

Laboratorium Nr 2
"Monitory w javie"
Radosław Kopeć
13.10.2020

I. Zadanie 1

- **Treść zadania**

Zaimplementować semafor binarny za pomocą metod wait i notify, użyć go do synchronizacji programu "Wyścig" (patrz sprawozdanie 1)

- **Koncepcja rozwiązania**

Semafor dzięki słowu kluczowemu synchronized zapewnia atomowy dostęp do metod obiektu. Pozwoli nam to zmienić zmienną typu boolean, która będzie determinować to czy kolejny wątek ma czekać czy może przejść dalej. Metoda V() będzie zwalniać jeden wątek.

- **Implementacja i wyniki:**

```
public class Semaphore {
    private boolean state = true;
    private int waitingThreads = 0;

    public Semaphore() {
    }

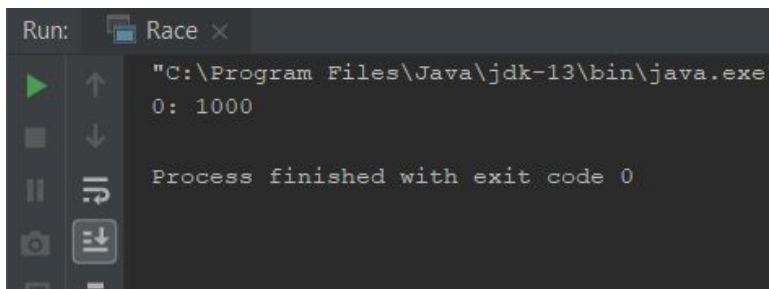
    public synchronized void P() {
        if(!state){
            waitingThreads++;
            while(!this.state) {
                try {
                    wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            waitingThreads--;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

```

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

```

Wynik testu dla tysiąca wywołań wyścigu z poprzedniego laboratorium:



```

Run: Race x
"C:\Program Files\Java\jdk-13\bin\java.exe"
0: 1000

Process finished with exit code 0

```

- **Wnioski**

- Semafor pozwala nam rozwiązać problem wyścigu używając mechanizmów wbudowanych w jave. Wymaga to dodatkowego nakładu pracy. "Pilnuje" on dostępu do jednego zasobu np. Licznika.
- słówko kluczowe synchronized zapewnia atomowy dostęp do metod obiektu.

II. Zadanie 2

- **Treść zadania**

Pokazać, że do implementacji semafora za pomocą metod wait i notify nie wystarczy instrukcja if tylko potrzeba użyć while. Wyjaśnić teoretycznie dlaczego i potwierdzić eksperymentem w praktyce. (wskazówka: rozważyć dwie kolejki: czekając na wejście do monitora obiektu oraz kolejkę związana z instrukcją wait , rozważyć kto kiedy jest budzony i kiedy następuje wyścig).

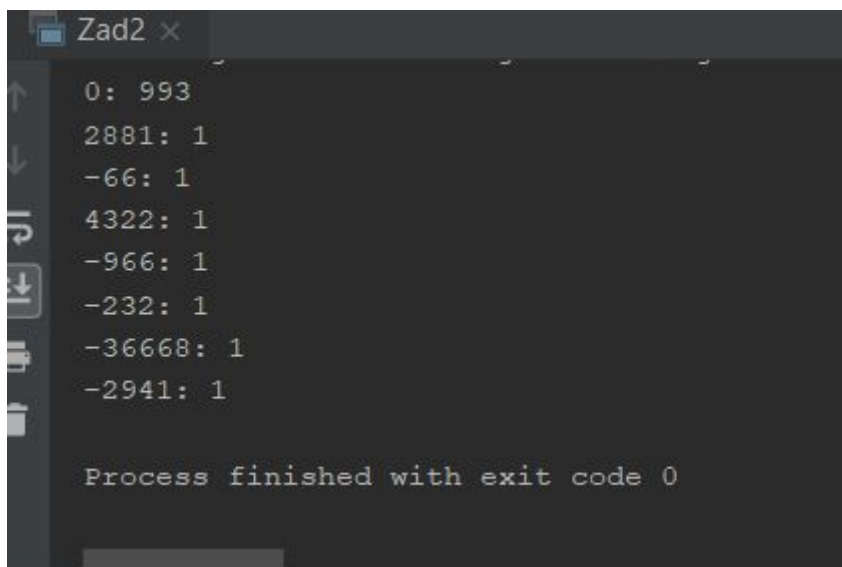
- **Koncepcja rozwiązania**

Zamienić "wait" w metodzie P() w klasie Semaphore na "if"

- Implementacja i wyniki:

```
public synchronized void P() {  
    if(!state){  
        waitingThreads++;  
  
        try {  
            wait();  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
        waitingThreads--;  
    }  
    this.state = false;  
}
```

Wynik testu dla tysiąca wywołań wyściugu z poprzedniego laboratorium:



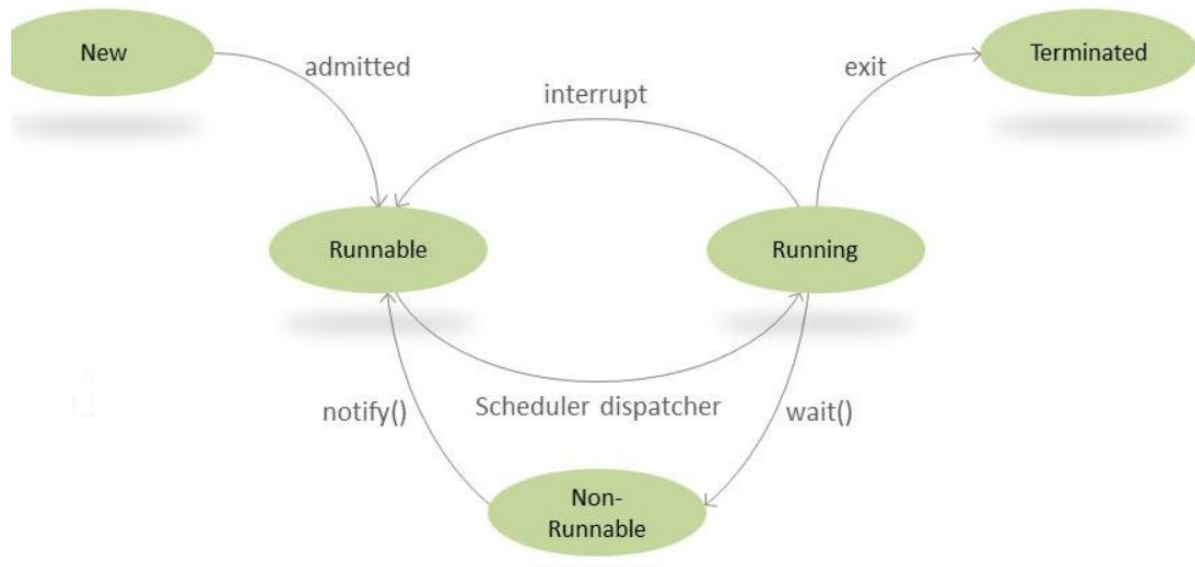
```
Zad2 x  
0: 993  
2881: 1  
-66: 1  
4322: 1  
-966: 1  
-232: 1  
-36668: 1  
-2941: 1  
  
Process finished with exit code 0
```

- Wnioski i wyjaśnienie teoretyczn

-Jak widać zastępując "while" przez "if" uzyskaliśmy nie poprawne wyniki.

Wyjaśnienie Teoretyczne!

Aby zrozumieć dlaczego while jest potrzebny należy dobrze zrozumieć diagram stanów wątku:



Po wykonaniu metody `notify()` na śpiącym wątku, ten nie uzyskuje od razu dostępu do bloku oznaczonym `synchronized` lecz zmienia stan na "Runnable" co oznacza, że dopiero czeka na przydział procesora przez planistę, trafia do kolejki do **procesora**.

Rozważmy zatem następującą sytuację dla poprawnego programu:

1. Wątek 1 wykonuje metodę `P()` uzyskuje dostęp do sekcji krytycznej.
2. Wątek 1 wychodzi z metody `P()`
3. Wątek 2 wchodzi do metody `P()` widzi ustawioną flagę na `false` więc zatrzymuje się na `wait()`.
4. Wątek 1 wykonuje metodę `V()` i budzi wątek 2.
5. Wątek 2 budzi się i czeka na przydział procesora.
6. W tym czasie monitor obiektu nie jest zajęty więc wątek 1 wykonuje metodę `V()` **wchodzi do sekcji krytycznej**. Kończy metodę `V()` więc monitor obiektu jest wolny.
7. Wątek 2 dostaje procesor, próbuje dostać się do bloku `synchronized` i uzyskuje dostęp bo kolejka monitora jest pusta. Wykonuje metodę `V()` i również **wchodzi do sekcji krytycznej**.

Jak widać istnieje niebezpieczeństwo, gdyby wątek 2 w punkcie(7) jeszcze raz sprawdził warunek z flagą, nie doszłoby do tego, zatem pętla while() jest niezbędna. Flaga byłaby ustawiona na false przez wątek 1 w punkcie 6, nie mogłoby być inaczej bo dostępu do metody pilnuje monitor obiektu.

III. Zadanie 3

- **Treść zadania**

Zaimplementować semafor licznikowy (ogólny) za pomocą semaforów binarnych. Czy semafor binarny jest szczególnym przypadkiem semafora ogólnego?

- **Koncepcja rozwiązania**

Koncepcja opierająca się o dwa semaforey, jeden odpowiedzialny za blokowanie dostępu do zmiennej liczącej ile zostało jeszcze dostępnych zasobów. Drugi semafor będzie odpowiedzialny za czekanie gdy zasobów jest zbyt mało.

- **Implementacja i wyniki:**

Do Semaphore dodajemy możliwość przekazania początkowego stanu w argumencie konstruktora.

```
public class Semaphore {
    private boolean state;
    private int waitingThreads = 0;

    public Semaphore(boolean state) { this.state = state; }

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

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

Tworzymy klasę CounterSemaphore, będącą implementacją semafora ogólnego.

```
public class CounterSemaphore {  
  
    private int resources;  
    private Semaphore semaphore;  
    private Semaphore semaphore2;  
  
    CounterSemaphore(int resources){
```

posiada ona pole resources oznaczające liczbę dostępnych zasobów i dwa semaforey binarne (tak jak w koncepcji rozwiązania)

Metody P() i V():

```
public void P() {  
  
    semaphore2.P();  
    resources -= 1;  
  
    if( resources < 0){  
        semaphore2.V();  
        semaphore.P();  
    }  
    else {  
        semaphore2.V();  
    }  
}  
  
public void V() {  
  
    semaphore2.P();  
  
    resources +=1;  
    if(resources <= 0) {  
        semaphore.V();  
    }  
    semaphore2.V();  
}
```

Wynik testu dla tysiąca wywołań wyścigu z poprzedniego laboratorium:

```
"C:\Program Files\Java\jdk-13\bin\java.exe" -Dide
0: 1000

Process finished with exit code 0
```

• Wnioski i odpowiedź na pytanie

- Za pomocą semaforów binarnych można stworzyć semafor ogólny.
- Semafor ogólny można stosować w przypadku gdy chcemy dopuścić pewną ilość wątków do pewnego zasobu.
- Czy semafor binarny jest szczególnym przypadkiem semafora ogólnego? :

Moja intuicja mi podpowiada, że jest to semaphore ogólny z jednym dostępnym zasobem.

IV. Bibliografia

obrazek z zadania 2:

<https://www.baeldung.com/java-wait-notify?fbclid=IwAR0kNATbxWX9wYD9px0IWMmBV0BlBE08fZa9B8vjwIKQYRZivSTPPNWiHyw>
zad3.

Wykłady dr Jarosława Koźlaka, konkretniej ten slajd:

Semafory: Semafor binarny

- Semafor binarny: może przyjmować dwie wartości: 0 lub 1
- Implementacja semafora zliczającego S przy użyciu semaforów binarnych

```
var S1: semafor-binarny;  
S2: semafor-binarny;  
C:integer;  
Inicjacja: S1=1; S2=0;  
C= wartość początkowa semafora zliczającego S;
```

czekaj (S) :

```
czekaj (S1);  
C:=C-1;  
if C<0 then begin  
    sygnalizuj (S1);  
    czekaj (S2);  
end  
sygnalizuj (S1);
```

sygnalizuj (S) :

```
czekaj (S1);  
C:=C+1;  
if C≤0  
    then sygnalizuj (S2)  
    sygnalizuj (S1);
```