Systemy Operacyjne 2

Projekt: Problem ucztujących filozofów

Mateusz Łysak 272994

Cel zadania

Zadanie polegało na zaimplementowaniu trzech wariantów rozwiązania problemu pięciu ucztujących filozofów, uwzględniających:

1. **Zakleszczenie** – sytuację, w której wszyscy filozofowie blokują się wzajemnie.
2. **Zagłodzenie** – sytuację, w którym jeden filozof jest trwale pomijany w dostępie do zasobów.
3. **Rozwiązanie poprawne** – gwarantujące brak zakleszczenia i zagłodzenia.

Realizacja

Program został napisany w języku C++ z wykorzystaniem biblioteki **ncurses** (interfejs tekstowy) oraz mechanizmów wielowątkowych (<thread>, <mutex>).

1. Wersja z zakleszczeniem (DeadlockVersion)

- **Mechanizm**:
  Każdy filozof najpierw próbuje zablokować **lewą pałeczkę**, a następnie **prawą**. Brak synchronizacji między wątkami.
- **Problem**:
  Jeśli wszyscy filozofowie jednocześnie zdobędą lewą pałeczkę, będą czekać w nieskończoność na prawą (żaden nie zwolni swojej).

2. Wersja z zagłodzeniem (StarvationVersion)

- **Mechanizm**:
  Filozof o id=0 (Yoda) próbuje najpierw **prawą pałeczkę** z użyciem try_lock(), a po niepowodzeniu czeka dłużej niż inni. Pozostali filozofowie działają standardowo.
- **Problem**:
  Yoda, z powodu nietypowego schematu (prawa-lewa) i dłuższych opóźnień, jest stale wyprzedzany przez innych.

3. Wersja poprawna (CorrectVersion)

- **Mechanizm**:
  Wprowadzono **arbitra (globalny mutex)**, który wymusza sekwencyjne pobieranie pałeczek. Tylko jeden filozof może próbować je zdobyć w danej chwili.
- **Zalety**:
  - Eliminuje zakleszczenie (brak cyklicznego oczekiwania).
  - Zapobiega zagłodzeniu (wszyscy mają równy dostęp do arbitra).

Zakleszczenie:

```
115 class DeadlockVersion : public DiningPhilosophers {
116 public:
117     void philosophize(int id) override {
118         int left = id;
119         int right = (id + 1) % NUM_PHILOSOPHERS;
120 
121         while (running) {
122             states[id] = PhilosopherState::HUNGRY;
123             display_status(id);
124 
125             chopsticks[left].lock();
126             {
127                 lock_guard<mutex> lock(display_mutex);
128                 last_chopstick_users[left] = current_chopstick_users[left];
129                 current_chopstick_users[left] = id;
130             }
131             display_status(id);
132             this_thread::sleep_for(chrono::milliseconds(100));
133 
134             chopsticks[right].lock();
135             {
136                 lock_guard<mutex> lock(display_mutex);
137                 last_chopstick_users[right] = current_chopstick_users[right];
138                 current_chopstick_users[right] = id;
139             }
140 
141             states[id] = PhilosopherState::EATING;
142             meals_eaten[id]++;
143             display_status(id);
144 
145             this_thread::sleep_for(chrono::seconds(eat_dist(gen)));
146 
147             {
148                 lock_guard<mutex> lock(display_mutex);
149                 last_chopstick_users[left] = current_chopstick_users[left];
150                 current_chopstick_users[left] = -1;
151                 last_chopstick_users[right] = current_chopstick_users[right];
152                 current_chopstick_users[right] = -1;
153             }
154             chopsticks[left].unlock();
155             chopsticks[right].unlock();
156 
157             states[id] = PhilosopherState::THINKING;
158             display_status(id);
159 
160             this_thread::sleep_for(chrono::seconds(think_dist(gen)));
161         }
162     }
163 };
```

Zagłodzenie:

```
class StarvationVersion : public DiningPhilosophers {
public:
    void philosophize(int id) override {
        int left = id;
        int right = (id + 1) % NUM_PHILOSOPHERS;

        while (running) {
            states[id] = PhilosopherState::HUNGRY;
            display_status(id);

            if (id == 0) {
                if (!chopsticks[right].try_lock()) {
                    this_thread::sleep_for(chrono::milliseconds(think_dist(gen) * 2));
                    continue;
                }

                {
                    lock_guard<mutex> lock(display_mutex);
                    last_chopstick_users[right] = current_chopstick_users[right];
                    current_chopstick_users[right] = id;
                }

                this_thread::sleep_for(chrono::milliseconds(200));
                if (!chopsticks[left].try_lock()) {
                    chopsticks[right].unlock();
                    {
                        lock_guard<mutex> lock(display_mutex);
                        last_chopstick_users[right] = current_chopstick_users[right];
                        current_chopstick_users[right] = -1;
                    }
                    this_thread::sleep_for(chrono::milliseconds(think_dist(gen) * 2));
                    continue;
                }

                {
                    lock_guard<mutex> lock(display_mutex);
                    last_chopstick_users[left] = current_chopstick_users[left];
                    current_chopstick_users[left] = id;
                }
            }
            else {
                chopsticks[left].lock();
                {
                    lock_guard<mutex> lock(display_mutex);
                    last_chopstick_users[left] = current_chopstick_users[left];
                    current_chopstick_users[left] = id;
                }

                this_thread::sleep_for(chrono::milliseconds(50));

                chopsticks[right].lock();
                {
                    lock_guard<mutex> lock(display_mutex);
                    last_chopstick_users[right] = current_chopstick_users[right];
                    current_chopstick_users[right] = id;
                }
            }
```

```
222
223             states[id] = PhilosopherState::EATING;
224             meals_eaten[id]++;
225             display_status(id);
226
227             this_thread::sleep_for(chrono::seconds(eat_dist(gen)));
228
229             {
230                 lock_guard<mutex> lock(display_mutex);
231                 last_chopstick_users[left] = current_chopstick_users[left];
232                 current_chopstick_users[left] = -1;
233                 last_chopstick_users[right] = current_chopstick_users[right];
234                 current_chopstick_users[right] = -1;
235             }
236             chopsticks[left].unlock();
237             chopsticks[right].unlock();
238
239             states[id] = PhilosopherState::THINKING;
240             display_status(id);
241
242             this_thread::sleep_for(chrono::seconds(id == 0 ? think_dist(gen) * 2 : think_dist(gen)));
243         }
244     }
245 };
```

Poprawnie działający:

```
class CorrectVersion : public DiningPhilosophers {
private:
    mutex arbitrator;

public:
    void philosophize(int id) override {
        int left = id;
        int right = (id + 1) % NUM_PHILOSOPHERS;

        while (running) {
            states[id] = PhilosopherState::HUNGRY;
            display_status(id);

            arbitrator.lock();

            chopsticks[left].lock();
            {
                lock_guard<mutex> lock(display_mutex);
                last_chopstick_users[left] = current_chopstick_users[left];
                current_chopstick_users[left] = id;
            }
            chopsticks[right].lock();
            {
                lock_guard<mutex> lock(display_mutex);
                last_chopstick_users[right] = current_chopstick_users[right];
                current_chopstick_users[right] = id;
            }

            states[id] = PhilosopherState::EATING;
            meals_eaten[id]++;
            display_status(id);

            this_thread::sleep_for(chrono::seconds(eat_dist(gen)));

            {
                lock_guard<mutex> lock(display_mutex);
                last_chopstick_users[left] = current_chopstick_users[left];
                current_chopstick_users[left] = -1;
                last_chopstick_users[right] = current_chopstick_users[right];
                current_chopstick_users[right] = -1;
            }
            chopsticks[left].unlock();
            chopsticks[right].unlock();

            arbitrator.unlock();

            states[id] = PhilosopherState::THINKING;
            display_status(id);

            this_thread::sleep_for(chrono::seconds(think_dist(gen)));
        }
    }
};
```