SPRAWOZDANIE

Systemy operacyjne 2

Dokumentacja do projektu

Autor:
Patryk Dulęba 259213
Środa 11:15

Prowadzący:
Mgr inż. Mateusz Gniewkowski

1 Wstęp

Niniejsza dokumentacja ma na celu przedstawienie projektu z przedmiotu Systemy Operacyjne 2, który polegał na napisaniu aplikacji w języku Java, rozwiązującej problem ucztujących filozofów. Problem ten jest klasycznym zagadnieniem w informatyce, które ilustruje potencjalne trudności występujące w systemach rozproszonych, takich jak dostęp do wspólnych zasobów.

Głównym celem projektu było zaprojektowanie i implementacja programu, który umożliwiłby filozofom prowadzenie swoich "uczt" w sposób bezkonfliktowy i sprawiedliwy. W problemie ucztujących filozofów uczestniczy pewna liczba filozofów (reprezentowanych jako wątki), siedzących wokół okrągłego stołu. Na stole znajduje się n talerzyków i n widelców, gdzie n jest liczbą filozofów.

Każdy filozof ma swoje własne zadanie - ucztować. Jednakże, przed rozpoczęciem posiłku musi on zająć dwa sąsiadujące ze sobą widelce. Problem polega na tym, że widelce są zasobami współdzielonymi między filozofami (sekcja krytyczna). Każdy filozof musi podjąć decyzję, kiedy sięgać po widelec, a kiedy je oddać, aby uniknąć zakleszczeń i zagłodzenia.

2 Implementacja

2.1 Opis fragmentów kodu

```
public void start() {
        Philosopher[] philosophers = new Philosopher[5];
        Object[] forks = new Object[philosophers.length];

        Arrays.setAll(forks, i -> new Object());

        for (int i = 0; i < philosophers.length; i++) {

            Object leftFork = forks[i];
            Object rightFork = forks[(i + 1) % forks.length];

            if(i != philosophers.length-1) {
                philosophers[i] = new Philosopher(leftFork, rightFork);
            } else {
                philosophers[i] = new Philosopher(rightFork, leftFork);
            }
                int finalI = i;
                philosophers[i].setPsi(
                        new ShowUpdate() {
                            public void showStatus() {
                                labels[finalI].setText("P " + (finalI + 1) + " " +
    philosophers[finalI].getActualActivity());
                            }
                        }
                );

            Thread t = new Thread(philosophers[i], "Philosopher " + (i + 1));
            allThreads.add(t);
            t.start();
        }
    }
```

Powyższa metoda **start()** jest odpowiedzialna za rozpoczęcie symulacji problemu ucztujących filozofów. Na początku są tworzeni filozofowie oraz widelce, a następnie każdemu filozofowi przypisywane są dwa sąsiadujące sztućce. Każdy z filozofów jest uruchamiany jako osobny wątek. W celu uniknięcia zakleszczenia (deadlock), ostatniemu filozofowi przypisywane są widelce w odwrotnej kolejności niż pozostałym filozofom.

```
public void run() {
        try {
            while (true) {
                actualActivity = "Thinking";
                psi.showStatus();
                Thread.sleep(((int) (Math.random() * 1000)));
                synchronized (leftFork) {
                    synchronized (rightFork) {
                        actualActivity = "Eating";
                        psi.showStatus();
                        Thread.sleep((int) (Math.random() * 2000));
                    }
                    actualActivity = "Thinking";
                    psi.showStatus();
                    Thread.sleep(((int) (Math.random() * 1000)));
                }
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
```

Powyższy kod ustawia stan, w jakim obecnie znajduje się dany wątek. Aby filozof mógł zacząć, jeść musi uzyskać dostęp do dwóch widelców. Jeśli widelce są zajęte przez innych filozofów, to aktualny filozof czeka, aż widelce zostaną zwolnione. Do synchronizacji dostępu do sekcji krytycznej wykorzystałem metodę **synchronized()**.

2.2 Opis mechanizmu synchronizacji

W Javie słowo kluczowe synchronized służy do synchronizacji dostępu do bloku kodu lub metody. Gwarantuje ono, że tylko jeden wątek na raz może wejść do oznaczonego fragmentu kodu, zwany sekcją krytyczną.

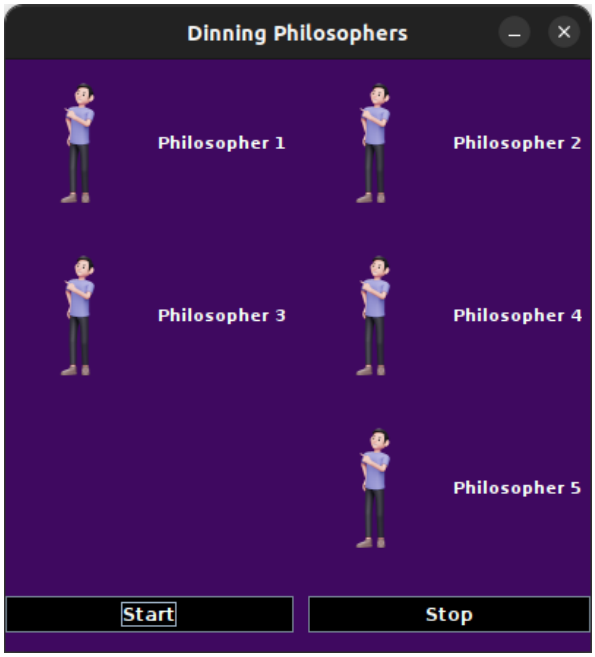
Kiedy wątek chce wejść do bloku kodu oznaczonego jako synchronized, sprawdza, czy żadne inne wątki nie mają już dostępu do tego bloku. Jeśli blok jest wolny, wątek zyskuje dostęp i wykonuje kod wewnątrz bloku. W tym czasie inne wątki, które próbują uzyskać dostęp do tego samego bloku, muszą czekać.

Jeśli inny wątek już zajmuje blok synchronized, wątek próbujący wejść do tego bloku zostanie zawieszony (wstrzymany) do momentu, aż blok będzie dostępny. W momencie, gdy wątek, który posiadał blok synchronized, zakończy wykonanie kodu wewnątrz bloku lub opuści blok, inny wątek zostaje obudzony i uzyskuje dostęp.

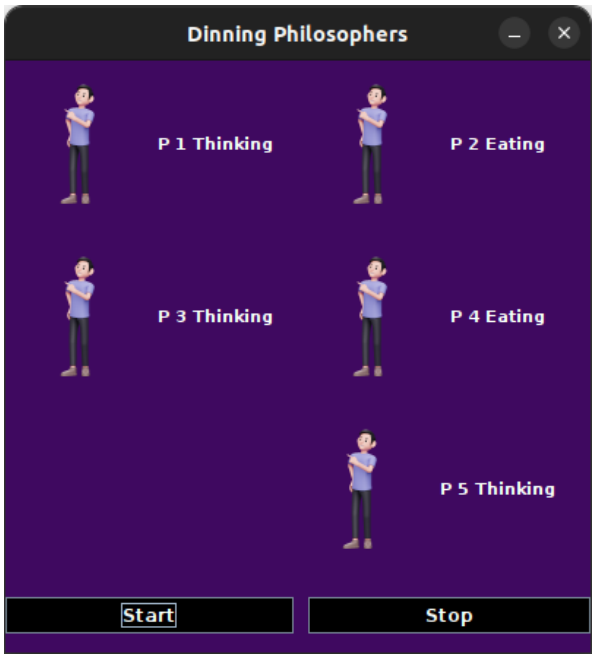
Istnieją również inne bardziej zaawansowane techniki synchronizacji, takie jak semafory czy monitorowanie, które mogą być wykorzystane do rozwiązania problemu ucztujących filozofów. Jednakże, przy użyciu metody synchronized() można zaimplementować podstawowe rozwiązanie tego problemu.

2.3 Działanie programu

Do końcowej wersji programu zaimplementowany został prosty interfejs graficzny przy użyciu biblioteki **Swing**. Po naciśnięciu przycisku **Start** przy każdym filozofie zostaje wyświetlony stan, w jakim aktualnie się znajduje. W celu zatrzymania działania programu możemy nacisnąć przycisk **Stop**.



Rysunek 1: Uruchomienie aplikacji



Rysunek 2: Działanie aplikacji

3 Uruchomienie

W celu uruchomienia programu należy upewnić się, czy na twoim systemie jest zainstalowana Java Development Kit (JDK), ponieważ jest ona niezbędna do uruchamiania plików .jar. Następnie należy uruchomić terminal i przejść do folderu, w którym znajduje się nasz program i wykonać poniższą komendę:

```
java -jar SO2.jar
```

4 Wnioski

W początkowej wersji programu nie było zabezpieczenia przed zakleszczeniem. Jeśli każdy filozof chwycił jeden widelec i czekał na drugi, mogło dojść do zakleszczenia, gdzie żaden z filozofów nie był w stanie kontynuować. Aby temu zapobiec, można zastosować różne strategie, takie jak: numerowanie widelców, ograniczenie liczby filozofów jednocześnie chwytających widelec, zastosowanie hierarchii przydzielania widelców itp. Problem został rozwiązany poprzez zamienienie kolejności widelców ostatniemu z filozofów.

W programie nadal istnieje problem zagłodzenia, który polega na tym, że istnieje możliwość, iż jeden z filozofów będzie miał znacznie mniejszą liczbę okazji do jedzenia w porównaniu do pozostałych.