Lab 2 - Teoria Współbieżności

Piotr Świerzy

Ćwiczenie 2

1)

Nie jest to poprawne działanie.

Np.

Czas / Krok	Proces P	Proces K	dzialaj
T1	P1	K1	true
T2	P1	K2	true
T3	P1	K3	true
T4	P2	K3	false
T5	**P3**	**K3**	**false**

czyli dochodzi do wzajemnego wykluczania

2)

Nie jest to poprawne działanie.

Np.

Czas / Krok	Proces P	Proces K	dzialaj
T1	P1	K1	true
T2	P2	K1	true
T3	P2	K2	true
T4	P3	K2	false
T5	P3	K3	false
T6	**P4**	**K3**	**false**

3)

Jest to poprawne działanie. Chyba że jeden z procesów przestanie działać, wtedy drugi będzie deadlocked.

4)

Nie poprawne

Czas / Krok	Proces P	Proces K	turaP	turaK
T1	P1	K1	false	false
T2	P2	K1	false	false
T3	P2	K2	false	false
T4	P3	K2	true	false
T5	P3	K3	true	true
T6	P4	K3	true	true
T7	**P4**	**K4**	**true**	**true**

5)

Nie poprawne

Czas / Krok	Proces P	Proces K	turaP	turaK
T1	P1	K1	false	false
T2	P2	K1	true	false
T3	P2	K2	true	true
T4	P3	K2	true	true
T5	**P3**	**K3**	**true**	**true**

Dochodzi do deadlocku

6)

Nie jest zapewniony brak zagłodzenia.

Czas / Krok	Proces P	Proces K	turaP	turaK
T1	P1	K1	false	false
T2	P2	K1	true	false
T3	P3	K1	true	false
T4	P6	K1	true	false
T3	P7	K1	false	false
T3	P1	K1	false	false
T3	P2	K1	true	false
T3	P3	K1	true	false
T3	**P6**	**K1**	**true**	**false**

Ćwiczenie 2a

1) Semafor Binarny

```
public class SemaforBinarny {
    private boolean stan = true;
```

```
    private int czeka = 0;
    public SemaforBinarny() {
    }
    public synchronized void P() {
        czeka++;
        while(!stan) {
            try{
                wait();
            } catch(InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
        czeka--;
        stan = false;
    }
    public synchronized void V() {
        stan = true;
        notify();
    }
}
```

2) if vs while

```
public class SemaforIF {
    private boolean stan = true;
    private int czeka = 0;
    public SemaforIF() {
    }
    public synchronized void P() {
        czeka++;
        if(!stan) {
            try{
                wait();
            } catch(InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
        czeka--;
        stan = false;
    }
    public synchronized void V() {
        stan = true;
        notify();
    }
}
```

Gdy uzywamy IF zamiast WHILE:

- Wątek A wywołuje wait() i zwalnia monitor
- Wątek B wywołuje notify() -> budzi wątek (np. A)
- Wątek A przechodzi do wejścia
- ALE w tym czasie chodzi wątek C i zmienia warunek
- Wątek A wchodzi, ale ze względu na ifa nie sprawdza warunku jeszcze raz
- Wątek A kontynuuje mimo że warunek nie jest spełniony

Co powoduje problemy (klasę Wyścig pokaże w punkcie 4)

3) Semafor Licznikowy

```
public class SemaforLicznikowy {
    private int wartosc;
    private final SemaforBinarny mutex;
    private final SemaforBinarny kolejka;
    private int czekajace = 0;

    public SemaforLicznikowy(int poczatkowaWartosc) {
        this.wartosc = poczatkowaWartosc;
        this.mutex = new SemaforBinarny();
        this.kolejka = new SemaforBinarny();
        this.kolejka.P();
    }


    public void P() {
        mutex.P();

        wartosc--;
        if (wartosc < 0) {
            czekajace++;
            mutex.V();
            kolejka.P();
            czekajace--;
        } else {
            mutex.V();
        }
    }


    public void V() {
        mutex.P();

        wartosc++;
        if (czekajace > 0) {
```

```
            kolejka.V();
        }

        mutex.V();
    }
}
```

4) Wyścig i przedstawienie różnic

```
public class Wyscig {
    private static int licznik = 0;
    private static final int ITERACJE = 100000;
    private static SemaforBinarny semaforBinarny = new SemaforBinarny();
    private static SemaforIF semaforIF = new SemaforIF();

    static class InkrementujacyWatek extends Thread {
        @Override
        public void run() {
            for(int i = 0; i < ITERACJE; i++) {
                semaforBinarny.P();
                licznik++;
                semaforBinarny.V();
            }
        }
    }
    static class InkrementujacyWatekIF extends Thread {
        @Override
        public void run() {
            for(int i = 0; i < ITERACJE; i++) {
                semaforIF.P();
                licznik++;
                semaforIF.V();
            }
        }
    }

    public static void main(String[] args) throws InterruptedException {
        // Test 1: Bez semafora
        System.out.println("Test 1: BEZ synchronizacji");
        licznik = 0;
        Thread[] t = new Thread[5];
        for(int i = 0; i < 5; i++) {
            t[i] = new Thread(() -> {
                for (int j = 0; j < ITERACJE; j++) {
                    licznik++;
                }
```

```
            });
            t[i].start();
        }
        for (Thread thread : t) {
            thread.join();
        }
        System.out.println("Oczekiwana wartosc: " + (5 * ITERACJE));
        System.out.println("Faktyczna wartosc:  " + licznik);
        System.out.println("Blad wyscigu: " + ((5 * ITERACJE) - licznik) + "\n");

        // Test 2: Z semaforem
        System.out.println("Test 2: Z semaforem binarnym");
        licznik = 0;
        Thread[] t2 = new Thread[5];
        for (int i = 0; i < 5; i++) {
            t2[i] = new InkrementujacyWatek();
            t2[i].start();
        }
        for (Thread thread : t2) {
            thread.join();
        }
        System.out.println("Oczekiwana wartosc: " + (5 * ITERACJE));
        System.out.println("Faktyczna wartosc:  " + licznik);
        System.out.println("Blad: " + ((5 * ITERACJE) - licznik));

        // Test 3: Semafor z if zamiast while
        System.out.println("Test 3: Z semaforem z if");
        licznik = 0;
        Thread[] t3 = new Thread[5];
        for (int i = 0; i < 5; i++) {
            t3[i] = new InkrementujacyWatekIF();
            t3[i].start();
        }
        for (Thread thread : t3) {
            thread.join();
        }
        System.out.println("Oczekiwana wartosc: " + (5 * ITERACJE));
        System.out.println("Faktyczna wartosc:  " + licznik);
        System.out.println("Blad: " + ((5 * ITERACJE) - licznik));
    }
}
```

Wynik:

```
Test 1: BEZ synchronizacji
Oczekiwana wartosc: 500000
Faktyczna wartosc:  118980
```

```
Blad wyscigu: 381020

Test 2: Z semaforem binarnym
Oczekiwana wartosc: 500000
Faktyczna wartosc:  500000
Blad: 0

Test 3: Z semaforem z if
Oczekiwana wartosc: 500000
Faktyczna wartosc:  499996
Blad: 4
```