Sprawozdanie

Teoria Współbieżności

Laboratorium 2

Joanna Bryk, środa 17:50

1. Wstęp

Celem laboratorium 2. była nauka działania semaforów w Javie, zapoznanie się z metodami *wait* i *notify*. Dzięki tym narzędziom możliwe było ulepszenie programu laboratorium 1. tak, aby wątki były poprawnie zsynchronizowane.

2. Rozwiązanie zadań

2.1. Implementacja semafora binarnego.

```
public class BinarySemaphore {
    private boolean state = true;
    private int waiting = 0;

    public BinarySemaphore() {
    }

    public synchronized void P() {  //opuszczanie
        if(!state){
            waiting++;
            while(!state){
                try {
                    wait();
                }
                catch (InterruptedException e){
                        e.printStackTrace();
                }
            }
            waiting--;
        }
        state = false;

    }

    public synchronized void V() { //podnoszenie
        if(waiting > 0){
            this.notify();
        }
        state = true;
    }
}
```

Zaimplementowaną klasę użyłam do synchronizacji programu.

```
public class Counter {
    private BinarySemaphore sem;
    private int _val;
    public Counter(int n) {
        _val = n;
        sem = new BinarySemaphore();
    }
    public void inc() {
        sem.P();
        _val++;
        sem.V();
    }
    public void dec() {
        sem.P();
        _val--;
        sem.V();
    }
    public int value() {
        return _val;
    }
}
```

```
public class Race {
    public static void main(String[] args) throws InterruptedException {

            Counter cnt = new Counter(0);
            IThread iThread = new IThread(cnt);
            DThread dThread = new DThread(cnt);
            iThread.start();
            dThread.start();

            iThread.join();
            dThread.join();

            System.out.println(cnt.value());
    }
}
```

Semafor spełnił swoje zadanie, wyniki wychodziły poprawne.

2.2. Kolejnym zadaniem było sprawdzenie, czy w implementacji semafora binarnego wystarczający jest warunek if (bez while).

```
public class BinarySemaphore {
    private boolean state = true;
    private int waiting = 0;

    public BinarySemaphore() {
    }

    public synchronized void P() {
        if(!state){
            waiting++;
                try {
                    wait();
                }
                catch (InterruptedException e){
                    e.printStackTrace();
            }
```

```
            waiting--;
        }
        state = false;
    }

    public synchronized void V() {
        state = true;
        if(waiting > 0){
            this.notify();
        }
    }
}
```

W tym przypadku stan zmiennej Counter na końcu działania programu często był różny od 0 – semafor binarny w takiej postaci nie działa poprawnie.

2.3. Następnie zaimplementowałam semafor ogólny (licznikowy) przy pomocy semafora binarnego. W tej implementacji potrzebne są dwa semafory binarne i jedna zmienna (*resources*).

```
public class Semaphore {
    private final BinarySemaphore mutexSemaphore;
    private final BinarySemaphore waitSemaphore;
    private int resources;

    public Semaphore(int resources){
        this.resources = resources;
        mutexSemaphore = new BinarySemaphore(true);
        waitSemaphore = new BinarySemaphore(false);
    }

    public void P() {
        mutexSemaphore.P();
        resources-=1;

        if(resources < 0){
            mutexSemaphore.V();
            waitSemaphore.P();
        }
        mutexSemaphore.V();
    }

    public void V() {
        mutexSemaphore.P();
        resources+=1;
        if(resources <= 0){
            waitSemaphore.V();
        }
        else
            mutexSemaphore.V();
    }
}
```

Semafor również zsynchronizował wątki. Wyniki wychodziły poprawne.

3. Podsumowanie

3.1. Semafor binarny jest odpowiednim narzędziem do synchronizacji wątków.
3.2. Przy użyciu funkcji *wait* należy użyć pętli *while* (nie wystraczy warunek *if*). Pętla *while* musi tam zostać użyta, aby „zatrzymać" program przed wykonywaniem kolejnych intstrukcji aż stan będzie ustawiony na true. *If* jest niewystarczający, ponieważ jedynie sprawdza warunek, nie pozwala kontynuować „oczekiwania".
3.3. Semafor binarny nie jest szczególnym przypadkiem semafora licznikowego, ponieważ semafor licznikowy jest zmienną całkowitą a semafor ogólny jest to zmienna logiczna, pojedynczy bit przyjmujący wartości 0 lub 1. Semafor binarny nie pamięta liczby wykonanych na nim operacji podnoszenia.

4. Bibliografia

- Zbigniew Weiss, Tadeusz Gruźlewski, *Programowanie współbieżne i roproszone w przykładach i zadanich,* Wydawnictwo Naukowo-Techniczne