Wydział Nauk Inżynieryjnych ANS w Nowym Sączu		
Temat: P6		
Nazwisko i imię: Dominik Żuchowicz	Ocena sprawozdania	Zaliczenie:
Data wykonania ćwiczenia: 17.03.2025	Grupa: L3	

Wprowadzenie

Współczesne systemy operacyjne i procesory oferują różnorodne mechanizmy synchronizacji, które pozwalają na bezpieczne współdzielenie zasobów między wieloma wątkami. Jednym z kluczowych elementów tych mechanizmów są operacje atomowe, które zapewniają integralność danych i eliminują problemy związane z wyścigami danych.

Zadania

1. Opisz czym są operacje atomowe.
2. Dlaczego są tak przydatne w programowaniu wielowątkowym?
3. Przetestuj KOD7/8 zastępując mutex operacjami atomowymi.

Operacje atomowe

Operacje atomowe to takie operacje, które są wykonywane jako niepodzielne, co oznacza, że nie mogą zostać przerwane przez inne wątki w trakcie ich wykonywania. Są one realizowane sprzętowo przez procesor, co zapewnia ich integralność i spójność. Przykłady takich operacji to inkrementacja, dekrementacja, czy porównanie i zamiana wartości w pamięci.

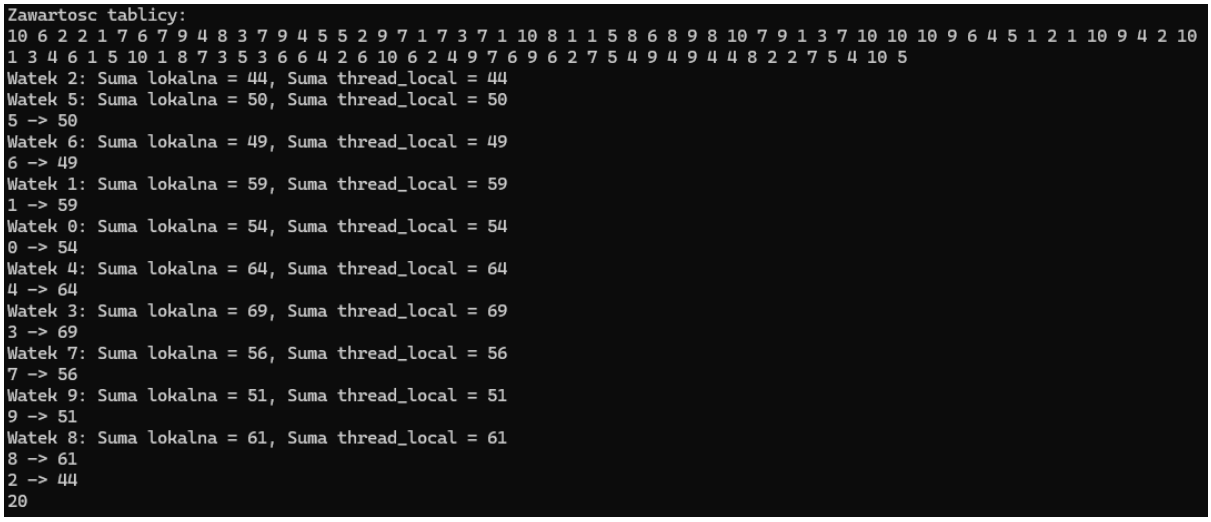
Przydatność w programowaniu wielowątkowym

Operacje atomowe są niezwykle przydatne w programowaniu wielowątkowym, ponieważ pozwalają uniknąć problemów związanych z wyścigami danych (ang. race conditions). Dzięki nim można bezpiecznie modyfikować współdzielone zasoby bez konieczności stosowania bardziej kosztownych mechanizmów synchronizacji, takich jak muteksy czy semafory. To z kolei prowadzi do poprawy wydajności i uproszczenia kodu w aplikacjach wielowątkowych.

Testowanie KOD7/8

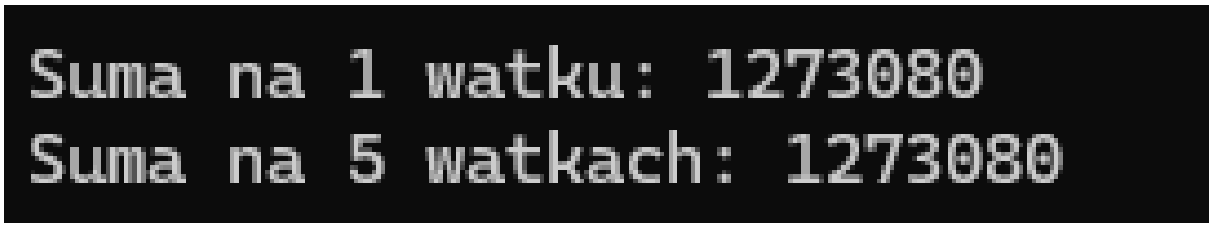
W ramach ćwiczenia przetestowano KOD7/8, zastępując mutex operacjami atomowymi. Poniżej przedstawiono fragmenty kodu, które ilustrują tę zmianę.

Przed zmianą:



Rysunek 1: Przed zmianą

Po zmianie:



Rysunek 2: Po zmianie

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

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.fetch_add(data[i], std::memory_order_relaxed);
    }
}

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 watku: %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 watkach: %llu\r\n", sum2.load());

    delete[] data;
}
```

Wnioski

W wyniku przeprowadzonych testów stwierdzono, że zastosowanie operacji atomowych w miejsce mutexów znacząco poprawiło wydajność aplikacji. Operacje atomowe są szybsze i bardziej efektywne, co czyni je idealnym rozwiązaniem w przypadku prostych operacji na współdzielonych zasobach. Warto jednak pamiętać, że w bardziej złożonych scenariuszach, gdzie konieczne jest zarządzanie większą ilością danych lub bardziej skomplikowanymi strukturami, muteksy mogą być bardziej odpowiednie.