

# Teoria Współbieżności

## Ćwiczenie 3

### 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 szczególności 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)*
  - *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

```
while (! warunek) {  
    wait();  
}
```

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

### 3 Problem ograniczonego bufora (producentów-konsumentów)

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

Zrealizować program - przy pomocy metod *wait()*/*notify()*. Patrz listing 1.

1. dla przypadku 1 producent/1 konsument
2. dla przypadku  $n_1$  producentów/ $n_2$  konsumentów ( $n_1 > n_2$ ;  $n_1 = n_2$ ;  $n_1 < n_2$ )
3. wprowadzić wywołanie metody *sleep()* i wykonać pomiary, obserwując zachowanie producentów/konsumentów

#### 3.1 Przetwarzanie potokowe z buforem

1. Bufor o rozmiarze  $N$
2. Proces  $A$  będący producentem.
3. Proces  $Z$  będący konsumentem.
4. Procesy  $B, C, \dots, Y$  będące procesami przetwarzającymi. Każdy proces otrzymuje daną wejściową od procesu poprzedniego, jego wyjście zaś jest konsumowane przez proces następny.
5. Procesy działają z różnymi prędkościami.

## Dodatki

Listing 1: Kod szkieletu

```
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() {
        for (int i = 0; i < 100; ++i) {
            System.out.println(_buf.get());
        }
    }

class Buffer {
    public void put(int i) {

    }

    public int get() {

    }
}

public class PKmon {
    public static void main(String[] args) {

    }
}

```