Społeczna Akademia Nauk w Łodzi

Programowanie współbieżne

Laboratorium 1

Prowadzący
mgr inż. Krystian Gumiński

2025

Zasady zaliczenia

Laboratorium ma na celu zapoznanie Państwa z zagadnieniami programowania wielowątkowego. Podczas zajęć omówione zostaną kwestie współbieżności oraz równoległości wykonywanych zadań. Zajęcia prowadzone są w języku Java, dopuszczam jednak realizację zadań w innych językach (chyba, że w zadaniu wskazano inaczej) wspierających paradygmat wielowątkowości (proszę sprawdzić czy język, którego Państwo pragną użyć wspiera ten paradygmat).

Zadania zostaną opublikowane na platformie Teams. Termin oddania zadania jest podany przy treści zadania; zegar Teams służy jako oficjalny znacznik czasu. Prace przesłane po terminie będą obniżone o 1 punkt za każdy dzień opóźnienia. Zasada jest stosowana maksymalnie do 3 dni opóźnienia (maks. -3 punkty). Po upływie 3 dni zadanie jest traktowane jako nieoddane i oceniane na 0. Brak przesłanej pracy = ocena 0. Ocena końcowa jest średnią ocen z przesłanych zadań z uwzględnieniem obecności; wynik wpisywany jest na ostatnich zajęciach.

Thread kontra Runnable

Listing 1: Thread i Runnable

```
1  class MyOwnTask extends Thread {
2      int counter = 0;
3
4      @Override
5      public void run() {
6          counter++;
7          System.out.println(Thread.currentThread().getName() +
               " is running. Counter: " + counter);
8      }
9  }
10
11 class MyOwnTaskBetter implements Runnable {
12     int counter = 0;
13
14     @Override
15     public void run() {
16         counter++;
17         System.out.println(Thread.currentThread().getName() +
               " is running. Counter: " + counter);
18     }
19 }
```

Rodzaje wątków i priorytety

Listing 2: Thread i Runnable

```
1     Thread t3 = new Thread(writer, "TELE2-Thread");
2     Thread t4 = new Thread(new StatsTask(), "Stats");
3     t4.setDaemon(true);
4     t4.setPriority(Thread.MIN_PRIORITY);
```

Metody na wątkach

Listing 3: Thread i Runnable

```
1     t3.start();
2     t4.start();
3
4     t0.join();
5     Thread.sleep(15000);
6
7     t1.interrupt();
8     t2.interrupt();
```

wait() i notify()

Listing 4: Klasa Data z wait i notify

```
1     static class Data {
2
3     private String data = null;
4
5     public synchronized void writeData(String data) throws
          InterruptedException {
6          while (this.data != null) {
7               wait();
8          }
9          this.data = data;
10         notify();
11    }
12
13    public synchronized String readData() throws
          InterruptedException {
14         while (data == null) {
15              wait();
```

```
16          }
17          String tmp = data;
18          data = null;
19          notify();
20          return tmp;
21      }
22  }
```

Diagnostyczne dane

Listing 5: Klasa Data z wait i notify

```
1  static class StatsTask implements Runnable {
2
3      @Override
4      public void run() {
5          while (!Thread.currentThread().isInterrupted()) {
6              try {
7  MemoryMXBean memoryMXBean = ManagementFactory.getMemoryMXBean();
8  MemoryUsage heapMemoryUsage = memoryMXBean.getHeapMemoryUsage();
9
10 System.out.println("Heap Memory Usage:");
11 System.out.println("Init: " + heapMemoryUsage.getInit());
12 System.out.println("Used: " + heapMemoryUsage.getUsed());
13 System.out.println("Committed: " +
14  heapMemoryUsage.getCommitted());
15     System.out.println("Max: " + heapMemoryUsage.getMax());
16     Thread.sleep(1_000);
17                 } catch (InterruptedException e) {
18                 Thread.currentThread().interrupt();
19             }
20         }
21     }
22 }
```

Współdzielone zasoby

Listing 6: Klasa Data z wait i notify

```
1      static class MyTaskWriter implements Runnable {
2
3      Data data;
4
```

```
5       public MyTaskWriter(Data data) {
6           this.data = data;
7       }
8
9       @Override
10      public void run() {
11          while (!Thread.currentThread().isInterrupted()) {
12              System.out.println("=== " + Thread.currentThread().
                    getName() + "  Working...");
13              try {
14                  data.writeData(Thread.currentThread().getName());
15                  Thread.sleep(ThreadLocalRandom.current().nextInt
                        (500, 1000));
16              } catch (InterruptedException e) {
17                  System.out.println("ARIWEDERCZI ROMA");
18                  Thread.currentThread().interrupt();
19              }
20          }
21      }
22  }
23
24  static class MyTaskReader implements Runnable {
25
26          Data data;
27
28          public MyTaskReader(Data data) {
29              this.data = data;
30          }
31
32          @Override
33          public void run() {
34              while (!Thread.currentThread().isInterrupted()) {
35                  System.out.println("=== " + Thread.currentThread()
                        .getName() + "  Working...");
36                  try {
37                      System.out.println("READING " + data.readData
                            ());
38                      Thread.sleep(ThreadLocalRandom.current().
                            nextInt(500, 1000));
39                  } catch (InterruptedException e) {
40                      System.out.println("ARIWEDERCZI ROMA");
41                      Thread.currentThread().interrupt();
42                  }
```

```
43                      }
44                  }
45  }
```

1 Zadanie

Napisz program wielowątkowy, który spełnia poniższe wymagania:

1. Zasób współdzielony: Utwórz jedną współdzieloną zmienną typu liczbowego. Zmienna może przyjmować wartości liczbowe lub być ustawiona na null.
2. Wątki producentów Wątek T1: losuje liczbę z przedziału [21–37] i zapisuje ją do zmiennej współdzielonej. Wątek T2: losuje liczbę z przedziału [1337–4200] i zapisuje ją do zmiennej współdzielonej.
3. Wątek konsumenta: Wątek T3: odczytuje bieżącą wartość zmiennej współdzielonej. Po odczycie dodaje tę wartość do swojej sumy (np. suma += odczytana_wartosc). Następnie ustawia zmienną współdzieloną na null.
4. Wątek monitorujący: Wątek T4: cyklicznie (np. co 1 sekundę) wypisuje: aktualny stan zmiennej współdzielonej, stan wszystkich wątków (T1–T3).
5. Czas działania programu: Cały program powinien działać 30 sekund, po czym wszystkie wątki mają zostać zakończone.

Zastosuj poznane metody i funkcję na zajęciach. Pokaż działanie programu na screenach pokazujące działanie programu w sposób jasny, że zastosowane mechanizmy działają. Zwróć zadanie w postaci sprawozdanie + kod.

Wymóg 1.1. Nazwij wątki numerem swojego indeksu na przykład:
"99467#Writer#1",
"[numer_albumu]#[Writer/Reader]#numer_wątku"

Dodatkowo napisz krótko:

Różnica między PID a TID,

Czym różni się wątek użytkownika od demonicznego?

Czy dobrą praktyką jest nienazywanie wątków?

Jak ważne jest ustawienie priorytetu wątku?

Czym jest monitor w ujęciu programowania wielowątkowego?