U1Programacion-Multiproceso.pdf

hugoomazariio_

Programación de servicios y procesos

1º Desarrollo de Aplicaciones Multiplataforma

Estudios España

Accede al documento original

PROGRAMACIÓN DE SERVICIOS Y PROCESOS

U1. PROGRAMACIÓN MULTIPROCESO

THE CORE CENTRO DE FORMACIÓN PROFESIONAL

Garbine Sukia

ÍNDICE

U1. PROGRAMACIÓN MULTIPROCESO

1. PROGRAMAS Y PROCESOS

- ✓ **Programa** ⇨ Conjunto de instrucciones que se pueden ejecutar en un ordenador. Normalmente se almacena en un fichero binario en un disco duro.
- ✓ **Proceso** ⇨ Instancia de un programa que se ha cargado en memoria y está en ejecución.
- ✓ Pasos en la ejecución de un programa:
  - ✓ Carga de las sentencias que componen el programa en la memoria principal.
  - ✓ Creación de un proceso, al que se le asigna dicha memoria.
  - ✓ El procesador (CPU) ejecuta el proceso, va cargando en los registros del procesador las instrucciones a ejecutar, desde la memoria principal.
  - ✓ La Unidad Aritmético Lógica (UAL) realiza las operaciones cuyo resultado se almacena en un registro para transferirlo después a la memoria.
  - ✓ Los procesos tienen acceso a los dispositivos de entrada/salida (E/S) para guardar la información.

1. PROGRAMAS Y PROCESOS

- ✓ **Servicio** ⇨ Proceso que no interactúa directamente con el usuario sino con otros procesos a los que proporciona un servicio determinado.
- ✓ **Servicio en red** ⇨ Servicio que se presta a procesos remotos, situados en otro ordenador, realizando la comunicación mediante protocolos estándares de red.
- ✓ **Sistema monoprocesador** ⇨ Sistema/ordenador que solo dispone de un procesador para realizar su labor.
- ✓ **Sistema multiprocesador** ⇨ Sistema que dispone de varios procesadores.
- ✓ **Sistema distribuido** ⇨ Sistema multiprocesador en el que los procesadores están distribuidos en varios ordenadores independientes que se comunican a través de una red de comunicación.

1. PROGRAMAS Y PROCESOS

- ✓ Los sistemas operativos ejecutan múltiples procesos simultáneamente, incluso en los sistemas que cuentan con un único procesador.
- ✓ La **multitarea** consiste en la ejecución simultánea de varios procesos en un procesador a lo largo de un intervalo de tiempo.
- ✓ En un instante dado solo se está ejecutando un proceso, pero el SO toma el control a intervalos de tiempo regulares de su ejecución y decide parar la ejecución del proceso e iniciar otro.
- ✓ Considerando un intervalo de tiempo más largo, el resultado es similar a que la ejecución de los procesos fuese simultánea.
- ✓ **Cambio de contexto** ⇨ Cuando se para un proceso se guarda su estado de ejecución para poder retomarlo después exactamente en el mismo punto y se cargan los valores del nuevo proceso en ejecución.
- ✓ En un sistema multiprocesador también se realiza esta acción, dado que el número de procesos en ejecución siempre es mayor que el número de procesadores disponibles.

OFERTAS BLACK FRIDAY

msi

Esta oferta

es como una doble victoria tuya: disfrútala ahora porque no pasa dos veces.

Ver ofertas

HASTA -40%

Tu viejo portátil ya dio lo que tenía que dar. Pásate a MSI: rápido, potente y sin dramas. Lo enciendes y estás listo para todo. Aprovecha las ofertas y despídete del modo “se cuelga cada dos por tres”.

2. PROCESOS

Elementos de un proceso

- ✓ Un proceso puede crear otros procesos, que pueden crear nuevos procesos a su vez, generándose una jerarquía de procesos.
- ✓ La información asociada a un proceso se almacena en el **Bloque de Control de Proceso** (BCP).
- ✓ Cada proceso tiene su propio BCP con información sobre:
  - Identificación de proceso.
  - Estado del proceso.
  - Contador de programa.
  - Registros de CPU.
  - Información de planificación de CPU.
  - Información de gestión de memoria.
  - Información contable como tiempo de CPU y tiempo real consumido.
  - Información de estado de E/S como lista de dispositivo asignados, archivos...

2. PROCESOS

Procesos en el sistema operativo

- En windows ⇨ Ctrl + Alt + Supr. → Administrador de tareas

Administrador de tareas

Archivo Opciones Vista

Procesos | Rendimiento | Historial de aplicaciones | Inicio | Usuarios | Detalles | Servicios

Nombre	Estado	77% CPU	45% Memoria	6% Disco	0% Red	4% GPU	Motor de GPU
Aplicaciones (6)							
Administrador de tareas		0,7%	25,9 MB	0,1 MB/s	0 Mbps	0%	
Adobe Acrobat		0%	88,0 MB	0 MB/s	0 Mbps	0%	
eM Client (32 bits) (2)		0%	259,1 MB	0 MB/s	0 Mbps	0%	
Explorador de Windows (3)		0,7%	82,4 MB	0,1 MB/s	0 Mbps	0%	
Google Chrome (22)		0%	1.169,2 MB	0,1 MB/s	0 Mbps	0%	
Microsoft PowerPoint (2)		0,3%	172,8 MB	0 MB/s	0 Mbps	0,2%	GPU 0 - 3D
Procesos en segundo plano (8...							
Acrobat Update Service (32 bits)		0%	0,3 MB	0 MB/s	0 Mbps	0%	
Adobe AcroCEF		0%	23,1 MB	0 MB/s	0 Mbps	0%	
Adobe AcroCEF		0%	3,2 MB	0 MB/s	0 Mbps	0%	

2. PROCESOS

Procesos en el sistema operativo

- ✓ En windows ⇨ cmd ⇨ tasklist

```
C:\Users\Garbi>tasklist

Nombre de imagen              PID Nombre de sesión Núm. de ses Uso de memor
========================= ======== ================ =========== ============
System Idle Process             0 Services                   0         8 KB
System                          4 Services                   0     4.024 KB
Registry                      124 Services                   0    42.976 KB
smss.exe                      504 Services                   0     1.112 KB
csrss.exe                     772 Services                   0     5.804 KB
wininit.exe                   860 Services                   0     7.060 KB
services.exe                  932 Services                   0    11.296 KB
lsass.exe                     952 Services                   0    24.392 KB
```

```
Nombre de imagen              PID Servicios
========================= ======== ============================================
svchost.exe                   968 BrokerInfrastructure, DcomLaunch, PlugPlay,
                                  Power, SystemEventsBroker
svchost.exe                   880 RpcEptMapper, RpcSs
svchost.exe                  1032 LSM
svchost.exe                  1212 NcbService
```

- /V Muestra información detallada de tareas.
- /SVC Muestra los servicios hospedados en cada proceso.
- /M [module] Enumera todas las tareas que actualmente usan el módulo exe/dll dado. Si no se especifica el módulo se muestran todos los módulos cargados.
- /FO formato Especifica el formato de salida. Valores válidos: TABLE, LIST, CSV.
- /FI filtro Muestra un conjunto de tareas que coinciden con el criterio especificado por el filtro.

2. PROCESOS

Procesos en el sistema operativo

- ✓ En Linux ⇨ comando ps (process status).

```
jfduran@ubuntu:~$ ps
    PID TTY          TIME CMD
   1948 pts/0    00:00:00 bash
   1954 pts/0    00:00:00 ps
```

PID: Process IDentifier, o identificador del proceso.
TTY: terminal asociado de que lee y al que escribe. Si no hay, aparece una interrogación (?)
TIME: tiempo de ejecución asociado. Es la cantidad total de tiempo de CPU que el proceso ha utilizado desde que nació.
CMD: nombre del proceso.

```
jfduran@ubuntu:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
jfduran     1948    1940  0 07:02 pts/0    00:00:00 bash
jfduran     1995    1948  0 07:04 pts/0    00:00:00 ps -f
```

UID: User IDentifier, o identificador del usuario.

PPID: PID del padre de cada proceso

C: porcentaje de recursos de CPU utilizado por el proceso.

STIME: hora de inicio del proceso.

```
jfduran@ubuntu:~$ ps -AF
UID          PID    PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
root           1       0  1 41880 11372   0 07:00 ?        00:00:03 /sbin/init a
root           2       0  0     0     0   1 07:00 ?        00:00:00 [kthreadd]
root           3       2  0     0     0   0 07:00 ?        00:00:00 [rcu_gp]
root           4       2  0     0     0   0 07:00 ?        00:00:00 [rcu_par_gp]
root           5       2  0     0     0   0 07:00 ?        00:00:00 [kworker/0:0
root           6       2  0     0     0   0 07:00 ?        00:00:00 [kworker/0:0
root           7       2  0     0     0   0 07:00 ?        00:00:00 [kworker/0:1
root           8       2  0     0     0   0 07:00 ?        00:00:00 [kworker/u25
root           9       2  0     0     0   0 07:00 ?        00:00:00 [mm_percpu_w
root          10       2  0     0     0   0 07:00 ?        00:00:00 [rcu_tasks_r
root          11       2  0     0     0   0 07:00 ?        00:00:00 [rcu_tasks_t
root          12       2  0     0     0   0 07:00 ?        00:00:00 [ksoftirqd/0
root          13       2  0     0     0   1 07:00 ?        00:00:00 [rcu_sched]
root          14       2  0     0     0   0 07:00 ?        00:00:00 [migration/0
```

C: porcentaje de CPU utilizado por el proceso.

SZ: tamaño virtual de la imagen del proceso.

RSS: tamaño de la parte residente en memoria en kilobytes.

PSR: procesador que el proceso tiene actualmente asignado

2. PROCESOS

Procesos en el sistema operativo

Se llama **Kernel** (núcleo) a la parte central del sistema operativo. Es una pequeña parte, muy optimizada, que cuenta con un mecanismo de gestión de interrupciones que pueden producirse:

- ✓ Eventos generados por el hardware: tecla pulsada, movimiento de ratón, fin de operación de escritura de disco…
- ✓ Llamadas al sistema: para proporcionar a las aplicaciones acceso a las funcionalidades del sistema operativo.
- ✓ Interrupciones periódicas que llaman al **planificador de procesos a corto plazo** (CPU scheduler) que es quien decide cuándo se realiza un cambio de contexto para ejecutar otro proceso.

Cuando se produce una interrupción el procesador para la ejecución del proceso actual y ejecuta una rutina de tratamiento de interrupción.

Modos de funcionamiento del procesador:

- ✓ **Modo kernel** o supervisor: es en este modo en el que se ejecutan las rutinas de tratamiento de interrupciones.
- ✓ **Modo usuario**

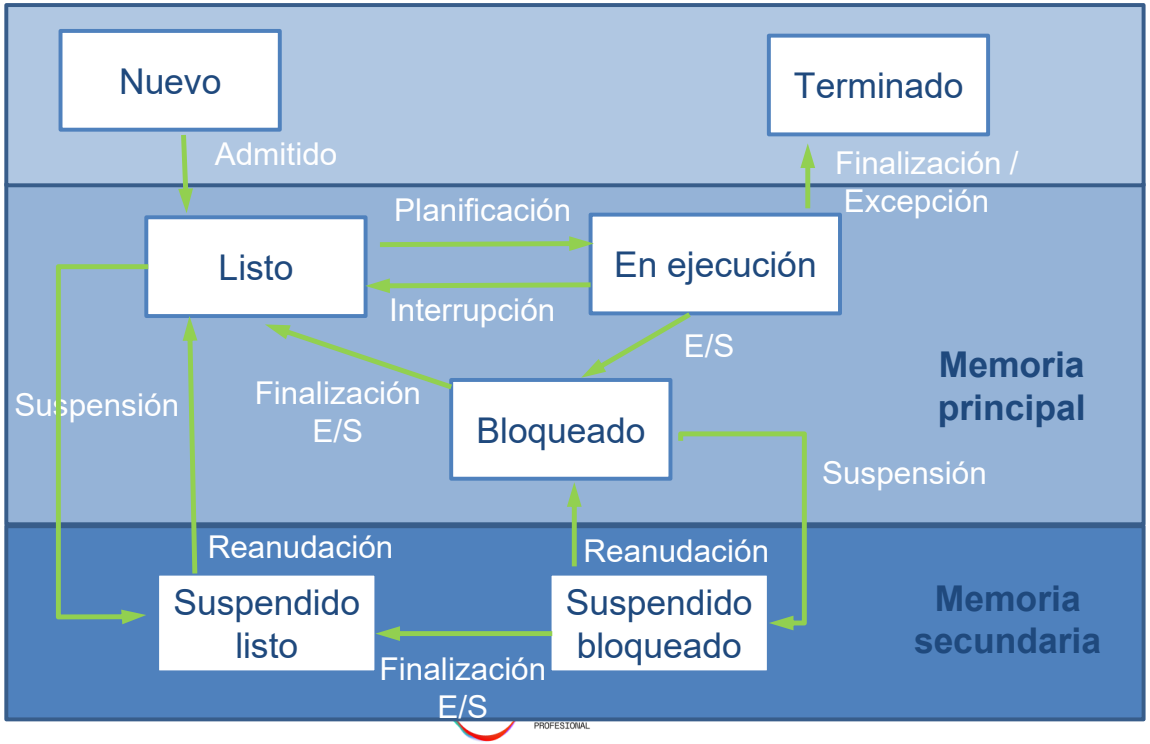
2. PROCESOS

Estados de un proceso

Planificación a largo plazo
Job scheduler

Planificación a corto plazo
CPU scheduler

Planificación a medio plazo



2. PROCESOS

Estados de un proceso

- **Nuevo** → Creado a partir del fichero ejecutable.
- **Listo** → Parado temporalmente y listo para ejecutarse.
- **En ejecución** → El proceso se está ejecutando.
- **Bloqueado** → El proceso está parado hasta que no ocurra un evento externo.
- **Terminado** → El proceso ha finalizado su ejecución y libera su memoria. El proceso debe indicar al sistema que terminó su ejecución o el propio sistema lo puede finalizar mediante una excepción.

3. SISTEMAS MULTITAREA

En un sistema multitarea existen varios procesos en ejecución a la vez.

Programa → conjunto de instrucciones sobre unos datos de entrada para obtener una salida.

Proceso → parte activa de un programa con recursos asociados.

Un programa puede dar lugar a varios procesos que ejecutan una parte del mismo. Ejemplo: Navegador web → 1. acciones del usuario, 2. peticiones al servidor.

✓ **PROCESOS CONCURRENTES**.

Varias instancias ejecutándose al mismo tiempo en el SO pero no al unísono. No hay simultaneidad, hay intercalado o solapamiento.

✓ **PROCESOS PARALELOS** → Varias instancias ejecutándose simultáneamente.

En la programación concurrente varios procesos pueden colaborar para la consecución de un fin y otros pueden competir por los recursos del sistema.

3. SISTEMAS MULTITAREA

Programación concurrente

Disciplina que se encarga del estudio de las notaciones que permiten especificar la ejecución concurrente de acciones en un programa.

BENEFICIOS

- ✓ Mejor aprovechamiento de la CPU. Mejor rendimiento del sistema.
- ✓ Mayor velocidad de ejecución. Menor tiempo de respuesta.
- ✓ Solución de problemas de naturaleza concurrente, como son:
  - ➢ Sistema de control: con captura de datos y análisis para la actuación. Ej. Sistemas en tiempo real.
  - ➢ Tecnologías web: que deben atender peticiones de múltiples usuarios.
  - ➢ Aplicaciones basadas en GUI: el usuario utiliza la aplicación mientras ésta realiza otra tarea.
  - ➢ Simulación: programas que modelan sistemas físicos con autonomía.
  - ➢ Sistemas gestores de bases de datos.

3. SISTEMAS MULTITAREA

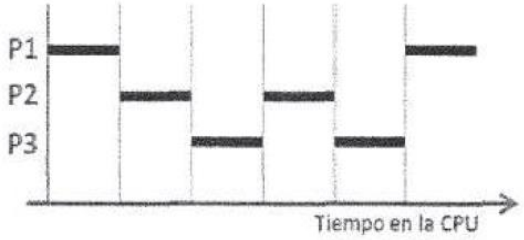
Programación concurrente

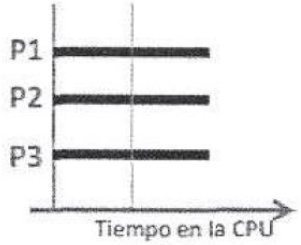
CONCURRENCIA Y HARDWARE

✓ **Sistema Monoprocesador** → Puede haber concurrencia pero no paralelismo.

Se usan ranuras temporales aprovechando los ciclos de procesador. La programación concurrente de varios procesos en un sistema monoprocesador se conoce como *multiprogramación*. Los procesos comparten la misma memoria. Los procesos se comunican mediante variables compartidas.

✓ **Sistemas multiprocesador** → Podemos tener un proceso en cada procesador.





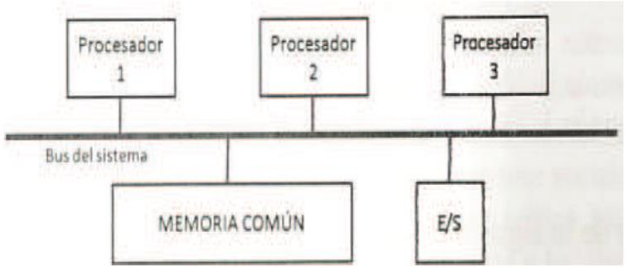
Paralelismo real

3. SISTEMAS MULTITAREA

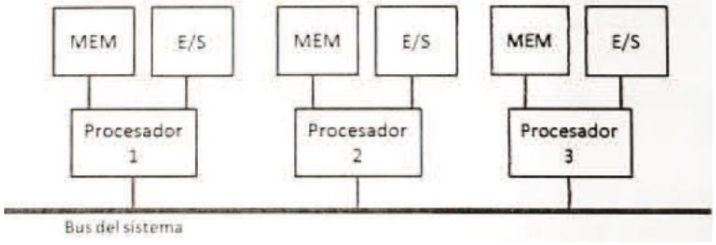
Programación concurrente

CONCURRENCIA Y HARDWARE

- ✓ **Sistemas fuertemente acoplados** → Tienen la memoria compartida por todos los procesadores.



- ✓ **Sistemas débilmente acoplados** → Los procesadores cuentan con memorias locales no compartidas.



3. SISTEMAS MULTITAREA
Programación concurrente

PROBLEMAS INHERENTES

- ✓ El principal problema es la dificultad de implementar los mecanismos de **sincronización y comunicación entre procesos.**
- ✓ **Exclusión Mutua** → Puede ocurrir que varios procesos accedan a la misma variable al mismo tiempo para actualizarla, pudiendo provocar inconsistencia en los datos.

  Solución → *Región Crítica* asociada a la variable. El acceso debe realizarse siempre dentro de dicha región. Sólo un proceso podrá acceder a ella. El resto debe esperar un tiempo finito.
- ✓ **Condición de sincronización** → Necesidad de coordinar los procesos para sincronizar sus actividades, pues unas pueden depender del resultado de las otras. Existen mecanismos para bloquear un proceso hasta que ocurra un evento.

  Solución → Región crítica, semáforos, región crítica condicional, buzones, sucesos, monitores y sincronización por rendez-vous.

3. SISTEMAS MULTITAREA

Programación paralela

La programación paralela se da en sistemas multiprocesador en los que muchos elementos de procesos independientes trabajan al unísono.

- ✓ Único equipo con varios procesadores o núcleos.
- ✓ Equipos conectados por una red.
- ✓ Problema dividido en partes independientes. Una parte puede realizarse de forma paralela, otras partes del programa deben ejecutarse de forma secuencial.
- ✓ Podemos tener un proceso corriendo en cada procesador.
- ✓ Se necesita intercambiar información con los otros procesadores.

Modelos de programación paralela:

- ✓ **Modelos de memoria compartida (Multiprocesadores)** → Todos acceden al mismo espacio de direcciones.
- ✓ **Modelo de paso de mensajes (Multicomputadores, memoria distribuida)** → Cada procesador dispone de su propia memoria accesible sólo por él.
  - ✓ Petición de datos a otro procesador.
  - ✓ Envío de datos al procesador que los ha pedido.
  - ✓ Clústers → Son sistemas de procesamiento paralelo y distribuido, con varios ordenadores en red.

3. SISTEMAS MULTITAREA

Programación paralela

VENTAJAS E INCONVENIENTES

Ventajas:

- ✓ Ejecución simultánea de tareas.
- ✓ Disminuye tiempo de ejecución.
- ✓ Permite la resolución de problemas grandes y complejos.
- ✓ Uso de recursos no locales (en red).
- ✓ Disminución de coste, utilizando varios ordenadores en lugar de uno más potente y costoso.

Inconvenientes:

- ✓ Los compiladores y entornos de programación para sistemas paralelos más difíciles de desarrollar.
- ✓ Programas más difíciles de escribir.
- ✓ Mayor consumo de energía al contar con varios sistemas interconectados.
- ✓ Mayor complejidad en el acceso a los datos.
- ✓ Comunicación y sincronización entre subtareas.

Google Gemini: Plan Pro a 0€ durante 1 año.

Tu ventaja por ser estudiante

Entra en wlh.es/estudiacongemini

Consigue la oferta

Sintetiza horas de investigación en minutos.

Oferta válida hasta el 9 de diciembre de 2025

3. SISTEMAS MULTITAREA

Programación distribuida

- ✓ La compartición de recursos es uno de los principales motivos de la programación distribuida.
- ✓ Internet es un claro ejemplo de sistema distribuido.
- ✓ Cloud Computing → servicio en la nube.
- ✓ **SISTEMA DISTRIBUIDO** → Aquel en el que los componentes hardware y software, localizados en computadores unidos mediante una red, comunican y coordinan sus acciones mediante el paso de mensajes. Esto implica:
  - ✓ Concurrencia.
  - ✓ Inexistencia de reloj global.
  - ✓ Fallos independientes.
- ✓ La arquitectura cliente-servidor es una arquitectura típica para el desarrollo de sistemas distribuidos . Ej. Servidor web.

3. SISTEMAS MULTITAREA

Programación distribuida

Modelos de programación para la comunicación entre procesos:

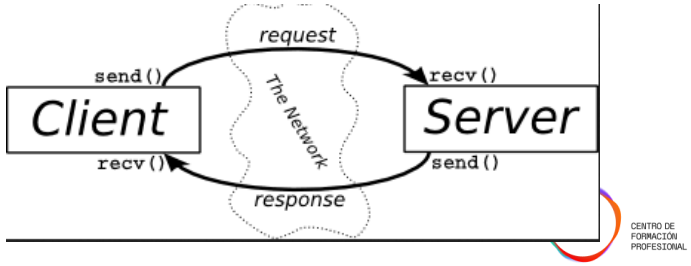
1. **Sockets** → extremos de la comunicación.
2. **RPC (Remote Procedure Call)** →

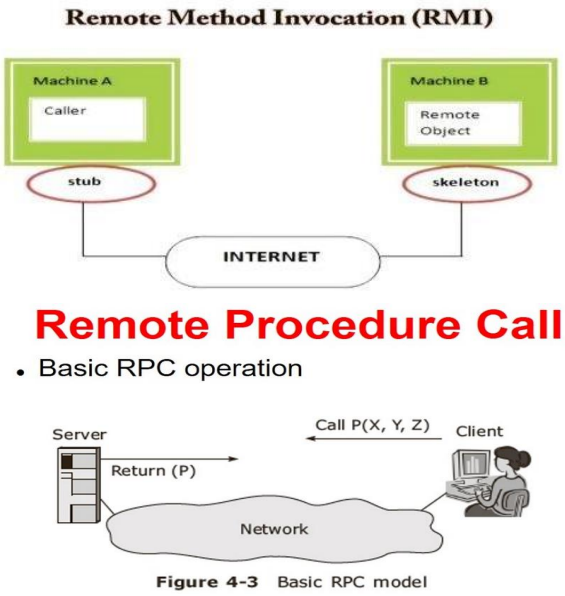
   Programa cliente llama a procedimiento remoto en servidor.

   El Servidor define los procedimientos disponibles para ser llamados.

3. **Invocación Remota de Objetos** →

   Objetos de diferentes procesos se comunican entre sí por medio de RMI (Remote Method Invocation).





3. SISTEMAS MULTITAREA
Programación distribuida

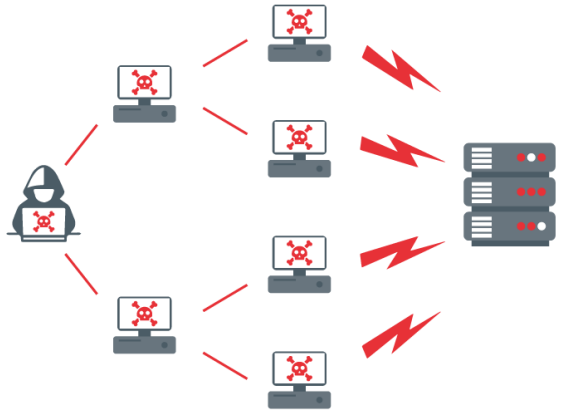
VENTAJAS E INCONVENIENTES

Ventajas:

- ✓ Compartición de recursos y datos.
- ✓ Crecimiento incremental.
- ✓ Flexibilidad en la distribución de carga.
- ✓ Alta disponibilidad.
- ✓ Soporte de aplicaciones distribuidas.
- ✓ Carácter abierto y heterogéneo.

Inconvenientes:

- ✓ Mayor complejidad → nuevo tipo de software.
- ✓ Redes de comunicación: cuello de botella, pérdida de mensajes...
- ✓ Problemas de seguridad → fuerza bruta, denegación de servicios (DDoS).



Distributed
Denial of Service

4. GESTIÓN DE PROCESOS

Concurrencia en Java

- Java está diseñado para soportar la programación concurrente, tanto la máquina virtual, como el lenguaje y la biblioteca estándar.
- **Process** ⇨ Clase que proporciona la funcionalidad básica para procesos.

  Se trata de una clase abstracta. Proporciona métodos para:

  - Lanzar un proceso.
  - Obtener información de su estado.
  - Controlar su ejecución.
  - Realizar la entrada del proceso.
  - Obtener su salida.
  - Esperar que se complete.
  - Comprobar su estado de salida.

  También se pueden crear objetos de la clase *Process* con métodos de otras clases, como *ProcessBuilder.start()* y *Runtime.exec().*
- **Thread** ⇨ Clase que proporciona la funcionalidad básica para hilos. Implementa la interfaz *Runnable*.
- **Java.util.concurrent** ⇨ Paquete para programación concurrente.

4. GESTIÓN DE PROCESOS

Clase Runtime

- Todo programa en Java tiene asociada una instancia de la clase **Runtime** que le permite obtener información de su entorno de ejecución e interactuar con él.
- **getRuntime()** ⇨ Método estático que devuelve la instancia de la clase Runtime de un programa.
- **exec()** ⇨ Método que permite lanzar un nuevo proceso (hay varias variantes de este método).

MÉTODOS

`Process exec(String command)`
`Process exec(String[] comandoYArgs)`
`Process exec(String[] comandoYArgs, String[] envp)`
`Process exec(String[] cmdarray, String[] envp, File dir)`
	Ejecuta un comando. Algunas variantes permiten especificar el entorno de ejecución y el directorio para ejecutar el comando.

`void exit(int status)` `void halt(int status)`	Termina la ejecución de la máquina virtual. El primer método lo hace de manera ordenada, y el segundo, de manera abrupta.
`int availableProcessors()`	Devuelve el número de procesadores disponibles para la máquina virtual de Java.
`long freeMemory()`	Devuelve la cantidad de memoria disponible para la máquina virtual.

4. GESTIÓN DE PROCESOS

Clase ProcessBuilder

- La clase **Process** es abstracta, pero se pueden obtener instancias de sus subclases con métodos de la clase **ProcessBuilder.**
- **ProcessBuilder** ⇨ paquete java.Lang. Permite configurar el entorno de los procesos que crea y redirigir su entrada y salida.

MÉTODOS

`ProcessBuilder(String... comando)` `ProcessBuilder(List<String> comando)`	Construye un `ProcessBuilder` para un comando con los correspondientes argumentos de línea de comandos, en su caso.
`ProcessBuilder directory(File directorio)` `File directory()`	El primer método establece el directorio de trabajo del proceso, y el segundo método lo devuelve. Por defecto, el segundo método normalmente devuelve `null`. Si es así, se puede obtener el directorio de ejecución del proceso actual con `System.getProperty("user.dir")`.
`Map<String,String> environment()`	Devuelve el entorno de trabajo del proceso, que es una lista de asignación de valores para variables de entorno.
`Process start()`	Crea e inicia un proceso.
`ProcessBuilder inheritIO()`	Redirige la salida estándar y de error de los subprocesos creados hacia las del proceso padre, y su entrada estándar desde la del proceso padre.

4. GESTIÓN DE PROCESOS

Clase ProcessBuilder

ProcessBuilder → Gestiona los siguientes atributos de un proceso:

- **Un comando** → lista de cadenas referidas al programa que se invoca y sus atributos.
- **Un entorno** → con sus variables.
- **Un directorio de trabajo.**
- **Una fuente de entrada estándar** → desde una tubería. La entrada puede ser redirigida a otra fuente usando *redirectInput*().
- **Un destino para la salida estándar** y **de error** → desde tubería. También puede ser redirigida con *redirectOutput*() y *redirectError*().
- **Una propiedad** → *redirectErrorStream*.

```
Process p = new ProcessBuilder("Comando", "Argum1").start();
Process pb = new ProcessBuilder("CMD", "/C","DIR").start ();
```

4. GESTIÓN DE PROCESOS

Clase ProcessBuilder

- Cada instancia de **ProcessBuilder** gestiona una colección de atributos.
- El método **start()** crea una nueva instancia con estos atributos. Se puede llamar varias veces dentro de la instancia para crear nuevos subprocesos con idénticos atributos o diferentes.

```
ProcessBuilder pb = new ProcessBuilder("CMD", "/C", "DIR");

Process p = pb.start();
```

- Para redirigir sus flujos de entrada y salida se usan los métodos:
  - **redirectInput ():** Devuelve el valor de la entrada estándar del ProcessBuilder o establece su valor.
  - **redirectOutput():** Devuelve el valor de la salida estándar del ProcessBuilder o establece su valor.

4. GESTIÓN DE PROCESOS

Clase ProcessBuilder

MÉTODOS	MISIÓN
ProcessBuilder command (String argumentos ...)	Define el programa que se quiere ejecutar indicando sus argumentos como una lista de cadenas separadas por comas.
List < String > command ()	Devuelve todos los argumentos del objeto **ProcessBuilder**.
Map < String , String > environment ()	Devuelve en una estructura Map las variables de entorno del objeto **ProcessBuilder**.
ProcessBuilder redirectError (File file)	Redirige la salida de error estándar a un fichero.
ProcessBuilder redirectInput (File file)	Establece la fuente de entrada estándar en un fichero.
ProcessBuilder redirectOutput (File file)	Redirige la salida estándar a un fichero.
File directory()	Devuelve el directorio de trabajo del objeto **ProcessBuilder**.
ProcessBuilder directory(File directorio)	Establece el directorio de trabajo del objeto **ProcessBuilder**.
Process start ()	Inicia un nuevo proceso utilizando los atributos del objeto **ProcessBuilder**.

4. GESTIÓN DE PROCESOS

Clase Process

MÉTODOS	
`void destroy()` `public Process destroyForcibly()`	Termina el proceso. El primer método permite una terminación limpia y ordenada del proceso. El segundo lo termina inmediatamente.
`int exitValue()`	Devuelve el valor de salida, o código de retorno, del proceso. Por convención, un valor 0 indica terminación normal, y otro valor se interpretará como un código de error. Se puede terminar un programa en Java con un código de retorno distinto de 0 con `System.exit(código)`.
`ProcessHandle.Info info()`	Devuelve la información actual del proceso.
`boolean isAlive()`	Comprueba si el proceso está vivo.
`long pid()`	Devuelve el PID o identificador de proceso.
`int waitFor()`	Hace que el hilo en ejecución espere hasta que el proceso haya terminado. Devuelve el valor de salida del proceso. Un valor cero se entiende que corresponde a una ejecución sin errores, mientras que un valor distinto de cero corresponde a un código de error. Si el proceso es de un programa en Java, es el valor devuelto por `System.exit()`, o cero si no se terminó la ejecución con `System.exit()`.
`boolean waitFor(long timeout, TimeUnit unit)`	Hace que el hilo en ejecución espere hasta que el proceso haya terminado, durante un tiempo máximo indicado por `timeout`. Devuelve `true` si el proceso ha terminado por sí mismo antes del tiempo máximo indicado, y `false` en caso contrario.

4. GESTIÓN DE PROCESOS

Clase Process

MÉTODOS

MÉTODOS	MISIÓN
InputStream getInputStream ()	Devuelve el flujo de entrada conectado a la salida normal del subproceso. Nos permite leer el stream de salida del subproceso, es decir, podemos leer lo que el comando que ejecutamos escribió en la consola.

4. GESTIÓN DE PROCESOS

Creación de un proceso. Ejemplo 1

- Programa en Java que lanza un nuevo proceso en el que ejecuta la aplicación de Windows de bloc de notas, Notepad.

```
public class LanzaNotepad {
    public static void main(String[] args) throws IOException {
        ProcessBuilder pb = new ProcessBuilder(command:"NOTEPAD");
        Process p = pb.start();

    }
}
```

- O en una sola línea:

```
Process p = new ProcessBuilder("NOTEPAD").start();
```

4. GESTIÓN DE PROCESOS

Creación de un proceso. Ejemplo 2

- Si queremos lanzar desde nuestro programa en Java un comando de Windows que no tenga ejecutable, como por ejemplo DIR, necesitamos abrir primero la consola utilizando el comando CMD.exe.
- El objeto ProcessBuilder a construir tendrá por lo tanto, los siguientes argumentos: “CMD”, “/C”, “DIR”.
- Para leer la salida del comando DIR usamos el método read() de Inputstream, que devuelve la salida producida por el comando, carácter a carácter.

```
public class LanzaDIR {
    public static void main(String[] args) throws IOException {
        Process p = new ProcessBuilder(command:"CMD", command:"/C",command:"DIR").start();
        InputStream is =  p.getInputStream();
        int c=0;
        while ((c=is.read()) != -1) {
          System.out.print((char) c);
        }
    }
}
```

4. GESTIÓN DE PROCESOS

Ejemplo 3

- Desarrolla un programa que cree un proceso hijo para un comando que se le pase como parámetro de la línea de comandos (args).
- Utiliza el método inheritIO() para que el proceso hijo herede la entrada y salida estándares de su padre.
- Usa el método waitFor() para que se pare la ejecución del programa, proceso padre, hasta que finalice la ejecución del proceso hijo.
- Captura las posibles excepciones IOException e InterrptedException.

4. GESTIÓN DE PROCESOS

Ejemplo 3

```
public static void main(String[] args) {
      if(args.length <=0){
         System.out.println("Debe indicarse comando a ejecutar.");
         System.exit(1);
      }
      ProcessBuilder pb = new ProcessBuilder (args);
      // se hace que el proceso herede la entrada/salida estándar del padre
      pb.inheritIO();
      try {
          Process p = pb.start();
          //espera a que termine la ejecución del proceso hijo y obtiene el código de
retorno
          int codRet = p.waitFor();
          System.out.println("La ejecución de "+ Arrays.toString(args) + " devuelve "
+ codRet + " " + (codRet==0 ? "(ejecución correcta)" : "(ERROR)"));
      }
      catch (IOException e) {
          System.out.println("Error durante la ejecución del proceso");
          e.printStackTrace();
          System.exit(2);
      } catch (InterruptedException ex) {
          System.out.println("Proceso interrumpido");
          System.exit(3);
      }
   }
```

5. SERVICIOS

Los servicios son un tipo particular de proceso que proporcionan servicios a otros procesos

Se ejecutan en segundo plano (background) y no los utilizan los usuarios directamente

Algunos son iniciados directamente por el sistema operativo

Pueden proporcionar servicio a procesos en el mismo equipo o en otros equipos en red

Pueden crear un hilo para responder a cada petición diferente o tener un pool de hilos ya creado y asignar un hilo a cada petición

Esto se conoce como servidor multihilo y siempre tienen un hilo principal que solo se encarga de recibir peticiones