Laboratorium II – Teoria współbieżności

Weronika Szybińska, 24.10.2022

Treść zadania:

1. Zaimplementować semafor binarny za pomocą metod wait i notify, użyć go do synchronizacji programu Wyścig.

2. Pokazać, ze 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: czekająca na wejście do monitora obiektu oraz kolejkę związaną z instrukcja wait , rozważyć kto kiedy jest budzony i kiedy następuje wyścig).

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

Opis rozwiązania oraz wnioski:

1. Do wykonania pierwszego zadania wykorzystany został szkielet klasy dostarczony wraz z treścią. Został on zmodyfikowany w następujący sposób:

```
17  class Semafor {
18      private boolean _stan = true;
19      //private int _czeka = 0;
20
21      public Semafor() {
22      }
23
24      public synchronized void P() throws InterruptedException {
25          while(!_stan){
26              wait();
27          }
28          _stan = false;
29      }
30
31      public synchronized void V() throws InterruptedException {
32          _stan = true;
33          notify();
34      }
35  }
```

```java
class IThread extends Thread {
    private Counter _cnt;
    public Semafor semafor;
    public IThread(Counter c, Semafor semafor) {
        _cnt = c;
        this.semafor = semafor;
    }
    public void run() {
        for (int i = 0; i < 100; ++i) {
            try {
                semafor.P();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            _cnt.inc();
            try {
                semafor.V();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```java
class DThread extends Thread {
    private Counter _cnt;
    public Semafor semafor;
    public DThread(Counter c, Semafor semafor) {
        _cnt = c;
        this.semafor = semafor;
    }

    public void run() {
        for (int i = 0; i < 100; ++i) {
            try {
                semafor.P();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            _cnt.dec();
            try {
                semafor.V();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Semafor w pierwszym zadaniu jest semaforem binarnym, dlatego do blokowania i odblokowywania segmentu pamięci używana jest zwykła flaga true/false. Operacja P blokuję zasób pamięci, dzięki czemu wątek może go swobodnie modyfikować. Następnie wywoływana jest metoda V, która zmienia flagę na true i wybudza drugi wątek dając mu możliwość dostępu do zasobu.

2. Po zamianie funkcji while() w metodzie P() na funkcje if(), program zgodnie z oczekiwaniami zaczął zwracać błędne wyniki.

```
> Task :Race2.main()
value=14
```

```
> Task :Race2.main()
value=-12
```

```
> Task :Race2.main()
value=-17
```

Dzieje się tak, gdyż specyfikacja języka Java pozwala na fałszywe wybudzenia (ang. spurious wake-ups). Są to wybudzenia, które mogą wystąpić nawet gdy nie było odpowiadającego im powiadomienia – wywołania metody notify. W przypadku wystąpienia fałszywego wybudzenia w trakcie działa programu z użyciem funkcji if wątki zaczynają tak zwany wyścig o zasób. Powoduje to jak w przypadku programu z poprzednich laboratoriów niekontrolowane modyfikacje zmiennej i powstanie fałszywych wyników.

3. Po modyfikacji utworzonego wcześniej semafora binarnego, zaimplementowany został semafor licznikowy.

```
17  class Semafor {
18      //private boolean _stan = true;
19      private int _czeka;
20
21      public Semafor(int availableResources) {
22          _czeka = availableResources;
23      }
24
25
26      public synchronized void P() throws InterruptedException {
27          while(_czeka <= 0){
28              wait();
29          }
30          _czeka -= 1;
31      }
32
33      public synchronized void V() throws InterruptedException {
34          _czeka += 1;
35          notify();
36      }
37  }
```

Semafor binarny jest szczególnym przypadkiem semafora licznikowego. Dowodem na to jest powyższy przykład. Semafor licznikowy dostaje na wstępie ilość dostępnych zasobów. Na podstawie tej ilości modyfikowanej na bieżąco, rozdziela je pomiędzy wątki. W naszym przypadku ilość tych zasobów jest równa 1 (modyfikujemy jedną zmienną), przez co zmienna _czeka (w której zapisane jest ile w danym momencie jest wolnych zasobów) może osiągać tylko dwie wartości: 0 oraz 1. Łatwo zauważyć że 0 oraz 1 to inny sposób przedstawienia zmiennych boolean: true/false.

Bibliografia:

https://home.agh.edu.pl/~funika/tw/lab2/

https://pl.wikipedia.org/wiki/Semafor_(informatyka)

https://www.samouczekprogramisty.pl/watki-w-jezyku-java/

http://smurf.mimuw.edu.pl/node/953