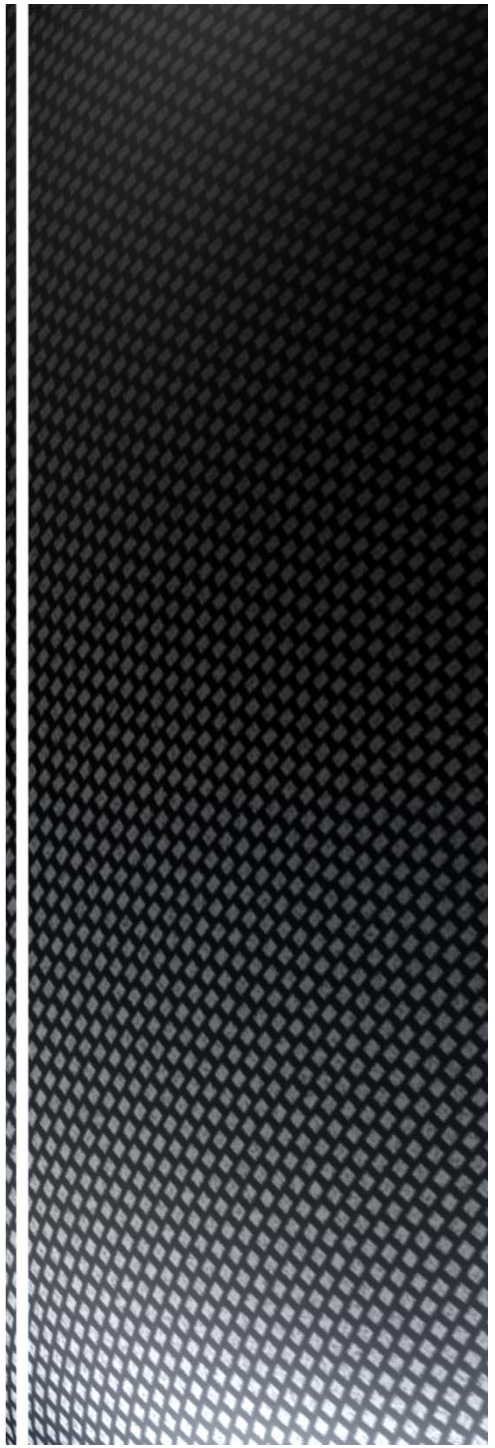
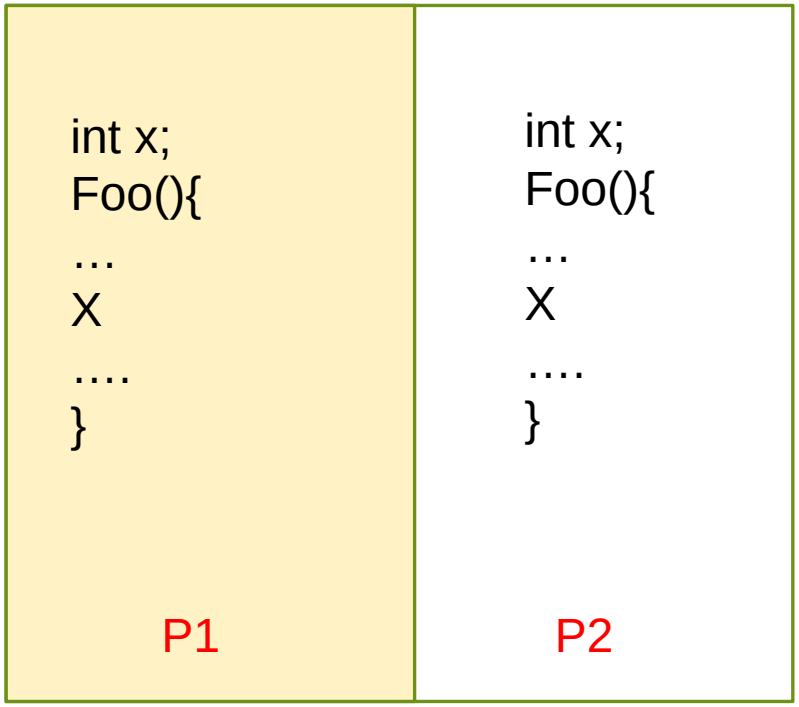
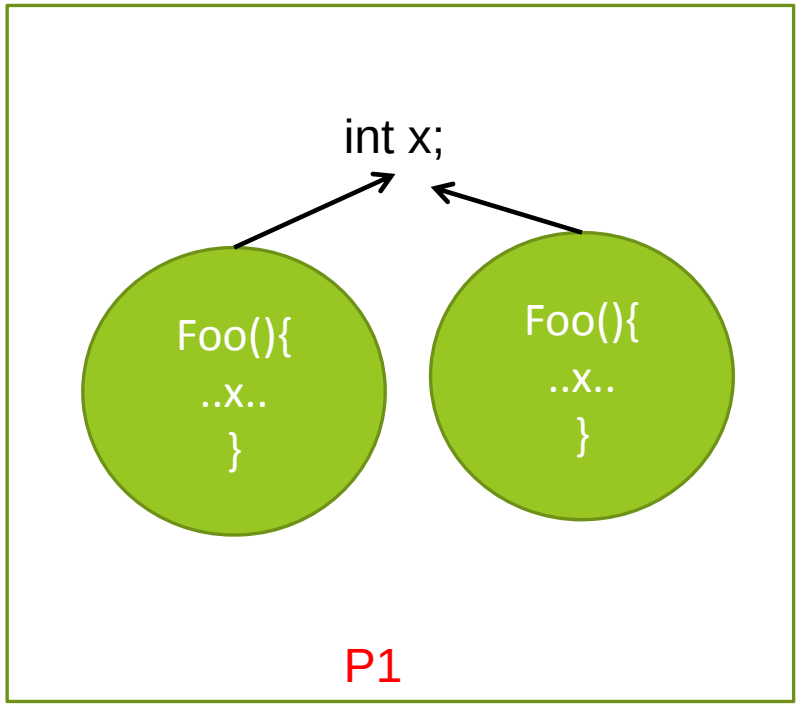
Android

THREADS



Procesos vs Threads



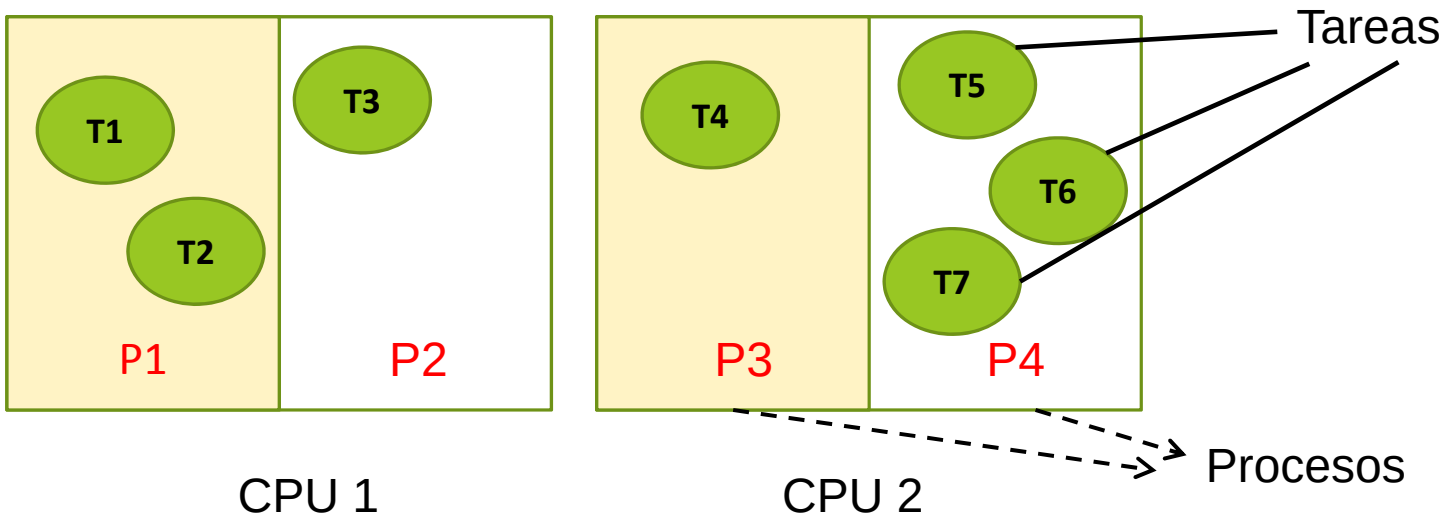


Los procesos no suelen compartir memoria

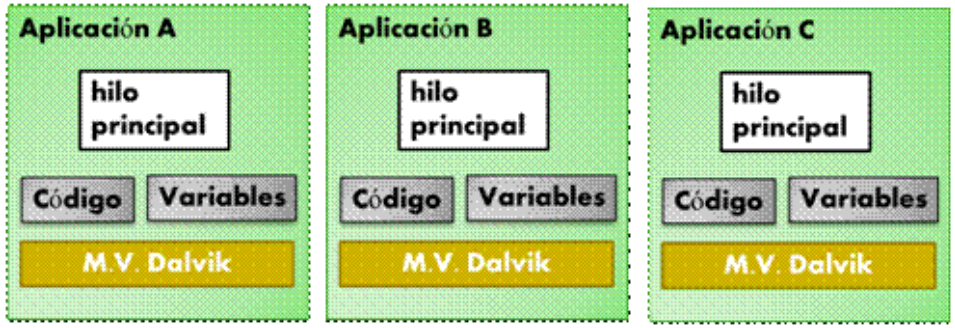
Theads dentro de un proceso pueden compartir memoria

Threads

- Un thread nos permite realizar varias tareas al mismo tiempo dentro de un proceso del sistema operativo.
- Desde el punto de vista de implementación cada thread tiene su propio contador de programa, y su pila de ejecución, pero comparte el área de memoria statica y el heap con los demás threads de un proceso.



Cada vez que se lanza una nueva aplicación en Android el sistema crea un nuevo proceso Linux para ella y la ejecuta en su propia máquina virtual Dalvik (Por supuesto si está programada en Java, si lo estuviera en código nativo no haría falta la máquina virtual).



Trabajar en procesos diferentes nos garantiza que desde una aplicación no se pueda acceder a la memoria (código o variables) de otras aplicaciones.

T

Cuando trabajamos con varios hilos, estos pueden acceder a las variables de forma simultánea. **Hay que tener cuidado de que un hilo no modifique el valor de una variable mientras otro hilo está leyéndola.** Este problema se resuelve en Java definiendo **secciones críticas** mediante la palabra reservada synchronized. Trataremos este problema más adelante.

Threads

Cuando se lanza una nueva aplicación el sistema crea un nuevo hilo de ejecución (thread) para esta aplicación conocida como hilo principal o hilo de interfaz de usuario, UI Thread.

El UI Thread Se encarga de atender los eventos de los distintos componentes (ejecuta los métodos onCreate(), onDraw(), onKeyDown(), …)

Todas las actividades y servicios de una aplicación son ejecutados por el UI Thread.

Cuando tu aplicación ha de realizar trabajo intensivo como respuesta a una interacción de usuario no debería ejecutarla en el UI Thread, puesto que bloqueará su ejecución (no responderá a acciones del usuario) hasta que termine el trabajo y el usuario pensará que la aplicación se ha colgado.

Si el UI Thread se bloquea por más de 5 segundos el sistema mostrará un cuadro de dialogo al usuario "La aplicación no responde" para que el usuario decida si quiere esperar o detener la aplicación.

Threads

La solución es crear un nuevo hilo de ejecución, para que realice este trabajo intensivo.

Las herramientas del interfaz de usuario de Android han sido diseñadas para ser ejecutadas desde un único hilo de ejecución, el UI Thread.

No se permite manipular el interfaz de usuario desde otros hilos de ejecución.

Descargar, Analizar y Ejecutar:
ThreadingNoThreading.rar

Threads

Un Thread en Java se crea asi:

```
Thread miThread = new Thread();
```

Para lanzar la ejecución de un Thread se llama a su método .start()

```
miThread.start();
```

En este ejemplo no hemos especificado el código que tenemos que ejecutar en el nuevo Thread. Terminará nada más comenzar pues no ejecuta nada.

Hay dos formas de especificar el código que debe ejecutar un Thread.

- Crear una subclase de Thread y sobreescribir su método run().
- Pasar en el constructor del Thread un objeto que implemente la interfaz Runanble.

Thread Subclass

El método run() es lo que se ejecuta tras llamar al método start() del Thread.

```
public class MyThread extends Thread {
    public void run(){
       System.out.println("MyThread running");
    }
  }
```

Para crear y lanzar el thread:

```
MyThread myThread = new MyThread();
myTread.start();
```

El método start() termina tran pronto como el thread arranca, no espera a que el método run() termine.

Se puede crear una subclase anónima de un thread asi:

```
Thread thread = new Thread(){
    public void run(){
      System.out.println("Thread Running");
    }
  }
  thread.start();
```

Thread Runnable

Un Runnable es un trozo de código que implementa el interfaz Runnable, por lo que tiene un método .run()

Tenemos que @overridde el metodo .run() de nuestro objeto Runnable

```
public class MyRunnable implements Runnable {
    public void run(){
       System.out.println("MyRunnable running");
    }
  }
```

Para que el método run() del Runnable sea ejecutado por un Thread hay que pasarlo en el constructor del Thread.

```
Thread thread = new Thread(new MyRunnable());
thread.start();
```

Usando una implementación anónima de un Runnable:

```
Runnable myRunnable = new Runnable(){
     public void run(){
        System.out.println("Runnable running");
     }
   }
   Thread thread = new Thread(myRunnable);
   thread.start();
```

Thread Names

Se pude dar un nombre a un Thread para distinguirlo de otros.

```
Thread thread = new Thread("New Thread") {
      public void run(){
        System.out.println("run by: " + getname());
      }
   };
   thread.start();
   System.out.println(thread.getName());
```

El nombre de un Thread se puede obtener con el método getName() de la clase Thread.

También se puede pasar el nombre cuando se pasa un runnable:

```
MyRunnable runnable = new MyRunnable();
Thread thread = new Thread(runnable, "New Thread");
thread.start();
System.out.println(thread.getName());
```

Pero como la clase MyRunnable() no es subclase de Thread no tiene acceso a getName()

Se puede obtener una referencia al thread actual con

```
Thread.currentThread()
```

y así ejecutar en el runnable:

```
String threadName = Thread.currentThread().getName();
```

E0902-ThreadingSimple (A y B)

Threading UI Thread Updates

- El trabajo se realiza en un Thread
- Las actualizaciones de interfaz se realizan en el UI Thread

Android proporciona varios métodos que garantiza que son ejecutados en el UI Thread.

Dos de los métodos que tenemos para actualizar el UI Thread son los siguientes:

- En la clase View el método post

```
boolean  View.post(Runnable action)
```

- En la clase Activity el método

```
void Activity.runOnUiThread(Runnable action)
```

- En la clase Activity el método

```
public boolean postDelayed (Runnable action, long delayMillis)
```

Como vemos ambos reciben un Runnable.

E0902-ThreadingSimple C