Beata Jankowska
Instytut Automatyki, Robotyki i Inżynierii Informatycznej

VIII. Wielowątkowość. Modelowanie obiektowe przy użyciu języka UML

1. Wielowątkowość

1.1. Realizacja programu mającego swój kod i swoje dane, zajmującego własną przestrzeń adresową i korzystającego z zasobów systemowych nazywa się **procesem**. System operacyjny przypisuje procesowi pewien określony kwant czasu. Regulamin przypisywania tych kwantów poszczególnym procesom może być różny dla różnych systemów operacyjnych.

1.2. Na proces może się składać kilka **wątków**. Wątek to najmniejsza jednostka programu, której system operacyjny może przydzielić czas procesora. Wątek może wykonywać dowolną część kodu programu. Ten sam fragment kodu może być wykonywany przez kilka współistniejących wątków. Wątki współdzielą tę samą wirtualną przestrzeń adresową co procesy, od których "pochodzą". Każdy wątek posiada natomiast odrębne: rejestry, stos, mechanizm wejścia oraz kolejkę wiadomości.

1.3. Korzystanie z wątków wiąże się więc z używaniem wspólnych obszarów danych przez współistniejące "mini-procesy". W celu zagwarantowania bezpiecznego dostępu do tych danych, trzeba używać odpowiednich mechanizmów synchronizacji.

1.4. Do komunikacji pomiędzy wątkami służą, między innymi, zmienne zwane **semaforami**. Specjalne **semafory wzajemnego wykluczania** mają na celu zagwarantowanie, że we wskazanym obszarze danych będzie się wykonywał równolegle co najwyżej jeden wątek. Przy wejściu do tego obszaru wątek blokuje (opuszcza) semafor, zakazując wejścia tam – na czas swego działania – innym wątkom. Odblokowanie (podniesienie) semafora przy opuszczaniu przez wątek chronionego

obszaru danych stwarza możliwość wejścia do niego kolejnego wątku oczekującego pod semaforem.

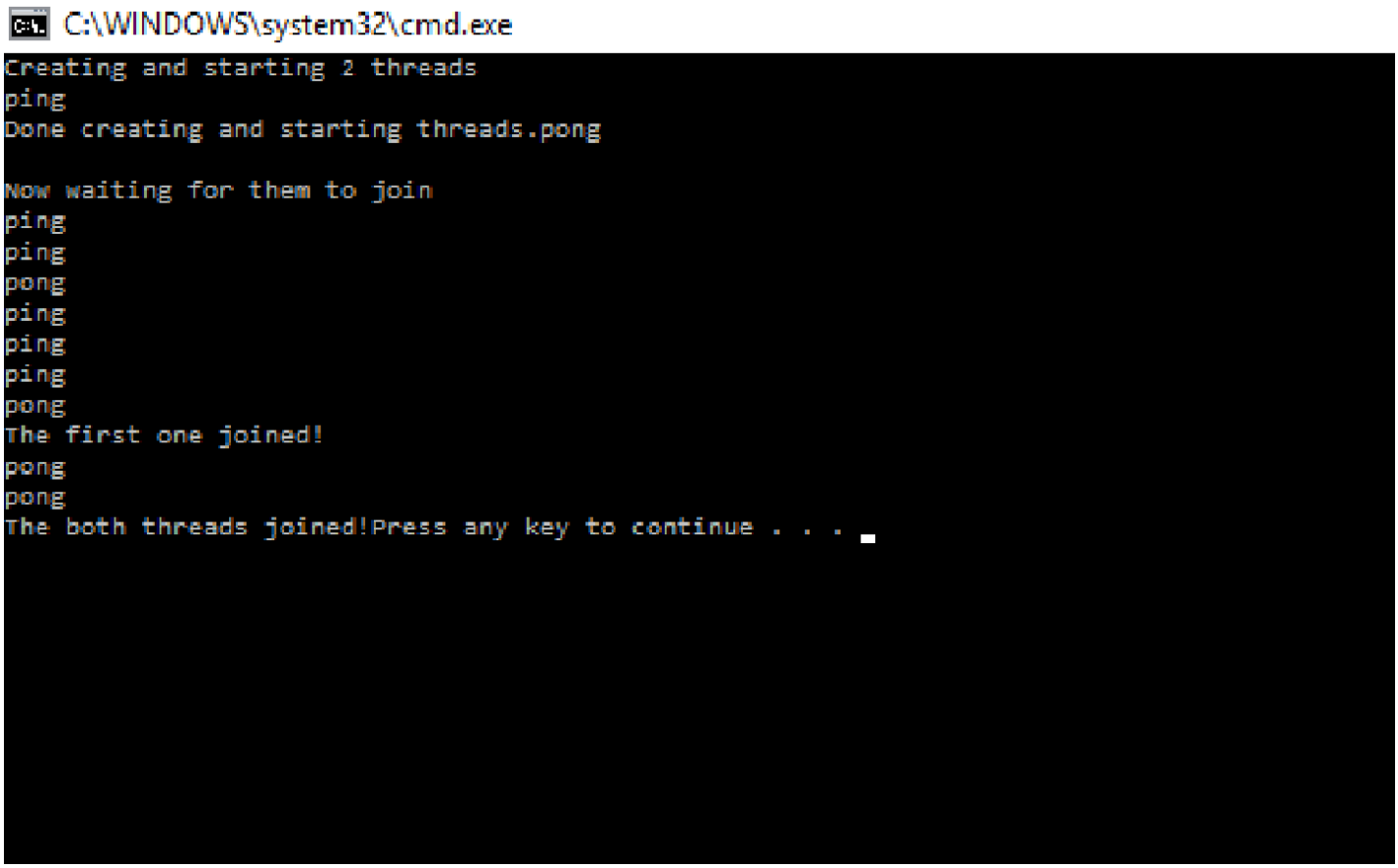
1.5. Pierwszym standardem języka C++ w którym przewidziano mechanizmy do definiowania i obsługi wątków jest standard C++11, zatwierdzony przez ISO w sierpniu 2011. We wcześniejszych implementacjach języka konstruowano specjalne biblioteki z kompletnym zestawem funkcji do tworzenia wątków i ich synchronizacji.
Oto wielowątkowa wersja pewnego programu (C++11), korzystająca z klasy `thread` (konstruktor, metoda `join`) oraz z metod zgrupowanych w przestrzeni nazw `this_thread`, zawierającej metody odnoszące się do bieżącego wątku (np. metoda `sleep_for`).

```
#include "stdafx.h"
#include <string>
#include <iostream>
#include <thread>
#include <chrono>
using namespace std;

void write_and_pause_thread(string s, int howmany,
                            int sleep)
{
    for (int i = 1; i <= howmany; i++) {
        cout << s << endl;
        std::this_thread::sleep_for
                        (std::chrono::seconds(sleep));
    }
}

int main()
{
    std::cout << "Creating and starting 2 threads"
              << endl;
    std::thread t1(write_and_pause_thread,
                   "ping", 6, 3);
    std::thread t2(write_and_pause_thread,
                   "pong", 5, 8);
    std::cout << "Done creating and starting threads."
              << endl << "Now waiting for them to join"
              << endl;
    t1.join();
    std::cout << "The first one joined!" << endl;
    t2.join();
    std::cout << "The both threads joined!";
    return 0;
}
```

Beata Jankowska
Instytut Automatyki, Robotyki i Inżynierii Informatycznej



C:\WINDOWS\system32\cmd.exe

```
Creating and starting 2 threads
ping
Done creating and starting threads.
pong
Now waiting for them to join
ping
ping
pong
ping
ping
ping
pong
The first one joined!
pong
pong
The both threads joined!Press any key to continue . . .
```

C:\WINDOWS\system32\cmd.exe

```
Creating and starting 2 threads
ping
pong
Done creating and starting threads.
Now waiting for them to join
ping
ping
pong
ping
ping
ping
pong
The first one joined!
pong
pong
The both threads joined!Press any key to continue . . .
```

```
#include "stdafx.h"
#include <string>
#include <iostream>
#include <thread>
#include <chrono>
#include <mutex>
using namespace std;
mutex mtx;
void write_and_pause_thread(string s, int howmany,
                            int sleep)
{
     for (int i = 1; i <= howmany; i++) {
          mtx.lock();
          cout << s << endl;
          mtx.unlock();
          std::this_thread::sleep_for
                     (std::chrono::seconds(sleep));
     }
}

int main()
{
     mtx.lock();
     std::cout << "Creating and starting 2 threads"
               << endl;
     mtx.unlock();
     std::thread t1(write_and_pause_thread,
                    "ping", 6, 3);
     std::thread t2(write_and_pause_thread,
                    "pong", 5, 8);
     mtx.lock();
     std::cout << "Done creating and starting threads."
               << endl <<"Now waiting for them to join"
               << endl;
     mtx.unlock();
```

```
    t1.join();
    mtx.lock();
    std::cout << "The first one joined!" << endl;
    mtx.unlock();
    t2.join();
    mtx.lock();
    std::cout << "The both threads joined!";
    mtx.unlock();
    return 0;    }
```

Jaka będzie odpowiedź systemu? Co się zmieni po zastosowaniu semafora wykluczającego `mutex mtx` ?

1.6. Wielowątkowość jest immanentną cechą języka Java. Oto – podobny do poprzedniego – program wielowątkowy napisany w języku Java [A.van Hoof et al., Java]:

```
public class Printer implements Runnable
{
  Thread printerThread;
  String string;
  int count;
  int sleeptime;

  public Printer(String s, int howMany,
                 int sleep)
  { count = howmany;
    string = s;
    sleeptime = sleep;
    printerThread = new Thread(this);
    printerThread.start();
  }
```

```
  public void run()
  {  while (count-- > 0)
    {  System.out.println(string);
       try
       { Thread.sleep(sleeptime);}
       catch (Exception e)
       {return;}
    }
  }
  public static void main(String args[])
  { new Printer("Ping", 5, 300);
    new Printer("Pong", 5, 300);
  }
}
```

Beata Jankowska
Instytut Automatyki, Robotyki i Inżynierii Informatycznej

2. Modelowanie obiektowe przy użyciu języka UML

2.1. Przeznaczenie języka UML, historia powstania UML
2.2. Diagramy UML (kategoryzacja)
2.3. Diagram klas
2.4. Diagram komponentów
2.5. Diagram pakietów
2.6. Diagram przypadków użycia
2.7. Diagram stanów
2.8. Diagram czynności

Wykład na podstawie materiałów dostępnych na stronie:

http://wazniak.mimuw.edu.pl/index.php?title=In%C5%BCynieria_oprogramowania

- język UML, cz.1: http://wazniak.mimuw.edu.pl/index.php?title=Io-5-wyk-toc
- język UML, cz.2: http://wazniak.mimuw.edu.pl/index.php?title=Io-6-wyk-toc