Systemy operacyjne

Problem ucztujących filozofów

Michał Wróblewski

Data wykonania ćwiczenia: 8.01.2024 r.

Wprowadzenie

Problem jedzenia filozofów jest klasycznym przykładem synchronizacji w programowaniu wielowątkowym. Symuluje on sytuację, w której filozofowie siedzą przy okrągłym stole i jedzą. Pomiędzy każdymi dwoma filozofami znajduje się widelec. Filozofowie mogą myśleć lub jeść, ale aby zjeść, muszą użyć dwóch widelców znajdujących się po bokach od nich.

Implementacja w C++

Poniżej znajduje się implementacja problemu jedzenia filozofów w języku C++:

```
#include <iostream>
#include <thread>
#include <mutex>
#include <condition_variable>
#include <vector>
#include <string>

#ifdef _WIN32
#include <windows.h> // Dodane dla funkcji czyszczenia konsoli na
↪  systemie Windows
#else
#include <unistd.h> // Dodane dla funkcji czyszczenia konsoli na
↪  systemach Unix
#endif

using namespace std;

const int N = 5; // liczba filozofów
const int EAT_TIME = 10; // czas jedzenia w sekundach
const int THINK_TIME = 10; // czas myślenia w sekundach
```

```
mutex mtx; // mutex do synchronizacji dostępu do konsoli
condition_variable cv; // zmienna warunkowa do sygnalizowania
↪ dostępności widelców
vector<bool> forks(N, true); // wektor przechowujący stan widelców
↪ (true - wolny, false - zajęty)
vector<int> eating(N, 0); // wektor przechowujący liczbę posiłków
↪ zjedzonych przez każdego filozofa

// Funkcja do czyszczenia konsoli
void clear_console() {
#ifdef _WIN32
    system("cls");
#else
    system("clear");
#endif
}

// funkcja pomocnicza do wypisywania stanu filozofów na konsolę
void print_status(int id, string status) {
    unique_lock<mutex> lock(mtx); // blokujemy mutex do konsoli
    clear_console(); // czyszczenie konsoli
    cout << "Filozof " << id << " " << status << ".\n"; // wypisujemy
    ↪ status filozofa
    cout << "Filozofowie, ktorzy jedza: "; // wypisujemy filozofów,
    ↪ którzy jedzą
    for (int i = 0; i < N; i++) {
        if (eating[i] > 0) {
            cout << i << " ";
        }
    }
    cout << "\nFilozofowie, ktorzy mysla: "; // wypisujemy filozofów,
    ↪ którzy myślą
    for (int i = 0; i < N; i++) {
        if (eating[i] == 0) {
            cout << i << " ";
        }
    }
    cout << "\n\n";
    lock.unlock(); // odblokowujemy mutex do konsoli
}

// funkcja symulująca zachowanie filozofa
void philosopher(int id) {
```

```
    int left = id; // indeks lewego widelca
    int right = (id + 1) % N; // indeks prawego widelca
    while (true) {
        // filozof myśli
        print_status(id, "mysli");
        this_thread::sleep_for(chrono::seconds(THINK_TIME));

        // filozof próbuje zjeść
        print_status(id, "jest glodny");
        unique_lock<mutex> lock(mtx); // blokujemy mutex do widelców
        cv.wait(lock, [left, right] {return forks[left] &&
        ↪  forks[right]; }); // czekamy na wolne widelce
        forks[left] = forks[right] = false; // bierzemy widelce
        eating[id]++; // zwiększamy liczbę posiłków filozofa
        lock.unlock(); // odblokowujemy mutex do widelców

        // filozof je
        print_status(id, "je");
        this_thread::sleep_for(chrono::seconds(EAT_TIME));

        // filozof kończy jeść
        lock.lock(); // blokujemy mutex do widelców
        forks[left] = forks[right] = true; // odkładamy widelce
        eating[id]--; // zmniejszamy liczbę posiłków filozofa
        cv.notify_all(); // informujemy innych filozofów o zwolnieniu
        ↪  widelców
        lock.unlock(); // odblokowujemy mutex do widelców
    }
}

int main() {
    vector<thread> philosophers; // wektor wątków dla filozofów
    for (int i = 0; i < N; i++) {
        philosophers.push_back(thread(philosopher, i)); // tworzymy i
        ↪  uruchamiamy wątek dla każdego filozofa
    }
    for (auto& p : philosophers) {
        p.join(); // czekamy na zakończenie wątków (nigdy się nie
        ↪  skończą)
    }
    return 0;
}
```

Rozwiązanie problemu

Aby uniknąć zakleszczeń i zapewnić, że filozofowie będą mogli efektywnie dzielić się widelcami, używamy mutexów i warunkowych z biblioteki standardowej C++. Mutexy są używane do synchronizacji dostępu do konsoli oraz do widelców, a warunkowe do informowania filozofów o dostępności widelców.

Demonstracja działania

Podczas działania programu, filozofowie będą naprzemiennie myśleć i jeść, starając się unikać konfliktów o widelce. Status każdego filozofa oraz aktualnie zajętych widelców jest wyświetlany na konsoli.

Podsumowanie

Symulacja problemu jedzenia filozofów jest ciekawym przykładem zastosowania synchronizacji w programowaniu wielowątkowym. Implementacja w C++ wykorzystuje mutexy i zmienne warunkowe do unikania problemów związanych z dostępem do wspólnych zasobów. Program pozwala zobaczyć, jak filozofowie efektywnie dzielą się widelcami, jednocześnie unikając zakleszczeń.