UF2. Programació de processos i fils

Processos i Fils

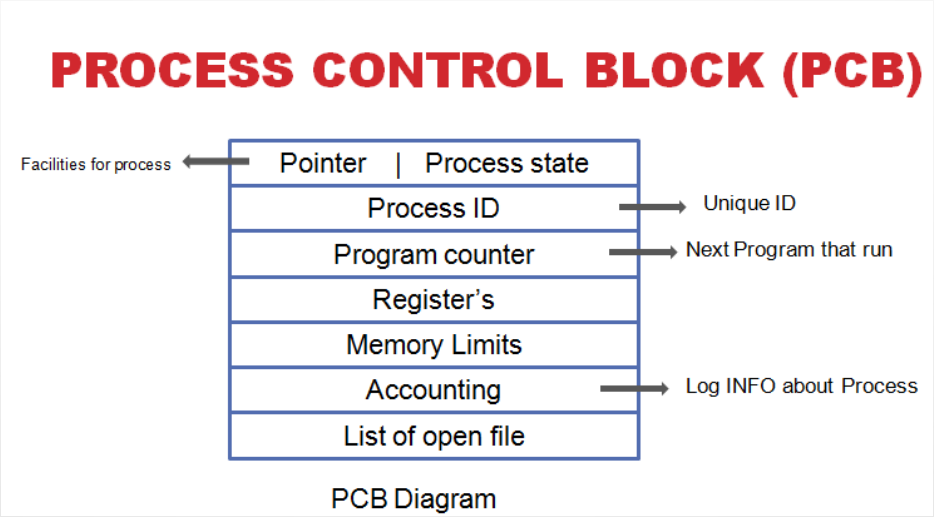
INS Vidreres
Adrià Tregon Muniesa

Introducció

- Des de fa temps, les aplicacions informàtiques necessiten realitzar gran quantitat de càlculs numèrics per donar resposta immediata a les necessitats humanes.
- Per agilitzar aquests càlculs, es decideix **dividir la feina** per guanyar en temps i recursos: es divideix en **processos**.

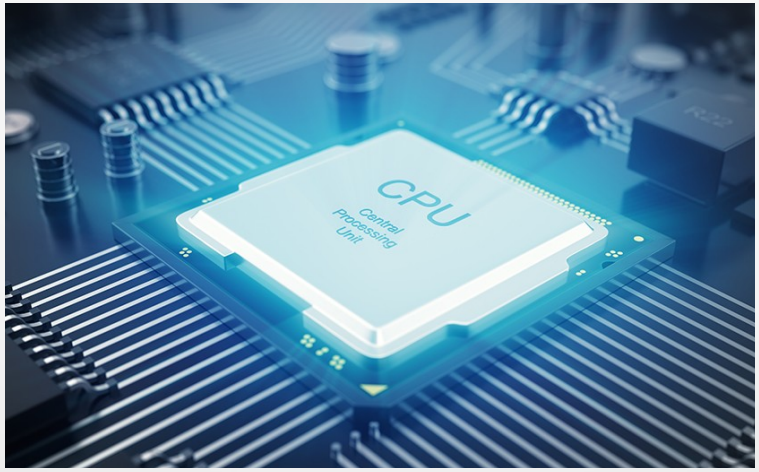
Processos

- Un **programa** és un conjunt d'instruccions que resolen un determinat problema. No varia, és estàtic.
- En canvi, un **procés** és un tros de programa en execució. Els valors varien, és dinàmic.
- El **BCP (Bloc de control del procés)** són els atributs associats a la imatge del procés utilitzats pel S.O per fer-ne les crides.



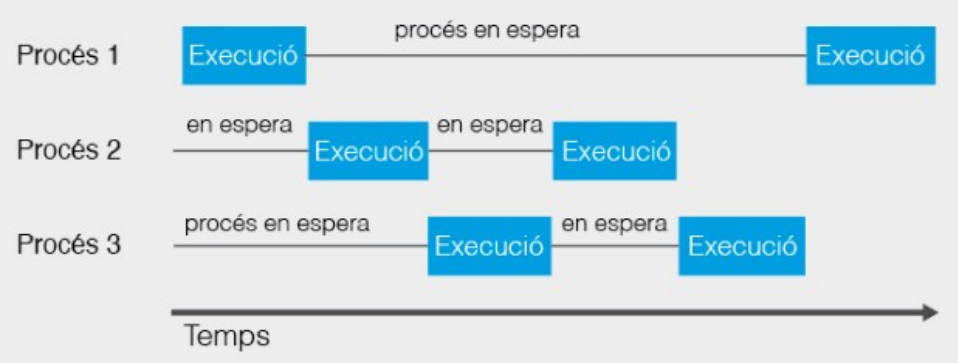
Processos

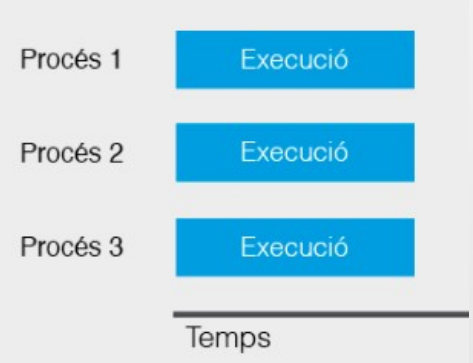
- Els **processadors** són els encarregats d'executar els processos. Si un sistema de informació n'utilitza diversos, s'anomena **multiprocés** o **multiprocessador**, atès que pot executar-ne més d'un de manera concurrent.
- Per contra, el sistema **monoprocessador** és aquell que està format únicament per un processador.



Processos

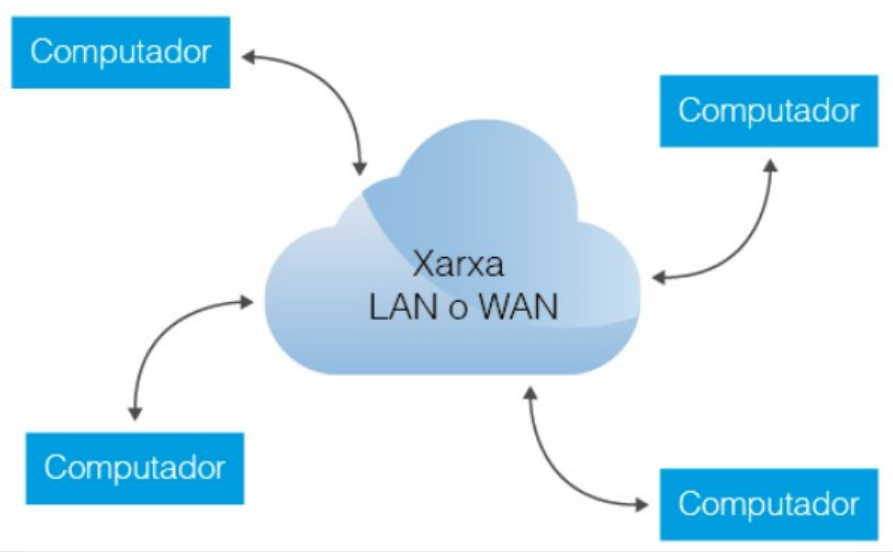
- S’utilitza el concepte de **multiprogramació** o **programació concurrent** per fer referència a la gestió que duu a terme el S.O per a l’execució ‘concurrent’ de processos.
- En canvi, quan el sistema és multiprocessador, es parla de **programació paral·lela**.





Processos

- La **programació distribuïda** és un tipus de programació concurrent en la qual els processos són executats en una xarxa de processadors autònoms.
- Des del punt de vista de l'usuari, el sistema es veu com una sola computadora.



Estats dels processos

- El sistema operatiu serà l'encarregat de controlar en cada moment quin procés entra en execució, i per a prendre aquesta decisió, utilitza l'**estat dels processos,** que es pot definir com la descripció de la seva activitat en un moment concret.
- Actualment s'utilitza un model de 7 estats però aquest va evolucionar dels seus predecessors, els models de 3 i 5 estats.

Estats dels processos

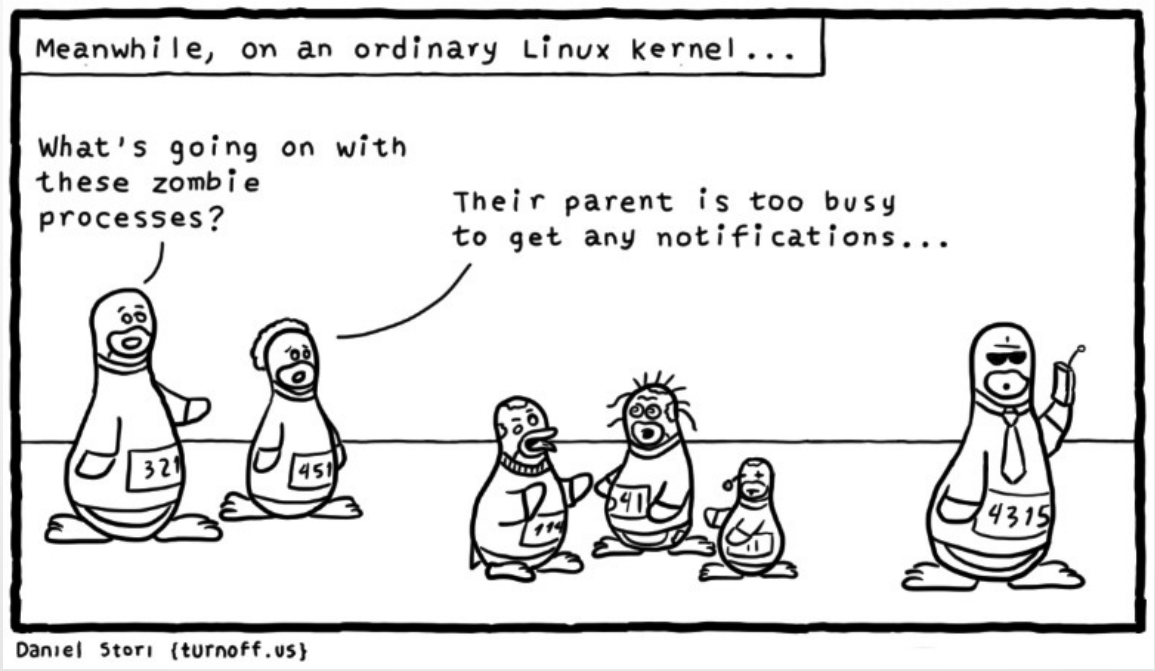
- Model 3 estats:
  - **Execució:** està utilitzant la CPU en aquest moment.
  - **Preparat**: amb possibilitats d'entrar en execució.
  - **Bloquejat**: sense possibilitat d'entrar en execució.

Estats dels processos

- Model 5 estats: s'introdueixen dos estats més:
  - **Nou**: procés que acaba de crear-se i no està preparat per executar-se.
  - **Acabat**: procés que ha finalitzat la seva execució o és abandonat per un error irrecuperable.

Estats dels processos

- Model 5 estats:
  - Linux utilitza un altre estat anomenat **zombie** o **difunt** que són processos que ja han finalitzat abans que el seu pare pugui esperar per ells (crida al sistema wait) per obtenir el seu identificador i poder alliberar-los.



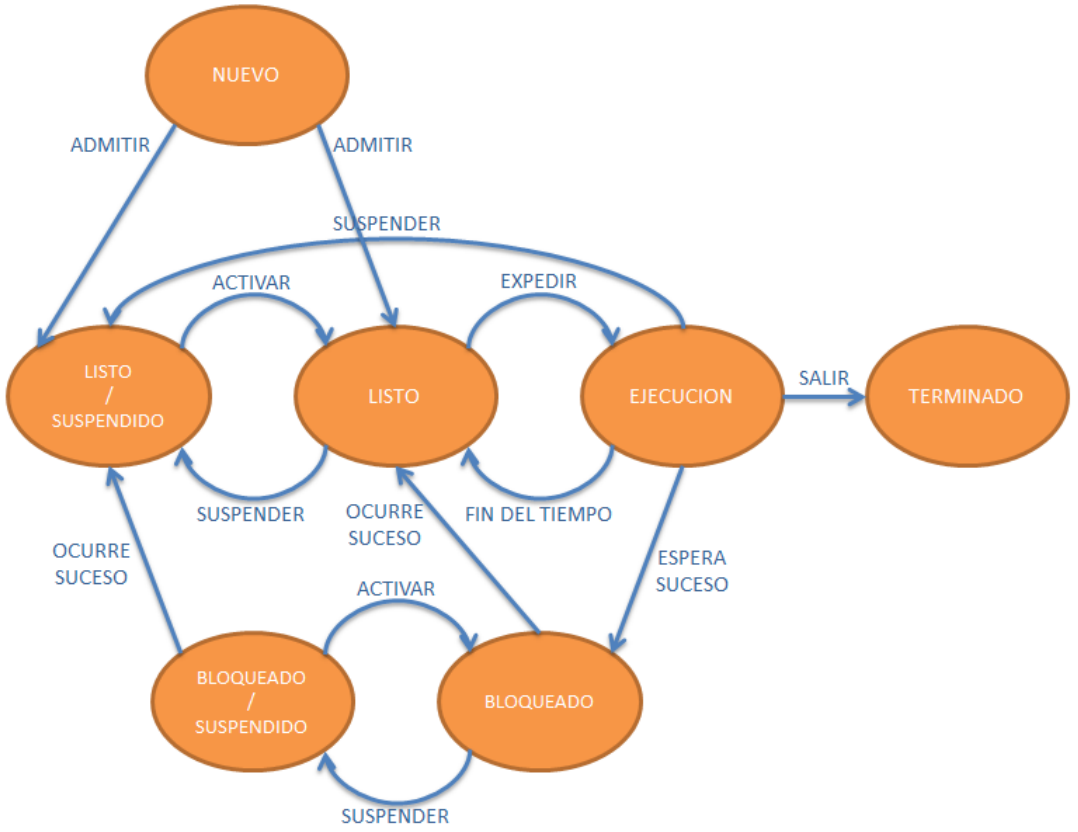
Estats dels processos

- Model 5 estats:
  - Si el procés pare acaba abans que ho facin també els fills, aquests passen a l'estat **Orfe** i són heretats pel procés init, pare de tots els processos (PID=1).

```
adriatregon@adriatregon-X550VX ~ $ ps -elf | head -1; ps -elf | awk '{if ($5 == 1 && $3 != "root") {print $0}}' | head
F S UID          PID  PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
4 S syslog       989     1  0  80   0 - 64100 -      mar03 ?        00:00:00 /usr/sbin/rsyslogd -n
4 S avahi        996     1  0  80   0 - 11230 -      mar03 ?        00:00:00 avahi-daemon: running [adriatregon-X550VX.lo
4 S message+    1019     1  0  80   0 - 11127 -      mar03 ?        00:00:05 /usr/bin/dbus-daemon --system --address=syst
4 S oracle      1876     1  0  80   0 - 11322 -      mar03 ?        00:00:00 /lib/systemd/systemd --user
```

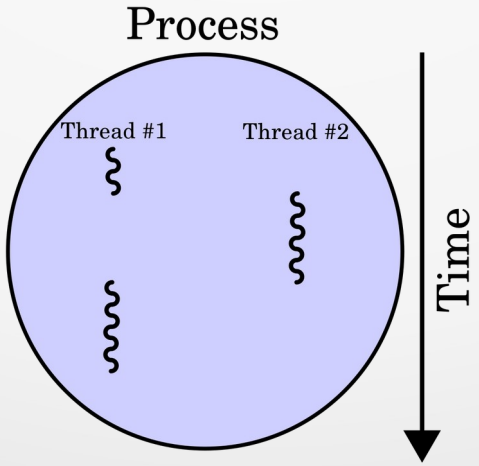
Estats dels processos

- Model 7 estats:
  - **Bloquejat i suspès:** els processos estan bloquejats i esperen en memòria secundaria.
  - **Preparat i suspès:** els processos estan preparats i esperen en memòria secundaria.



Fils

- Alguns sistemes operatius, amb l'objectiu d'optimitzar els recursos, permeten que dins d'un mateix procés puguin coexistir vàries línies d'execució denominades **fils** (*threads*) o **processos lleugers**.
- Són una forma d'executar paral·lelament diferents parts d'un mateix programa. Per exemple, en un editor de text, un fil pot controlar l'ortografia, un altre les pulsacions, etc.



Fils

- Les **avantatges** son:

  - El S.O tarda molt menys en crear, finalitzar o intercanviar fils que el que tarda amb els processos, ja que la majoria de la informació del BCP és compartida per tots els fils del procés.

  - Els fils, al compartir posicions de memòria comuns, la comunicació entre ells és immediata.

  - Poden ser executats de forma paral·lela en un sistema multiprocés.

Fils

- Parlem de **processos multifil** quan s'executen diversos fils concurrentment, realitzant diferents tasques i col·laborant entre ells.
- Com a mínim sempre n'hi ha un (*fil principal*). Un fil pot crear nous fils pel que es creen **relacions pare-fills**.
- Quan un procés finalitza, tots els seus fils també ho fan. De forma equivalent, quan finalitzen tots els fils d'un procés, el procés també acaba i tots els seus recursos són alliberats.

Fils

- Java defineix la funcionalitat principal de la gestió dels fils a la classe **Thread.** Aquesta conté un mètode **run()** a sobreescriure que contindrà el codi a executar pel fil.
- Run() no s'executa quan s'instancia el fil, sinó quan es crida el mètode **start()**:

```
--- exec-maven-plugi
Final Fil Principal
Fil Y: 0
Fil Y: 1
Fil Y: 2
Fil Y: 3
Fil Y: 4
Fil X: 0
Fil X: 1
Fil X: 2
Fil X: 3
Fil X: 4
------------------
```

```java
public class Fil extends Thread {

    String nFil;

    public Fil(String strP) {
        nFil=strP;
    }

    public void run(){
        for(int x=0;x<5;x++){
            System.out.println(nFil+ ": " + x);
        }

    }

    public static void main(String[] args) {

        Thread primer = new Fil("Fil X");
        Thread segon = new Fil("Fil Y");

        //S'executen de forma paral·lela, és igual quin cridem primer:
        primer.start();
        segon.start();

        System.out.println("Final Fil Principal");
```

Fils

- Com que Java no suporta l'herència múltiple, la classe que hereti de Thread no podria heretar de cap altre superclasse.
- Per evitar això, s'implementa una classe amb la interfície **Runnable** que conté també el mètode run(), i es passa com a paràmetre en el constructor del Thread.

```
--- exec-maven-plugi
Final Fil Principal
Fil X: 0
Fil X: 1
Fil X: 2
Fil X: 3
Fil X: 4
Fil Y: 0
Fil Y: 1
Fil Y: 2
Fil Y: 3
Fil Y: 4
------------------
```

```java
public class Fil implements Runnable {

    String nFil;

    public Fil(String strP) {
        nFil=strP;
    }

    public void run(){
        for(int x=0;x<5;x++){
            System.out.println(nFil+ ": " + x);
        }
    }

    public static void main(String[] args) {

        Fil objPrimer = new Fil("Fil X");
        Fil objSegon = new Fil("Fil Y");

        Thread primer = new Thread(objPrimer);
        Thread segon = new Thread(objSegon);

        primer.start();
        segon.start();

        System.out.println("Final Fil Principal");
```

Fils

- El mètode **join()** força els fils a esperar el fi de l'execució d'un altre (fil pare).

```
Final Fil Principal
Suma Total: 0
Fil X: 0
Fil X: 1
Fil X: 2
Fil X: 3
Fil X: 4
Fil Y: 0
Fil Y: 1
Fil Y: 2
Fil Y: 3
Fil Y: 4
```

```
public class Fil implements Runnable {

    String nFil;
    static int suma;

    public Fil(String strP) {
        nFil=strP;
        suma=0;
    }

    public void run(){
        for(int x=0;x<5;x++){
            System.out.println(nFil+ ": " + x);
            suma=suma+x;
        }
    }

    public static void main(String[] args) {

        Fil objPrimer = new Fil("Fil X");
        Fil objSegon = new Fil("Fil Y");
        Thread primer = new Thread(objPrimer);
        Thread segon = new Thread(objSegon);
        primer.start();
        segon.start();
        System.out.println("Final Fil Principal");
        System.out.println("Suma Total: "+suma);
    }
}
```

```
Fil X: 0
Fil X: 1
Fil X: 2
Fil Y: 0
Fil Y: 1
Fil Y: 2
Fil Y: 3
Fil Y: 4
Fil X: 3
Fil X: 4
Final Fil Principal
Suma Total: 20
```

```
public static void main(String[] args) {

    try {
        Fil objPrimer = new Fil("Fil X");
        Fil objSegon = new Fil("Fil Y");

        Thread primer = new Thread(objPrimer);
        Thread segon = new Thread(objSegon);

        primer.start();
        segon.start();

        primer.join();
        segon.join();

        System.out.println("Final Fil Principal");
        System.out.println("Suma Total: "+suma);

    } catch (InterruptedException ex) {
        Logger.getLogger(Fil.class.getName()).log(Lev
    }
}
```

Sincronització de fils

- A Java, si es vol que un codi sigui executat per un sol fil en un moment donat (*exclusió mútua*), s'utilitza la paraula reservada **synchronized.**
- Si es declara en un mètode, cap altre fil que executi un mètode sincronitzat del mateix objecte, s'executarà mentre aquest ho faci.

```
public synchronized static void connect(FTPClient client){
```

- També es pot utilitzar solament en una part del codi:

```
public void connect(FTPClient client){

    String sFTP = "ftpupload.net";
    String sUser = "epiz_27783373";
    String sPassword = "9t8CEkeHXwo8C9C";
    boolean login = false;

    try {

        synchronized(this){
            //Connexió al servidor:
            client.connect(sFTP);
        }

        //Logueig al servidor:
        login = client.login(sUser,sPassword);
```

Sincronització de fils

- Només un fil pot executar un codi *synchronized* a la vegada de la mateixa instància:

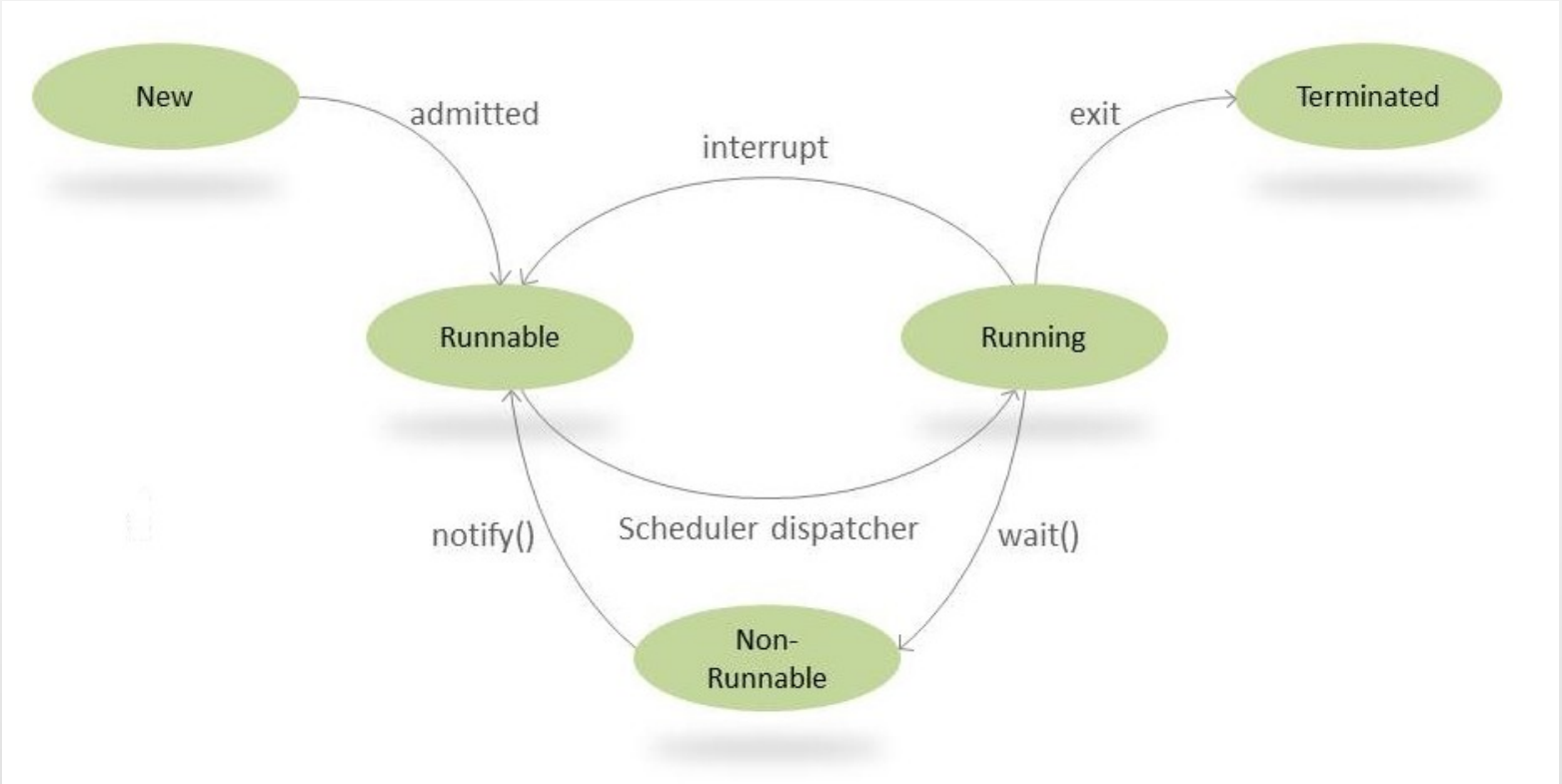
```
 * @author adriatregon
 */
public class Fil implements Runnable {

    static int suma;

    public Fil() {
        suma=0;
    }

    public void run(){
        suma();
    }

    public synchronized void suma(){

        for(int x=0;x<5;x++){
            try {
                System.out.println(Thread.currentThread().getName()+ ": " + x);
                suma=suma+x;
                Thread.sleep(2000);

            } catch (InterruptedException ex) {
                Logger.getLogger(Fil.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
```

```
public static void main(String[] args) {

    try {
        Fil obj = new Fil();
        Thread primer = new Thread(obj);
        Thread segon = new Thread(obj);
        primer.setName("Fil X");
        segon.setName("Fil Y");

        primer.start();
        segon.start();

        primer.join();
        segon.join();

        System.out.println("Final Fil Principal");
        System.out.println("Suma Total: "+suma);

    } catch (InterruptedException ex) {
        Logger.getLogger(Fil.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

```
Fil X: 0
Fil X: 1
Fil X: 2
Fil X: 3
Fil X: 4
Fil Y: 0
Fil Y: 1
Fil Y: 2
Fil Y: 3
Fil Y: 4
Final Fil Principal
Suma Total: 20
----------------------
```

Sincronització de fils

- Els fils, a l'igual que els processos, poden trobar-se en diferents **estats**: nou, preparat, en execució, bloquejat, adormit, en espera o acabat.



Sincronització de fils

- Els fils, a part de sincronitzar-se, poden comunicar-se entre ells per mitjà de mètodes per fer canvis d'estats.
  - **wait()**: treu el fil en curs i el passa a l'estat d'espera.
  - **notify() o notifyAll()**: un o molts fils de la cua d'espera passen a l'estat preparat. (Wait/notify sempre dins d'un synchronized)

```
public static void main(String[] args) {

    MainXY obj = new MainXY();
    Thread primer = new Thread(obj);
    Thread segon = new Thread(obj);

    primer.setName("Fil X");
    segon.setName("Fil Y");

    primer.start();
    segon.start();
}
```

```
public void run(){

    if("Fil X".equals(Thread.currentThread().getName())){
        filx();
    } else {
        fily();
    }
}
```

Sincronització de fils

```java
public void filx(){

    synchronized(this){
        try {
            System.out.println("Fil X: Esperant que el fil Y acabi de sumar...");
            wait();

        } catch (InterruptedException ex) {
            Logger.getLogger(ThreadA.class.getName()).log(Level.SEVERE,  msg: null, ex);
        }

        System.out.println("Fil X: suma total és: " + suma);
        notify();
    }

}
```

```
Fil X: Esperant que el fil Y acabi de sumar.
Fil Y: Ja he acabat de sumar!
Fil X: suma total és: 4950
Fil Y: Adeu!
```

```java
public void fily() {

    try {
        Thread.sleep( millis: 1000);
        synchronized(this){
            for(int i=0; i<100 ; i++){
                suma += i;
            }
            System.out.println("Fil Y: Ja he acabat de sumar!");
            notify();
            wait();
            System.out.println("Fil Y: Adeu!");
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Fils

- Un fil acostuma a crear nous fils pel que podem tenir un número elevat a gestionar. Per facilitar l'administració, es poden agrupar utilitzant la classe **ThreadGroup**.
- Tots els grup pertanyen a un fil pare que els ha creat.

```
ThreadGroup Grup1 = new ThreadGroup ("Grup1");
ThreadGroup Grup2 = new ThreadGroup (Grup1, "Grup2"); //Fill del Grup1
Thread fil = new Thread (Grup2, "Fil 1"); //Pertany al Grup2
```

- D'altre banda, cada fil té un número que determina la seva **prioritat** (d'1 a 10) i que utilitza el S.O per decidir quin posa en execució.

```
fil.setPriority(Thread.MAX_PRIORITY);
```

Fils

- Un fil pare pot interrompre els seus fills que ha creat per mitjà del mètode **interrupt**, el qual genera una excepció *InterruptedException*:

```
ThreadGroup g1 = new ThreadGroup( name: "G1");

Sum primer = new Sum(g1,  fil_suma: "Fil suma");
Mult segon = new Mult(g1,  fil_multiplicació: "Fil multiplicació");

primer.start();
segon.start();

primer.interrupt(); //interrompre un fil
g1.interrupt(); //interrompre un grup de fils
```

Fils

- Quan existeix més d'un fil d'execució simultani, els IDE's proporcionen eines de **depuració multifil**.
- Si tenim aplicat un breakpoint, podem seleccionar (*botó dret → Make current*) quin fil volem debugar en foreground