Teoria Współbieżności
Ćwiczenie 2

1 Cel ćwiczenia

Celem ćwiczenia jest zapoznanie studentów z mechanizmami synchronizacji w Java opartymi o oczekiwanie na warunek. Kolejnym celem jest rozpoznanie abstrakcyjnego problemu ograniczonego bufora przy producentach i konsumentach oraz jego symulacja przez implementację w Java. W szczególnoci studenci zostaną zapoznani z przetwarzaniem potokowym z buforem

2 Oczekiwanie na warunek (condition wait)

1. Oczekiwanie na spełnienie określonego warunku
2. Sposób na podział oczekujących zadań na grupy - zadania z każdej grupy czekają na spełnienie innego warunku
3. Zwykle realizowane przez zmienne warunkowe (condition variables)
   - *wait*(C) (w pseudokodzie „await")
   - *signal*(C)
4. Zmienne warunkowe skojarzone z monitorem to po prostu **nazwane kolejki** monitora.
5. Realizacja oczekiwania na warunek w Javie: **nie ma zmiennych warunkowych** - jest tylko jedna **anonimowa** kolejka *wait*
   - Oczekiwanie w pętli while (realizacja „await" w Javie)

```
while (! warunek) {
  wait(); // usypiamy sie i oddajemy procesor,
     liczymy ze ktos nas obudzi
}
```

- Inny proces zmienia wartość zmiennej *warunek* i wykonuje *notifyAll*()

3 Ćwiczenie implementacyjne

Problem ograniczonego bufora (producentów-konsumentów)

Dany jest bufor o rozmiarze m, do którego producent może wkładać dane, a konsument pobierać. Napisz program, który zorganizuje takie działanie producenta i konsumenta, w którym zapewniona będzie własność bezpieczeństwa i żywotności.

Do symulacji użyj metod metod *wait*() / *notify*(). Procesy działają z różnymi prędkościami, użyj np. *TimeUnit.SECONDS.sleep*(-) / *TimeUnit.MILLISECONDS.sleep*(-) .
Wykonaj pomiar, obserwując zachowanie producentów/konsumentów np. *... Watek P3 produkuje pizze numer 107... Watek K2 zjada pizze numer 104...*

3.1 Wersja „klasyczna”

1. dla przypadku 1 producenta i 1 konsumenta.
2. dla przypadku n_1 producentów i n_2 konsumentów:
   (a) $n_1 > n_2$
   (b) $n_1 = n_2$
   (c) $n_1 < n_2$

Do realizacji bufora możesz użyć tablicy albo zwykłej kolejki. Na dole pdfa jest startowy kod. Aby można było pokazać problem wyścigu zadbaj aby jedna klasa nadawała numery produkcjom

```
class UnikalneNumery{
        private static int numerId = 0;
        synchronized public static int nowyID(){
                return numerId++;
        }
}
```

3.2 Przetwarzanie potokowe z buforem

1. Proces P_0 bedący (tylko) producentem.
2. Proces P_{100} bedący (tylko) konsumentem.
3. Procesy $P_1, P_2, \ldots, P_{99}$ bedące procesami przetwarzającymi. Każdy proces otrzymuje daną wejściową od procesu poprzedniego, jego wyjście zaś jest konsumowane przez proces następny.

Jeśli nie wiesz jak rozpocząć możesz rozwiązać najpierw zadanie bez buforu używając *PipedWriter* oraz *PipedReader* .

```
import java.util.concurrent.*;

// pisanie do kanalu
private PipedWriter out = new PipedWriter();
out.write(  jakisChar );

// czytanie z kanalu
public PipedReader in = new PipedReader( out );
char ch = (char)in.read(); // czeka az cos przeczyta
```

Dopiero za rozwiązanie z dowolnym buforem (wielkości m dla kadżego procesu) dostaniesz komplet punktów.

Starter

Listing 1: Można wykorzystać poniższy kod

```
class Producer extends Thread {
        private Buffer _buf;

        public void run() {
          for (int i = 0; i < 100; ++i) {
                _buf.put(i);
          }
        }
}

class Consumer extends Thread {
        private Buffer _buf;

        public void run() {
```

```
15            for (int i = 0; i < 100; ++i) {
16                  System.out.println(_buf.get());
17            }
18          }
19  }
20
21  class Buffer {
22    public void put(int i) {
23
24    }
25
26    public int get() {
27
28    }
29  }
30
31  public class SymulacjaPK {
32    public static void main(String[] args) {
33
34    }
35  }
```

Literatura

[1] Bruce Eckel, “Thinking in Java” - rozdział o wątkach

Uwagi: sluzalec@agh.edu.pl