WĄTKI

Jakub Jarząbek, Jakub Janczak

Co to są wątki?

Wątki – podprogram (zbiór wykonywanych instrukcji) umożliwiającym jego realizację w oderwaniu od innych niezależnych podprogramów

- Wszystkie wątki korzystają z tej samej przestrzeni adresowej
- Każdy wątek działa tak, jakby nie było żadnych innych wątków, posiadając procesor “tylko dla siebie”

Zalety

- Tworzenie wrażliwego interfejsu użytkownika
- Optymalizacja przepustowości
- Zwiększenie przepustowości
- Wykonywanie kilku zadań równocześnie
- Zwiększenie szybkości dla komputerów wieloprocesorowych

Wady

- Dodatkowe wykorzystanie procesora na obsługę wątków
- Spowolnienie podczas oczekiwania na zasoby współdzielone
- Wzrost złożoności wynikający z błędnych decyzji projektowych
- Możliwości wystąpienia nowych błędów (patologi): zakleszczenie (problem filozofów), nieprawidłowy dostęp do zasobów, zagłodzenie, wyścig

Tworzenie wątków

Sposób pierwszy (klasa Thread):

- Rozszerzamy klasę java.lang.Thread
- Przeciążamy metodę public void run()
- W konstruktorze wywołujemy metodę Thread.start() (konfiguruje wątek, wywołuje metodę run(), jeśli jej nie wywołamy wątek nie zostanie uruchomiony!
- Każdy obiekt nowo utworzonej klasy to osobno działający wątek

Tworzenie wątków cd.

Sposób drugi (interfejs Runnable):

- Tworzymy klasę i implementujemy interfejs java.lang.Runnable (należy przeciążyć metodę public void run())
- Tworzymy nowy wątek jako parametr podając klasę implementującą interfejs Runnable:

```
class X implements Runnable {
  public void run() {/*kod w tku*/}
}
new Thread( new X() );
```

Przykład

```
class SimpleThread extends Thread {
   public SimpleThread() {
      super();
      start();
   }
   public void run() {
        int d = 0;
        while( d < 100000 ) {
           d++; /* inne operacje... */
        }
   }
   public static void main( String[] args ) {
        for( int i=0; i<10; i++ )
             new SimpleThread();
   }
}
class MyThread implements Runnable {
   public void run() {
        int d = 0;
        while( d < 100000 ) {
           d++; /* inne operacje... */
        }
   }
   public static void main( String[] args ) {
        for( int i=0; i<10; i++ )
             new Thread(new MyThread());
   }
}
```

Sterowanie wątkami

Podpowiedzi dla zarządcy wątków:

- Metoda Thread.yield():

  przesłanie mechanizmowi zarządzającemu informacji, że wątek zrobił co do niego należało i można przekazać sterownie do innego wątku

- Metoda Thread.sleep():

  przerywa działanie wątku na określoną liczbę milisekund. Wywołanie sleep() umieszczamy w bloku try-catch, gdyż mając referencję do wątku można wywołać metodę Thread.interrupt(). Wyjątkiem, który wtedy otrzymujemy jest InterruptedException

Sterowanie wątkami cd

- Metoda Thread.setPriority():

  zmienia priorytet danego wątku. Istnieje 10 priorytetów.

  Predefiniowane priorytety:

  - Thread.MAX_PRIORITY
  - Thread.MIN_PRIORITY
  - Thread.NORM_PRIORITY

  Priorytet odczytujemy metodą getPriority()

- Metoda Thread.join()

  jeśli jakiś wątek wywoła metodę join() innego wątku, to realizacja wątku wywołującego metodę zostanie odłożona do czasu zakończenia wątku dla którego metoda została wywołana

Wątki - demony

Demon – wątek, który działa w tle programu, ale nie jest bezpośrednio związany z główną częścią programu. Jeśli wszystkie wątki nie będące demonami zakończą działanie , to program też.

Tworzenie:

wywołanie metody Thread.setDaemon(true) przed wywołaniem metody Thread.start().

Analogiczna metoda do sprawdzania czy wątek jest demonem – isDaemon()

UWAGA: klasy dziedziczące po klasie wątku demona są demonami

Współdzielenie zasobów

Nieprawidłowy dostęp do zasobów:

student.uci.agh.edu.pl/~jjarzabe/JTP/referat/examples

```
class Computer {
   private User loggedUser;
   private HashMap data = new HashMap();
   public void login(User u) {
     while (loggedUser != null)
         ;
         Thread.yield();
          loggedUser = u;
   }
   public void logout() {
      loggedUser = null;
   }
   public void writeData(String d) {
      data.put(loggedUser, d);
   }
   public String readData() {
      return(String)data.get(loggedUser);
   }
}
```

Współdziałanie wątków cd.

- Problem – niewłaściwy dostęp do zasobów
- Rozwiązanie – zastosowanie słowa kluczowego **`synchronized`** w stosunku do metod:

```
synchronized void f(){}
```

`synchronized` powoduje, że wywołując synchronizowaną metodę obiekt jest blokowany i żadna inna jego synchronizowana metoda nie może być wywołana, dopóki powyższa metoda się nie zakończy i nie zwolni blokady obiektu

Współdzielenie zasobów – sekcje krytyczne

- Problem: uniemożliwienie wątkom dostępu tylko do fragmentu kodu, a nie do całej metody
- Rozwiązanie: zastosowanie `synchronized` tylko w stosunku do bloku kodu:

```
synchronized( obiektSynronizowany ) {
  /* blok kodu */
} /* zwolnienie blokady */
```

Jeśli jakiś wątek posiada blokadę obiektu, wejście innego wątku do sekcji krytycznej jest niemożliwe.

UWAGA: synchronizowanie bloku kodu jest szybsze!

Współdziałanie wątków

- Metoda Object.wait():

  stosujemy, gdy dojdziemy do miejsca, w którym wątek musi czekać na zmianę jakiś warunków zewnętrznych. Aby nie marnować czasu procesora i bezczynnie czekać, sprawdzając warunki, wywołujemy metodę wait (). Metoda usypia wątek i oczekuje na „zmianę świata zewnętrznego”. Możliwe wywołania metody:

  - wait() - czeka dopóki nie zostanie obudzony
  - wait(timeout) - czeka maksimum timeout milisekund na obudzenie
  - wait(timeout, nanos) - czeka timeout milisekund i nanos nanosekund

  UWAGA: metoda wait() zwalnia blokadę obiektu

Współdziałanie wątków cd.

- Metoda Object.notify():

  budzi wątek uśpiony metodą wait(). Aby wywołać metodę notify() musimy przechwycić blokadę obiektu, który chcemy obudzić

- Metoda Object.notifyAll()

  budzi wszystkie wątki

UWAGA: wszystkie powyższe metody muszą być wywoływane wyłącznie w synchronizowanym bloku kodu! W przeciwnym wypadku zostanie wyrzucony wyjątek:

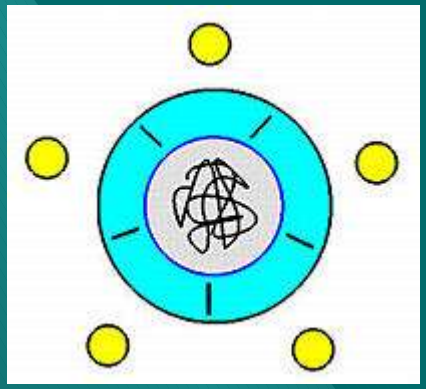
```
IllegalMonitorStateException
```

Przykład:
student.uci.agh.edu.pl/~jjarzabe/JTP/referat/examples

Zakleszczenie (ang. deadlock)

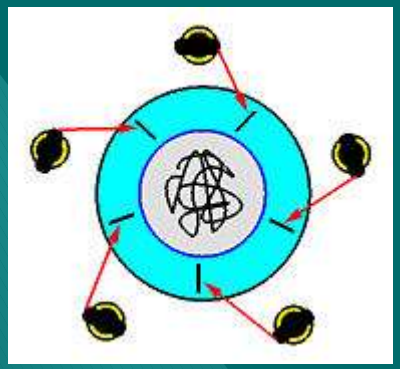
Problem ucztujących filozofów (twórca: Dijkstra)

Przy stole siedzi 5 filozofów. Każdy z nich albo je posiłek, albo medytuje. Do zjedzenia posiłku potrzebne są 2 sztućce (np. widelce). Niestety filozofowie są biedni i stać ich tylko na zakup 5 widelców. Zakładamy, że filozofowie są “kulturalni” i nie wydzierają sobie z rąk raz zabranych przez kogoś sztućców (np. widelców).



Zakleszczenie cd.

Zakleszczenie w “problemie filozofów”: pierwszy bierze najpierw lewy widelec, drugi w tym samym czasie także bierze swój lewy widelec, itd. Na końcu każdy z filozofów ma w widelec tylko w lewej ręce i czeka na zwolnienie drugiego widelca przez innego filozofa.



Zakleszczenie cd.

Problem filozofów

rozwiązanie: jeden z filozofów musi pobrać pałeczki w odwrotnej kolejności, niż pozostali

Niebezpieczne metody

Thread.stop(), Thread.suspend(), Thread.resume()

- Używanie metody stop() nie zwalnia blokad, które posiada wątek, w tym czasie obiekt może być w niestabilnym stanie, o czym inne wątki nie będą wiedziały
- Metody suspend() i resume() mogą łatwo spowodować zakleszczenie. Jeśli wywołamy metodę suspend() blokada obiektu nie są zdejmowana, więc jeśli inny wątek chce uzyskać blokadę, aby wywołać metodę resume() dla tego wątku powoduje to zakleszczenie.
- Rozwiązanie – użycie flagi