Entrada/salida

José Ramón Herrero Zaragoza
Teodor Jové Lagunas
Enric Morancho Llena

PID_00169384

UOC
Universitat Oberta de Catalunya

© 2014, FUOC. Se garantiza permiso para copiar, distribuir y modificar este documento según los términos de la GNU Free Documentation License, Version 1.2 o cualquiera posterior publicada por la Free Software Foundation, sin secciones invariantes ni textos de cubierta delantera o trasera. Se dispone de una copia de la licencia en el apartado "GNU Free Documentation License" de este documento.

Índice

Introducción

En este módulo didáctico, analizamos uno de los elementos necesarios en la ejecución de un programa: **la entrada/salida**. Ésta permite que un proceso pueda recibir o enviar información desde el exterior y hacia el exterior.

Nos centraremos en el análisis de los diversos tipos de dispositivos, el tratamiento que hace el sistema operativo y la percepción que tiene el usuario mediante este tratamiento. Veremos que la variedad de dispositivos hace deseable independizar los programas con respecto a los dispositivos, es decir, que un mismo código pueda funcionar con diferentes dispositivos sin ser modificado. Veremos cómo se puede conseguir mediante una interfaz de sistema operativo con operaciones uniformes que trabajan en dispositivos virtuales. Con el fin de consolidar estos conocimientos mostraremos sistemas reales. Utilizaremos el sistema operativo UNIX y lo analizaremos tanto desde el punto de vista del intérprete de comandos como desde el punto de vista del programador. Seguiremos también el caso de Windows.

Objetivos

En los materiales didácticos facilitados en este módulo, encontraréis las herramientas necesarias para alcanzar los objetivos siguientes:

1. Entender el concepto d*e* entrada/salida (E/S) de un proceso.

2. Conocer la diversidad de dispositivos que existen y sus modos de trabajo.

3. Entender la necesidad de independizar los programas de los dispositivos con los que trabajan.

4. Aprender el concepto de dispositivo virtual.

5. Conocer el concepto de independencia de los dispositivos basados en los dispositivos virtuales y en la uniformidad de las operaciones.

6. Distinguir los dispositivos físicos de los dispositivos lógicos y de los dispositivos virtuales.

7. Aprender el significado de sesión de trabajo con un dispositivo y conocer las operaciones básicas que se efectúan durante la sesión de trabajo.

1. El concepto de entrada/salida (E/S)

Un proceso, como contenedor de los recursos necesarios para ejecutar un programa, dispone de un espacio lógico de direcciones donde guardar el código, los datos y la pila[1]. Cuando el usuario pide ejecutar un programa, el sistema operativo crea un proceso y, entre otras cosas, le reserva memoria, carga este programa y sus datos y lo deja listo para ejecutarse. Así, el proceso puede realizar operaciones basadas en todo lo que tiene almacenado en la memoria. Sin embargo, con eso el programa siempre trabajaría con los mismos datos, que serían los especificados por el programador al establecer el programa. Por cuestiones de seguridad, el sistema operativo se encarga de aislar la memoria de cada proceso en ejecución de forma que ninguno de ellos pueda interferir, ya sea por error o por mala intención, en la ejecución de otros procesos o del sistema operativo mismo. Por lo tanto, un proceso no puede, en principio, ir a buscar datos en otras zonas de memoria externas a su espacio lógico.

(1)Recordad que la pila sirve para almacenar las variables locales de las subrutinas, así como los parámetros pasados al invocarlas (a menos que se pasen a través de registros).

Algo similar pasa a otra escala con un sistema computador, formado en un nivel elemental por el procesador y la memoria. Si sólo pudiéramos usar el computador para trabajar con lo que tiene inicialmente en la memoria, el uso que podríamos hacer sería muy limitado.

Existen dispositivos que permiten **enviar información desde el exterior hacia la memoria** y otros que permiten **enviar información desde la memoria hacia el exterior** del computador. El primer caso se llama **entrada** de información y los dispositivos que permiten hacerlo los llamamos dispositivos de entrada. El segundo se llama **salida** y se puede hacer con dispositivos de salida. Hablamos, por lo tanto, de hacer **entrada/salida**.

Estudiaremos los dispositivos de entrada/salida en el apartado siguiente.

Una vez introducido el concepto de entrada/salida al sistema computador, podemos volver a la escala de un proceso. Hay que tener mecanismos que nos permitan entrar información hacia la memoria de un proceso desde el exterior y mecanismos para poder enviar información desde la memoria hacia el exterior. Eso nos puede permitir, por ejemplo, introducir datos en un proceso desde un teclado. Con estos datos, el proceso puede hacer cálculos y los resultados de estos cálculos se pueden mostrar, por ejemplo, por pantalla.

2. El concepto de dispositivo de entrada/salida

El concepto de dispositivo de entrada/salida está muy relacionado con el de periférico. Entendemos por periféricos elementos como:

- el teclado, el ratón o una tarjeta digitalizadora, que permiten introducir información en el sistema para que pueda ser manipulada por los procesos,
- las pantallas, que sirven para mostrar al exterior la información que los procesos elaboran,
- los discos, que permiten almacenar información,
- los módems o las redes, destinados a transferir información entre procesos.

Figura 1
Con el teclado, introducimos información en el sistema.

A pesar de eso, y tal como veremos a lo largo de este módulo, un dispositivo de entrada/salida no tiene que estar a la fuerza asociado a un periférico, sino que puede ser cualquier objeto de hardware o software gestionado por el sistema que sea capaz de llevar a cabo el tipo de acciones que hemos descrito.

Un **dispositivo de entrada/salida** es un objeto gestionado por el sistema operativo con el que los procesos pueden establecer operaciones de lectura/escritura con la finalidad de obtener información, almacenarla, mostrarla o transferirla.

El sistema operativo es el encargado de gestionar los dispositivos proporcionando las operaciones necesarias para que los procesos puedan acceder a él. Tal como veremos, esta gestión tiene que ofrecer los elementos que indicamos a continuación:

- Una **interfaz de acceso al proceso/dispositivo** que sea **independiente** de las particularidades de cada uno de los dispositivos, pero que al mismo tiempo permita, si fuera necesario, explotar todas sus características.
- Un **entorno portable** que permita a las aplicaciones trabajar con dispositivos diferentes sin tener que modificarlas.

3. Las características de los dispositivos de entrada/salida

Aunque la función de los diferentes dispositivos es genéricamente la misma (permitir la entrada o la salida de datos), el uso que se le da y el resultado final que proporcionan pueden ser muy diferentes según el tipo y el modelo de dispositivo que utilizamos. Así pues, los dispositivos tienen características singulares que el sistema tiene que gestionar de formas diferentes: las impresoras vuelcan la información en un soporte material como el papel, los teclados permiten entrar datos o las pantallas visualizan las salidas, entre otros.

Ejemplo

No es lo mismo tener una impresora de inyección de tinta que tener una láser, ni tampoco es lo mismo tener la impresora conectada al ordenador mediante un puerto paralelo que tenerla conectada mediante un puerto USB.

Podemos establecer una **clasificación de los dispositivos** según sus características con el fin de analizar las diferencias que presentan. Visto su gran número y variedad, las hemos agrupado según **características físicas**, **de acceso** y **de control**. De todas las características que enumeraremos a continuación, sólo comentaremos con cierta profundidad las que afectan directamente a los usuarios. Tenemos que destacar que esta clasificación es una de las muchas que podríamos establecer basándonos en las características de los dispositivos y que, por lo tanto, se tiene que entender como un marco orientativo para lograr una comprensión óptima de la diversidad de los dispositivos.

3.1. Características físicas

Las características físicas hacen referencia a propiedades intrínsecas de los dispositivos. Son las siguientes:

1) Los dispositivos pueden ser extraíbles o fijos: hay dispositivos que son fijos, como un disco duro interno, y otros que son extraíbles, como un disco externo, un lápiz de memoria (*pen drive*) o un reproductor de música en formato MP3, que se pueden conectar a un ordenador a través de un puerto USB con el fin de transferir información entre el ordenador y el dispositivo externo. A veces, el dispositivo está formado por una unidad de acceso que permite acceder a contenedores o volúmenes de información extraíbles. Un ejemplo lo encontramos en la unidad de lectura o escritura de discos compactos y los discos compactos (como CD o DVD).

Los dispositivos extraíbles necesitan un tratamiento adicional; hay que proporcionar los mecanismos necesarios para poder insertar o extraer dinámicamente estos dispositivos sin necesidad de detener el sistema. Por cuestiones de eficiencia, el sistema puede guardar en la memoria información sobre los cambios que se quieren realizar en un dispositivo y retrasar la escritura en el

mismo. Por lo tanto, es necesario materializar los cambios en el dispositivo en cuestión antes de proceder a su extracción, ya que en caso de no hacerlo estos cambios se pierden[2].

(2)Estas optimizaciones se utilizan en la mayoría de sistemas de gestión de ficheros. Por eso es importante detener el sistema de forma correcta pidiendo al propio sistema que se detenga (*shutdown*). De esta manera nos aseguramos de que el sistema sincroniza el contenido de la memoria y el de los dispositivos antes de detenerse y no se pierden las actualizaciones realizadas.

2) La capacidad de almacenaje de los dispositivos: los dispositivos pueden servir para visualizar información, recogerla o almacenarla. Los dispositivos de almacenamiento tienen una capacidad, o un espacio, concreto que puede variar entre los diferentes tipos de dispositivos. El sistema operativo es el encargado de gestionarla. Las características de estos dispositivos varían en función de los aspectos siguientes:

a) La geometría: los dispositivos de almacenamiento pueden tener el espacio organizado de formas diversas, siguiendo geometrías diferentes así, por ejemplo, un disco organiza el espacio en pistas, caras y sectores y una cinta, en bloques.

b) El tipo de almacenaje, permanente o temporal: los dispositivos de almacenamiento pueden contener la información de manera permanente, hasta que se borra explícitamente, o bien pueden guardarla de forma temporal. En este último caso, la información se borra automáticamente cuando se cumple una condición concreta. Por ejemplo, en un fichero temporal la información almacenada se destruye al cerrarlo.

3) El tipo de unidad de transferencia: las unidades de transferencia entre el sistema y los dispositivos pueden ser diversas. En general, distinguimos los dos tipos siguientes:

a) Dispositivos de caracteres: son dispositivos que transfieren **caracteres**, por ejemplo, un terminal es un dispositivo de caracteres de entrada/salida formado por una pantalla y un teclado porque cada vez que pulsamos una tecla el carácter se envía al sistema.

b) Dispositivos de bloques: son dispositivos que transfieren bloques de caracteres, entendiendo como *bloque* un conjunto de caracteres. Por ejemplo, el disco es un dispositivo de bloques, ya que cada vez que se quiere leer o escribir en el disco se tiene que transferir como mínimo un bloque, por ejemplo un sector del disco con 512 bytes.

4) La velocidad de transferencia de los dispositivos: la velocidad de transferencia, o ancho de banda del dispositivo, es otro de los factores que diferencian los dispositivos. Se mide en bits/segundo o en bytes/segundo. Por ejemplo, un cable USB 1.0 puede llegar a una velocidad de 12 Mbit/s, mientras que un USB 2.0 puede llegar a los 480 Mbit/s y un USB 3.0 puede alcanzar los 4,8 Gbit/s.

La velocidad de transferencia de los dispositivos normalmente es mucho más pequeña que la velocidad del procesador. Esta diferencia de velocidades hace que el sistema, al gestionar los dispositivos, tenga que aplicar técnicas que permitan reducir los tiempos de espera de los procesos. Veremos algunas de estas técnicas en el apartado sobre optimización.

5) El tipo de codificación de la información: la información que se lee o se escribe en un dispositivo tiene que seguir unas normas de codificación que dependen de cada dispositivo en concreto. Por ejemplo, un terminal orientado a caracteres puede visualizar la información codificándola de diferentes maneras, que pueden ser ASCII de 7 bits o de 8 bits, UTF-8, ISO-8859-1 o muchas otras. En este último caso, los caracteres por encima del código ASCII 127 dependen del alfabeto (catalán o castellano, por ejemplo) y del fabricante. Por lo tanto, al visualizarlos, el resultado que se obtiene puede ser diferente del que se esperaba.

Cuando intentamos visualizar el contenido de un fichero ejecutable (codificado en binario) en la pantalla, se produce una situación sorprendente porque muchos caracteres son no imprimibles.

6) Condiciones de error: los tipos de error posibles, la forma de notificarlos, sus consecuencias y las posibles respuestas difieren de un dispositivo a otro. Por ejemplo, si intentamos leer de una impresora, tenemos que recibir un error, ya que ésta es un dispositivo de salida pero no de entrada. En cambio, con un ratón pasa lo contrario, ya que es un dispositivo de entrada y tiene que notificar un error si intentamos escribir.

3.2. Características de acceso

Las características de acceso hacen referencia a la manera como el sistema operativo y los procesos acceden a la información que contienen los dispositivos. Son las siguientes:

1) Acceso por entrada o salida: los dispositivos pueden ser de entrada, de salida o de los dos tipos al mismo tiempo. En función de esta característica, una operación de acceso puede tener sentido o no. El sistema es el encargado de controlar qué tipo de acceso se intenta establecer y si es posible.

2) Acceso compartido o exclusivo: a un dispositivo de uso exclusivo sólo puede acceder un usuario al mismo tiempo, al contrario de lo que sucede con los dispositivos compartidos. Un ejemplo de dispositivo exclusivo es la impre-

sora, ya que no puede escribir dos documentos al mismo tiempo; si se mezclaran las líneas de dos o más documentos en una sola hoja de papel, ésta quedaría inservible. En el apartado de optimizaciones, veremos cómo se puede aligerar la restricción de no poder trabajar con dispositivos no compartibles.

3) Acceso secuencial, directo o indexado: el acceso a una información concreta contenida en un dispositivo de almacenaje se puede establecer básicamente de las tres formas siguientes:

- **Secuencialmente**: se accede a la información que sigue a la última información a la que se ha accedido. Implica el mantenimiento de esta posición, mediante el puntero de lectura/escritura, por parte del sistema operativo.

- **Directamente**: se accede de forma directa a una posición concreta.

- **De manera indexada**: se accede a la información mediante una clave lógica.

Cada dispositivo permite, en función de su estructura, uno o más de estos métodos de acceso. El sistema, mediante su gestión, puede ofrecer aquellos métodos que el dispositivo no acepta directamente.

4) Acceso síncrono o asíncrono: en el momento de acceder a un dispositivo puede ser que la información no esté disponible. Entonces, según el modo de funcionamiento utilizado, tenemos los casos siguientes:

- En **modo síncrono**, si la información no está disponible, el proceso que efectúa el acceso se esperará hasta que lo esté. El proceso pasará al estado *blocked*.

- En **modo asíncrono**, el proceso lanza una petición al sistema para realizar una entrada/salida, pero no se espera para conocer el resultado, lo que permite que el proceso se siga ejecutando. Cuando quiera saber si la entrada/salida ha acabado, se tendrá que sincronizar haciendo una petición al sistema operativo. Eso es necesario para saber si ya disponemos de un dato que se ha pedido antes de operar con ella. También puede ser necesario saber si se ha escrito un dato que queríamos escribir con el fin de reutilizar una posición de memoria o liberar un recurso.

Ved también

En el módulo 2, explicamos más ampliamente cómo funciona el estado *blocked*.

5) Otras características del acceso a los dispositivos de entrada: cuando se introducen datos desde algunos dispositivos, como los terminales, es necesario ver lo que se entra y, en caso de equivocación, se tiene que poder rectificar. Estas dos características de los terminales hacen referencia respectivamente a lo que se llama *echo* y al modo de edición *cooked*/*raw* (cocinado/crudo):

- **El hecho de activar o no el modo *echo***: no utilizamos el *echo* cuando, por ejemplo, queremos introducir una contraseña.

Ejemplo

El carácter ASCII 127 (*delete*) produce la eliminación de uno de los caracteres entrados.

- **El modo de edición *cooked*** consigue que algunos caracteres que componen la entrada se interpreten como pedidos que alteran la información que se ha introducido. Con este modo activado, todos los caracteres que controlan la edición se interpretarán y, por lo tanto, no se entregarán al proceso que efectúa la operación de entrada. Así pues, un mismo dispositivo se puede comportar de maneras muy diferentes según cuál de estos dos modos esté activo. En general, el sistema es el encargado de implementarlas por software; sin embargo, algunos dispositivos las pueden tener implementadas directamente por hardware.

3.3. Características de control

Las características de control hacen referencia a la forma de indicar a los dispositivos lo que se quiere que hagan. Podemos distinguir los dos tipos siguientes:

1) Las características del controlador: el sistema controla los periféricos mediante el acceso a un conjunto de puertos que pertenecen al hardware llamado controlador. Las características de estos controladores condicionan la manera como el sistema gestiona las entradas/salidas, que puede ser por interrupciones, por encuesta o con acceso directo a la memoria (DMA[3]), entre otros. El sistema operativo esconde a los usuarios todas estas particularidades.

(3)DMA es el acrónimo correspondiente a la expresión inglesa *Direct Memory Access*.

2) Los lenguajes de pedidos: algunos dispositivos utilizan lenguajes de control para configurar el tipo de entrada/salida que tienen que hacer. Por ejemplo, algunas impresoras láser utilizan el lenguaje *PostScript* para especificar el formato de la información que tienen que imprimir. Otro ejemplo serían los caracteres de control que se utilizan en un terminal para editar las entradas o para indicar si la salida tiene que estar en negrita, subrayado o como sea.

3.4. Consecuencias de la diversidad de los dispositivos

Tal como se puede ver en la relación de características que acabamos de mencionar, los dispositivos son muy diferentes entre sí. Eso ocurre no sólo entre tipos diferentes, sino también dentro de los del mismo tipo. Como consecuencia de estas diferencias, tienen lugar las situaciones siguientes:

- Se accede a los dispositivos y se manipulan mediante **operaciones diferentes** y con parámetros diferentes.
- Los dispositivos pueden producir **resultados diferentes** como respuesta a operaciones que en un principio podrían parecer iguales.
- Finalmente, y como resultado de las dos consideraciones anteriores, los dispositivos producen **errores diferentes**, incluso en situaciones análogas.

Esta diversidad hace que, como usuarios, deseemos independizarnos de esta multitud de casos diferentes. El sistema operativo es el encargado de conseguir este objetivo; mediante su gestión proporciona una máquina virtual que logra que toda esta diversidad sea transparente para el usuario y así le ofrece un grado más elevado de independencia de los dispositivos. En los apartados siguientes, veremos cómo se puede conseguir.

Ved también

Recordad que presentamos el concepto de máquina virtual en el apartado 1 del módulo 2, "El sistema operativo: una máquina virtual".

4. Los dispositivos físicos, los lógicos y los virtuales

En este apartado, clasificamos los dispositivos en tres categorías, cada vez más abstractas, orientadas a permitir la escritura de programas independientes de los dispositivos. Se trata de los dispositivos:

- físicos
- lógicos
- virtuales

4.1. Los dispositivos físicos

Un dispositivo físico es un dispositivo que existe físicamente. Lo forman el periférico y su hardware de control (*device controller*), que constituyen la parte física, y el software que los gestiona, que llamamos programa controlador (*driver*). El programa controlador se comunica directamente con el controlador del dispositivo a través de registros de control.

Por norma general, son dispositivos físicos todos los que están asociados a un periférico. Las impresoras, los teclados, los ratones o los escáneres son algunos ejemplos de dispositivos físicos.

4.2. Los dispositivos lógicos

Los dispositivos físicos son los dispositivos más evidentes en un sistema y rápidamente los identificamos como tales. No obstante, en el sistema hay otro tipo de dispositivos que no están necesariamente asociados a un periférico, son los dispositivos lógicos.

Ejemplo

Un disco simulado en la memoria es un dispositivo que no existe físicamente, sino que es el resultado de un algoritmo que simula, en este caso, un dispositivo físico, un disco.

Un **dispositivo lógico** es el resultado de un software del sistema que define este dispositivo. Un dispositivo lógico puede tener asociado un dispositivo físico o no. Así pues, los dispositivos lógicos, a diferencia de los físicos, pueden estar formados únicamente por su programa controlador.

Un ejemplo de dispositivo lógico es un fichero. Probablemente nos interesa guardar el fichero y lo almacenamos, por ejemplo, en un disco o en un lápiz USB. Por lo tanto, en este caso hay algún dispositivo físico asociado al dispositivo lógico fichero. Sin embargo, en algunos casos el dispositivo lógico no tiene asociado un dispositivo físico de entrada/salida, sino que es el sistema operativo el que gestiona y almacena la información necesaria en su memoria.

Eso pasa, por ejemplo, con un dispositivo llamado *pipe*, que permite comunicar información entre procesos a través de él mismo y consiste, básicamente, en una zona de memoria (memoria intermedia) gestionada por el sistema operativo mismo.

Lo que el sistema operativo nos ofrece son los dispositivos lógicos. Como usuarios de un sistema, podemos indicar que queremos usar un dispositivo lógico. De hecho, no podremos acceder directamente a los dispositivos físicos, eso queda reservado al sistema operativo. Sin embargo, sí podremos usar los dispositivos físicos indirectamente a través de los dispositivos lógicos. Por ejemplo, no indicaremos que queremos leer una cara, una pista y un sector de un disco, sino que indicaremos que queremos leer un fichero y eso puede provocar el acceso a unas posiciones concretas de un disco.

Otros **ejemplos de dispositivos lógicos** son los que presentamos a continuación:

1) El **nulo**, que es un dispositivo de entrada/salida en el que podemos escribir todo lo que queramos y que siempre está vacío. En realidad, es un dispositivo que ignora cualquier cosa que se escriba y que en lectura no devuelve nunca ninguna información. Se utiliza para dejar de lado resultados no deseados. Este dispositivo se basa exclusivamente en el software que lo define, sin ningún dispositivo físico asociado.

El nulo se utiliza para dejar de lado los mensajes sobre el estado de la evolución de un proceso que salen generalmente en pantalla y que indican que el proceso trabaja.

2) La **ventana**, que es un dispositivo lógico de entrada/salida que combina cuatro dispositivos físicos: una pantalla, la memoria, un teclado y un dispositivo apuntador, como puede ser un ratón o un lápiz óptico. La ventana es un área gráfica almacenada en la memoria que se representa total o parcialmente en una pantalla. De los cuatro dispositivos mencionados, tenemos que:

- la pantalla es el dispositivo donde se reflejan las salidas,
- el teclado es el medio para introducir las entradas,
- el dispositivo apuntador nos permite llevar a cabo ciertas operaciones de control en la ventana, como cerrarla, variar el área y la posición o seleccionar una entre varias para poder efectuar entradas desde el teclado,
- la memoria almacena toda la información relacionada con la ventana.

A diferencia del dispositivo nulo, el software que configura las ventanas se basa en la manipulación de los programas controladores de los dispositivos físicos que lo componen.

3) La **cola de mensajes**, que es un dispositivo de entrada/salida pensado para comunicar procesos y que puede tener las dos resoluciones siguientes:

- una totalmente lógica, basada en una estructura de datos y unas operaciones que la manipulan, que se utiliza para comunicar procesos que se ejecutan en una misma máquina;

- otra basada en dispositivos físicos, como el módem o la placa de acceso a la red, que se utiliza para comunicar procesos que se ejecutan en máquinas diferentes.

4) El **espacio lógico**, que es una agrupación de información almacenada a la memoria física mediante el uso de la memoria virtual. El espacio lógico de un proceso también se puede ver como un dispositivo que se puede leer o en el que se puede escribir. Esta visión de los espacios lógicos no está lejos del concepto de fichero y, por lo tanto, puede recibir el mismo tratamiento.

Ved también

Consultad el apartado 3 del módulo 3, "La gestión de la memoria", dedicado a la memoria virtual.

El espacio lógico tiene utilidad para aplicaciones como los depuradores de programas (*debugger*), que necesitan acceder al espacio lógico del proceso que están depurando.

4.3. Los dispositivos virtuales

Los **dispositivos virtuales**, también conocidos como **canales virtuales** o simplemente **canales**, representan el nivel más abstracto e independiente del dispositivo. Un dispositivo virtual es un dispositivo que, a priori, no está asociado a ningún dispositivo concreto.

El programa lo utiliza igual que cualquier otro dispositivo, pero sin saber de entrada en qué dispositivo en concreto se efectuarán las operaciones que se le especifican. En el tiempo de ejecución del programa será donde el sistema asocie un dispositivo concreto al dispositivo virtual mediante el código del sistema operativo, como mostramos en la figura 2. Podremos asociar un dispositivo virtual a alguno de los dispositivos visibles como usuarios del sistema operativo, es decir, a un dispositivo lógico.

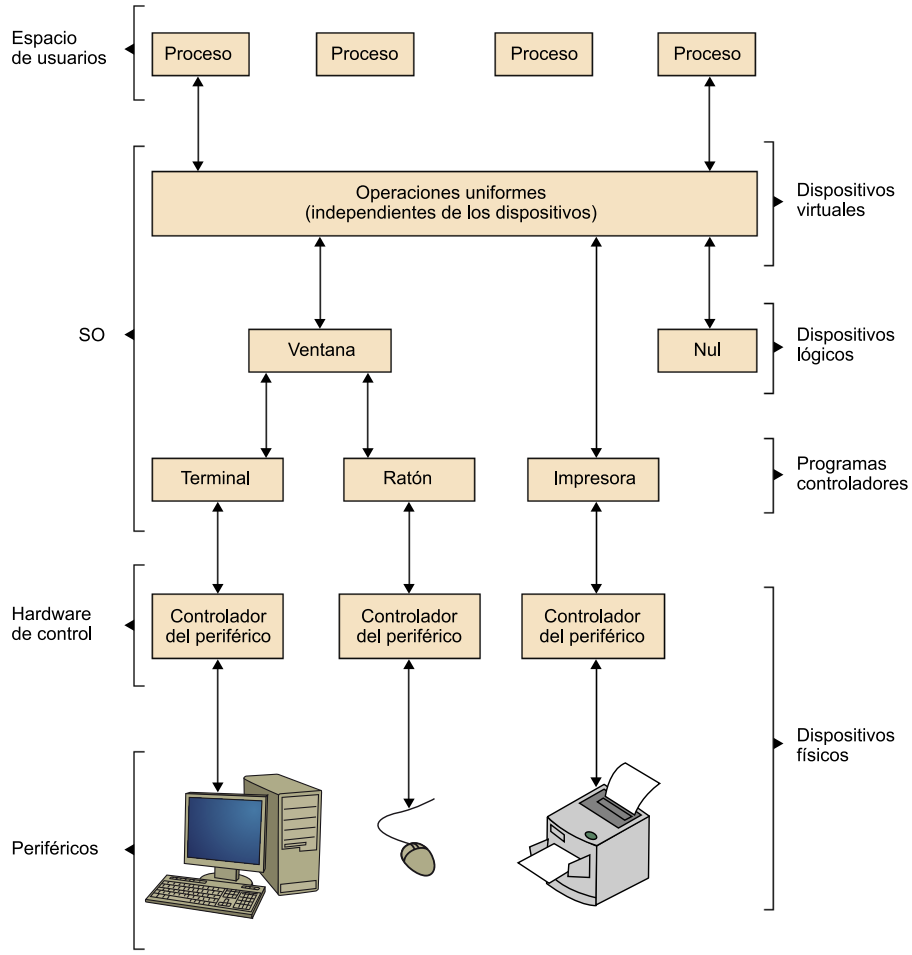


Figura 2. Dispositivos físicos, lógicos y virtuales

5. La independencia de los dispositivos

Como programadores, nos interesa poder hacer programas que sean portables y que se puedan adaptar a situaciones diferentes. Esta necesidad incluye el entorno de entrada/salida en el que se ejecutarán. Hemos visto que los dispositivos tienen características que hacen que su gestión y las operaciones que se pueden llevar a cabo sean diferentes. El objetivo del sistema operativo es proporcionar una máquina virtual que uniformice los dispositivos a la vez que respeta las particularidades para facilitar así a los usuarios el uso que harán del mismo. Eso se consigue independizando el código de los programas respecto de los dispositivos que utilizan y de las operaciones con las que se dirigen.

Algunos casos, como un programa que copia ficheros, una sencilla máquina de escribir, un editor de líneas y un visualizador de ficheros, se pueden reducir a un solo programa que escribe en un dispositivo lo que ha leído en otro. A pesar de que se pueda hacer esta simplificación, y dado que los dispositivos con los que tendrán que trabajar los programas anteriores son tan diferentes, difícilmente podemos pensar que un único programa fuente pueda servir para todos los dispositivos. A lo largo de este apartado veremos cómo, gracias a la gestión del sistema operativo, es posible efectuar todas estas operaciones con un único código fuente.

Ved también

Consultad el ejemplo comodín del subapartado 7.1.2 de este módulo didáctico.

5.1. Uso de los dispositivos virtuales

El primer paso para independizar los programas del entorno de entrada/salida con el que trabajarán se basa en la utilización de los dispositivos virtuales. En el apartado anterior hemos dicho que podemos asociar un dispositivo virtual a alguno de los dispositivos visibles como usuarios del sistema operativo, es decir, a un dispositivo lógico. Esta **asociación entre un dispositivo lógico y un dispositivo virtual** se puede materializar de las dos maneras siguientes:

a) En **la asociación implícita**, también llamada **redireccionamiento de las entradas/salidas**, el programa ya encuentra la asociación hecha en el instante en el que inicia su ejecución. El sistema y el proceso que han iniciado la ejecución del programa son los encargados de establecer la asociación. Los dispositivos virtuales asociados de manera implícita se llaman **dispositivos estándar** y, en general, tenemos tres, que son el estándar de entrada (*standard input*), el de salida (*standard output*) y el de error (*standard error*).

b) La asociación explícita entre un dispositivo lógico y uno virtual hecha por el programa mismo durante la ejecución. Para efectuarla, necesita una operación específica que asocie un dispositivo virtual a un dispositivo lógico. A partir del momento en el que se ha hecho la asociación, el programa especificará todas las operaciones de entrada/salida del dispositivo mediante el dispositivo virtual.

Ved también

Ved los dispositivos estándar en el apartado 7.1 de este módulo didáctico.

En el apartado 4 del módulo 5, veremos la manera como las operaciones utilizadas en la asociación explícita se utilizan para llevar a cabo verificaciones de protección.

5.2. Definición de operaciones uniformes

Para conseguir la independencia de los programas respecto de las entradas/salidas no basta con utilizar dispositivos virtuales. No sirve de nada hacer que los programas trabajen en dispositivos virtuales si éstos tienen que utilizar operaciones específicas de los dispositivos concretos con los que están asociados. Para evitar que los programas tengan que conocer las particularidades de los dispositivos reales, el sistema ha de uniformizar todas las operaciones que se pueden hacer en los dispositivos.

Para hacerlo, hay que conocer cuáles son las operaciones más utilizadas por los programas y cuáles son las más comunes en la mayoría de los dispositivos. Estas operaciones se tienen que definir en un conjunto nuevo de operaciones independientes de las características de los dispositivos, que tiene que ser ofrecido por el sistema. Con el fin de proporcionar esta nueva funcionalidad, el sistema añade una capa de software por encima de los programas controladores (código específico de cada dispositivo), que son las operaciones independientes de los dispositivos. El programador usará esta interfaz uniforme para indicar las operaciones que hay que hacer en los dispositivos virtuales. Internamente, el sistema operativo utilizará las rutinas específicas para el dispositivo concreto asociado al dispositivo virtual. Esta información se determina en tiempo de ejecución gracias a la información almacenada internamente en el momento de establecer la asociación entre el dispositivo virtual y un dispositivo lógico.

Si analizamos las operaciones de entrada/salida podemos ver que hay operaciones básicas de acceso y de control, orientadas a dar un acceso uniforme, y operaciones específicas de control, orientadas a poder gestionar las particularidades de cada dispositivo. A continuación, estudiaremos cada una de estas operaciones.

5.2.1. Operaciones básicas de acceso

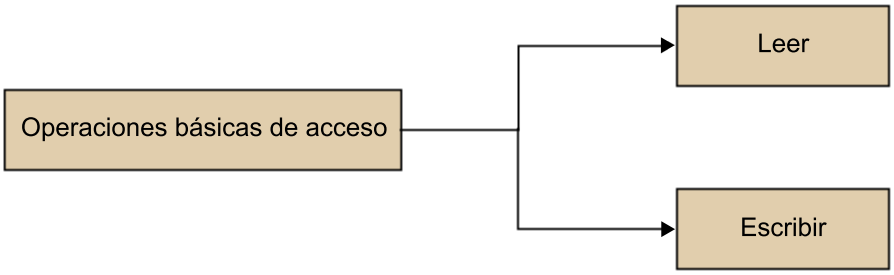


Figura 3

Las operaciones básicas de acceso son dos, **leer** y **escribir**, y nos tienen que permitir acceder a la información de manera independiente del dispositivo. Su funcionalidad tiene que ser compatible con la dinámica de funcionamiento de todos los dispositivos. Esta dinámica incluye el tipo de acceso (secuencial o directo), el tipo de los datos que se tienen que transferir y su estructura.

El tipo de acceso más habitual es el secuencial. No obstante, se tiene que dar opción al hecho de que las aplicaciones puedan establecer accesos directos a la información. Para permitirlo, es necesario proporcionar la operación *posicionar*, que se analiza junto con las operaciones básicas de control.

Ved también

Encontraréis información sobre el acceso secuencial en el subapartado 3.2 de este módulo didáctico.

Para especificar qué información se tiene que transferir basta con dos parámetros:

- *buf*: especifica la zona de memoria desde donde se extraerán los datos con la operación o en los que se entrarán.
- *cont:* indica el número de datos máximo que se tienen que transferir.

Si no introducimos más parámetros, tenemos que suponer que accedemos siempre al mismo tipo de datos. Si no fuera así, habría que incluir otro parámetro que especificara con cuál de los tipos de datos reconocidos por el sistema se quiere establecer la operación de entrada/salida.

Ahora bien, la mayoría de sistemas reducen todos los tipos de datos posibles a uno solo, el más elemental, que es el byte. El motivo de esta decisión es sencillo: no se pueden reconocer todos los tipos posibles; por más tipos que reconociera el sistema, siempre habría aplicaciones con tipos nuevos. Por otra parte, cualquier tratamiento específico se puede hacer en la biblioteca o en el programa. Esta idea encaja perfectamente con uno de los principios de los sistemas operativos, que es el de ofrecer herramientas para que los usuarios del sistema o su administrador puedan establecer políticas que se adapten a sus necesidades particulares.

Con el fin de completar la lista de parámetros, se debe indicar el dispositivo virtual en el que se quiere trabajar (*disp*) y, finalmente, una vez ejecutadas estas operaciones, el sistema tiene que devolver información sobre el estado en el que han finalizado.

Así pues, las operaciones de acceso mínimas quedan tal como se indica a continuación:

- estado = leer (*disp, buff, cont*)
- estado = escribir (*disp, buff, cont*)

5.2.2. Operaciones básicas de control

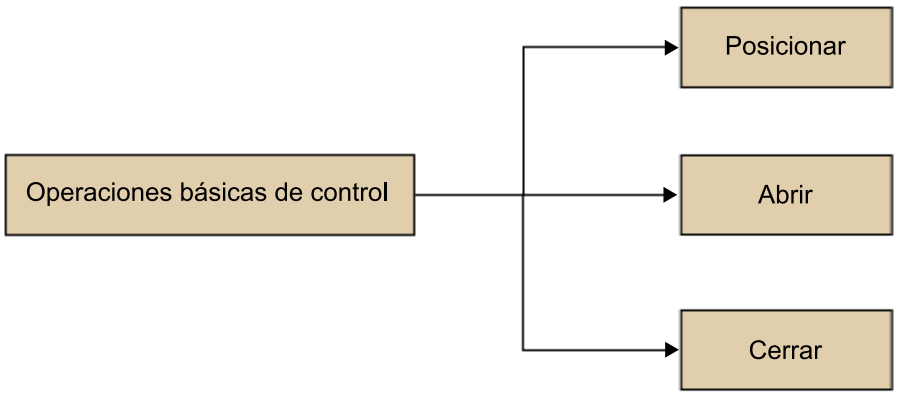


Figura 4

Existen tres operaciones básicas de control, que son **posicionar**, **abrir** y **cerrar** y que permiten llevar a cabo las operaciones de control más comunes que se ejecutan en los dispositivos.

Tal como hemos visto en el caso de las operaciones de acceso, es necesario proporcionar un mecanismo para poder establecer accesos directos. La operación *posicionar* nos tiene que permitir acceder directamente a cualquier posición dentro de la información de los dispositivos que lo permitan. Los parámetros necesarios para ejecutar esta operación son los dos siguientes:

- el dispositivo virtual en el que se quiere efectuar la operación (*disp*),
- la posición en la que, o desde donde, se quiere llevar a cabo la operación de acceso siguiente (*pos*).

El inicio y el final de una sesión de acceso a un dispositivo se definen con las operaciones *abrir* y *cerrar*. Las aplicaciones no suelen acceder a los dispositivos de manera aislada, sino que normalmente establecen una serie de accesos que siguen una misma pauta: o bien se escribe todo un fichero de manera secuencial o bien se modifica parcialmente mediante un acceso directo o bien se lee

secuencialmente, entre otros. En cada una de estas sesiones, los accesos que la componen tienen necesidades comunes. Podemos resumir estas necesidades en tres ideas básicas:

1) Asociar un dispositivo lógico a un dispositivo virtual de manera explícita.
2) Inicializar las estructuras de datos internos del sistema operativo necesarias para establecer los accesos (secuenciales y directos).
3) Verificar los derechos de acceso que tiene el proceso sobre el dispositivo.

Si no existieran las operaciones *abrir* y *cerrar* que enmarcan todos los accesos de una sesión, habría que cubrir las necesidades comunes para cada acceso y, en consecuencia, se multiplicaría el trabajo que tiene que hacer el sistema.

Al abrir **una sesión de trabajo con un dispositivo**, mediante la operación *abrir*, hay que indicar los datos siguientes:

- el nombre del dispositivo lógico (*nombre*),
- el tipo de acceso[4] que se quiere efectuar (*op*).

Con estos parámetros, el sistema tiene que verificar la existencia del dispositivo y los derechos de acceso del proceso a este dispositivo. Si el dispositivo existe y se tiene derecho a acceder a él, el sistema generará un dispositivo virtual que lo asociará, para esta sesión de trabajo, al dispositivo lógico. Al mismo tiempo inicializará, con el valor adecuado, el puntero de lectura/escritura necesario para establecer un acceso secuencial; este valor será el inicio de la información del dispositivo en caso de lectura o modificación o bien el final de la información en caso de añadir (*append*). Eso se hará para los dispositivos en los que tenga sentido, como por ejemplo en el caso de los ficheros.

Al **cerrar la sesión de trabajo con el dispositivo**, es decir, al ejecutar la operación *cerrar*, el sistema liberará las estructuras de datos necesarios para el acceso y deshará la asociación entre el dispositivo virtual y el lógico.

Las operaciones *abrir* y *cerrar* se utilizan para llevar a cabo de manera explícita todas las operaciones. Sin embargo, el sistema las puede efectuar de manera implícita al iniciar o finalizar un proceso, tal como hemos comentado en el caso de la asignación de dispositivos lógicos a virtuales.

Así, pues, las operaciones básicas de control podrían ser las siguientes:

- estado = posicionar(*disp, pos*)
- disp = abrir(*nom, op*)
- estado = cerrar(*disp*)

(4)Los accesos pueden ser de lectura, de escritura o de lectura y escritura simultáneamente.

Ved también

Encontraréis más información sobre el control de accesos a los dispositivos en el subapartado 4.3 del módulo didáctico 5.

5.2.3. Operaciones específicas de control

Finalmente, el sistema permite, mediante la operación *control*, que los usuarios que lo necesiten puedan gestionar las particularidades de los dispositivos.

Según el sistema, podemos encontrar la operación de control de las dos formas siguientes:

a) como una operación única con parámetros variables que dependen de la función que se quiere efectuar y del dispositivo en el que se efectúa,
b) como tantas operaciones diferentes como funciones se deben realizar.

6. Optimización de la entrada/salida

Es interesante conocer algunas de las optimizaciones que establece el sistema con respecto a la gestión de la entrada/salida, como el almacenaje en la memoria intermedia (*buffering*) y la gestión de las colas (*spooling*). Acto seguido veremos estas dos técnicas con más detalle.

6.1. El almacenamiento en la memoria intermedia

A veces, la velocidad de los dispositivos no se adapta a la velocidad con la que el proceso se está ejecutando, ya sea porque los dispositivos son más lentos o bien porque el proceso trabaja a rachas. Con el fin de solucionar este problema y conseguir que ni los dispositivos ni los procesos se tengan que esperar mutuamente, el sistema operativo utiliza la memoria intermedia (*buffer*). Así, esta memoria tiene la función de amortiguar las diferencias de velocidad.

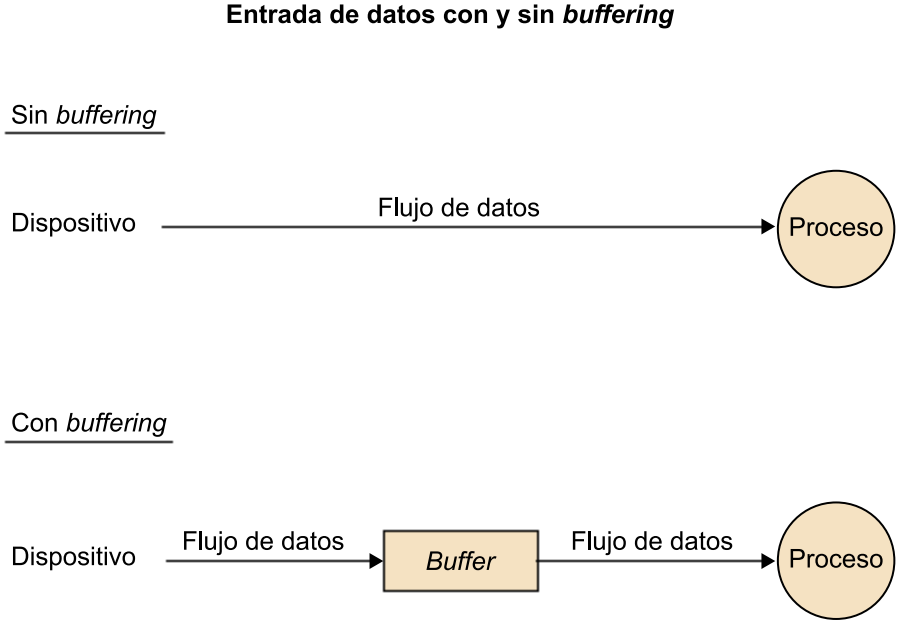


Figura 5. El almacenaje en la memoria intermedia

En el esquema con la memoria intermedia, el dispositivo produce información y el proceso la consume. El dispositivo tiene que pasar la información a la memoria intermedia y sólo surgen problemas si está llena. Por otra parte, el proceso consume únicamente información de la memoria intermedia y sólo se tiene que esperar si ésta está vacía.

6.2. La gestión de las colas

La técnica de la gestión de colas o SPOOL (*Simultaneous Peripherical Operation On-Line*) se basa en el hecho de que las entradas/salidas de un proceso se implementen con un paso intermedio utilizando dispositivos compartibles de

gran capacidad de almacenaje y rápidos (como discos o memoria). Los procesos de usuario no usan directamente el dispositivo final, sino que ponen en cola su petición en una memoria intermedia predeterminada (por ejemplo, un fichero en un disco o en una zona de memoria) y pueden continuar su trabajo sin esperar que finalice la entrada/salida. El sistema operativo, a través de un proceso encargado de este trabajo y llamado gestor de colas, irá tratando las peticiones de cola en la memoria intermedia a medida que el dispositivo final esté disponible. Eso permite mejorar el rendimiento en un sistema multiprogramado cuando se utilizan periféricos más lentos que el procesador, ya que los procesos no deben esperar que el recurso final esté disponible ni que se complete la entrada/salida. Permite también que diversos procesos puedan avanzar de forma concurrente a pesar de intentar utilizar un dispositivo no compartible, como por ejemplo una impresora.

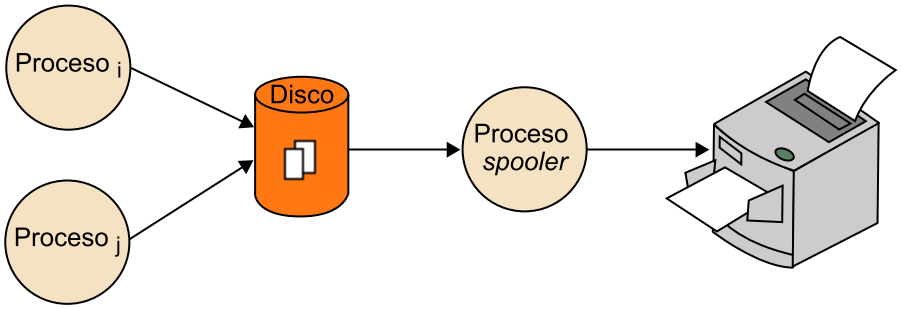


Figura 6. Impresión de un documento

Un caso muy corriente es el de trabajar con un procesador de textos e iniciar una impresión de un documento relativamente largo. Si no se utiliza la técnica de la gestión de colas (*spool*), el computador, y por lo tanto tampoco el usuario, no podrá hacer nada más hasta que no acabe la impresión. Un efecto adicional de la utilización de la técnica de la gestión de colas es la sensación que tienen los usuarios de estar utilizando todos al mismo tiempo la impresora, aunque es un dispositivo no compartible. Eso es posible gracias al hecho de que el dispositivo intermedio (en este caso el disco) es compartible.

Ved también

Podéis consultar la ejecución concurrente de procesos en el subapartado 2.3 del módulo 1, "Introducción a los sistemas operativos".

Ved también

Ved los dispositivos compartibles y no compartibles en el subapartado 3.2 de este módulo.

7. Ejemplos

7.1. Entrada/salida en el UNIX

El UNIX, como todos los sistemas operativos, ofrece una visión de los dispositivos que hace que las características propias de cada uno sean transparentes para el usuario. En el UNIX, un dispositivo es cualquier objeto en el que se pueden llevar a cabo operaciones de lectura o escritura.

El punto de vista del usuario del UNIX es el de un dispositivo en el que se puede escribir o leer secuencias de bytes utilizando un entorno único y bien definido.

El sistema operativo UNIX reconoce los dos tipos básicos de dispositivos que mencionamos a continuación:

1) Los **dispositivos de bloques**, como los discos, las cintas y los ficheros. Permiten el acceso directo, ya que la unidad de direccionamiento es un bloque de tamaño fijado por el sistema.

Para estos tipos de dispositivos, el UNIX utiliza una memoria caché de bloques con la finalidad de mejorar el rendimiento en sus accesos, ya que el tiempo de acceso a la memoria es mucho menor que el tiempo de acceso a algunos dispositivos de bloques. Básicamente, consiste en utilizar una zona de memoria, compuesta por diversas memorias intermedias y que gestiona el sistema operativo con el fin de evitar accesos al disco, reutilizando información previamente leída o retrasando la escritura de información nueva de forma que las actualizaciones parciales se hagan en la memoria[5].

[(5)]Sin embargo, hay que sincronizar de vez en cuando la información que hay en la memoria con la que hay en el dispositivo, con el fin de evitar inconsistencias en caso de sufrir una caída del sistema. También hay que hacerlo en el momento en el que se cierra el sistema (*shutdown*) como se avanzó en el apartado 3.1 de este módulo didáctico.

Figura 7

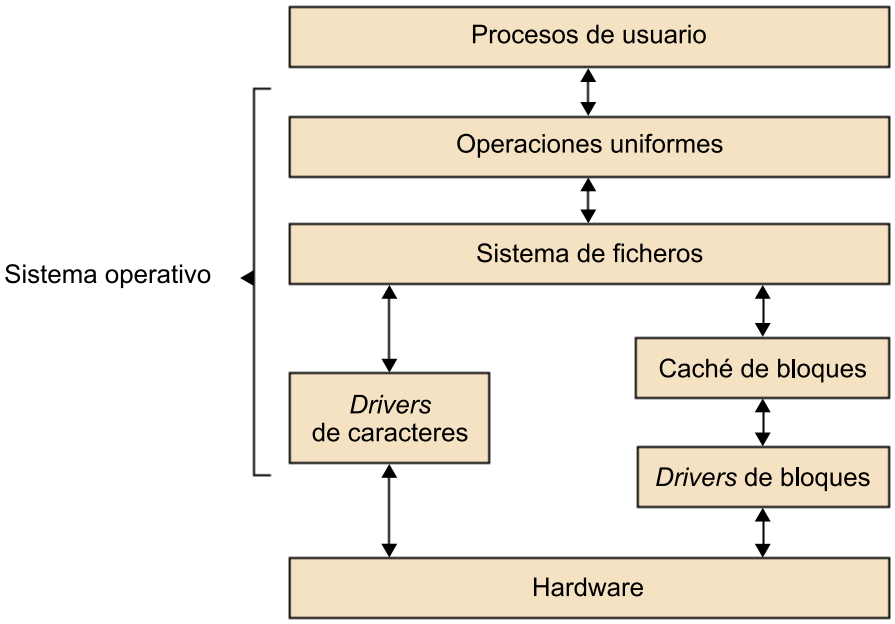
2) Los **dispositivos de caracteres**, como los terminales, las impresoras o los ratones, en general, todos aquellos dispositivos que no utilizan la memoria caché de bloques. Suelen ser dispositivos en los que se tienen que establecer los accesos de manera secuencial.

Internamente, los diferentes tipos de dispositivos se identifican según si son de bloques o de caracteres y por un número llamado *major*. Los diferentes dispositivos de un mismo tipo se identifican mediante otro número, llamado *minor*.

Así pues, internamente, un dispositivo se identifica completamente mediante el trío tipo básico, *major*, *minor*.

El sistema gestiona los dispositivos mediante programas controladores asociados a cada tipo de dispositivo, tal como muestra la figura siguiente:

Figura 8. Esquema de las entradas/salidas en UNIX



El usuario que quiere dirigirse a los dispositivos lógicos del UNIX lo tiene que hacer a través del sistema de ficheros, mediante el cual el UNIX designa **ficheros especiales**.

Ejemplo

En el UNIX, una ventana tiene el nombre /dev/pts/0, el disco se llama /dev/sda, etc.

El acceso básico a los dispositivos que ofrece el UNIX a los procesos es el acceso secuencial. No obstante, también proporciona llamadas para que se puedan establecer accesos directos.

Otra característica importante es el control de los accesos simultáneos en un mismo dispositivo. Por defecto, el UNIX permite que más de un proceso acceda de forma concurrente al mismo dispositivo, aunque proporciona las herramientas necesarias, mediante las llamadas de control, para que los usuarios puedan trabajar con modelos de exclusión mutua.

Los procesos utilizan los *file descriptors*[6] para acceder a los dispositivos una vez ya se han abierto. Desde el punto de vista del usuario, un ***file descriptor*** es un identificador del dispositivo virtual que forma parte del entorno de ejecución del proceso.

[6]Los *file descriptors* son los dispositivos virtuales, o canales, del UNIX.

En el UNIX existen tres *file descriptors*, que normalmente están asignados de manera implícita y que por convenio tienen un significado concreto, que son el 0 o canal de entrada estándar (*standard input*), el 1 o canal de salida estándar (*standard output*) y el 2 o canal de error estándar (*standard error*). Los dispositivos de entrada y de salida estándares hacen referencia a la entrada/salida por defecto. El canal de error estándar se utiliza para comunicar las situaciones anómalas o de control en las que se encuentra el proceso durante su ejecución.

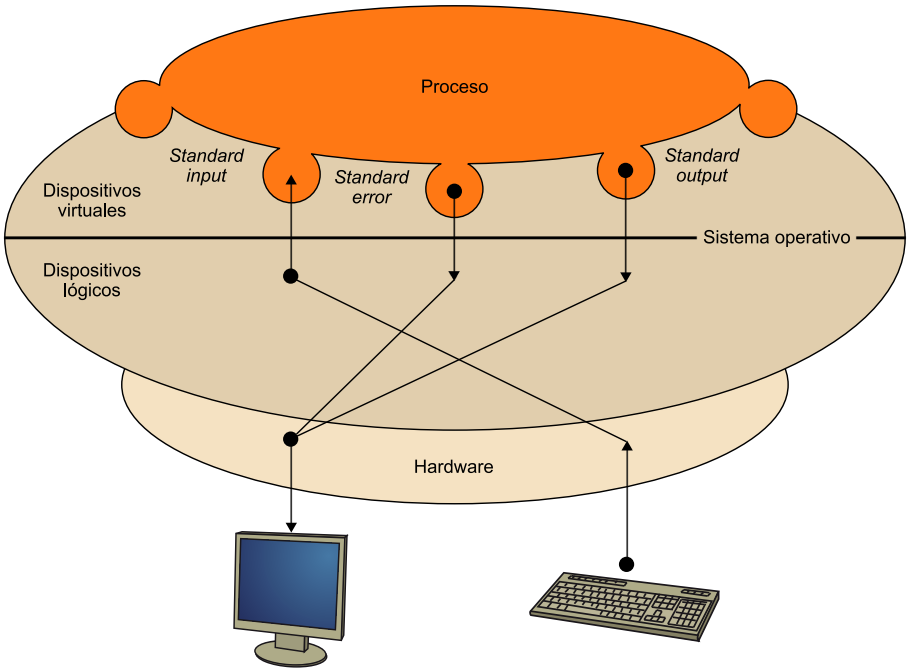


Figura 9. Entrada/salida estándar de los procesos

Ejemplo

Un compilador puede leer el fichero fuente por su entrada estándar, escribir el ejecutable con el de salida estándar e indicar los errores o hacer advertencias sobre el estilo del programa por su canal de error estándar.

7.1.1. Las *pipes*, un caso especial de dispositivo

Una ***pipe*** es un dispositivo lógico destinado a comunicar procesos. Su funcionamiento es el de una cola de caracteres con una longitud fija en la que los procesos pueden leer y/o escribir.

Cuando un proceso quiere utilizar las *pipes* se puede encontrar con los dos problemas siguientes:

- Si intenta leer una *pipe* vacía, quedará bloqueado hasta que algún otro proceso escriba algún carácter a fin de que el proceso bloqueado pueda efectuar la lectura. El lector de una *pipe* vacía sólo se bloqueará si hay escritores potenciales, es decir, procesos que tengan la *pipe* abierta para escribir.

- Si intenta escribir en una *pipe* llena, quedará bloqueado hasta que algún otro proceso lea suficientes caracteres de la *pipe* para que el proceso bloqueado pueda efectuar la escritura.

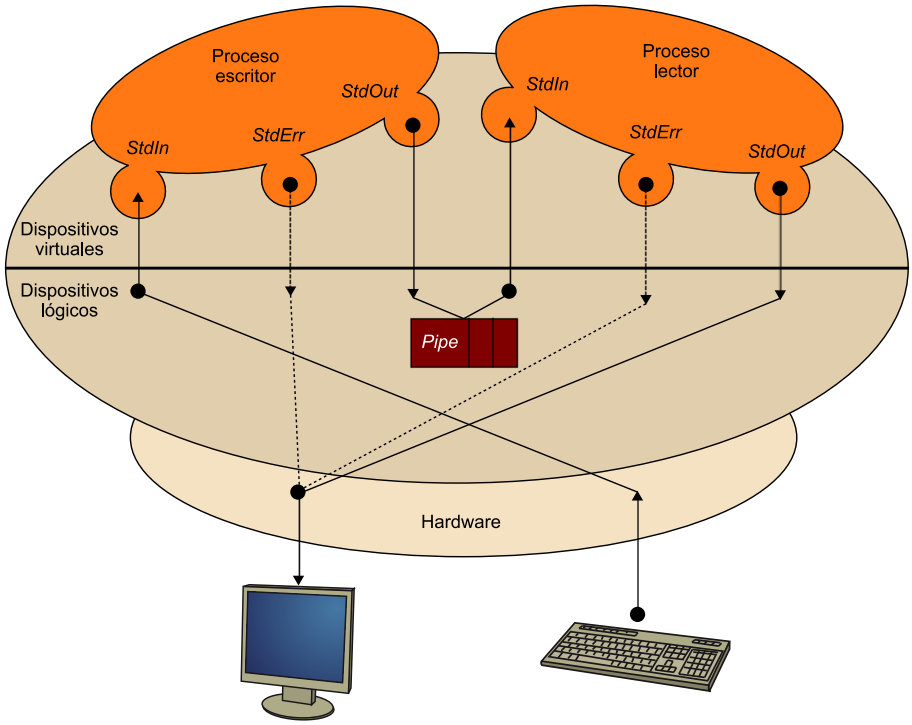


Figura 10. Comunicación de dos procesos de UNIX mediante una *pipe*

Existen dos tipos de *pipes*, que son las las llamadas simplemente *pipes* y *named pipes*.

- Una ***pipe*** es un dispositivo que se crea en el momento en el que un proceso lo abre y que se destruye cuando el último proceso que lo tiene abierto lo cierra. Es un dispositivo que no se puede abrir de manera explícita mediante la operación *abrir* (*open*). Una vez creada, los diferentes procesos que quieren utilizarlo lo tienen que asociar a un dispositivo virtual[7] de manera implícita. A causa de esta particularidad, las *pipes* no tienen asociado ningún nombre que aparezca en el sistema de ficheros.

- Una ***named pipe*** tiene un comportamiento similar al de una *pipe* pero, a diferencia de ésta, aparece en el sistema de ficheros. Por lo tanto, los

[7] En el UNIX, un dispositivo virtual es un *file descriptor*.

procesos la pueden abrir como si fuera un fichero, siempre que tengan derechos para hacerlo.

7.1.2. El punto de vista desde las llamadas al sistema

El objetivo de este subapartado es dar un repaso breve por las principales llamadas al sistema relacionadas con la entrada/salida. En ningún caso pretendemos establecer una descripción exhaustiva de todas sus posibilidades ni de su sintaxis.

En general, el sistema operativo UNIX tiene una estructura de llamadas al sistema idéntica a la que hemos descrito a lo largo de este módulo. Las correspondencias entre las llamadas del UNIX y las que hemos visto en las operaciones uniformes son las que podéis ver en la tabla siguiente:

Operaciones uniformes	
Llamadas al sistema	Llamadas de UNIX
leer	read
escribir	write
posicionar	lseek
abrir	open
cerrar	close
control	ioctl
	fcntl

Operaciones de control

ioctl controla el dispositivo lógico asociado a un *file descriptor*.

fcntl controla las características del *file descriptor* y de la sesión de entrada/salida que representa.

La interfaz de las llamadas al sistema y los ejemplos de utilización están disponibles en los recursos del aula.

El proceso comodín

Un ejemplo de uso de las llamadas de entrada/salida del UNIX es un proceso que escribe en la salida estándar lo que ha leído en la entrada estándar.

```
/* PROCESO COMODÍN */
main()
{

      char buff;
      int fi_in,fi_out;
```

```
        fi_out = 1;
        fi_in = read(0,&buff,1);

        /*mientras la entrada no encuentre el final del fichero,*/
        /*y no se produzcan errores de acceso*/
        while((fi_in>0)&&(fi_out>0))
        {
              fi_out = write(1,&buff,1);
              fi_in = read(0,&buff,1);
        }
}
```

En el caso del UNIX, los ficheros no tienen una marca de final de fichero. Si estamos leyendo, sabremos que hemos llegado al final del fichero cuando la llamada read vuelva a un 0.

Por sencillez, se muestra el código con las llamadas básicas, pero sería recomendable establecer un control de errores.

Un caso especial son las llamadas pipe y dup:

- La llamada al sistema pipe crea una *pipe* y dos *file descriptors*, uno para lectura y el otro para escritura. Una vez creada, se accede a la *pipe* como si fuera un dispositivo más.

- La llamada al sistema dup asocia un *file descriptor* nuevo al dispositivo asociado al *file descriptor* que se le pasa como parámetro.

Ved también

Podéis ver cómo se puede utilizar la llamada `dup` para redireccionar y enlazar los *file descriptores* de diferentes procesos en el apartado 4.1.2 del módulo 6, "La gestión de procesos".

7.1.3. El punto de vista desde el intérprete de comandos

En este subapartado, igual que en el anterior, introducimos las características principales de los intérpretes de comandos o *shells* del UNIX, sin pretender ser exhaustivos.

El tema más interesante del UNIX, desde el punto de vista de las entradas/salidas, es la facilidad con la que se pueden redireccionar los *file descriptors* estándares de los procesos desde el *shell* y el potencial que este hecho representa.

Para entender este potencial, hay que analizar los dos factores que mencionamos a continuación:

1) El primer concepto que se tiene que analizar es el de **filtro**. Un filtro es un programa que lee la entrada estándar, manipula la información y escribe el resultado de esta manipulación en la salida estándar. Esta operación la repite mientras no detecta un final de fichero.

Ejemplo de filtro

El comando tail del UNIX es un filtro que saca por la salida estándar las últimas diez líneas que ha leído por la entrada estándar.

Marca de fin de fichero

El sistema avisa a los procesos mediante la señal de fin de fichero que lee de un dispositivo en el que ya no se puede conseguir más información. Por ejemplo, si se accede secuencialmente a un fichero y se intenta leer información más allá del último carácter almacenado o bien en el caso de acceder a una *pipe* si ya se ha leído toda la información que contenía y no quedan más procesos que la tengan abierta para escribir. En el UNIX, esta señal se traduce en el hecho de que la operación read devuelve un número positivo menor que el número de caracteres que se ha solicitado leer.

2) El **redireccionamiento** es el segundo concepto. Mediante la sintaxis del *shell* se pueden redireccionar los *file descriptors* estándar de un proceso desde el pedido que los invoca.

El símbolo < se utiliza para redireccionar el estándar de entrada y el símbolo >, para el estándar de salida.

	Ejemplos de redireccionamiento
$ ps	ps muestra el estado de los procesos por la salida estándar actual.
$ ps > nom	Guarda al estado de los procesos en un fichero nom
$ tail	tail muestra en la salida estándar las últimas diez líneas que entran por la entrada estándar.
$ tail < nom	Muestra las diez últimas líneas del fichero nom por la salida estándar actual.

3) En tercer lugar, desde un pedido podemos **encadenar la ejecución de más de un programa** mediante *pipes*. Eso se consigue redireccionando la salida estándar de un proceso y la entrada estándar de otro a una *pipe*.

Para indicar que se conectan dos procesos mediante una *pipe*, se utiliza el símbolo |.

Ejemplos de redireccionamiento con el proceso comodín

Copia el contenido del fichero origen en el fichero destino.

```
$ comodin <origen >destino
```

Actúa como una máquina de escribir y envía todo lo que escribimos en el terminal hacia la impresora.

```
$ comodin </dev/tty >/dev/lp0
```

Edita el fichero destino. Actúa como un editor de líneas sencillo.

```
$ comodin </dev/tty >destino
```

Muestra el contenido del fichero origen.

```
$ comodin <origen >/dev/tty
```

Ejemplo de encadenamiento

Podemos combinar el comando ps con el filtro sort (ordena lo que lee por la entrada estándar y lo escribe en la salida estándar) y seleccionar las primeras líneas con head para conseguir los dos procesos en ejecución que han utilizado durante más tiempo la UCP.

```
$ ps -ef | sort -r -k 7 | head -3
UID      PID   PPID   C   STIME   TTY   TIME        CMD
albert   2973  1      6   12:16   ?     00:38:33    /usr/lib/firefox-3.5.8/firefox
root     1180  1172   1   10:40   tty7  00:07:39    /usr/bin/X
```

En un sistema UNIX podemos encontrar información sobre cada comando y sus opciones con man nombre_comando.

Como podéis ver, si tenemos un conjunto de filtros que ejercen funciones concretas podemos, mediante las *pipes* y las redirecciones, efectuar operaciones realmente complejas sin tener que escribir ni una sola línea de código.

7.2. Entrada/salida en Windows

Muchos de los conceptos que hemos aprendido en este módulo son aplicables a los sistemas Windows. En este sistema, podemos utilizar la **Windows API**, también llamada **WinAPI**. El programa siguiente ilustra el uso de algunas llamadas relacionadas con el sistema de entrada/salida. Las operaciones de lectura y escritura trabajan con estructuras de tipo HANDLE que corresponden a los dispositivos virtuales. Podemos observar que nos podemos referir a la entrada estándar con STD_INPUT_HANDLE y a la salida estándar como STD_OUTPUT_HANDLE. ¿Qué creéis que hace el siguiente programa?

```
#include <windows.h>
#include <stdio.h>

#define MAX_SIZE 64
void panic(char *miss)
{
  fprintf(stderr, "Error in %s\n", miss);
  ExitProcess(1);
}

int main(int argc, char *argv[])
{
    HANDLE h_in, h_out;

    DWORD num_read, num_written;
    BOOL ret;
    char buf[MAX_SIZE];

    if (argc == 1) {
        h_in = GetStdHandle(STD_INPUT_HANDLE);
        if (h_in == INVALID_HANDLE_VALUE) panic("GetStdHandle");
    }
```

```
    else {
        h_in = CreateFile(argv[1], GENERIC_READ, 0, NULL, OPEN_EXISTING, 0,
                          NULL);
        if (h_in == INVALID_HANDLE_VALUE) panic("CreateFile");
    }

    h_out = GetStdHandle(STD_OUTPUT_HANDLE);
    if (h_out == INVALID_HANDLE_VALUE) panic("GetStdHandle");

    while (1) {
          ret = ReadFile(h_in, (void *)buf, MAX_SIZE, &num_read, NULL);
          if (ret) {
            if (num_read > 0) {
                    ret = WriteFile(h_out (void*)buf, num_read, &num_written,
                                    NULL);
                    if (!ret) panic("WriteFile");
                    if (num_read != num_written) panic("Missing chars");
            }
            else break;
          }
          else panic("ReadFile");
    }
}
```

Resumen

En este módulo hemos estudiado el concepto de **entrada/salida** básicamente desde el punto de vista del usuario del sistema. Hemos visto que existen diferentes **tipos de dispositivos (físicos y lógicos)** con características y particularidades muy diferentes. Ante esta realidad, el objetivo del sistema operativo consiste en ofrecer una visión de los dispositivos que, en principio, haga transparente para los usuarios todas estas particularidades a fin de que las aplicaciones sean el máximo de independientes de los dispositivos que utilizan. Ahora bien, el sistema operativo debe permitir también que aquellas aplicaciones que lo necesiten puedan manipular las características y extraer todo el rendimiento que éstas ofrecen. Para lograrlo, la gestión de los dispositivos que hace el sistema operativo ofrece una interfaz de acceso formada por los elementos siguientes:

- **operaciones uniformes** independientes de las características concretas de los dispositivos,
- **dispositivos virtuales** en los que trabajan estas operaciones independientes.

Como caso concreto, hemos analizado el **sistema de entrada/salida del UNIX**, en el que podemos destacar:

- el **redireccionamiento** de los dispositivos estándar de entrada/salida desde el intérprete de comandos;
- las ***pipes*** como un caso de dispositivo lógico que permite comunicar procesos entre sí;
- los **programas filtro**, que permiten efectuar operaciones complejas a partir del encadenamiento de pequeños programas mediante *pipes*.

De forma complementaria, hemos introducido la manera como se puede programar la parte de entrada/salida de un proceso en un sistema Windows utilizando la **Windows API**.

Actividades

1. Estudiad en el manual del UNIX qué redireccionamientos se pueden establecer desde los intérpretes de comandos Bash.

2. Analizad los comandos del UNIX y familiarizaos especialmente con aquellos que se puedan considerar filtros, como por ejemplo sort, wc, tail, head, cut, grep, uniq, cat o tee.

3. Confrontad vuestra experiencia como usuarios de Linux, Windows o Mac con lo que hemos estudiado en este módulo didáctico.

4. Buscad información sobre las llamadas al sistema del UNIX y localizad las llamadas relacionadas con la entrada/salida. Podéis encontrar información en los recursos del aula y en los manuales en línea de un sistema UNIX mediante el pedido man.

5. Buscad información sobre la Windows API y localizad las llamadas relacionadas con la entrada/salida.

Ejercicios de autoevaluación

1. ¿Cuáles de los ficheros siguientes se pueden imprimir?

a) un fichero con código fuente

b) un fichero objeto

c) un fichero con ensamblador

d) un fichero ejecutable

Justificad la respuesta.

2. ¿Creéis que el reloj del sistema se puede considerar un dispositivo? Justificad la respuesta.

3. Decid ventajas e inconvenientes del hecho de que el sistema operativo reconozca la estructura de la información. Justificad la respuesta.

4. Cuáles de estos dispositivos utilizan la técnica del *buffering*:

a) la pantalla

b) el ratón

c) el teclado

d) el disco

Justificad la respuesta.

5. ¿En qué caso los procesos que utilizan ventanas pueden recibir una señal de fin de fichero?

6. ¿Qué parámetros tendría que tener una operación de lectura o escritura que diera acceso secuencial a la información si no existieran las operaciones abrir y cerrar? ¿Y si el acceso fuera directo? Justificad las respuestas.

7. Los dispositivos virtuales se pueden asignar a dispositivos lógicos de manera implícita o explícita. ¿Qué consecuencias puede tener el hecho de disponer sólo de la asignación explícita? ¿Y si sólo tenemos la implícita? Justificad las respuestas.

8. ¿Qué consecuencias tiene el hecho de redireccionar la entrada y la salida estándar del proceso comodín hacia un terminal? ¿Cómo se podría solucionar? Justificad las respuestas.

9. ¿Qué hace el programa de ejemplo de uso del Windows API de la sección 7.2?

Solucionario

Ejercicios de autoevaluación

1. Se pueden imprimir **a**, **c** y **d** porque la impresora es un dispositivo orientado a caracteres y, por lo tanto, se pueden imprimir todos los ficheros de texto. Debemos tener claro que puede haber ficheros ejecutables que sean de texto, como los *scripts*. Los ficheros objeto o binarios no son imprimibles, ya que su codificación no se basa en caracteres. En caso de que se impriman, el resultado será ininteligible.

2. Sí, ya que por definición un dispositivo de entrada/salida es un objeto gestionado por el sistema operativo en el que los procesos pueden establecer una operación de lectura o escritura con la finalidad de obtener información, almacenarla, mostrarla o transferirla. Esta definición se puede aplicar al reloj del sistema, ya que se pueden iniciar tanto operaciones de lectura como de escritura (consultar la hora actual y poner el reloj en hora, por ejemplo) con la finalidad de mostrarlo por pantalla, sincronizar procesos o sincronizar máquinas.

3. La principal ventaja del hecho de que el sistema operativo reconozca la estructura de la información es que se puede controlar que las manipulaciones que se hacen de la información sean correctas. Por ejemplo, que sólo se puedan editar ficheros de texto y no de dispositivo o que los directorios no se puedan crear a mano como lo haríamos con un fichero de texto (eso da más seguridad), entre otros. Como principal desventaja está el hecho de que el sistema no puede prever todos los tipos posibles de ficheros y, por lo tanto, puede ser que el hecho de tener que conocer todas las estructuras posibles de los ficheros se convierta en una barrera que impida la adaptación del sistema a situaciones nuevas. Otra desventaja es que nos da menos flexibilidad a la hora de establecer operaciones fuera de lo habitual, como por ejemplo editar un fichero objeto para ver su contenido.

4. La utilizan **b** y **c**. Ambos son dispositivos de entrada y tienen una velocidad muy diferente de la del procesador. Con el fin de reducir esta diferencia se utiliza el almacenaje en la memoria intermedia. Para verlo más claro, sólo nos tenemos que fijar en que, cuando el intérprete de comandos está ejecutando un comando, ya se puede escribir el siguiente. Este segundo comando se guarda en la memoria intermedia hasta que el intérprete lo reclama. De esta forma, se agiliza todo el proceso de entrada/salida.

También la **d**: el disco es un dispositivo que también trabaja a una velocidad significativamente más baja que la del procesador. A diferencia del teclado y del ratón, el disco es un dispositivo tanto de entrada como de salida y, en ambos casos, se puede utilizar la técnica del almacenaje en la memoria intermedia para reducir el impacto de esta diferencia de velocidad. En el caso de la entrada, permite leer bloques (unidad de transferencia del disco) y almacenarlos en la memoria intermedia hasta que los procesos necesitan la información. En el caso de la salida, los procesos dejan la información en la memoria intermedia y sigue su ejecución sin esperar que el sistema haya finalizado la escritura en el disco.

La pantalla es un dispositivo que no tiene el problema de las diferencias de velocidad y, por lo tanto, no necesita utilizar la técnica del almacenaje en la memoria intermedia.

5. Una señal de fin de fichero avisa a los procesos de que están leyendo un recurso del que ya no podrán extraer más información. Se podría deducir que un proceso que utiliza una ventana recibirá una señal de fin de fichero cuando se haya cerrado la ventana.

6. Si no existieran las operaciones abrir y cerrar, el sistema no tendría conocimiento de las sesiones de trabajo con los ficheros y no podría distinguir qué operaciones se establecen asociadas a una sesión o a otra. Como consecuencia, no se podría guardar información del estado de cada uno de las sesiones ni de su evolución.

Esto se traduce en lo siguiente:

a) Con los accesos secuenciales, el sistema no tendría constancia de cuál era la última posición a la que se ha accedido. Esta información se tendría que gestionar desde los programas y pasarla como parámetro al sistema. En consecuencia, las operaciones de acceso secuencial no existirían y se tendrían que construir por programa mediante operaciones de acceso directo.

b) Las operaciones de acceso tendrían que utilizar dispositivos físicos o lógicos, pero no virtuales, ya que durante la operación abrir es cuando se establece la asociación con estos dispositivos.

c) Finalmente, aunque no afecte a los parámetros de las operaciones de acceso, se tiene que destacar que cada operación de acceso tendría que verificar los derechos de acceso al dispositivo.

Así pues, las operaciones de acceso serían todas directas y tendrían los parámetros siguientes:

estado = *acceso(nom,pos,buff,cont*)

Donde *nom* es el nombre del dispositivo lógico, *pos* es la posición del dispositivo al que se quiere acceder, *buff* es la variable que define dónde se efectuará el acceso o bien desde dónde se hará y *cont* es la cantidad de información que se tiene que transferir.

7. Si sólo se dispusiera de la asignación explícita, no se podrían hacer redireccionamientos de la entrada/salida de los programas y estaríamos limitados a las entradas/salidas definidas en el código.

Si sólo se dispusiera de la asignación implícita, los programas no podrían especificar para qué dispositivos querrían hacer las entradas/salidas. Estarían obligados a utilizar los dispositivos estándar que hubiera previsto el sistema. El hecho de poder hacer redireccionamientos o no estaría condicionado por el tipo de mecanismo que proporcionara el sistema.

8. La consecuencia principal es que todo lo que se escribe se ve dos veces, una por efecto del modo *echo* del terminal como dispositivo de entrada y otra como resultado de la salida que efectúa el programa comodín. Para evitarlo y ver únicamente una vez lo que se escribe, se tendría que desactivar el modo *echo* del terminal.

9. El programa se comportará de forma similar al pedido cat del UNIX, es decir, copiará en la salida estándar el contenido del fichero cuyo nombre se pasa como parámetro. En caso de que se llame al programa sin parámetros, entonces copiará lo que llegue por la entrada estándar.

Glosario

almacenamiento en la memoria principal *m* Técnica de entrada/salida que utiliza una memoria intermedia llamada *buffer* con el fin de amortiguar las diferencias de velocidad de los dispositivos respecto de la velocidad de ejecución de los procesos.
en *buffering*

***cooked/raw* (cocinado/crudo)** *m* Modos de edición asociados a los dispositivos de entrada, como los terminales. El modo *cooked* hace que algunos caracteres que componen la entrada se interpreten como comandos que alteran la información que se ha introducido. El modo *raw* no interpreta ningún carácter y, por lo tanto, todos los caracteres se entregan en el proceso que hace la operación de entrada.

dispositivo de entrada/salida *m* Objeto gestionado por el sistema operativo en el que los procesos pueden entablar operaciones de lectura/escritura con la finalidad de obtener información, almacenarla, mostrarla o transferirla.

dispositivo físico *m* Dispositivo que existe físicamente. Está formado por el periférico y por su hardware de control, que constituyen la parte física, así como por el software que los gestiona, llamado programa de control.

dispositivo lógico *m* Dispositivo que no existe físicamente. Es el resultado de un software del sistema que lo crea.

dispositivo nulo *m* Dispositivo de entrada/salida en el que se puede escribir todo lo que se quiere y siempre sigue vacío. Ignora cualquier cosa que se escriba y en lectura no devuelve nunca ninguna información. Se basa exclusivamente en el software que lo define.

dispositivo virtual *m* Dispositivo que no está asociado a priori a ningún dispositivo concreto. El programa lo utiliza igual que lo haría con cualquier dispositivo, pero sin conocer a priori en qué dispositivo en concreto se efectuarán las operaciones que se especifican. Estará en tiempo de ejecución del programa que el sistema asociará un dispositivo lógico al dispositivo virtual mediante el código del sistema operativo.

file descriptors *m* Nombre que reciben el dispositivos virtuales, o canales, en el UNIX.

echo *m* Modo de los dispositivos de entrada/salida, como los terminales, que provoca que se realice una salida de toda la información que entra por el dispositivo con el objetivo de verificar y, si es necesario, corregir la información que se está entrando (ved *cooked/raw*).

gestión de colas *f* Técnica que utilizan las SPOOL (*Simultaneous Peripherical Operation On-Line*) que permite que las entradas/salidas de un proceso tengan un paso intermedio para dispositivos de gran capacidad de almacenaje. Hacen que el computador trabaje con un proceso en concreto mientras, de forma concurrente, los dispositivos finales van sacando o incorporando información de manera más lenta.
en *spooling*

independencia de los dispositivos *f* Objetivo del sistema operativo que consiste en proporcionar una máquina virtual que uniformice los dispositivos y respete sus particularidades. Eso se consigue independizando el código de los programas respecto de los dispositivos que utilizan y de las operaciones con las que se dirigen mediante la utilización de dispositivos virtuales y operaciones independientes.

major* y *minor *m* Pareja de números que identifican internamente los dispositivos del UNIX. El *major* identifica el tipo de dispositivo y el *minor* identifica un dispositivo concreto del tipo *major*.

named pipe *m* Dispositivo lógico del UNIX que se comporta como una *pipe*, con la diferencia de tener un nombre que consta en el espacio de nombres del sistema de ficheros.

operaciones uniformes *m* Uniformización de las operaciones que proporciona el sistema operativo con el fin de conseguir la independencia de los programas respecto de las entradas/salidas. Con el fin de proporcionar esta nueva función, el sistema añade una capa de software por encima de los programas controladores que dependen de los dispositivos, que son las operaciones uniformes de entrada/salida (son independientes de los dispositivos).

pipe *m* Dispositivo lógico del UNIX destinado a comunicar procesos. Su funcionamiento es el de una cola de caracteres, con una longitud fija, en la que los procesos pueden escribir y leer. Es un dispositivo que no figura en el sistema de ficheros, se crea cuando lo abre un proceso y se destruye cuando el último proceso que lo tiene abierto se cierra.

programa controlador *m* Software que gestiona un dispositivo de entrada/salida.

en *driver*

Bibliografía

Bibliografía básica

Silberschatz, A.; Galvin; Gagne, G. (2008). *Operating Systems Concepts* (8.ª ed.). John Wiley & Sons.

Tanembaum, A. (2009). *Modern Operating Systems*. Prentice-Hall.

Bibliografía complementaria

Robbins, K.; Robbins, S. (2000). *UNIX: programación práctica*. Prentice Hall.

Recursos del Aula. Manual de llamadas al sistema UNIX y programas de ejemplo.