Développement d'applications réparties

Serveurs Multi-Threadés

Khaled Barbaria
khaled.barbaria@ensta.org

Faculté des Sciences de Bizerte
GLSI3

2022

Plan

Threads et Processus I

Processus

- Un processus est un programme en cours d'exécution complet.
- A son propre espace d'adressage.
- L'OS assure :
  - La protection mémoire
  - L'accès au CPU de chacun des processus
  - Changement de contexte : mémoriser/restituer l'état du processus processus arrêté/activé.

Threads et Processus II

Thread

- Flot de contrôle au sein du processus
- Pas de protection de ressources entre les Threads d'un seul processus, le changement de contexte est beaucoup plus rapide qu'entre les processus.
- Le programmeur doit faire attention à la protection des ressources partagées .

Threads et Processus III

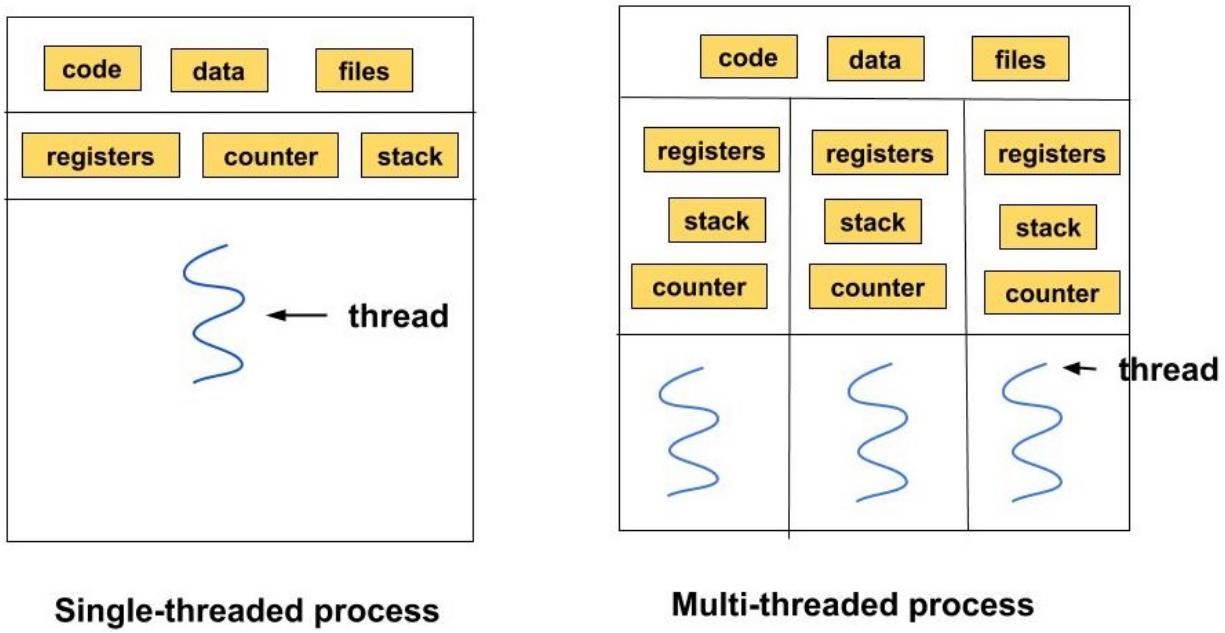

Figure 1 – Processus mono et multi-threadés

Threads et Processus IV

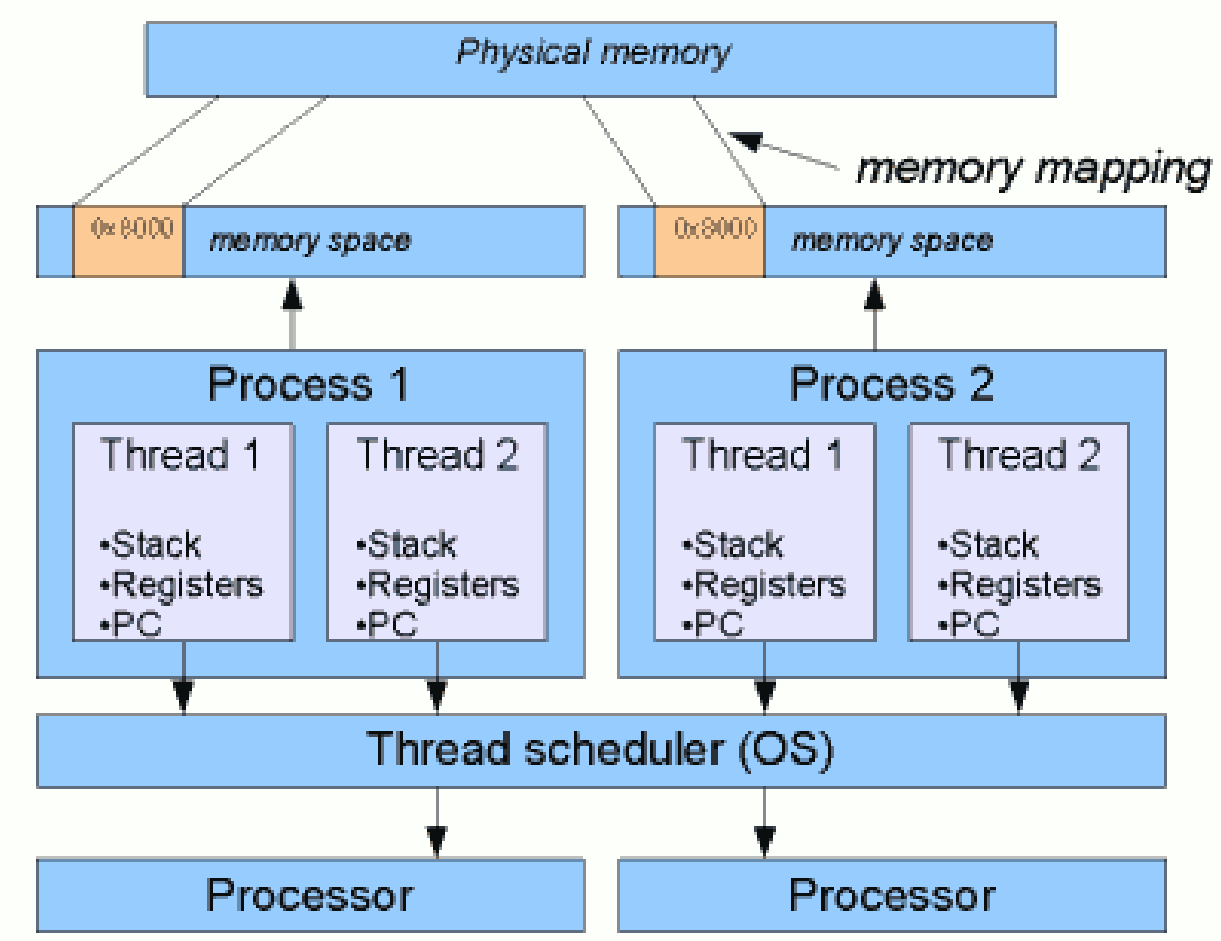

Figure 2 – Organisations des Processus et des Threads

Exemple : jeu vidéo en réseau

- Un seul processus .. trois threads
- Un Thread pour l'affichage graphique
- Un Thread pour l'interaction avec l'utilisateur (clavier, souris, manette„ etc.)
- Un Thread pour la communication avec les processus distants.

Discussion

- Avantages
  - Plus facile à programmer : un thread par tâche concurrente
  - Pas besoin d'écouter *simultanément* sur plusieurs sources d'évènements (on définit un thread par source)
  - Meilleures performances : partage des ressources, utilisation de plusieurs CPU (en cas de disponibilité)
- Inconvénients
  - Si le nombre de threads est important : problèmes de performances
  - Risque de corruption des données : cohérence des données assurée par le programmeur
  - Risque d'inter-blocage

Création des Threads en java I

Deux méthodes

- Étendre la classe `java.lang.Thread`
  - Java ne supporte pas l'héritage multiple, donc hériter de la classe `Thread` empêche d'hériter des autres classes de l'application
  - Hériter de la classe `Thread` peut causer le coût excessif d'héritage de cette classe
- Implémenter l'interface `java.lang.Runnable`
  - Évite les problèmes de la première méthode
  - Plus léger, et donne lieu à un programme mieux structuré

Création des Threads en java II

Listing 1 – Héritage de la classe `Thread`

```
1  class Output extends Thread {
2      private String msg;
3      public Output(String m){msg = m;}
4      public void run(){
5    while(true){
6        try{
7      System.out.println(msg);
8      //attente pendant (au moins) 250 ms
9      sleep(250);
10       }
11       catch(InterruptedException e){}}}
12     public static void main(String [] args) {
13   Output thr1 = new Output("A");
14   Output thr2 = new Output("B");
15   thr1.start(); thr2.start();
16     }
17 }
```

Création des Threads en java III

Listing 2 – Implémentation de la classe `Runnable`

```
1  class OutputR implements Runnable{
2      private String msg;
3      public OutputR(String m){msg = m;}
4      public void run(){
5    while(true){
6        try{
7      System.out.println(msg);
8      //attente pendant (au moins) 250 ms
9      Thread.sleep(250);
10       }
11       catch(InterruptedException e){}}}
12     public static void main(String [] args) {
13   Thread thr1 = new Thread(new OutputR("A"));
14   Thread thr2 = new Thread(new OutputR("B"));
15   thr1.start(); thr2.start();
16     }
17 }
```

Création des Threads en java IV

Remarques générales

- Le Thread du main thread n'est qu'un autre thread (qui commence en premier)
- Le Thread du main peut se terminer avant les autres
- Tout thread peut lancer d'autres threads
- L'appel de la méthode `start` déclare à la machine virtuelle que le Thread peut commencer son exécution
- L'appel de la méthode `run` ne crée pas de nouveau thread mais exécute cette méthode dans le contexte du thread qui appelé `run`

Synchronisation en Java I

Verrous Java

- À chaque object qui a du code (méthode, bloc) `synchronized` est associé un verrou
- L'utilisation des verrous permet une exécution en exclusion mutuelle
- Le verrou est obtenu avant que le code du bloc (ou de la méthode) synchronized ne soit exécuté.
- Le verrou est relâche lors de la sortie du bloc.

Variables conditionnelles Java

- La classe Object définit les méthodes `wait()`, `notify()`, `notifyAll()`

Synchronisation en Java II

Listing 3 – sans synchronized

```
1  class Power{
2      void printPower(int n){//method not synchronized
3    int temp = 1;
4    for(int i=1;i<=5;i++){
5        System.out.println(Thread.currentThread().getName()
6                + ":- " +n + "^"+ i + " value: " + n*temp);
7        temp = n*temp;
8        try{
9      Thread.sleep(100);
10       }catch(Exception e){System.out.println(e);}
11   }
12       }
13 }
14
15 class Thread1 extends Thread{
16     Power p;
17     Thread1(Power p){
18   this.p=p;
19       }
20     public void run(){
21   p.printPower(5);
```

Synchronisation en Java III

```
22        }
23    }
24    class Thread2 extends Thread{
25        Power p;
26        Thread2(Power p){
27      this.p=p;
28        }
29        public void run(){
30      p.printPower(8);
31        }
32    }
33
34    public class SynchronizationExample{
35        public static void main(String args[]){
36      Power obj = new Power();//only one object
37      Thread1 p1=new Thread1(obj);
38      Thread2 p2=new Thread2(obj);
39      p1.start();
40      p2.start();
41        }
42    }
43
44
```

Synchronisation en Java IV

```
45  /*
46     Thread-1:- 8^1 value: 8
47     Thread-0:- 5^1 value: 5
48     Thread-0:- 5^2 value: 25
49     Thread-1:- 8^2 value: 64
50     Thread-0:- 5^3 value: 125
51     Thread-1:- 8^3 value: 512
52     Thread-0:- 5^4 value: 625
53     Thread-1:- 8^4 value: 4096
54     Thread-0:- 5^5 value: 3125
55     Thread-1:- 8^5 value: 32768
56  */
```

Synchronisation en Java V

Listing 4 – avec synchronized

```
1  public class SynchronizationExampleOK{
2      public static void main(String args[]){
3    Power obj = new Power();//only one object
4    Thread1 p1=new Thread1(obj);
5    Thread2 p2=new Thread2(obj);
6    p1.start();
7    p2.start();
8      }
9  }
10 /*
11 Thread-0:- 5^1 value: 5
12 Thread-0:- 5^2 value: 25
13 Thread-0:- 5^3 value: 125
14 Thread-0:- 5^4 value: 625
15 Thread-0:- 5^5 value: 3125
16 Thread-1:- 8^1 value: 8
17 Thread-1:- 8^2 value: 64
18 Thread-1:- 8^3 value: 512
19 Thread-1:- 8^4 value: 4096
20 Thread-1:- 8^5 value: 32768
21 */
```

Synchronisation en Java VI

Listing 5 – ”avec synchronized ; mais l’objet n’est pas partagé”

```
1  public class SynchronizationExample2{
2      public static void main(String args[]){
3    Thread1 p1=new Thread1(new Power());
4    Thread2 p2=new Thread2(new Power());
5    p1.start();
6    p2.start();
7      }
8  }
9  /*
10 Thread-0:- 5^1 value: 5
11 Thread-1:- 8^1 value: 8
12 Thread-1:- 8^2 value: 64
13 Thread-0:- 5^2 value: 25
14 Thread-0:- 5^3 value: 125
15 Thread-1:- 8^3 value: 512
16 Thread-0:- 5^4 value: 625
17 Thread-1:- 8^4 value: 4096
18 Thread-0:- 5^5 value: 3125
19 Thread-1:- 8^5 value: 32768
20 */
```

Exemple de serveur multi-threadé en Java I

```
1  import java.net.*;import java.io.*;import java.util.*;
2
3  class ClientHandler implements Runnable{
4      private Socket clientSocket = null;
5      private String msg;
6      public ClientHandler (Socket cs, String m) {
7          this.clientSocket = cs; msg=m;
8      }
9      public void run() {
10         try {
11       OutputStream output = clientSocket.getOutputStream();
12             String time = new Date().toString();
13             output.write(("HTTP/1.1 200 OK<H1>\n\nWorkerRunnable: "
                   +
14         this.msg + " - " + time + "<H1>").getBytes());
15       clientSocket.close();
16       System.out.println("Request processed: " + time);
17         } catch (IOException e) {
18       e.printStackTrace();
19         }
20     }
21 }
```

Exemple de serveur multi-threadé en Java II

```
1  public class MultiThreadedServer{
2      public static void main(String args[]) throws Exception{
3    ServerSocket ss = new ServerSocket(8080);
4    while (true){
5        Socket clientSocket=ss.accept();
6        new Thread(new ClientHandler(clientSocket,
7             "Mulithreaded Server!")).start();
8          }
9      }
10 }
```