Arquitecturas cliente/servidor

Creación de Sockets Cliente Servidor

Creación de Sockets Cliente/Servidor

- ☑ Sockets en TCP
- Concepto de Hilos
- Definición de DAEMON
- Sockets en UDP

THREADS

¿Qué es un thread?

- Un thread es un flujo secuencial de control dentro de un programa.

- Un thread (hilo o contexto de ejecución) es una secuencia de instrucciones en ejecución.

Threads

- El concepto de thread es similar al de proceso, excepto que múltiples thread pueden ejecutarse dentro del mismo proceso, compartiendo los datos y el código de programa.

- La mayoría de los programas se pueden considerar como de thread único, es decir, existe un único camino de ejecución, el programa tiene un inicio, se realizan una serie de cálculos y se finaliza.

Threads

- Un thread es un proceso ligero (más rápido y económico)

- Generalmente un proceso (padre) genera diversos threads (procesos-hijo), formando así un árbol de procesos

- Los programas multihilo pueden tener varios flujos de control en diferentes partes del código ejecutándose "simultáneamente".

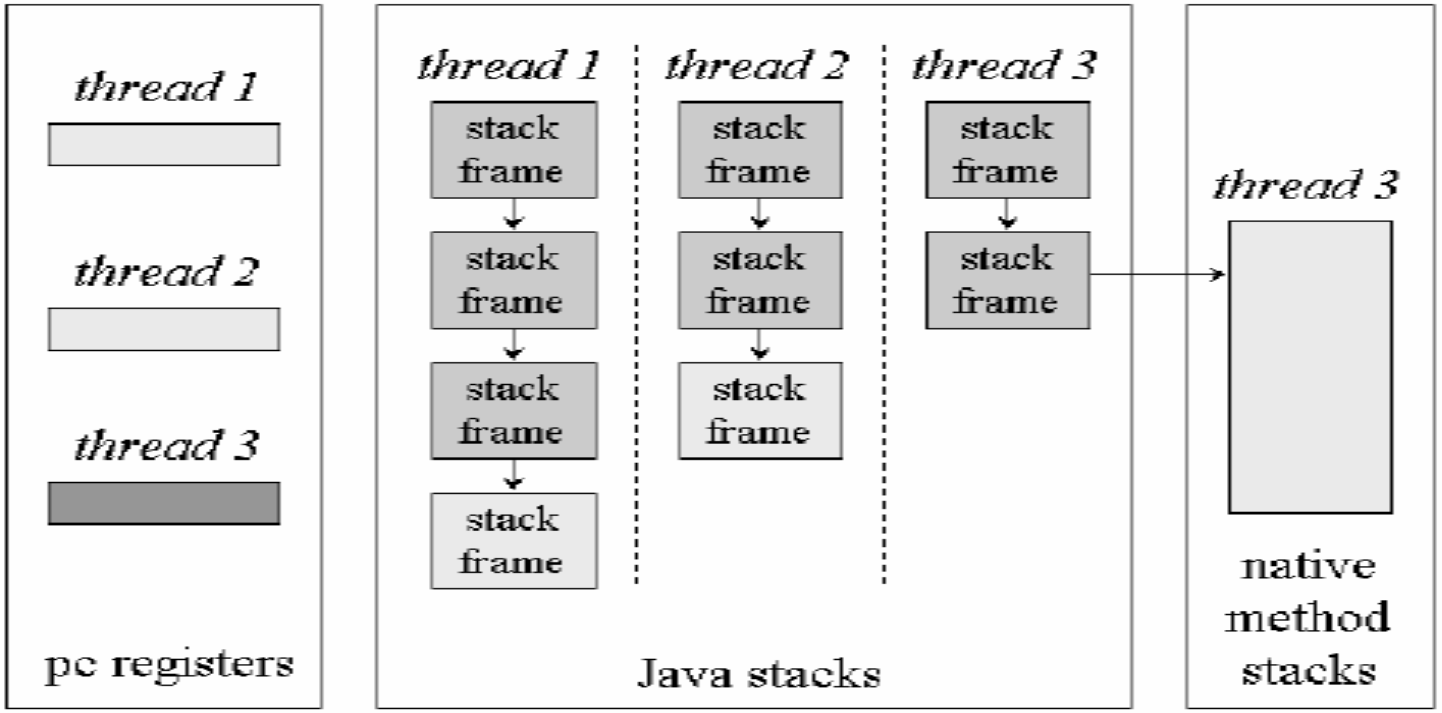
Cambio de contexto

- La alternación de threads que tiene que hacer el sistema operativo es escaso.

- Entendiendo como tal el cambio que deberá realizar el sistema para colocar un proceso en ejecución y colocar el que se está ejecutando en espera.

- Este mecanismo se denomina cambio de contexto.

Características de los threads



Creación de un Thread

- Para crear un nuevo thread es necesario generar una instancia de la clase java.lang.Thread.

- Un objeto Thread representa un verdadero hilo en ejecución para el intérprete de Java y es utilizado para el control y sincronización durante su aplicación.

- Mediante Thread, podemos dormir el hilo, ejecutarlo, interrumpirlo, etc.

Implementación

- Hay dos alternativas posibles :
  - Declarar el hilo con una clase extendida de la clase java.lang.Thread, mediante la herencia.
  - Implementar la interfaz java.lang.Runnable.
- La implementación de la interfaz java.lang.Runnable es la forma habitual de crear threads, ya que proporciona una forma de abstracción a la hora de definir una clase.

Interface Runnable

- Un objeto que desee utilizar los métodos que proporciona la clase **java.lang.Thread** se puede declarar mediante la implementación de la interfaz **java.lang.Runnable**, que simplemente se define de la siguiente manera descrita:

```
public interface Runnable {
  abstract public void run();
}
```

El método run( )

- El inicio de cada hilo quedara definido mediante la ejecución del método **run**() para cada objeto creado, que contendrá el código que se va a ejecutar.

- run() es un método público, no acepta parámetros y no lanza ninguna excepción.

- Cada objeto de esta clase sería **Runnable** y por tanto se podrá utilizar con la clase **java.lang.Thread**.

Implementación de la interfaz Runnable

- Como se ha descrito, la clase debe implementar la interfaz **Runnable**, por lo que para empezar,la clase debe tener la siguiente definición:

```
public class miThread implements Runnable{

}
```

Funcionamiento run( ) y start( )

- Es posible indicar que comience la ejecución del hilo de ejecución mediante el uso de los métodos que proporciona la clase java.lang.Thread.

- El método start() es el encargado de llamar al método run().

- El método start() de java.lang.Thread() arranca el objeto tipo Thread como tal.

Ejemplo:

- Solo queda utilizar la clase como un verdadero objeto tipo Thread.

- Por lo tanto, se genera un objeto tipo java.lang.Thread y se pasa un objeto del tipo de la clase generada:

```
public class Mensaje implements Runnable{
   public void run(){
            System.out.println("Hola mundo");
   }
   public static void main(String args[]){
            Mensaje miHilo = new Mensaje();
            new Thread(miHilo).start();
   }
}
```

Herencia

- El otro método para la creación de hilos consiste en heredar un objeto de la clase java.lang.Thread.

- Se trata de una forma más sencilla que la anterior, ya que la propia clase java.lang.Thread contiene el método run() que ejecuta el código.

- Para que la clase pueda ser ejecutada como un hilo, se hereda de la clase java.lang.Thread y se escribe el código dentro del procedimiento run().

Sintaxis

- La clase java.lang.Thread redefine el método run()

```
class miHilo extends Thread {
    public void run () {
    // Código a ejecutar
    }
}
```

Creación y ejecución

- La creación del hilo y su ejecución sería de la forma siguiente:

```
miHilo mio = new miHilo();
mio.start();
```

- Como se observa, la ejecución también se realiza con la llamada al método start()

Observaciones (Herencia)

La implementación de los threads mediante este método es mucho más fácil de utilizar.

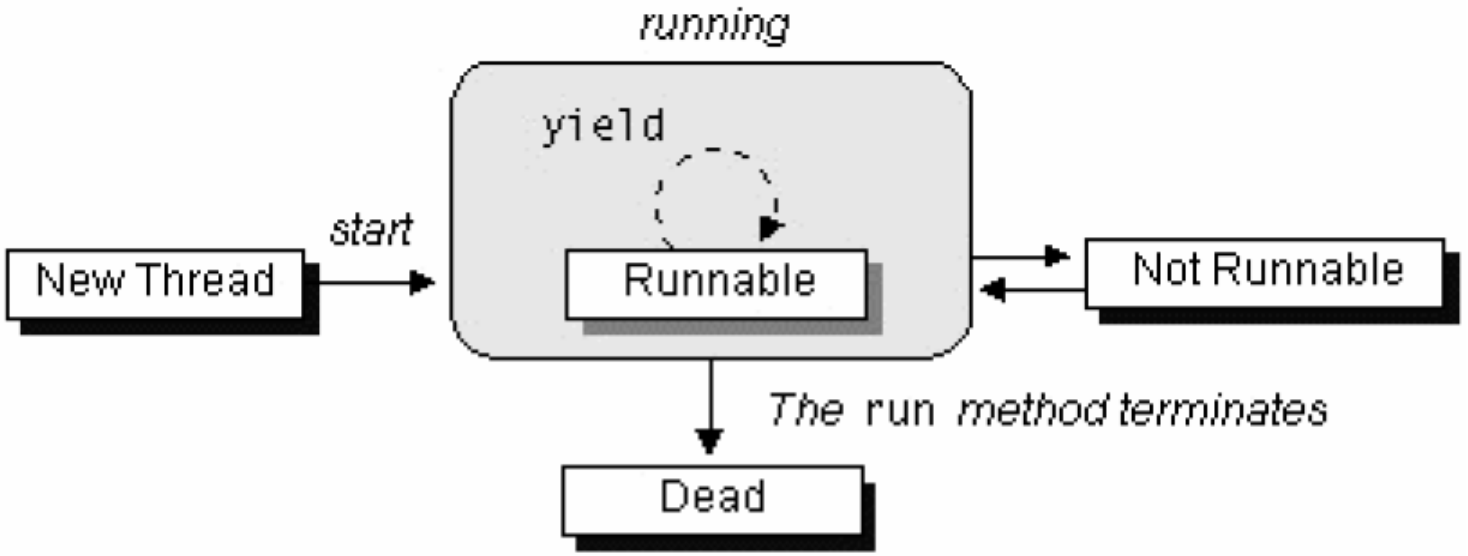
Si se quiere crear un thread que

herede de otra clase que no se java.lang.Thread, como por ejemplo Applet, no se podrá realizar.

Cabe señalar que la clase Thread en sí implementa a la interfaz Runnable, hay un par de constructores de la clase Thread que reciben objetos tipo Runnable

Estados de un thread

Durante el ciclo de vida de un thread se pueden encontrar diferentes estados.



Algunos métodos de la clase Thread

- void **start**(): usado para iniciar el cuerpo de la thread definido por el método run().
- void **sleep(tiempo**): pone a dormir una thread por un tiempo mínimo especificado.
- void **join**(): usado para esperar por el término de la thread, por ejemplo por término de método run().
- void **yield**(): Mueve el thread desde el estado de en ejecución al final de la cola de procesos en espera por la CPU, en otras palabras cede el control del microprocesador a otro hilo.
- boolean **isAlive**(): Indica si el thread esta vivo o no.

Distribución del procesador

- Por defecto los hilos con prioridad igual se ejecutan con el sistema round robin.

- Si un thread comienza a ejecutarse, lo seguirá haciendo hasta que ocurra alguna de las siguientes circunstancias:

  - Se llame al método sleep() o wait()
  - Se espere una entrada/salida
  - Si se ejecuta un método sincronizado.
  - Se llame al método yield()

Prioridades de los Threads

- Java garantiza de cierta manera la gestión de los threads en cuanto al nivel de prioridad asignado a cada uno de ellos, de tal forma que cada hilo tiene una prioridad asociada.

- Si un hilo con mayor prioridad pasa a activo, que el hilo que se esta ejecutando, éste pasa a dormido.

Manejo de Prioridades

- La prioridad por defecto de un thread está definida por la variable estática Thread.NORM_PRIORITY = 5

- Existen otras dos variables estáticas definidas en el paquete java.lang.Thread

  - public static final int MAX_PRIORITY 10
  - public static final int MIN_PRIORITY 1
  - public static final int NORM_PRIORITY 5

Ejemplo

```
class MiHilo extends Thread {
      String mensaje;
      MiHilo( String cadena) {
        mensaje = cadena ;
      }
      public void run() {
        while(true) {
              System.out.println( mensaje );
        }
      }
      public static void main( String args[] ) {
        new MiHilo("K1 ").start();
        new MiHilo("K2 ").start();
        MiHilo obj2 = new MiHilo("L ");
        obj2.setPriority( Thread.NORM_PRIORITY+1);
        obj2.start();
      }
}
```

Salida: K1 K1 K1 K1 K1 K2 L K1 K2 L K1 K2 L K1 K2 L L

Definición de DAEMON

- Es un tipo especial de proceso que se ejecuta de forma:
  - Continua (infinitamente) .
  - Transparente para el usuario.
  - Simultánea con otros procesos, por eso suelen encontrarse en sistemas multitarea.

Además…

- No disponen de una interfaz directa con un humano, ya sea gráfica o textual.
- No hacen uso de la entradas y salidas estándar para comunicar errores o registrar su funcionamiento.
- Por el contrario usan archivos del sistema en zonas especiales (/var/log/) o utilizan otros demonios especializados en dicho registro como el syslogd

Por ejemplo:

- httpd
- demonios "cronológicos" para mantenimiento del sistema en segundo plano.
- proceso de sistema de protección en tiempo real de antivirus .
- proceso de protección de firewall de capa de red/aplicación.

Programas Demonios

- Un thread demonio (daemon) es un thread de baja prioridad que consta de un bucle infinito y se suele utilizar para prestar servicios cuando se necesite, por ejemplo, refresco de memoria, mostrar imágenes, etc. Java utiliza un thread daemon, el recolector de basura para realizar todas las gestiones sobre la creación y destrucción de los objetos en memoria.

- Un thread demonio se declara de la siguiente manera:
  - **thread.setDaemon**();
- Y se puede comprobar si un thread es tipo "demonio" mediante el siguiente método:
  - **isDaemon();**

Observaciones sobre threads

La diferencia entre los **threads** normales y los demonios es que Java no espera la muerte de los demonios para detener su ejecución

Otra característica consiste en que cualquier **thread** que se crea desde otro que es demonio también es un **thread** demonio.