Programowanie wielowątkowe: podstawowe koncepcje, narzędzia w Javie

J. Starzyński, JiMP2

Tematyka

- Wprowadzenie
- Podstawowe pojęcia
- Tworzenie i uruchamianie wątków
- Zatrzymywanie wątków
- Współdzielenie pamięci – synchronizacja
- Komunikacja pomiędzy wątkami
- Hierarchie i priorytety wątków

Wprowadzenie

- Większość programów działa w środowiskach sekwencyjnych, gdzie procesor wykonuje ciąg instrukcji jedna po drugiej.
- Wiele programów wymaga jednak “równoległego” wykonywania więcej niż jednego ciągu operacji.
- Taką “równoległość” symuluje się przez cykliczne przełączanie procesora.

Wprowadzenie, c.d.

- *Proces* to samodzielnie wykonujący się program z *własną przestrzenią adresową*.
- Wielozadaniowy system operacyjny jest w stanie uruchomić jednocześnie wiele procesów. Każdy z nich może działać niezależnie od innych, a system przydziela każdemu pewien przedział czasu procesora.
- *Wątek* to niezależne podzadanie w ramach procesu. Może współdzielić pamięć z innymi wątkami.

Wprowadzenie, c.d.

```
...
while( 1 ) {
   t1=
    getTemperature( sensor1 );
   t2=
    getTemperature( sensor2 );
   t3=
    getTemperature( sensor3 );
   p1= getPressure( sensor5 );
   p2= getPressure( sensor6 );
   if( t1 > TMAX || t2 > TMAX )
      stopHeater( h1 );
   if( t1 < t2 )
      switchHeater( h2 );
   if( t3 > TMAX && p1 > PMAX )
      openValve( );
 ...
```

- W Javie można tworzyć wątki jako rozdzielne sekwencje sterowania.
- Do czego się to przydaje?
  - sterowanie
  - obsługa urządzeń
  - programowanie grafiki interakcyjnej
  - programowanie równoległe i rozproszone

Wprowadzenie, c.d.

Model odsłuchowy

```
...
while( n_nodes > 4 ) {
   check_sharp_edges( &n_nodes,  nodes, x, y, np, nop, ne );
   if( n_nodes < 5 )
      break;
   if( userInput() )  /* sprawdzamy, czy uzytkownik sie nie znudzil */
      break;
   find_edge_seq( &n_nodes, nodes, x, y, np, nop, ne, &l_seq, &ptr );
   if( userInput() )  /* sprawdzamy, czy uzytkownik sie nie znudzil */
      break;
   if( l_seq > 1 )
      close_edge_seq( &n_nodes, nodes, x, y, np, nop, ne, &l_seq, &ptr );
   else
      catalizer( &n_nodes, nodes, x, y, np, nop, ne );
   if( n_nodes < 5 )
      break;
   if( userInput() )  /* sprawdzamy, czy uzytkownik sie nie znudzil */
      break;
}
if( userInput() ) /*  uzytkownik sie znudzil */
   processUserInput();
...
```

Wprowadzenie, c.d.

Model zdarzeniowy

```
...
int kbd_listener( int key_pressed ) {
  if( key_pressed == CTRL_C )
    ask_if_really_stop(....);
  ...
}
...
  register_listener( kbd_listener );
  while( n_nodes > 4 ) {
     check_sharp_edges( &n_nodes,  nodes, x, y, np, nop, ne );
     if( n_nodes < 5 )
        break;
     find_edge_seq( &n_nodes, nodes, x, y, np, nop, ne, &l_seq, &ptr );
     if( l_seq > 1 )
        close_edge_seq( &n_nodes, nodes, x, y, np, nop, ne, &l_seq, &ptr );
     else
        catalizer( &n_nodes, nodes, x, y, np, nop, ne );
  }
...
```

Tworzenie wątków – metoda A

- Klasę-wątek możemy utworzyć rozszerzając szkieletową klasę Thread
- Jeżeli wątek ma coś robić, to musimy w naszej klasie utworzyć metodę
  public void run()
- Ta metoda jest uruchamiana przez wywołanie metody start.
- Wątek ,,żyje” tak długo, jak długo działa jego metoda run

Tworzenie wątków – przykład A

```
class W1 extends Thread {
   String name;
   long delay;
   long nRepet;

   W1( String w, long dt, long n ) {
      name= w;
      delay= dt;
      nRepet= n;
   }
...
```

```
...
  public void run() {
     try {
        for( int i= 0; i < nRepet; i++) {
           System.out.print( name + " " );
           sleep( delay );
        }
     } catch( InterruptedException e ) {
        System.out.println( "Exiting " + name + "\n" );
        return;
     }
  }

}
```

Tworzenie wątków – przykład A,

```
class W1Test {

   public static void main( String args[] )
      throws InterruptedException
   {
     W1 ping= new W1( “Ping”, 500, 10 );
     W1 pong= new W2( “Pong”,500, 20);
     ping.start();
     pong.start();
   }
}
```

Tworzenie wątków – przykład A,

```
class W1Test {

   public static void main( String args[] )
      throws InterruptedException
   {
     for( int i= 0; i < args.length; i += 3 )
            new W1( args[i],
                  Long.valueOf( args[i+1] ).longValue(),
                  Long.valueOf( args[i+2] ).longValue()
                ).start();
      }
   }
}
```

Tworzenie wątków – metoda B

- Klasę-wątek możemy utworzyć implementując interfejs Runnable
- Pozwala to utworzyć klasę-wątek, która rozszerza jakąś klasę, która nie jest wątkiem
- Podobnie, jak w metodzie A musimy w naszej klasie utworzyć metodę
  public void run()
- Klasę implementującą Runnable uruchamiamy w nieco bardziej skomplikowany sposób

Tworzenie wątków – przykład B

```
class W2 implements Runnable {
   String name;
   long delay;
   long nRepet;

   W2( String w, long dt, long n ) {
      name= w;
      delay= dt;
      nRepet= n;
   }
...
```

```
...
   public void run() {
      try {
         for( int i= 0; i < nRepet; i++) {
            System.out.print( name + " " );
            Thread.sleep( delay );
         }
      } catch( InterruptedException e ) {
         System.out.println( "Exiting " + name + "\n" );

}
```

Tworzenie wątków – przykład B,

```
class W2Test {
  public static void main( String args[] )
      throws InterruptedException
  {
        for( int i= 0; i < args.length; i += 3 ) {
          Runnable r= new W2( args[i],
                Long.valueOf( args[i+1] ).longValue(),
                Long.valueOf( args[i+2] ).longValue()
              );
          new Thread( r ).start();
        }
    }
  }
```

Zatrzymywanie wątków

- Program działa, dopóki działa choć jeden z jego wątków (z wyj. wątków-demonów).
- Każdy program ma przynajmniej jeden wątek – jest nim metoda main uruchamianej klasy
- Wątek można zatrzymać komunikując się z jego metodą run – tak jest najlepiej
- Można też zatrzymać wątek wysyłając mu przerwanie (wyjątek InterruptException)
- W starej Javie można też było wywołać metodę stop (zgłasza wyjątek ThreadDeath), ale jest to sposób przestarzały (niebezpieczny)

Zatrzymywanie wątków

```
class Wx extends Thread {

   public boolean canRun;

   public void start() {
      canRun= true;
      super.start();
   }

   public void run() {
        ...
        while( canRun ) {
          ...
        }
        ... // sprzątanie
   }
```

```
class WxUser {
   ...
   Wx t= new Wx();
   ...
   t.start();
   ...
   if( t_should_stop )
     t.canRun = false;
   ...
}
```

Możliwe stany wątku

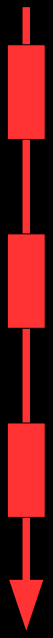
- Utworzony, ale nie uruchomiony
- Uruchomiony
- Zakończony (koniec run(), stop(), destroy())
- Zablokowany
  - sleep()
  - suspend() .... resume()
  - yield()
  - wait()
  - we/wy
  - czeka na zwolnienie monitora

Problem: dzielony dostęp do danych

Stan konta nie uwzględnia wpłaty!

Wątek Kasa:

```
stanKonta = stanKonta + wpłata;
```



stanKonta → rejestr

rejestr += wpłata

rejestr → stanKonta

Wątek Bankomat:

```
stanKonta = stanKonta - wypłata;
```

stanKonta → rejestr

rejestr -= wpłata

rejestr → stanKonta

Synchronizacja

- Każdy obiekt zawiera *monitor* – blokadę, która jest automatycznie jego częścią.
- Można wykorzystać ją (po to jest) do blokowania obiektu.
- Pierwszym sposobem jest deklarowanie metod, które powinny blokować obiekt jako

  synchronized
- Drugim sposobem jest użycie

  bloków synchronizowanych