Bezpieczeństwo oprogramowania - lab 13

Zadanie 1

Kod programu

```
#include <iostream>
#include <mutex>
#include <thread>

int main() {
    int numberOfPhilosophers;
    std::cout << "How any philosophers: ";
    std::cin >> numberOfPhilosophers;

    struct Fork {
    public:
        std::mutex mutex;
    };

    auto eat = [](Fork &leftFork, Fork &rightFork, int index) {

        std::unique_lock<std::mutex> leftLock(leftFork.mutex);
        std::unique_lock<std::mutex> rightLock(rightFork.mutex);

        std::cout << "Philosopher " << index << " is eating." << std::endl;

        std::this_thread::sleep_for(std::chrono::milliseconds(1500));

        std::cout << "Philosopher " << index << " finished eating." <<
std::endl;
    };

    Fork forks[numberOfPhilosophers];
    std::thread philosophers[numberOfPhilosophers];

    for (int i = 0; i < numberOfPhilosophers; i++) {
        std::cout << "Philosopher " << (i + 1) << " is thinking." << std::endl;
        philosophers[i] = std::thread(eat, std::ref(forks[i]),
```

```
                                        std::ref(forks[(i + numberOfPhilosophers
- 1) % numberOfPhilosophers]), (i + 1));
    }

    for (auto &philosopher: philosophers) {
        philosopher.join();
    }

    return 0;
}
```

Problem da się rozwiązać bez uzycia blokad - za pomocą operacji atomowych.

Zadanie 2

`read-and-write` - operacja atomowa, która jednocześnie odczytuje lokalizację pamięci i zapisuje
do niej nową wartość - zupełnie nową lub z pewną funkcją poprzedniej wartości

`compare-and-swap` - operacja atomowa, która polega na porównaniu zawartości pewnej lokacji w pamięci z zadaną wartością a następnie, jeśli obie wartości są równe, jej zmodyfikowaniu do nowej wartości