Utilizarea firelor de executie în Java si C#

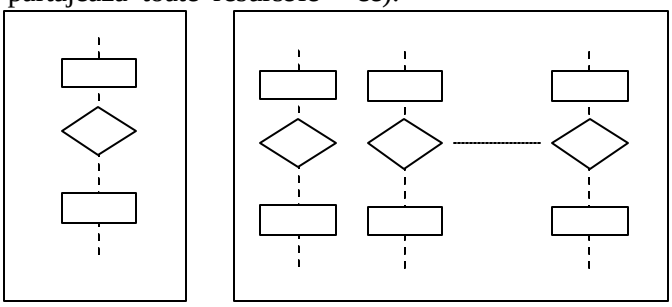
Lect.dr. Paul POCATILU
Catedra de Informatica Economica, A.S.E. Bucuresti

Using multithreading programming is very useful in many applications and it can solve different problems. In this paper are presented the main characteristics of the multithreading and how it can be used in two object oriented programming languages, Java and C#.
***Keywords**: process, thread, synchronization, multitasking.*

Introducere

Un proces consta dintr-unul sau mai multe fire de executie, date si alte resurse (figura 1). Fiecare proces are un fir principal de executie, asociat aplicatiei. Firele de executie sunt unitati independente de executie si reprezinta o cale într-un program. Un fir de executie are alocata o stiva, registrii microprocesorului si o intrare în lista de executie a componentei de alocare a executiilor.

Firele de executie partajeaza toate resursele procesului, cum ar fi fisierele deschise sau memoria alocata dinamic. Fiecare fir de executie lucreaza independent si nu intra în contact cu alte fire de executie, decât în cazul în care acest lucru este realizat în mod specific de catre programator. În acest caz, daca sunt partajate resurse comune, este necesara coordonarea executiei firelor prin intermediul semafoarelor sau a altor mecanisme de comunicare între procese (mutex, sectiuni critice).



a) b)

Fig. 1. Proces cu un singur fir de executie a) si cu mai multe fire de executie b)

Un fir se executa în momentul în care componenta de alocare a executiilor da controlul acestuia. Predarea controlului unui fir de executie se realizeaza dupa un anumit interval de timp, pentru o durata de timp limitata. De asemenea, fiecare fir de executie are asociata o prioritate de care se tine cont în momentul în care se da controlul executiei, firele de executie cu prioritate mare fiind executate înaintea celor cu prioritate mica.

Utilizarea firelor de executie este necesara în diverse aplicatii. Firele de executie sunt utilizate în aplicatii care necesita prelucrari de lunga durata, aplicatii care urmaresc anumite evenimente si aplicatii de tip GUI care ofera posibilitatea întreruperii unei actiuni [FERG02]. De exemplu încarcarea unui document este realizata de un fir de executie si nu firul principal, ceea ce permite utilizatorului interactiunea cu fereastra care a lansat firul de executie.

Limbajele Java si C# suporta firele multiple de executie. Pentru exemplificare este creata o aplicatie de tip consola în limbajele Java si C# în care se utilizeaza doua fire de executie. Un fir de executie afiseaza un mesaj la consola, care contine numele firului de executie. Oprirea executiei aplicatiei se realizeaza tastând Ctrl+C.

Implementarea firelor de executie în Java

Limbajul Java ofera doua clase (*Thread* si *ThreadGroup*) si o interfata (*Runnable*) pentru dezvoltarea de aplicatii cu fire multiple de

executie. Clasa Thread implementeaza la rândul ei interfata *Runnable*.

Utilizarea clasei *Thread* presupune derivarea unei clase din aceasta. În cadrul clasei derivate, se redefineste metoda *run* definita în clasa de baza, metoda care contine logica firului de executie. Principalele metode existente în clasa *Thread* sunt:

- *start* – lansarea în executie a firului;
- *run* – metoda care se va executa la pornirea firului;
- *yield* – cedeaza executia altor fire concurente;
- *setPriority* – stabilirea prioritatii firului de executie cu valori cuprinse între Thread.MIN_PRIORITY si Thread.MAX_PRIORITY;
- *destroy* – utilizata pentru terminarea executiei firului fara eliberarea resurselor ocupate de firul de executie;
- *sleep* – opreste executia firului curent pe o durata precizata ca parametru, masurata în milisecunde;
- *join* – asteapta ca firul curent sa se termine.

Metodele *stop*, *suspend* si *resume* sunt depreciate, având în vedere faptul ca prin utilizarea acestora pot sa apara blocari ale firelor de executie.

În listingul 1 este prezentat codul sursa Java al aplicatiei implementata folosind clasa FM derivata din clasa *Thread*.

Listing 1 – Java

```
class FM extends Thread
{      private String nume=null;
       FM(String n)
       {    super();
            nume=n;
       }
       public void run()
       {    while(true)
                {       System.out.println("Firul "+nume);
                        try
                        {       sleep(10);
                        }
                        catch(InterruptedException ie)
                        {       ie.printStackTrace();
                        }
                }
       }
}

public class TestFire
{
public static void main(String arg[])
{  FM fm1=new FM("Unu");
   FM fm2=new FM("Doi");
   fm1.start();
   fm2.start();
 }
}
```

A doua posibilitate presupune implementarea interfetei *Runnable* în cadrul clasei si definirea metodei *run*. Din listingul 2 se observa modalitatea de creare a firelor de executie prin implementarea interfetei *Runnable*.

Listing 2 – Java

```
class FM implements Runnable
{      private String nume=null;
       private Thread fm=null;
```

```
       FM(String n)
       {super();
        nume=n;
       }
public void start()
{     fm=new Thread(this,nume);
      fm.start();
}
       public void run()
       {        Thread th=Thread.currentThread();
                while(true)
                {       System.out.println("Firul "+nume);
                        try
                        {      th.sleep(10);
                        }
                        catch(InterruptedException ie)
                        {      ie.printStackTrace();
                        }
                }
       }
}

public class TestFire
{
public static void main(String arg[])
{  FM fm1=new FM("Unu");
   FM fm2=new FM("Doi");
   fm1.start();
   fm2.start();
 }
}
```

Alegerea unei metode sau a alteia se face în primul rând în functie de necesitatea mostenirii multiple. Astfel, daca o clasa utilizeaza fire de executie si este derivata dintr-o alta clasa, neexistând posibilitatea mostenirii multiple, se foloseste interfata *Runnable*. În figura 2 sunt prezentate cele mai importante stari în care se poate afla un fir de executie.

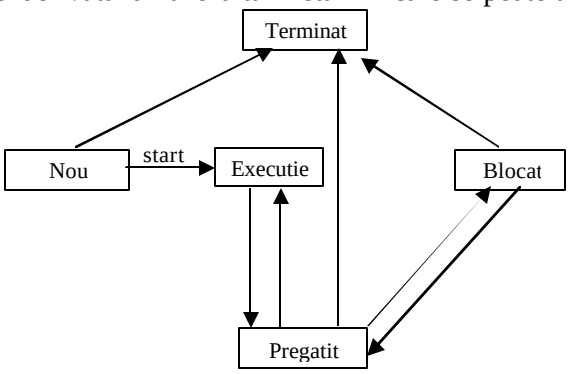


Fig. 2. Starile unui fir de executie

Prin intermediul clasei *ThreadGroup* este posibila utilizarea mai multor obiecte de tip *Thread* ca un tot unitar. Pentru sincronizarea firelor de executie se utilizeaza atributul *syncronized* în momentul în care se acceseaza o resursa comuna.

Fire de executie în C#

Spatiul de nume *System.Threading* contine clasele pentru implementarea de aplicatii .NET care sa utilizeze fire de executie multiple si astfel sa fie dezvoltate aplicatii concurente.

Biblioteca de clase Microsoft .NET furnizeaza clasa *Thread* pentru utilizarea firelor de executie. Clasa Thread are atributul *sealed*, ceea ce nu permite derivarea de noi clase din aceasta. Clasa Thread furnizeaza o serie de metode de operare asupra firelor de executie, dintre care cele mai importante sunt prezentate în continuare.

Constructorul clasei *Thread* este utilizat pentru a initializa firul de executie. Parametrul constructorului este un delegat de tipul *ThreadStart*, prin intermediul caruia se transmite adresa functiei corespunzatoare firului de executie. În tabelul 1 sunt prezentate principalele metode din clasa *Thread*.

Tabelul 1 – Principalele metode ale clasei Thread

Metoda	Descriere
Start	începe executie unui fir
Interrupt	întrerupe executia unui fir
Suspend	suspenda executia unui fir
Resume	continua executia unui fir suspendat
Join	asteapta un alt fir
Sleep	opreste executia firului pe o durata de timp determinata

Pentru a utiliza clasa *Thread* se creeaza o clasa prin care este implementat comportamentul firului de executie. O metoda a clasei este transmisa ca parametru constructorului firului de executie prin intermediul unui delegat de tip *ThreadStart*. În listingul 3 este prezentata implementarea în C# a exemplului prezentat la începutul articolului. Metoda Run din clasa FM executa un ciclu infinit în care este afisat numele firului de executie curent.

Listing 3 – C#

```
using System;
using System.Threading;

class FM
{     string nume;
      public FM(string n)
      {     nume=n;
      }
      public void Run()
      {     while(true)
            {     Console.WriteLine("Firul {0}",nume);
                  Thread.Sleep(10);
            }
      }
}

class TestFire
{     static void Main()
      {     FM ofm1=new FM("Unu");
            FM ofm2=new FM("Doi");
            Thread fm1=new Thread(new ThreadStart(ofm1.Run));
            Thread fm2=new Thread(new ThreadStart(ofm2.Run));
            fm1.Start();
            fm2.Start();
      }
}
```

Pentru sincronizarea firelor de executie sunt disponibile în spatiul de nume *System.Threading* o serie de clase (*Monitor*, *Mutex*).

Concluzii

Prin intermediul firelor de executie sunt executate simultan mai multe secvente de cod.

Aplicabilitate firelor de executie este vasta. Crearea si utilizarea firelor de executie multiple trebuie realizata cu atentie, având în vedere posibilitatile de aparitie a blocarilor în cazul partajarii resurselor. Tratarea aspectelor privind sincronizarea firelor de executie necesita o abordare separata.

Bibliografie

[ECKE00] Eckel, Bruce – *Thinking in Java – Second Edition*, Prentice Hall, New Jersey, 2000

[FERG02] Ferguson, Jeff, Patterson, Brian, Beres, Jason, Boutquin, Pierre, Gupta, Meeta – *C# Bible*, Wiley Publishing, Inc., Indianapolis, 2002

[LAFO99] Lafore, Robert, Waite, Mitchell – *Structuri de date si algoritmi în Java*, Editura Teora, Bucuresti, 1999

[PUVV03] Jawahar Puvvala, Alok Pota – *.NET for Java Developers: Migrating to C#*, Addison Wesley, 2003

[ROTA96] Rotariu, Eugen – *Limbajul Java*, Editura Computer Press Agora, Tîrgu Mures, 1996

[TANA03] Tanasa, Stefan, Olaru, Cristian, Andrei, Stefan – *Java de la 0 la expert*, Editura Polirom, Iasi, 2003

http://java.sun.com

http://msdn.microsoft.com