

ACTIVIDAD 1.7

Modifica el *Ejemplo5.java* para que la salida del proceso y la salida de error se almacenen en un fichero de texto, y la entrada la tome desde otro fichero de texto.

Para llevar a cabo el redireccionamiento, tanto de entrada como de salida del proceso que se ejecuta, también podemos usar la clase **ProcessBuilder.Redirect**. El redireccionamiento puede ser uno de los siguientes:

- El valor especial **Redirect.INHERIT**, indica que la fuente de entrada y salida del proceso será la misma que la del proceso actual.
- **Redirect.from (File)**, indica redirección para leer de un fichero, la entrada al proceso se encuentra en el objeto **File**.
- **Redirect.to(File)**, indica redirección para escribir en un fichero, el proceso escribirá en el objeto **File** especificado.
- **Redirect.appendTo (File)**, indica redirección para añadir a un fichero, la salida del proceso se añadirá al objeto **File** especificado.

El ejemplo anterior usando esta clase quedaría de esta manera:

```
pb.redirectInput(ProcessBuilder.Redirect.from(fBat));
pb.redirectOutput(ProcessBuilder.Redirect.to(fOut));
pb.redirectError(ProcessBuilder.Redirect.to(fErr));
```

El siguiente ejemplo muestra en la consola la salida del comando DIR,

```
import java.io.IOException;
public class Ejemplo9 {
    public static void main(String args[]) throws IOException {
        ProcessBuilder pb = new ProcessBuilder("CMD", "/C", "DIR");
        pb.redirectOutput(ProcessBuilder.Redirect.INHERIT);
        Process p = pb.start();
    }
} // Ejemplo9
```

ACTIVIDAD 1.8

Usando **ProcessBuilder.Redirect**, modifica el *Ejemplo5.java* para que la salida del proceso se muestre en la consola, la entrada la tome desde un fichero de texto, y la salida la lleve a un fichero de texto. Realiza los ejercicios 7, 8 y 9.

1.3. PROGRAMACIÓN CONCURRENTE

El diccionario *WordReference.com* (<http://www.wordreference.com/definicion/>) nos muestra varias acepciones de la palabra concurrencia. Nos quedamos con la tercera: “*Acaecimiento o concurso de varios sucesos en un mismo tiempo*”. Si sustituimos sucesos por procesos ya tenemos una aproximación de lo que es la concurrencia en informática: la existencia simultánea de varios procesos en ejecución.

1.3.1. PROGRAMA Y PROCESO

Al principio del tema se definió un **proceso** como un programa en ejecución. Y ¿qué es un **programa**? podemos definir **programa** como un conjunto de instrucciones que se aplican a un conjunto de datos de entrada para obtener una salida. Un proceso es algo activo que cuenta con una serie de recursos asociados, en cambio un programa es algo pasivo, para que pueda hacer algo hay que ejecutarlo.

Pero un programa al ponerse en ejecución puede dar lugar a más de un proceso, cada uno ejecutando una parte del programa. Por ejemplo, el navegador web, por un lado está controlando las acciones del usuario con la interfaz, por otro hace las peticiones al servidor web. Entonces cada vez que se ejecuta este programa crea 2 procesos.

En la Figura 1.12 existe un programa almacenado en disco y 3 instancias del mismo ejecutándose, por ejemplo, por 3 usuarios diferentes. Cada instancia del programa es un proceso, por tanto, existen 3 procesos independientes ejecutándose al mismo tiempo sobre el sistema operativo, tenemos entonces 3 procesos concurrentes.

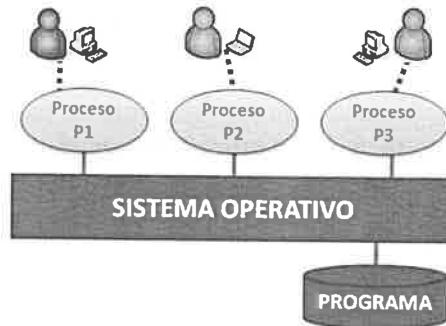


Figura 1.12. Un programa con 3 instancias ejecutándose.

Dos procesos serán concurrentes cuando la primera instrucción de uno de ellos se ejecuta después de la primera instrucción del otro y antes de la última. Es decir, existe un solapamiento o intercalado en la ejecución de sus instrucciones. No hay que confundir el solapamiento con la ejecución simultánea de las instrucciones, en este caso estaríamos en una situación de **programación paralela**, aunque a veces el hardware subyacente (más de un procesador) sí permitirá la ejecución simultánea.

Supongamos ahora que el programa anterior al ejecutarse da lugar a 2 procesos más, cada uno ejecutando una parte del programa, entonces la Figura 1.12 se convierte en la 1.13. Ya que un programa puede estar compuesto por diversos procesos, una definición más acertada de proceso es la de una actividad asíncrona susceptible de ser asignada a un procesador¹.



Figura 1.13. Un programa dando lugar a más de un proceso.

Cuando varios procesos se ejecutan concurrentemente puede haber procesos que colaboren para un determinado fin (por ejemplo, P1.1 y P1.2), y otros que compitan por los recursos del sistema (por ejemplo P2.1 y P3.1). Estas tareas de colaboración y competencia por los recursos exigen **mecanismos de comunicación y sincronización entre procesos**.

¹ Programación concurrente. José Tomás Palma Méndez y otros. Ed Paraninfo. ISBN: 9788497321846

1.3.2. CARACTERÍSTICAS

La programación concurrente es la disciplina que se encarga del estudio de las notaciones que permiten especificar la ejecución concurrente de las acciones de un programa, así como las técnicas para resolver los problemas inherentes a la ejecución concurrente (comunicación y sincronización).

BENEFICIOS

La programación concurrente aporta una serie de beneficios:

Mejor aprovechamiento de la CPU. Un proceso puede aprovechar ciclos de CPU mientras otro realiza una operación de entrada/salida.

Velocidad de ejecución. Al subdividir un programa en procesos, éstos se pueden “repartir” entre procesadores o gestionar en un único procesador según importancia.

Solución a problemas de naturaleza concurrente. Existen algunos problemas cuya solución se basa en la utilización de la metodología:

- Sistemas de control: son sistemas en los que hay captura de datos, normalmente a través de sensores, análisis y actuación en función del análisis. Un ejemplo son los sistemas de tiempo real.
 - Tecnologías web: los servidores web son capaces de atender múltiples peticiones de usuarios concurrentemente, también los servidores de chat, correo, los propios navegadores web, etc.
 - Aplicaciones basadas en GUI: el usuario puede interactuar con la aplicación mientras la aplicación está realizando otra tarea. Por ejemplo, el navegador web puede estar descargando un archivo mientras el usuario navega por las páginas.
 - Simulación: programas que modelan sistemas físicos con autonomía.
 - Sistemas Gestores de Bases de Datos: Los usuarios interactúan con el sistema, cada usuario puede ser visto como un proceso.

CONCURRENCIA Y HARDWARE

En un sistema **monoprocesador** (de un solo procesador) se puede tener una ejecución concurrente gestionando el tiempo de procesador para cada proceso. El S.O. va alternando el tiempo entre los distintos procesos, cuando uno necesita realizar una operación de entrada salida, lo abandona y otro lo ocupa; de esta forma se aprovechan los ciclos del procesador. En la Figura 1.14 se muestra como el tiempo de procesador es repartido entre 3 procesos, en cada momento sólo hay un proceso. Esta forma de gestionar los procesos en un sistema monoprocesador recibe el nombre de **multiprogramación**.

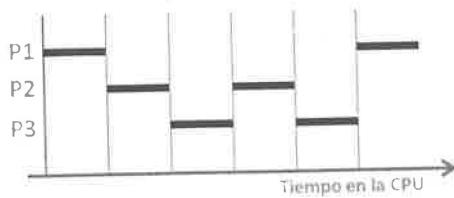


Figura 1.14. Concurrencia.

En un sistema **monoprocesador** todos los procesos comparten la misma memoria. La forma de comunicar y sincronizar procesos se realiza mediante variables compartidas.

En un sistema **multiprocesador** (existe más de un procesador) podemos tener un proceso en cada procesador. Esto permite que exista paralelismo real entre los procesos, véase Figura 1.15.

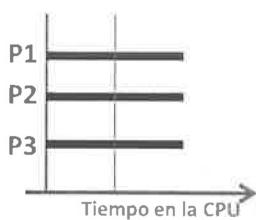


Figura 1.15. Paralelismo.

Estos sistemas se pueden clasificar en:

- Fuertemente acoplados: cuando poseen una memoria compartida por todos los procesadores, véase Figura 1.16.
- Débilmente acoplados: cuando los procesadores poseen memorias locales y no existe la compartición de memoria, véase Figura 1.17.

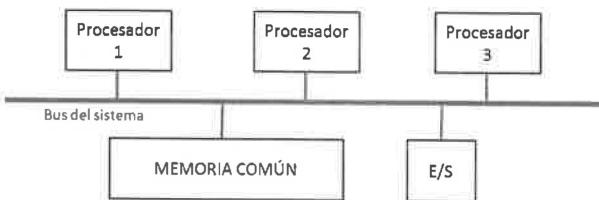


Figura 1.16. Fuertemente acoplados.

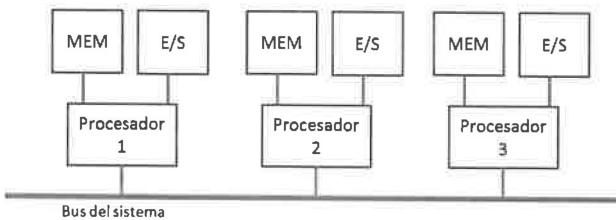


Figura 1.17. Débilmente acoplados.

Se denomina **multiproceso** a la gestión de varios procesos dentro de un sistema multiprocesador, donde cada procesador puede acceder a una memoria común.

1.3.3. PROGRAMAS CONCURRENTES

Un **programa concurrente** define un conjunto de acciones que pueden ser ejecutadas simultáneamente. Supongamos que tenemos estas dos instrucciones en un programa, está claro que el orden de la ejecución de las mismas influirá en el resultado final:

<code>x=x+1;</code>	<code>y=x+1;</code>	La primera instrucción se debe ejecutar antes de la segunda.
---------------------	---------------------	--

En cambio, si tenemos estas otras, el orden de ejecución es indiferente:

<code>x=1;</code>	<code>y=2;</code>	El orden no interviene en el resultado final.
	<code>z=3;</code>	

CONDICIONES DE BERNSTEIN

Bernstein definió unas condiciones para que dos conjuntos de instrucciones se puedan ejecutar concurrentemente. En primer lugar es necesario formar 2 conjuntos de instrucciones:

- **Conjunto de lectura:** formado por instrucciones que cuentan con variables a las que se accede en modo lectura durante su ejecución.
- **Conjunto de escritura:** formado por instrucciones que cuenta con variables a las que se accede en modo escritura durante su ejecución.

Por ejemplo, sean las siguientes instrucciones:

Instrucción 1:	$x := y + 1$
Instrucción 2:	$y := x + 2$
Instrucción 3:	$z := a + b$

Los conjuntos de lectura y escritura estarían formados por las variables siguientes:

	Conjunto lectura - L	Conjunto escritura - E
Instrucción 1- I1:	y	x
Instrucción 2- I2:	x	y
Instrucción 3- I3:	a,b	z

Se pueden expresar de la siguiente manera:

$L(I1) = \{y\}$	$E(I1) = \{x\}$
$L(I2) = \{x\}$	$E(I2) = \{y\}$
$L(I3) = \{a,b\}$	$E(I3) = \{z\}$

Para que dos conjuntos se puedan ejecutar concurrentemente se deben cumplir estas 3 condiciones:

- La intersección entre las variables leídas por un conjunto de instrucciones Ii y las variables escritas por otro conjunto Ij debe ser vacío, es decir, no debe haber variables comunes:

$$L(Ii) \cap E(Ij) = \emptyset$$

- La intersección entre las variables de escritura de un conjunto de instrucciones Ii y las variables leídas por otro conjunto Ij debe ser nulo, es decir, no debe haber variables comunes:

$$E(Ii) \cap L(Ij) = \emptyset$$

- Por último, la intersección entre las variables de escritura de un conjunto de instrucciones Ii y las variables de escritura de un conjunto Ij debe ser vacío, no debe haber variables comunes:

$$E(Ii) \cap E(Ij) = \emptyset$$

En el ejemplo anterior tenemos las siguientes condiciones, donde se observa que las instrucciones I1 e I2 no se pueden ejecutar concurrentemente porque no cumplen las 3 condiciones:

Conjunto I1 e I2	Conjunto I2 e I3	Conjunto I1 e I3
$L(I1) \cap E(I2) \neq \emptyset$	$L(I2) \cap E(I3) = \emptyset$	$L(I1) \cap E(I3) = \emptyset$
$E(I1) \cap L(I2) \neq \emptyset$	$E(I2) \cap L(I3) = \emptyset$	$E(I1) \cap L(I3) = \emptyset$
$E(I1) \cap E(I2) = \emptyset$	$E(I2) \cap E(I3) = \emptyset$	$E(I1) \cap E(I3) = \emptyset$

En los programas secuenciales hay un orden fijo de ejecución de las instrucciones, siempre se sabe por dónde va a ir el programa. En cambio, en los programas concurrentes hay un orden parcial. Al haber solapamiento de instrucciones no se sabe cuál va a ser el orden de ejecución, puede ocurrir que ante unos mismos datos de entrada el flujo de ejecución no sea el mismo. Esto da lugar a que los programas concurrentes tengan un comportamiento indeterminista donde repetidas ejecuciones sobre un mismo conjunto de datos puedan dar diferentes resultados.

1.3.4. PROBLEMAS INHERENTES A LA PROGRAMACIÓN CONCURRENTE

A la hora de crear un programa concurrente podemos encontrarnos con dos problemas:

- **Exclusión mutua.** En programación concurrente es muy típico que varios procesos accedan a la vez a una variable compartida para actualizarla. Esto se debe evitar, ya que puede producir inconsistencia de datos: uno puede estar actualizando la variable a la vez que otro la puede estar leyendo. Por ello es necesario conseguir la exclusión mutua de los procesos respecto a la variable compartida. Para ello se propuso la **región crítica**. Cuando dos o más procesos comparten una variable, el acceso a dicha variable debe efectuarse siempre dentro de la región crítica asociada a la variable. Sólo uno de los procesos podrá acceder para actualizarla y los demás deberán esperar, el tiempo de estancia es finito.
- **Condición de sincronización.** Hace referencia a la necesidad de coordinar los procesos con el fin de sincronizar sus actividades. Puede ocurrir que un proceso P1 llegue a un estado X que no pueda continuar su ejecución hasta que otro proceso P2 haya llegado a un estado Y de su ejecución. La programación concurrente proporciona mecanismos para bloquear procesos a la espera de que ocurra un evento y para desbloquearlos cuando este ocurra.

Algunas herramientas para manejar la concurrencia son: la región crítica, los semáforos, región crítica condicional, buzones, sucesos, monitores y sincronización por rendez-vous.

ACTIVIDAD 1.9

Responde a las siguientes cuestiones:

Escribe alguna característica de un programa concurrente.

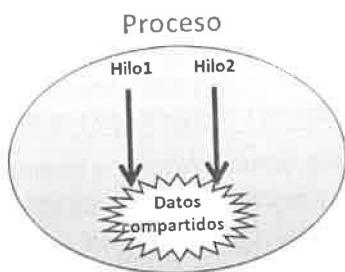
¿Cuál es la ventaja de la concurrencia en los sistemas monoprocesador?

¿Cuáles son las diferencias entre multiprogramación y multiproceso?

¿Cuáles son los dos problemas principales inherentes a la programación concurrente?

1.3.5. PROGRAMACIÓN CONCURRENTE CON JAVA

Al igual que el sistema operativo puede ejecutar varios procesos concurrentemente, dentro de un proceso podemos encontrarnos con varios hilos de ejecución. Un hilo es como una secuencia de control dentro de un proceso que ejecuta sus instrucciones de forma independiente, véase Figura 1.18. Los hilos comparten el contexto del proceso, pero cada hilo mantiene una parte local.

**Figura 1.18.** Hilos en un proceso.

Entre procesos e hilos hay algunas diferencias:

- Los hilos comparten el espacio de memoria del proceso, muchos comparten datos y espacios de direcciones; a diferencia de los procesos que generalmente poseen espacios de memoria de trabajo independientes e interactúan a través de mecanismos de comunicación dados por el sistema.
- Hilos y procesos pueden encontrarse en diferentes estados, pero los cambios de estado en los procesos son más costosos ya que los hilos pertenecen al mismo proceso. A los hilos también se les llama procesos ligeros.
- Se tarda menos tiempo en crear o en terminar un hilo que un proceso.
- En la comunicación entre procesos debe intervenir el núcleo del sistema, entre hilos no se necesita que intervenga el núcleo.

Para programar concurrentemente podemos dividir nuestro programa en hilos. Java proporciona la construcción de programas concurrentes mediante la clase **Thread** (hilo o hebra). Esta clase permite ejecutar código en un hilo de ejecución independiente.

En Java existen dos formas de utilizar o crear un hilo:

- Creando una clase que herede de la clase **Thread** y sobrecargando el método **run()**.
- Implementando la interface **Runnable**, y declarando el método **run()**. Se utiliza este modo cuando una clase ya deriva de otra. Por ejemplo, un applet deriva de la clase **Applet** por lo que no puede derivar también de **Thread**, en este caso tiene que implementar la interface **Runnable**.

El siguiente ejemplo crea un hilo de nombre *HiloSimple* heredando de la clase **Thread**. En el método **run()** se indican las líneas de código que se ejecutarán simultáneamente con las otras partes del programa. Cuando se termina la ejecución de ese método, el hilo de ejecución termina también:

```
public class HiloSimple extends Thread {
    public void run() {
        for (int i=0; i<5;i++)
            System.out.println("En el Hilo... " );
    }
}
```

Para usar el hilo creo la clase *UsaHilo*:

```
public class UsaHilo {
    public static void main(String[] args) {
        HiloSimple hs = new HiloSimple();
        hs.start();
        for (int i=0; i<5;i++)
```

```

        System.out.println("Fuera del hilo..");
    }
} //
```

Desde esta clase se arranca el hilo: primero se invoca al operador *new* para crear el hilo y luego al método *start()* que invoca al método *run()*. La compilación y ejecución muestra la siguiente salida, en la que se puede observar que se intercala las instrucciones del hilo y de fuera del hilo. La salida puede variar cada vez que ejecutemos el programa:

```

D:\CAPIT1>javac HiloSimple.java
D:\CAPIT1>javac UsaHilo.java
D:\CAPIT1>java UsaHilo
Fuera del hilo..
Fuera del hilo..
Fuera del hilo..
En el Hilo...
Fuera del hilo..
Fuera del hilo..
```

Las 2 clases anteriores implementando la interfaz **Runnable** quedarían así:

```

public class HiloSimple2 implements Runnable{
    public void run() {
        for (int i=0; i<5;i++)
            System.out.println("En el Hilo...");
    }
} //
```



```

public class UsaHilo2 {
    public static void main(String[] args) {
        HiloSimple2 hs = new HiloSimple2();
        Thread t = new Thread(hs);
        t.start();
        for (int i=0; i<5;i++)
            System.out.println("Fuera del hilo..");
    }
} //
```

En el siguiente capítulo se tratarán más ampliamente los hilos con Java.

1.4. PROGRAMACIÓN PARALELA Y DISTRIBUIDA

1.4.1. PROGRAMACIÓN PARALELA

Un **programa paralelo** es un tipo de programa concurrente diseñado para ejecutarse en un sistema multiprocesador. El procesamiento paralelo permite que muchos elementos de proceso independientes trabajen simultáneamente para resolver un problema. Estos elementos pueden ser un número arbitrario de equipos conectados por una red, un único equipo con varios procesadores o una combinación de ambos. El problema a resolver se divide en partes independientes de tal forma que cada elemento pueda ejecutar la parte de programa que le corresponda a la vez que los demás.

Recordemos que en un sistema **multiprocesador**, donde existe más de un procesador, podemos tener un proceso en cada procesador y todos juntos trabajan para resolver un problema. Cada procesador realiza una parte del problema y necesita intercambiar información con el resto. Según cómo se realice este intercambio podemos tener modelos distintos de programación paralela:

- **Modelo de memoria compartida:** los procesadores comparten físicamente la memoria, es decir, todos acceden al mismo espacio de direcciones. Un valor escrito en memoria por un procesador puede ser leído directamente por cualquier otro.
- **Modelo de paso de mensajes:** cada procesador dispone de su propia memoria independiente del resto y accesible sólo por él. Para realizar el intercambio de información es necesario que cada procesador realice la petición de datos al procesador que los tiene, y éste haga el envío. El entorno de programación PVM que veremos más adelante utiliza este modelo.

El intercambio de información entre procesadores depende del sistema de almacenamiento que se disponga. Según este criterio las arquitecturas paralelas se clasifican en: **Sistemas de memoria compartida o multiprocesadores:** los procesadores comparten físicamente la memoria; y **Sistemas de memoria distribuida o multicamputadores:** cada procesador dispone de su propia memoria.

Dentro de los sistemas de memoria distribuida o multicamputadores nos encontramos con los **Clusters**. Son sistemas de procesamiento paralelo y distribuido donde se utilizan múltiples ordenadores, cada uno con su propio procesador, enlazados por una red de interconexión más o menos rápida, de tal forma que el conjunto de ordenadores es visto como un único ordenador, más potente que los comunes de escritorio.

Tradicionalmente, el paralelismo se ha utilizado en centros de supercomputación para resolver problemas de elevado coste computacional en un tiempo razonable, pero en la última década su interés se ha extendido por la difusión de los procesadores con múltiples núcleos (combina dos o más procesadores independientes en un solo circuito integrado). Estos procesadores permiten que un dispositivo computacional exhiba una cierta forma del paralelismo a nivel de thread (thread-level parallelism) (TLP) sin incluir múltiples microprocesadores en paquetes físicos separados. Esta forma de TLP se conoce a menudo como multiprocesamiento a nivel de chip (chip-level multiprocessing) o CMP².

VENTAJAS E INCONVENIENTES

Ventajas del procesamiento paralelo:

- Proporciona ejecución simultánea de tareas.
- Disminuye el tiempo total de ejecución de una aplicación.
- Resolución de problemas complejos y de grandes dimensiones.
- Utilización de recursos no locales, por ejemplo, los recursos que están en una red distribuida, una WAN o la propia red internet.
- Disminución de costos, en vez de gastar en un supercomputador muy caro se pueden utilizar otros recursos más baratos disponibles remotamente.

Pero no todo son ventajas, algunos inconvenientes son:

² https://es.wikipedia.org/wiki/Procesador_multinúcleo

- Los compiladores y entornos de programación para sistemas paralelos son más difíciles de desarrollar.
- Los programas paralelos son más difíciles de escribir.
- El consumo de energía de los elementos que forman el sistema.
- Mayor complejidad en el acceso a los datos.
- La comunicación y la sincronización entre las diferentes subtareas.

La computación paralela resuelve problemas como: predicciones y estudios meteorológicos, estudio del genoma humano, modelado de la biosfera, predicciones sísmicas, simulación de moléculas... En algunos casos se dispone de tal cantidad de datos que serían muy lento o imposible tratar con máquinas convencionales.

ACTIVIDAD 1.10

Entra en la siguiente URL https://computing.llnl.gov/tutorials/parallel_comp/ y responde a las siguientes cuestiones:

Cita algunas características de la computación serie.

Cita algunas características de la computación en paralelo.

Ámbitos en los que se usa la computación en paralelo.

¿Cómo hace uso de la computación paralela el proyecto SETI @ home?

1.4.2. PROGRAMACIÓN DISTRIBUIDA

Uno de los motivos principales para construir un sistema distribuido es compartir recursos. Probablemente, el sistema distribuido más conocido por todos es Internet que permite a los usuarios donde quiera que estén hacer uso de la World Wide Web, el correo electrónico y la transferencia de ficheros. Entre las aplicaciones más recientes de la computación distribuida se encuentra el *Cloud Computing* que es la computación en la nube o servicios en la nube, que ofrece servicios de computación a través de Internet.

Se define un sistema distribuido como aquel en el que los componentes hardware o software, localizados en computadores unidos mediante una red, comunican y coordinan sus acciones mediante el paso de mensajes. Esta definición tiene las siguientes consecuencias³:

- **Concurrencia:** lo normal en una red de ordenadores es la ejecución de programas concurrentes.
- **Inexistencia de reloj global:** cuando los programas necesitan cooperar coordinan sus acciones mediante el paso de mensajes. No hay una temporalización, los relojes de los host no están sincronizados.
- **Fallos independientes:** cada componente del sistema puede fallar independientemente, permitiendo que los demás continúen su ejecución.

La programación distribuida es un paradigma de programación enfocado en desarrollar sistemas distribuidos, abiertos, escalables, transparentes y tolerantes a fallos. Este paradigma es el resultado natural del uso de las computadoras y las redes. Casi cualquier lenguaje de

³ Sistemas Distribuidos: Conceptos y Diseño. George Coulouris y otros. Ed: Addison-Wesley.

programación que tenga acceso al máximo al hardware del sistema puede manejar la programación distribuida, considerando una buena cantidad de tiempo y código⁴.

Una arquitectura típica para el desarrollo de sistemas distribuidos es la arquitectura **cliente-servidor**. Los clientes son elementos activos que demandan servicios a los servidores realizando peticiones y esperando la respuesta, los servidores son elementos pasivos que realizan las tareas bajo requerimientos de los clientes.

Por ejemplo, un cliente web solicita una página, el servidor web envía al cliente la página solicitada. Véase Figura 1.19. La comunicación entre servidores y clientes se realiza a través de la red.

Existen varios modelos de programación para la comunicación entre los procesos de un sistema distribuido:

- **Sockets.** Proporcionan los puntos extremos para la comunicación entre procesos. Es actualmente la base de la comunicación. Pero al ser de muy bajo nivel de abstracción, no son adecuados a nivel de aplicación. En el capítulo 3 se tratarán los sockets en Java.
- **Llamada de procedimientos remotos o RPC (*Remote Procedure Call*).** Permite a un programa cliente llamar a un procedimiento de otro programa en ejecución en un proceso servidor. El proceso servidor define en su interfaz de servicio los procedimientos disponibles para ser llamados remotamente.
- **Invocación remota de objetos.** El modelo de programación basado en objetos ha sido extendido para permitir que los objetos de diferentes procesos se comuniquen uno con otro por medio de una *invocación a un método remoto* o **RMI (*Remote Method Invocation*)**. Un objeto que vive en un proceso puede invocar métodos de un objeto que reside en otro proceso. **Java RMI** extiende el modelo de objetos de Java para proporcionar soporte de objetos distribuidos en lenguaje Java.



Figura 1.19. Cliente-servidor sobre web.

VENTAJAS E INCONVENIENTES

Ventajas que aportan los sistemas distribuidos:

- Se pueden compartir recursos y datos.
- Capacidad de crecimiento incremental.

⁴ http://es.wikipedia.org/wiki/Programaci%C3%B3n_distribuida

- Mayor flexibilidad al poderse distribuir la carga de trabajo entre diferentes ordenadores.
- Alta disponibilidad.
- Soporte de aplicaciones inherentemente distribuidas.
- Carácter abierto y heterogéneo.

Pero no todo son ventajas, algunos inconvenientes son:

- Aumento de la complejidad, se necesita nuevo tipo de software.
- Problemas con las redes de comunicación: pérdida de mensajes, saturación del tráfico.
- Problemas de seguridad como por ejemplo ataques de denegación de servicio en la que se “bombardea” un servicio con peticiones inútiles de forma que un usuario interesado en usar el servicio no pueda usarlo.

ACTIVIDAD 1.11

Busca en Internet aplicaciones de los sistemas distribuidos.

PROGRAMACION CONCURRENTE, PARALELA Y DISTRIBUIDA

Programación Concurrente: Tenemos varios elementos de proceso (hilos, procesos) que trabajan de forma conjunta en la resolución de un problema. Se suele llevar a cabo en un único procesador o núcleo.

Programación Paralela: Es programación concurrente cuando se utiliza para acelerar la resolución de los problemas, normalmente usando varios procesadores o núcleos.

Programación Distribuida: Es programación paralela cuando los sistemas están distribuidos a través de una red (una red de procesadores); se usa paso de mensajes.

1.4.3. PVM. INSTALACIÓN Y CONFIGURACIÓN

PVM (Parallel Virtual Machine - Máquina virtual en paralelo) es un conjunto de herramientas software que permiten emular un marco de computación concurrente, distribuido y de propósito general, utilizando para ello grupos de ordenadores conectados, de manera que ni los ordenadores ni las redes que los conectan tienen las mismas características arquitectónicas.

Permite conectar entre sí ordenadores Unix y Windows (WIN95, NT 3.5, NT 4.0) para ser usados como un único gran ordenador paralelo de alto rendimiento. Así, grandes problemas de cómputo se pueden resolver de manera más rentable aprovechando la potencia y memoria de muchos equipos conectados.

Cientos de sitios en todo el mundo están usando PVM para resolver importantes problemas científicos, industriales y médicos, además de su uso como una herramienta educativa para enseñar programación paralela.

El modelo de computación de PVM se basa en considerar que una aplicación es una colección de tareas que se comunican y sincronizan mediante el paradigma de paso de mensajes. El sistema de PVM se compone de 3 partes⁵:

⁵ Procesamiento paralelo teoría y programación. Sebastián Dormido Canto y otros, Ed: Sanz y Torres.