

Cours n° 5

Multiprogrammation et processus légers



Sommaire

1. Multiprogrammation

- Classe Thread
- Interface Runnable

2. Communication entre processus

- Mémoire partagée
- Flot partagé
- Concurrence d'accès à des ressources et synchronisation

Définitions (1/2)

Processus (ou tâche)

Ensemble d'instructions du processeur à exécuter (programme) associé à un espace mémoire (pile d'exécution, données, descripteurs de ressource)

Un seul processus actif par processeur

Multiprocessus à temps partagé ou quasi parallélisme

Liste de N processus (1 actif , N-1 en veille) et quantum de temps (1/100 sec)

Affectation du processeur à un processus différent à chaque quantum de temps

Gestion des priorités des processus

Impression de parallélisme

Définitions (2/2)

Processus légers (ou thread)

Ensemble de processus partageant des ressources communes (mémoire, ...)

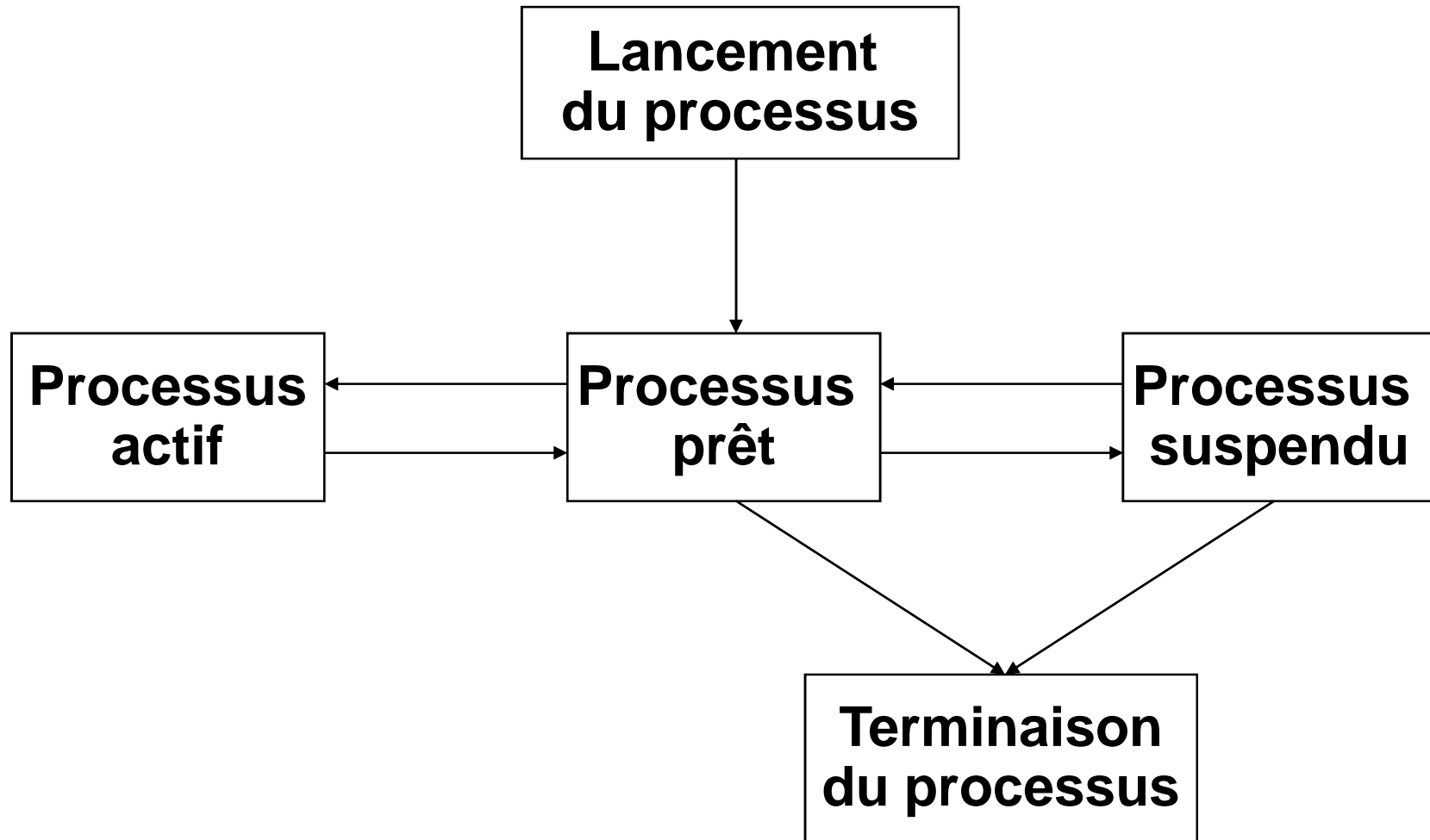
Synchronisation et concurrence

Gestion des accès à une ressource partagée par plusieurs processus légers

Mécanisme de synchronisation à base de verrous (moniteurs)

Processus en attente (suspendu) d'une ressource affectée à un autre processus

Vie d'un processus



Lancement, suspension et arrêt d'un processus

Constructeurs

Thread() Thread(**String** name)

Modification de l'état du processus

void start() Lancement du processus (appel par la JVM de la méthode run())

void yield() Passage de l'état exécution à l'état prêt

void sleep(**long** ms) Passage à l'état suspendu pendant un temps donné

void interrupt() Terminaison du processus

Accesseur sur l'état du processus

boolean isAlive() retourne vrai si la tâche est prête

boolean interrupted() retourne vrai si la tâche a été interrompue

Thread.State getState() retourne l'état du processus (NEW, RUNNABLE, BLOCKED, WAITING, TERMINATED)

1.1 CLASSE THREAD

Classe dérivée

```
public class Compteur extends Thread {
    JTextPane tp;
    int délai;
    // constructeur
    Compteur(int d) {
        délai = d;
        tp = new JTextPane();
        JFrame fen = new JFrame(); fen.setSize(100, 100);
        fen.getContentPane().add(tp); fen.setVisible(true);
    }
    // méthode appelée à l'exécution du thread
    public void run() {
        int i = 0;
        while (true) {
            tp.setText(Integer.toString(i)); i++;
            try {sleep(délai);} catch (InterruptedException e) {}
        }
    }
}
```

1.1 CLASSE THREAD

Lancement de processus

```
public class testCompteurs {  
  
    public static void main(String[] args) {  
  
        Compteur[] pa = new Compteur[5];  
        for (int i = 0; i < pa.length; i++) {  
            pa[i] = new Compteur(10*(i+1));  
  
            // lancement des compteurs  
            pa[i].start();  
        }  
  
    }  
}
```



Caractéristiques

Ajout à tout tâche de la possibilité d'être lancé dans un processus léger

Méthode à redéfinir

public void run() contient le code des traitements exécuté au lancement du processus

Lancement du processus

void start()

Adaptée en cas de classes existantes

1.2 INTERACE RUNNABLE

Lancement du processus d'écoute (1/2)

```
public class lancerEcouleurTCP {  
  
    /**  
     * @param args  
     */  
    public static void main(String[] args) {  
        int port = 32504;  
        InetAddress saddr = null;  
        Socket s = null;  
        try {  
            InetAddress addr = InetAddress.getLocalHost();  
            saddr = new InetAddress(addr, port);  
        }  
        catch (UnknownHostException exp){  
            System.out.println("machine inconnue");  
        }  
    }  
}
```

1.2 INTERACE RUNNABLE

Lancement du processus d'écoute (2/2)

```
try {  
    // création d'un écouteur de connexion  
    ServerSocket ss = new ServerSocket();  
    // attachement  
    ss.bind(saddr);  
    // acceptation de la connexion  
    while (true) {  
        s = ss.accept();  
        System.out.println("Connexion établie entre " +  
            s.getLocalSocketAddress() + " et " +  
            s.getRemoteSocketAddress());  
        new EcouleurTCP(s);  
    }  
} catch (IOException exp){  
    System.out.println("erreur d'ouverture");  
}  
}
```

1.2 INTERACE RUNNABLE

Constructeur

```
public class EcouteurTCP implements Runnable {

    BufferedReader br = null;
    PrintWriter pw = null;

    public EcouteurTCP(Socket s) {
        try {
            br = new BufferedReader(new InputStreamReader(s.getInputStream()));
            pw = new PrintWriter(s.getOutputStream(), true);
        }
        catch (IOException exp){
            System.out.println("erreur de création des flots");
        }

        new Thread(this).start();
    }
}
```

1.2 INTERACE RUNNABLE

Processus de communication (côté serveur)

```
public void run() {  
  
    Date d = new Date();  
  
    // Communication  
    try {  
        String ligne = br.readLine();  
        System.out.println(ligne);  
  
        pw.println("bienvenue sur le serveur à " + d.toString());  
    }  
    catch (IOException exp){  
        System.out.println("erreur d'entrée-sortie");  
    }  
}
```

1.2 INTERACE RUNNABLE

Processus de communication (côté clients)

Client 1

```
Connexion etablie entre /192.168.1.75:32506 et /192.168.1.75:32504  
bienvenue sur le serveur Thu Feb 24 14:17:24 CET 2022
```

Client 2

```
Connexion etablie entre /192.168.1.90:32507 et /192.168.1.75:32504  
bienvenue sur le serveur Thu Feb 24 14:18:34 CET 2022
```

Principes

Inter-Process Communication ou IPC

Ensemble de mécanismes permettant à des processus concurrents (ou distants) de communiquer

Outils permettant aux processus de s'échanger des données
mémoire partagé,
flot partagé (pipe, socket, ...)

Outils permettant de synchroniser les processus (planification)
blocage, déblocage par un processus tiers

Outils permettant de gérer les sections critiques (cohérence des données)
sémaphores,
moniteur
réseau de Petri

2.1 MEMOIRE PARTAGEE

Partage d'une variable de classe (1/2)

```
class ProcessusPartageM extends Thread
{
    static private int numero = 0;
    int numeroProcessus;
    // Constructeur
    ProcessusParallèles() {
        super(("Processus numero " + numero++));
        numeroProcessus = numero;
    }

    // Corps de l'application
    public void run() {
        for (int i=0; i < 10; i++) affiche(i);
    }

    void affiche(int i) {
        try { sleep(100); } catch (InterruptedException e) {}
        System.out.println(this.getName() + " à l'itération" + i);
    }
}
```


2.1 MEMOIRE PARTAGEE

Partage d'une variable de classe (2/2)

```
public class testPartage {  
    public static void main(String args[])  
    { //création et activation des 5 Processus  
        for (int i=0; i<5; i++)  
        { Thread p = new ProcessusPartageM(); p.start(); }  
    }  
}
```

```
Processus numero 4 à l'itération 0  
Processus numero 0 à l'itération 0  
Processus numero 2 à l'itération 0  
Processus numero 1 à l'itération 0  
Processus numero 3 à l'itération 0  
Processus numero 0 à l'itération 1  
Processus numero 4 à l'itération 1  
Processus numero 2 à l'itération 1  
Processus numero 3 à l'itération 1  
Processus numero 1 à l'itération 1
```

2.2 FLOT PARTAGE

Partage d'un tube (1/3)

```
public class testPartageF {  
  
    /**  
     * @param args  
     */  
    public static void main(String[] args) {  
        // création du tube  
        PipedWriter out = new PipedWriter();  
        PipedReader in = null;  
        try {  
            in = new PipedReader(out);  
            Emetteur e = new Emetteur(out);  
            Recepteur r = new Recepteur(in);  
            new Thread(e).start();  
            new Thread(r).start();  
        }  
        catch (IOException e) {}  
    }  
}
```

2.2 FLOT PARTAGE

Partage d'un tube (2/3)

```
public class Emetteur extends Thread {  
  
    PipedWriter out;  
  
    public Emetteur(PipedWriter out) {  
        this.out = out;  
    }  
  
    public void run() {  
        while (true) {  
            String mess = Keyboard.getString("message?");  
            try {  
                out.write(mess + "\n");  
            }  
            catch (IOException e) {}  
        }  
    }  
}
```

2.2 FLOT PARTAGE

Partage d'un tube (3/3)

```
public class Recepteur extends Thread {
    JTextPane tp;
    BufferedReader br;

    public Recepteur(PipedReader in) {
        br = new BufferedReader(in);
        tp = new JTextPane();
        JFrame fen = new JFrame(); fen.setSize(100, 100);
        fen.getContentPane().add(tp); fen.setVisible(true);
    }

    public void run() {
        String mess = null;
        while (true) {
            try {
                mess = br.readLine();
            } catch (IOException e) {}
            tp.setText(mess);
        }
    }
}
```

Problématiques et outils

Objets accessibles en écriture par plusieurs processus

Constitution de sections critiques (pas de modification de l'état du processus)

Modificateur **synchronized**

devant un objet protège son accès pendant le bloc d'instruction suivant
devant une méthode protège l'accès à la méthode

Méthode de synchronisation

void wait() **void wait(long)**

Demande d'accès à un objet (suspendu s'il est affecté)

void notify() **void notifyAll(long)**

Réveille un ou plusieurs processus suspendus

void join() **void join(long)**

Attente de la terminaison d'un processus