* 1. **PROGRAMAS Y PROCESOS**

Un programa contiene un conjunto de instrucciones que se pueden ejecutar directamente en una máquina física o virtual. Es un objeto estático almacenado en un fichero binario, y este en un disco duro.

Un proceso es la instancia de un programa en ejecución. Es una entidad dinámica. La ejecución de un programa comienza con la creación y ejecución de un proceso, y un proceso puede crear nuevos procesos sobre la marcha, pudiéndose crear una jerarquía de procesos (en Linux se puede ver con el comando pstree, y para ver el uso de la memoria y la CPU se usa top; en Windows se ve con el programa Process Explorer).

Para ejecutar un programa primero tiene que cargarse en memoria. Durante su ejecución se usan varios recursos del sistema.

* Memoria principal. Secuencia de celdas de memoria con la misma longitud en bits y cada una identificada por su posición. Antes de comenzar su ejecución, un programa debe cargarse en un bloque de la memoria principal, que se asigna al proceso que se crea para ejecutar el programa. Este proceso también puede obtener más memoria, dinámicamente, durante su ejecución.
* Procesador o CPU (Central Processing Unit). Ejecuta el proceso que se crea una vez cargado el programa en la memoria. Guarda, en un registro especial (contador de programa, PC, program counter), la dirección en la memoria de la instrucción que se está ejecutando. Un proceso no puede operar directamente con los contenidos de la memoria. Estos antes deben traerse de ella y cargarse en registros del procesador. Las operaciones se realizan en una unidad aritmético-lógica o UAL (arithmetic and logical unit, ALU). Su resultado se obtiene en un registro desde el que se puede transferir una posición de la memoria.
* Dispositivos de entrada/salida (E/S). Los procesos comparten los dispositivos de E/S. Debe guardarse información acerca de a qué procesos se les ha otorgado acceso a un dispositivo E/S, y del estado de las operaciones realizadas sobre él.

Toda la información asociada a un proceso se guarda en un bloque de control de proceso (PCB, process control block). Si el mismo programa se ejecuta varias veces, se creará cada vez un nuevo proceso, y cada uno tendrá su propio PCB.

* 1. **MULTITAREA**

Los sistemas operativos actuales permiten ejecutar múltiples procesos a la vez. Un usuario puede editar un fichero de texto, reproducir un vídeo, copiar ficheros de un directorio a otro, y descargar un fichero de internet. Todo ello, aunque exista un solo procesador en el sistema. A intervalos regulares de tiempo, la ejecución del proceso en curso se detiene y toma el control un programa especial del sistema operativo, que puede parar la ejecución del proceso actual y pasar a ejecutar otro proceso.

La multitarea (multitasking) consiste en la ejecución simultánea de más de un proceso en un procesador a lo largo de un intervalo de tiempo.

En cada momento hay un único proceso en ejecución, pero la rápida alternancia entre ellos hace que todos puedan seguir avanzando a lo largo del tiempo, de manera que, en resumidas cuentas, considerando un intervalo de tiempo, se puede considerar que múltiples procesos se ejecutan simultáneamente. Y así lo percibe, desde luego, un usuario del sistema.

Antes de empezar a ejecutar un nuevo proceso, se guarda el estado de ejecución del proceso actual, para poder más adelante retomar su ejecución en el mismo punto en que se interrumpió. En particular, se guardan los contenidos del contador de programa y de los registros del procesador, y se cargan los del nuevo proceso. Esto se conoce como cambio de contexto.

El rendimiento de un sistema, por supuesto, aumenta si dispone de más de un procesador (sistema multiprocesador). En este caso, se pueden ejecutar en paralelo varios procesos, uno en cada procesador.

Pero, normalmente, siempre habrá muchos más procesos que procesadores, por lo que, también en el caso de los sistemas multiprocesadores, se hará multitarea en cada procesador.

Se puede pensar que, cuando solo se dispone de un procesador, no compensa la multitarea, porque, al tiempo de ejecución de los procesos, hay que sumar el tiempo necesario para los cambios de contexto, además del tiempo que necesita el sistema operativo para planificar la ejecución de los distintos procesos. Pero esto no es así, por los siguientes motivos:

1. Tiempo dedicado: el tiempo que se dedica a esto es pequeño en comparación con el que se dedica a la ejecución de los procesos, y se puede justificar con las ventajas de ejecutar varios procesos al mismo tiempo, especialmente para sistemas interactivos en los que hay una frecuente interacción entre el usuario y el ordenador. La manera natural de trabajar de una persona es hacer varias cosas a la vez, y querrá que todas las tareas con las que está trabajando avancen, y una respuesta rápida cuando interactúa con cualquiera de ellas.
2. Uso del procesador: durante la ejecución de los procesos, suele haber períodos muy largos de tiempo en que no hacen uso del procesador, por estar esperando a que se terminen de realizar operaciones de E/S. Un procesador de textos, por ejemplo, pasa la mayor parte del tiempo desocupado, a la espera de que el usuario pulse una tecla o realice una acción con el ratón. También pasa intervalos de tiempo desocupado mientras obtiene el contenido un documento almacenado en un fichero, o graba un documento en un fichero.

La figura 1.5 muestra la ejecución de dos procesos que alternan periodos de uso del procesador con periodos de espera para que se realicen operaciones de E/S. Se puede ver que el tiempo total de ejecución es solo ligeramente superior al de cualquiera de ellos por separado, porque, en general, es posible utilizar el procesador para el otro proceso mientras uno está a la espera de que se complete una operación de E/S.

* 1. **PROCESOS Y SISTEMAS MONOPROCESADORES Y MULTIPROCESADORES**

Un sistema monoprocesador tiene un único procesador, y un sistema multiprocesador varios. En ambos tipos de sistemas, se pueden ejecutar varios procesos de manera simultánea. En este apartado se explican las técnicas utilizadas para ejecutar varios procesos concurrentes en ambos tipos de sistemas.

* Se denominan procesos concurrentes a los que se ejecutan simultáneamente durante un intervalo dado de tiempo, ya sea de forma real (en sistemas multiprocesadores) o simulada (en sistemas monoprocesadores, o en un mismo procesador de un sistema multiprocesador).
* Se habla de programación concurrente cuando se ejecutan varios procesos concurrentes en un sistema, y también para hacer referencia a las técnicas que permiten desarrollar programas que utilizan varios procesos concurrentes que funcionan de forma conjunta y coordinada, para realizar una tarea.

Si un programa lanza distintos procesos concurrentes para realizar una tarea, estos deben comunicarse y sincronizarse entre sí. Las técnicas de comunicación y sincronización serán distintas, normalmente, según se trate de sistemas monoprocesadores o multiprocesadores, y según el tipo de sistema multiprocesador.

* + 1. **Sistemas monoprocesadores**

La ejecución concurrente de varios procesos en un sistema monoprocesador se conoce con el nombre de multiprogramación. Ya se ha visto que la técnica del cambio de contexto permite una multitarea efectiva y un mejor aprovechamiento del procesador, que se asigna a otro proceso durante el tiempo en que un proceso está inactivo, o a la espera de que se termine de realizar una operación de E/S

* Los actuales procesadores multinúcleo (multicore en inglés: dual-core con dos núcleos, quad-core con cuatro núcleos, hexa-core con seis núcleos y octa-core con ocho núcleos) contienen varios núcleos o unidades de proceso integradas. Por lo tanto, se pueden considerar un tipo de sistemas multiprocesadores, aunque con importantes diferencias con respecto a sistemas multiprocesadores con procesadores independientes.
  + 1. **Sistemas multiprocesadores**

Los sistemas multiprocesadores disponen de más de un procesador. Se pueden clasificar, dependiendo de su arquitectura, en sistemas multiprocesadores fuertemente acoplados y sistemas multiprocesadores débilmente acoplados.

* La programación paralela consiste en la ejecución de varios procesos concurrentes en un sistema multiprocesador. Estos procesos se pueden ejecutar con paralelismo real. Se usa también este término para hacer referencia a las técnicas que permiten desarrollar programas que utilizan varios procesos concurrentes que pueden ejecutarse en distintos procesadores de un sistema multiprocesador. Estas serán distintas según el tipo de sistema multiprocesador de que se trate. Pero los lenguajes de alto nivel proporcionan bibliotecas de funciones o de clases que permiten hacer abstracción de la arquitectura y utilizar las mismas operaciones, que se implementarán de distinta manera según la plataforma concreta.

1. *Sistemas fuertemente acoplados*

En estos sistemas existe una memoria compartida para todos los procesadores, a la que todos ellos acceden a través de un bus de conexión. También acceden al sistema de E/S a través mismo bus.

Se pueden distinguir dos tipos de sistemas multiprocesadores fuertemente acoplados:

1. Sistemas multiprocesadores simétricos (SMP): en los que los procesadores del sistema funcionan todos de igual manera, y no hay ninguno diferenciado del resto.
2. Sistemas multiprocesadores asimétricos: en los que uno de los procesadores del sistema, el maestro o master, controla al resto de los procesadores (slave). Este tipo de sistemas también se denominan
3. master-slave.
4. *Sistemas débilmente acoplados*

En estos sistemas no hay memoria compartida, sino que cada procesador tiene su propia memoria y su propio sistema de E/S. Existe una red de comunicaciones entre los procesadores.

Una posibilidad es que los procesadores estén físicamente cercanos unos a otros y comunicados mediante una red a medida, de alta velocidad y con una estructura regular. Este tipo de sistemas se utilizan en supercomputación y para aplicaciones especializadas intensivas en cálculo.

Una alternativa a este tipo de sistemas son los clusters. Tienen también una estructura regular, pero, en lugar de procesadores de alto rendimiento interconectados mediante una red especializada y diseñada a medida, están formados por ordenadores convencionales conectados mediante una red de área local convencional.

Cabe destacar los clusters Beowulf. En ellos existe un ordenador que funciona como controlador del cluster y como servidor de ficheros para él, y que se usa como consola para todo el cluster. Los nodos no suelen tener teclados ni monitores, y se accede a ellos solo desde el controlador. El controlador y todos los nodos funcionan con sistemas operativos de tipo UNIX o Linux convencionales, en los que se instala determinado software. Para añadir un nuevo ordenador o nodo, solo hace falta conectarlo a la red.

Otra posibilidad, muy común hoy en día, es que los procesadores estén en ordenadores independientes comunicados mediante una red de área local (LAN) o incluso de área amplia (WAN)p a través de internet. Estos son los sistemas distribuidos. Se caracterizan por la heterogeneidad del hardware sobre el que están construidos y por la relativa independencia entre los sistemas que los constituyen.

* La programación distribuida consiste en la ejecución de varios procesos concurrentes en un sistema distribuido. Se usa también este término para hacer referencia al desarrollo de programas que utilizan varios procesos concurrentes que pueden ejecutarse en distintos procesadores de un sistema distribuido, y a las técnicas que lo hacen posible. En un sistema distribuido no hay memoria compartida ni red de conexión especifica entre los distintos ordenadores. La comunicación entre procesos se realiza mediante mensajes a través de la red de comunicaciones que comunica los distintos ordenadores, para los que se suelen utilizar los protocolos de red estándares TCP o UDP.

Las principales ventajas de los sistemas distribuidos es que son altamente escalables y configurables. Sus principales inconvenientes es que son sistemas heterogéneos y complejos de mantener. La sincronización entre procesos puede ser complicada también. La comunicación entre procesos se realiza a través de la red, que tiene una velocidad limitada, por lo que puede convertirse en un cuello de botella.

* 1. **VENTAJAS E INCONVENIENTES DE LA PROGRAMACIÓN CONCURRENTE**

La programación concurrente permite incrementar el rendimiento del sistema, porque los procesadores pasan a ejecutar otros procesos cuando el proceso que ejecutan está desocupado, o a la espera de que termine una operación de E/S. También hace posible un menor tiempo de respuesta, porque el tiempo de procesador se distribuye entre todos los procesos en ejecución. La mejora en el rendimiento dependerá, de todas formas, del tipo de aplicaciones.

Lo ideal es que una aplicación se pueda descomponer en procesos independientes que se ejecuten en paralelo con un mínimo de interacción y comunicación entre ellos.

Un sistema multiprocesador debe ser flexible para permitir distribuir dinámicamente la carga de trabajo entre los procesadores disponibles. Y debe ser escalable para permitir que la potencia de cálculo aumente con la instalación de nuevos procesadores.

Un sistema multiprocesador redundante tiene procesadores suplementarios que proporcionan alta disponibilidad (bajo tiempo de respuesta ante peticiones de los usuarios) y tolerancia a fallos, de manera que, si se produce un fallo en un procesador, se deshabilita para seguir funcionando con el resto.

Los principales inconvenientes de la programación concurrente pueden venir, sobre todo, de la dificultad de implementar mecanismos adecuados de sincronización y comunicación entre procesos. Según el caso, es posible que la sobrecarga que suponen anule la mejora en el rendimiento que proporciona la ejecución concurrente. En sistemas distribuidos, la red podría convertirse en un cuello de botella si no tiene la suficiente capacidad para el tráfico que genera el intercambio de mensajes entre procesos.

* 1. **KERNEL (O NÚCLEO) DEL SISTEMA OPERATIVO Y LLAMADAS AL SISTEMA**

La parte central del sistema operativo se denomina kernel o núcleo. Es una parte pequeña y muy optimizada del sistema operativo que da respuesta a multitud de eventos mediante un mecanismo de gestión de interrupciones. Las interrupciones pueden producirse por diversas causas:

* Eventos generados por el hardware. Por ejemplo: movimientos del ratón o pulsaciones del
* teclado, o el final de una operación de lectura o escritura en disco.
* Llamadas al sistema. El kernel proporciona acceso para las aplicaciones a determinados servicios o funcionalidades del sistema operativo mediante llamadas al sistema.
* Interrupciones periódicas que invocan al planificador de procesos a corto plazo del sistema operativo (CPU scheduler). Este decide si se sigue ejecutando el proceso actual o se hace un cambio de contexto para pasar a ejecutar otro proceso.

Cuando sucede una interrupción, el procesador deja de ejecutar el proceso en curso y pasa a ejecutar una rutina de tratamiento de interrupción. Cuando esta finaliza, se reanuda la ejecución en el mismo lugar en que estaba antes de la interrupción.

El procesador tiene dos modos de funcionamiento: modo kernel, o supervisor, y modo usuario. Las rutinas de tratamiento de interrupciones se ejecutan en modo kernel.

* Las llamadas al sistema suelen estar escritas en un lenguaje de bajo nivel, normalmente C, o incluso ensamblador. Estas llamadas se pueden invocar desde código en Co ensamblador. Pero, normalmente, los programas de aplicación no las llaman directamente, sino a través de APl de más alto nivel. Por ejemplo, POSIX en Linux, o Win32 o Win64 en Windows.
* Linux no es un sistema operativo, sino un núcleo o kernel que se puso a disposición del público bajo una licencia de software libre GPL. En torno a él se han desarrollado infinidad de aplicaciones que también se han puesto a disposición del público bajo la misma licencia. Esta fue creada por la FSF (Free Software Foundation), impulsora del proyecto GNU, que se creó con el objetivo de desarrollar un sistema operativo con software libre alternativo a UNIX. Este proyecto desarrolló con éxito gran cantidad de software. Pero a su kernel (llamado hurd) se adelantó un joven Linus Torvalds, que creó el suyo propio, al que llamó Linux, y que puso a disposición del público bajo licencia GNU. Con frecuencia, se habla del "sistema operativo Linux, pero esto no es correcto, estrictamente hablando. Es más correcto hablar de un sistema operativo GNU/LINUX, o de una distribución de GNU/Linux. Este es un sistema operativo construido en torno a un kernel Linux, y con software de sistema desarrollado para el proyecto GNU. Existen unas cuantas distribuciones de GNU/Linux, como Debian, Suse, Ubuntu o Mint. De hecho, cualquiera con los conocimientos y medios necesarios puede crear la suya porque tanto el núcleo Linux como las aplicaciones son software libre.
  1. **ESTADOS DE EJECUCIÓN DE UN PROCESO**

Desde que se crea inicialmente un proceso, puede pasar por diversos estados y cambiar de uno a otro. El sistema operativo gestiona los procesos y realiza los cambios de estado, teniendo en cuenta distintos eventos que pueden suceder durante el ciclo de vida del proceso, que se muestra en la figura 1.11.

Los cambios de estado de los procesos están controlados por el sistema operativo, con el objetivo de conseguir, por un lado, un máximo aprovechamiento de los recursos del sistema, y en particular del procesador o procesadores; y. por otro lado, una ejecución lo más eficiente posible de todos los procesos. Esto se consigue con una planificación a varios niveles:

1. Planificador a largo plazo o de trabajos (job scheduler). Un proceso debe estar en la memoria principal para poderse ejecutar. El planificador a largo plazo decide qué procesos son admitidos para su ejecución. Estos se cargan en la memoria principal y pasan a estado listo (ready). En sistemas operativos, como los que manejan la mayoría de los usuarios en sistemas de sobremesa, no hay propiamente planificación a largo plazo, sino que son los propios usuarios quienes lanzan las aplicaciones, y estas se cargan inmediatamente en la memoria, y pasan a estado listo. Es importante, en cambio, en sistemas donde hay muchos procesos no interactivos y es el propio sistema operativo el que gestiona su lanzamiento.
2. Planificador a corto plazo o de procesador (CPU scheduler). Se realiza para procesos cargados en la memoria principal. Su objetivo es repartir el tiempo de procesador entre todos los procesos, de manera que se consiga un máximo aprovechamiento del procesador, y que los procesos se ejecuten de la manera más eficiente posible. Un proceso en estado listo puede pasar a estado en ejecución y empezar a ejecutarse en el procesador. Un proceso en este estado puede volver a estado listo para permitir que otro proceso se ejecute. Cuando un proceso en ejecución realiza una operación de E/S, pasa a estado bloqueado, y se pasa a ejecutar un nuevo proceso cuyo estado cambia de listo a en ejecución. Cuando la operación de E/S haya concluido, el proceso volverá a estado listo. El planificador a corto plazo funciona basándose en interrupciones periódicas que hacen que el procesador pase inmediatamente a ejecutar una rutina de gestión de interrupción, que es parte del núcleo o kernel del sistema operativo. Esta rutina realiza la planificación a corto plazo, y decide si hay que continuar con la ejecución del proceso actual, o si hay que pasar a ejecutar un nuevo proceso y, en su caso, cuál.
3. Planificador a medio plazo. Gestiona el paso de procesos de la memoria principal a la secundaria (suspensión) y viceversa (reanudación). Los sistemas operativos actuales funcionan, en general, con memoria virtual. Esta es una técnica que utiliza un espacio de direcciones virtual, mayor que el disponible en la memoria física, y utiliza para ello un medio de almacenamiento secundario. El contenido de cualquier dirección de la memoria virtual puede estar en la memoria física (memoria principal) o en almacenamiento secundario (memoria secundaria). Pero el microprocesador solo puede acceder directamente a la memoria principal. Cuando un proceso necesita más memoria de la que hay disponible en ella, se consigue pasando parte de sus contenidos a la memoria secundaria. Cuando se necesitan contenidos que están en la memoria secundaria, deben pasarse antes a la memoria principal. Si no hay suficiente espacio en ella, antes hay que pasar algunos de sus contenidos a la memoria secundaria. Se produce así un intercambio o swap entre las memorias principal y secundaria. La memoria virtual funciona bien si los intercambios solo se producen ocasionalmente, porque las operaciones de lectura y escritura en almacenamiento secundario son muy lentas. Si son demasiado frecuentes, se produce un trasiego constante entre las memorias principal y secundaria (trashing), y el rendimiento del sistema se degrada enormemente. Los procesos en la memoria principal que no se están ejecutando pueden suspenderse, y pasan a la memoria secundaria en los estados suspendido listo y suspendido bloqueado. Desde el estado suspendido bloqueado puede pasar al estado suspendido listo si se completa la operación de E/S que provocó su paso a estado bloqueado. También pueden reanudarse los procesos en estos estados para pasar a la memoria principal.

Para la gestión de los procesos, el sistema operativo utiliza un bloque de control de procesos (PCB) para cada proceso. El sistema operativo mantiene colas de procesos para cada uno de los estados. También mantiene colas para cada dispositivo de E/S, con información de los procesos que tienen operaciones pendientes en dicho dispositivo. La utilización de estas colas permite realizar, de manera muy eficiente, la planificación de procesos.

* 1. **HILOS Y PROCESOS**

Hasta ahora se ha venido hablando de procesos. Para crear un proceso espacio en la memoria para él. Esto consume una cantidad considerable de tiempo y recursos del sistema. La comunicación entre procesos requiere de ciertos mecanismos especiales que se verán más adelante.

Los hilos o hebras (threads en inglés) se llaman veces procesos ligeros (lightweight processes). Un proceso en ejecución tiene inicialmente un hilo, pero se pueden crear más de manera sencilla y rápida. La ejecución de un proceso termina cuando finaliza la ejecución de todos sus hilos. Y a la inversa, si se termina la ejecución de un proceso, se acaba también la ejecución de todos sus hilos.

La creación de un nuevo hilo para un proceso ya existente no requiere reservar e inicializar espacio en la memoria, porque este es compartido por todos los hilos de un mismo proceso. Esto hace muy sencilla la comunicación entre ellos. Si un hilo de un proceso modifica un objeto situado en la memoria, los demás hilos del mismo proceso pueden ver los cambios realizados. Se necesitan mecanismos de sincronización entre hilos para evitar problemas que puedan darse si los distintos hilos modifican sin control objetos situados en la memoria compartida. El planificador a corto plazo, o CPU scheduler, gestiona de manera independiente los distintos hilos de un mismo proceso.

* 1. **SERVICIOS**

Los servicios son un tipo particular de procesos que proporcionan determinados servicios (de ahí su nombre) a otros procesos. Se ejecutan en segundo plano (en background) y no son directamente utilizados por los usuarios.

Otros procesos, en cambio, se ejecutan en primer plano (en foreground), y tienen una interfaz de usuario con la que interactúan los usuarios.

Los servicios se ejecutan continuamente. Normalmente, son iniciados por el sistema operativo durante su arranque, y se ejecutan en segundo plano (en background). Suelen proporcionar información acerca de su ejecución en ficheros de log, en los que van anotando todos los eventos que se producen durante su ejecución y detalles acerca de las acciones que realizan.

Los servicios pueden proporcionar servicios a otros procesos en el mismo ordenador, y también a otros procesos en otros ordenadores. En este caso, suelen comunicarse a través de una red de comunicaciones, utilizando protocolos de red estándares, como TCP o UDP, de la familia de protocolos TCP/IP. Este tipo de servicios en red son un componente fundamental de los sistemas distribuidos. En este contexto, las aplicaciones servidoras, o simplemente los servidores, prestan servicio a otras aplicaciones, y las aplicaciones clientes, o simplemente los clientes, utilizan estos servicios. Por supuesto, una aplicación puede proporcionar un servicio a otras aplicaciones y a la vez utilizar los servicios que proporciona otra aplicación. Es decir, puede actuar a la vez como cliente y como servidor.

Una técnica frecuentemente utilizada por los servicios para mejorar el tiempo de respuesta ante peticiones de servicios de otros procesos es crear un nuevo hilo para responder a cada nueva petición recibida, o bien mantener un pool de hilos, de manera que a cada nueva petición se le puede asignar inmediatamente un hilo ya disponible en el pool. De esta forma, el hilo principal queda libre para recibir nuevas peticiones. Un servidor de este tipo es un servidor multihilo.

Cada sistema operativo tiene sus propios mecanismos para gestionar los servicios, normalmente, se puede configurar cada uno para que arranque automáticamente cuando arranca el sistema. También existen mecanismos generales para arrancar o parar servicios.

* + 1. **Servicios en Windows**

En Windows, el administrador de servicios permite arrancar y detener servicios, así como consultar el estado de ejecución y obtener información adicional acerca de cada uno. También se puede configurar el modo de arranque para cada servicio: automático (para que arranque automáticamente cuando arranca el sistema operativo) o manual.

* + 1. **Servicios en Linux**

En Linux existen sofisticados sistemas para gestión, control y configuración de servicios. El antiguo sistema initd de SystemV está siendo desplazado por el nuevo systemd. Para systemd se utiliza el comando systemctl.

Con el siguiente comando, se muestra una lista con todos los servicios existentes en el sistema. Aquellos para los que se muestra enabled en la columna STATE Se arrancan automáticamente cuando arranca el sistema.

$ systemctl list-unit-files

Con el siguiente comando, se muestran todos los servicios cargados en el sistema. Que estén cargados no significa que estén en ejecución actualmente. El estado de ejecución aparece en la columna SUB.

$ systemctl --type=service

Para cada servicio, se pueden ejecutar comandos del tipo:

$ systemctl acción servicio

Cuadro del libro

* 1. **CONCURRENCIA EN JAVA**

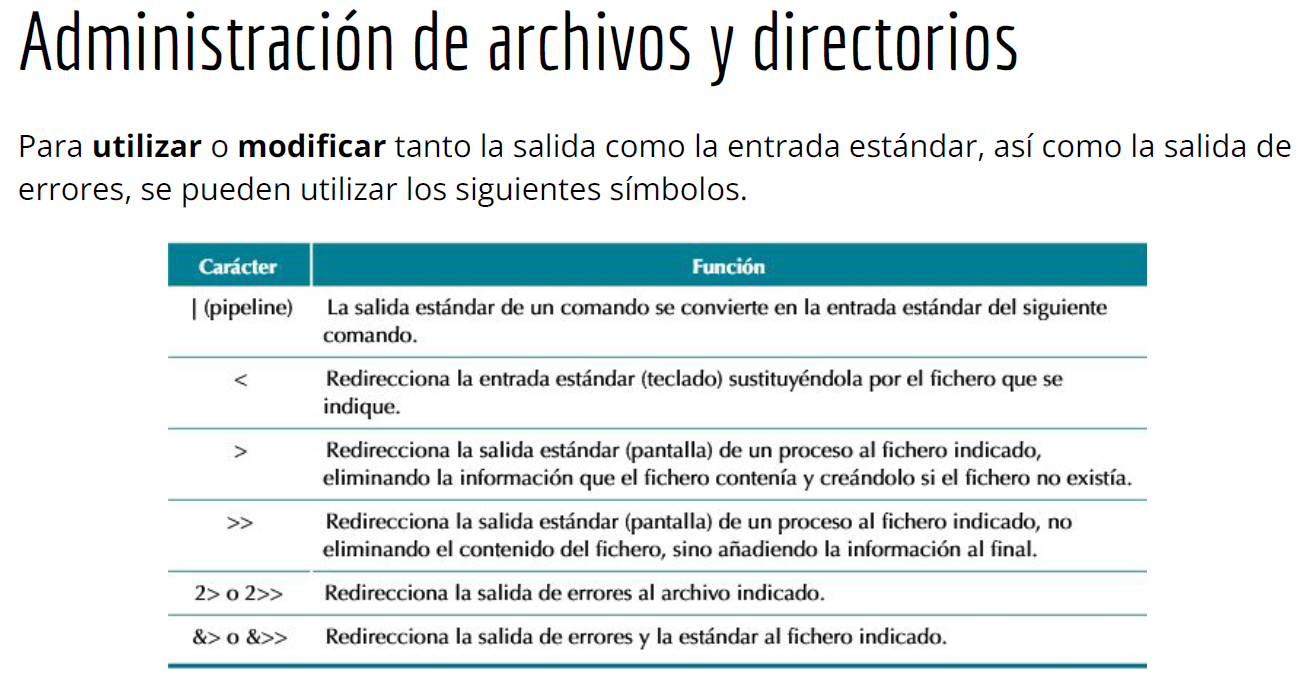
PDF de Jonathan (tema 3). También tengo pendiente hacer apuntes del tema 1 (procesos, RR, FIFO…)

* + 1. **Redirección de entrada y salida estándares y de error**

Las clases ProcessBuilder y Process proporcionan diversos mecanismos para redireccionar y gestionar la salida y entrada de los procesos. Esto podría no funcionar bien en algunas plataformas, debido a limitaciones en los buffers utilizados para ellas por el sistema operativo particular. Estos mecanismos se aprovechan en algunos de los programas de ejemplo que se presentan a continuación.

* Los conceptos de entrada y salida estándares y de salida de error se introdujeron en el sistema operativo UNIX, y posteriormente en Linux y Windows.
  + Entrada estándar o stdin (standard input). Es un stream de entrada del que puede leer datos. Normalmente, se leen datos sobre los que se realiza algún procesamiento. Por defecto, suele estar asociada al teclado, de manera que el programa puede leer todo el texto que se introduce por teclado.
  + Salida estándar o stout (standard output). Es un stream de salida en el que se pueden escribir datos. Una manera sencilla en que un programa puede generar su salida es escribirla en la salida estándar. Por defecto, suele estar asociada a la pantalla, de manera que todo el texto que se envía a la salida estándar se escribe en la pantalla.
  + Salida de error o stderr (standard output). Es un stream de salida en el que se pueden escribir datos. También suele estar asociada a la pantalla, de manera que todo el texto que se envía a la salida de error se escribe en la pantalla.

El proceso puede ser cualquier programa en ejecución. Para los siguientes ejemplos se utilizarán comandos existentes en Linux.



1. Mecanismos de redirección de la clase ProcessBuilder