że tylko jedna operacja zostanie wykonana na raz, a zmienna faktycznie zwiększy się o 1 (a nie o 2).

Zadanie nr 3: Zastosowanie operacji atomowych w kodzie 7 / 8

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

// #include <mutex>
#include <atomic>

// std::mutex counter_mutex;
// unsigned int counter = 0;
std::atomic<unsigned int> counter;

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) {
            std::cout << counter << " jest nieparzyste \r\n";
        }
        else {
            std::cout << counter << " jest parzyste \r\n";
        }
        //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: zmodyfikowany kod 7 / 8

```

auto end = std::chrono::steady_clock::now();
printf("Czas trwania 1: %llu\n",
    std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count()
);
1 jest nieparzyste
1 jest nieparzyste
2 jest parzyste
4 jest parzyste
5 jest nieparzyste
6 jest parzyste
7 jest nieparzyste
8 jest parzyste
9 jest nieparzyste
10 jest parzyste
Czas trwania 2: 10081
Czas trwania 1: 11066
Done
C:\Users\TadeK\Documents\PROJECTS\pwir05\x64\Debug\pwir05.exe (proces 10064) zakończono z kodem 0.
Naciśnij dowolny klawisz, aby zamknąć to okno...

```

Zrzut nr 2: uruchomienie kodu 7 / 8 po zmianach