Introducción a Sistemas Operativos: Empezando

Clips 1 a 5.
Francisco J Ballesteros

1. ¿Qué es el Sistema Operativo?

El sistema operativo es software que te permite utilizar el ordenador. Lo que esto signifique dependerá del punto de vista del usuario que utiliza el ordenador. Por ejemplo, para mi abuela el sistema operativo incluye no sólo Windows, sino también todos los programas instalados en el ordenador. Para un programador, la mayoría de las aplicaciones no forman parte del sistema operativo. No obstante, este usuario podría considerar los compiladores librerías y utilidades para programar como parte del sistema. Para un programador de sistemas, los programas que forman el sistema operativo serían muchos menos de los que otros usuarios considerarían como parte del sistema. Todo depende del punto de vista.

Este curso intenta enseñar cómo utilizar el sistema operativo de forma efectiva y qué abstracciones forman parte del mismo. En adelante, nos referimos al sistema operativo como *sistema*, para abreviar. Usar el sistema implica utilizar las funciones que incluye y los programas y lenguajes que forman parte del mismo y nos permiten utilizar el ordenador. El propósito es siempre el mismo: Hacer que la máquina haga el trabajo y evitar tener que hacerlo manualmente. La diferencia entre saber utilizar el sistema y no saber hacerlo es la diferencia entre necesitar horas o días para hacer el trabajo y necesitar un par de minutos para conseguirlo. La elección es tuya, aunque parece clara.

En este curso aprenderás a utilizar un sistema, a ver qué hace y que abstracciones proporciona y a tener una idea aproximada de cómo lo hace. Tienes otros libros que cubren otros aspectos:

- Para aprender C siempre tienes [1].
- Como libro de introducción a UNIX tienes [2].
- Puedes ver [3] para aprender conceptos teóricos de sistemas operativos y su relación con la implementación.
- El mejor libro para ver cómo está hecho el sistema es quizá [4]. Aunque describe la 6th edición de UNIX, sigue siendo el mejor. Una vez digieras este libro, podrías seguir con [5] y leer el código fuente de OpenBSD mientras lo lees.
- Para continuar aprendiendo a programar sobre UNIX cuando termines este curso puedes utilizar [6].
- En [7] tienes consejos útiles sobre como programar.
- Como referencia (para buscar funciones y programas) siempre tienes el manual en línea en cualquier sistema UNIX (y en internet los tienes todos).

Pero volvamos nuestra pregunta... ¿Qué es el sistema operativo? Es tan sólo un conjunto de programas que te permiten utilizar el ordenador. El hardware es complejo y está lejos de los conceptos que utilizas como programador (no digamos ya como usuario). Hay multitud de tipos de procesadores, dispositivos hardware para entrada/salida (o E/S, o I/O, por *input/output*), y muchos otros artefactos. Si tuvieras que escribir el software necesario para manejar todos los que usas, no tendrías tiempo de escribir el software de la aplicación que quieres escribir. El concepto es pues similar al de una librería (o biblioteca) de software. De hecho, los sistemas operativos empezaron como librerías utilizadas por aquellos que escribían programas para una máquina.

Cuando el ordenador arranca, el procesador comienza a ejecutar instrucciones de un programa que habitualmente se guarda en memoria no volátil. Este programa es un cargador cuyo trabajo es localizar en el disco o en algún otro dispositivo el código de otro programa, el núcleo del sistema operativo, y cargarlo en la memoria. Una vez cargado, se salta a su primera instrucción y lo que suceda a partir de ese momento depende del sistema operativo que estemos ejecutando. Se suele llamar *kernel* al núcleo del sistema operativo. Es un programa como cualquier otro programa, pero es importa puesto que permite que los programas de los usuarios puedan ejecutar y, por ello, permite a los usuarios utilizar el ordenador. Tener un sistema operativo tiene tres ventajas:

1. No es preciso escribir el código que incluye el sistema operativo, ya lo tenemos escrito y lo podemos utilizar sea cual sea la aplicación que queremos ejecutar.
2. Podemos olvidarnos de los detalles necesarios para utilizar el hardware. El sistema operativo se comporta como nuestra librería y ya incluye tipos abstractos de datos que empaquetan los servicios que nos da el hardware de un modo más apropiado.
3. Podemos olvidarnos de cómo se pueden repartir los recursos del hardware entre los distintos programas que queremos ejecutar en el mismo ordenador. El sistema operativo está hecho para que sea posible utilizarlo de forma simultánea desde varios programas en el mismo equipo.

La mayoría de los programas que has escrito utilizan discos, pantallas, teclados y otros dispositivos. No obstante, los has podido escribir sin saber cómo se manipulan. Dicho de otro modo, no has tenido que escribir el software que sabe como manejarlos (no has tenido que escribir los *manejadores* o *drivers* para ellos). Esto debería ser razón suficiente para convencerte de la utilidad del sistema operativo.

Pero hay más. Los tipos abstractos de datos son muy cómodos para escribir software. De hecho, son casi indispensables. Por ejemplo, has escrito programas que utilizan ficheros. No obstante, los discos donde guardas tus ficheros no saben nada respecto a ellos: ¡El hardware no sabe lo que es un fichero! El disco sólo sabe cómo almacenar bloques de bytes. Lo que es más, sólo sabe como almacenar bloques del mismo tamaño (normalmente llamados *sectores*). A pesar de ello, nosotros preferimos utilizar nombres para cada conjunto de datos de nuestro interés. Y preferimos que dichos datos persistan, almacenados en el disco. Nos los imaginamos con una serie contigua de bytes en el disco, empaquetados dentro de un "fichero". Es el sistema operativo el que se inventa el tipo de datos *fichero* y suministra las operaciones que permiten utilizarlo. Incluso el nombre de un fichero es parte de la abstracción. Es otra "mentira" implementada por el sistema operativo.

Esto es tan importante que incluso el hardware lo hace. Volviendo a pensar en un disco, el interfaz que utiliza el sistema operativo es habitualmente un conjunto de registros que permiten leer bloques del disco y escribir bloques en el. El sistema piensa que el disco es una sucesión (un *array*) de bloques contiguos, identificados por una dirección (el índice en el array). Todo esto es mentira. En el hardware de la tarjeta que controla el disco hay gran cantidad de software que está ejecutando e inventando esta abstracción (array de bloques). En la actualidad, sólo los que trabajan para un fabricante de disco saben realmente lo que hace el disco internamente. Todo lo demás son abstracciones implementadas por el software que, en este caso, podríamos decir que es el sistema operativo del disco, o el *firmware* del disco. Los discos pueden tener geometrías complejas para optimizar el acceso y pueden contener caches que mejoren su rendimiento. En cualquier caso, el sistema operativo piensa que un disco es básicamente un array de bloques. Exactamente lo mismo sucede cuando tus programas utilizan ficheros.

Utilizar tipos abstractos de datos tiene otra ventaja: La portabilidad. Si el hardware cambia, pero el tipo de datos que utilizas sigue siendo el mismo, no es preciso que cambies el programa. Tu programa seguirá funcionando. ¿Has visto que tus programas utilizan ficheros sin pensar en qué tipo de disco se utiliza para almacenarlos? Los ficheros se utilizan igual tanto si son ficheros almacenados en un disco magnético, en un disco USB o en cualquier otro medio de almacenamiento.

Piensa que el hardware puede cambiar también cuando lo reemplazas por hardware más moderno. Los sistemas operativos están hechos de tal forma que sea posible utilizar versiones antiguas del hardware. Dicho de otro modo, suelen tener *compatibilidad hacia atrás*. Esto quiere decir que están hechos intentando que

los programas sigan funcionando a pesar de que el hardware evolucione. Simplemente, en algún momento tendrás que actualizar el sistema instalando nuevos programas para el nuevo hardware (sus *drivers*). Tus programas seguirán ejecutando del mismo modo.

Por esta razón decimos que el sistema operativo es una *máquina virtual*, que corresponde a una máquina que no existe. Por eso se la denomina "virtual". La máquina suministra ficheros, procesos (programas en ejecución), ventanas, conexiones de red, etc. Todos estos artefactos son desconocidos para el hardware.

Dado que los ordenadores son extremadamente rápidos, es posible utilizarlos para ejecutar varios programas de forma simultánea. El sistema hace que sea sencillo mantener todos los programas ejecutando a la vez (o casi a la vez). ¿Has notado que resulta natural programar pensando que el programa resultante tendrá toda la máquina para el solo? No obstante, ese no será el caso. Es casi seguro que tendrás ejecutando al menos un editor, un navegador web y otros muchos programas. El sistema decide qué partes de la máquina, y en qué momentos, se dedican a cada programa. Esto es, el sistema *reparte*, o *multiplexa*, los recursos del ordenador entre los programas que lo utilizan. Las abstracciones que suministra el sistema tratan también de aislar unos programas de otros, de tal forma que sea posible escribir programas sin necesidad de pensar en todo lo que tenemos ejecutando dentro del ordenador.

Por esto decimos que el sistema operativo es un *gestor de recursos*, o un *multiplexor* de recursos. Asigna recursos a los programas y los reparte (o multiplexa) entre ellos. Algunos recursos, como la memoria, los podemos repartir dando a cada programa un trozo del recurso: los multiplexamos en el espacio. Unos programas utilizan unas partes de la memoria, y otros utilizan otras. El kernel también necesita utilizar su propia parte. Otros recursos, como el procesador, los tenemos que repartir dando a cada programa el recurso entero durante un tiempo. Pasado ese tiempo, el recurso se dedica a otro programa. Estos recursos se multiplexan en el tiempo. Dado que la máquina es tan rápida, da la impresión de que todos los programas están ejecutando a la vez (en paralelo). No obstante, si tenemos un único procesador (un *core*), sólo podemos ejecutar un programa en cada instante. Pero incluso en este caso, todos los programas ejecutan *concurrentemente* (de forma simultánea).

La última misión del sistema operativo tiene que ver con esto. Los humanos y los programas cometen errores. Además, los programas tienen *bugs* (que no son otra cosa que errores cometidos por sus programadores). Un error en un programa puede hacer que todo el ordenador se venga abajo y deje de funcionar, a menos que el sistema operativo tome medidas para evitar esto. No obstante, el sistema operativo no es una divinidad y tan sólo puede utilizar los mecanismos de protección que suministra el hardware para intentar proteger a unos programas de otros. Por ejemplo, una de las primeras cosas que hace el kernel cuando se inicializa es proteger su propia memoria. Esta se marca como privilegiada y se permite el acceso a la misma sólo para código que ejecute con el procesador en "modo privilegiado" (con un bit en un registro puesto a un valor determinado). El kernel ejecuta en modo privilegiado, pero tus programas no lo hacen. Al saltar hacia el código del usuario, el procesador de deja en modo no privilegiado y el efecto neto es que tus programas no pueden acceder a la memoria del kernel. Además, las protecciones de la memoria que utiliza un programa se ajustan de tal modo que otros programas no puedan acceder a la misma. Como resultado, un error en un programa no conseguirá habitualmente que otros programas dejen de funcionar. ¿Has notado que cuando tu programa tiene un bug, otros programas pueden seguir funcionando? ¿Podrías decir ahora por qué?

Resumiendo, el sistema operativo es un programa que te da abstracciones para utilizar el hardware, que te permiten programar de un modo mas simple. Y naturalmente, reparte dicho hardware entre los programas que lo usan. Para hacerlo, ha de gestionar los recursos y proteger a unos programas de otros. En cualquier caso, el sistema es tan sólo un programa.

2. ¿Qué es UNIX?

En este curso vamos a utilizar UNIX como sistema operativo. Hoy en día, eso quiere decir Linux, OpenBSD (u otros cuyo nombre termina en "BSD") o MacOS (OS X).

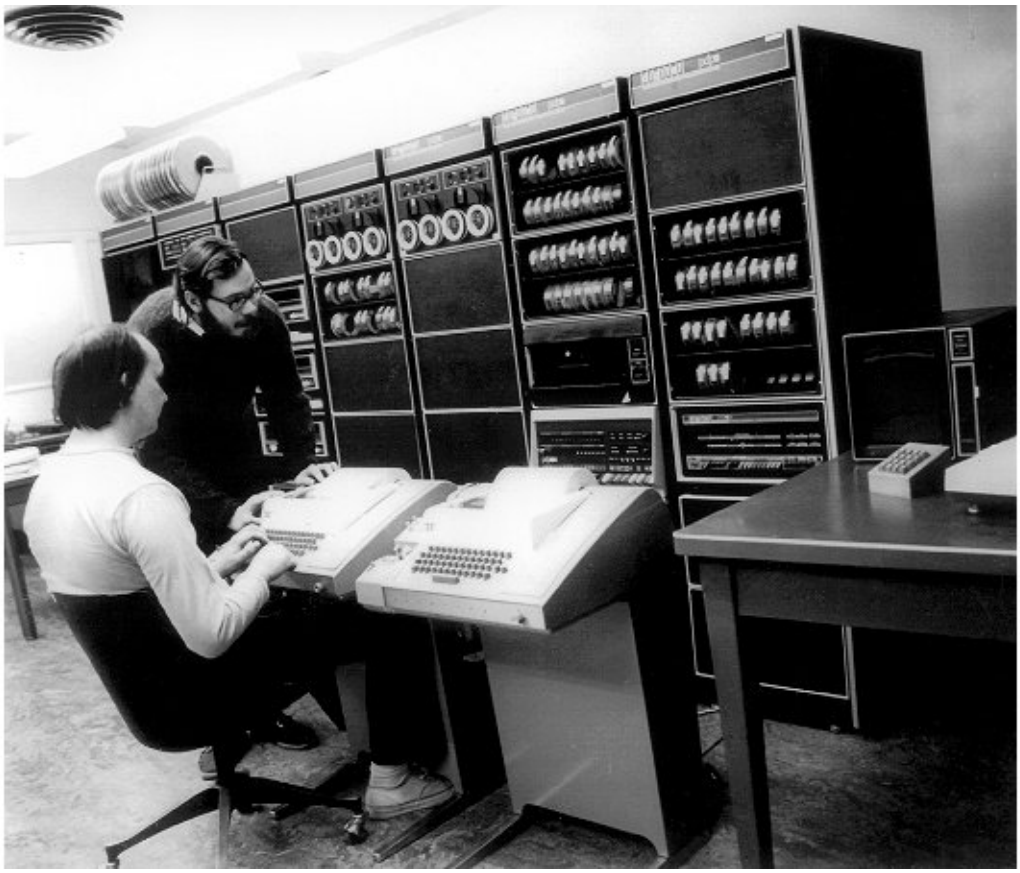

Figura 1: *Ken Thompson y Dennis Ritchie junto a la PDP. ¡Seguro que lo pasaban bien!*

Hace mucho tiempo (pero en esta galaxia) Ken Thompson y Dennis Ritchie hicieron un programa llamado UNIX para un ordenador llamado PDP de una empresa, Digital, que ya no existe. El sistema era tan fácil de utilizar y de adaptar para otros ordenadores que se extendió como la pólvora. Como su código fuente era fácil de entender (en comparación con los de otros) hay muchos que lo han modificado y, como resultado, hay toda una familia de sistemas operativos que descienden de UNIX (y se conocen como UNIX de hecho) pero que, naturalmente, difieren en alguna medida. Así pues, UNIX realmente no existe, aunque es el sistema más popular hoy día para programar y para mantener servicios en red. Consulta los libros que hemos mencionado para ver más sobre esto.

Oirás o leerás nombres como *POSIX* (*Portable Operating System Interface*) y como X/Open o SUS (*Single Unix Specification*). Estos se refieren a diversos estándares que intentan estandarizar (¡sorprendentemente!) UNIX. La práctica totalidad de los sistemas UNIX difieren de dichos estándares y, si se utilizan opciones para hacer que no lo hagan, en la mayoría de los casos encuentras problemas. En pocas palabras, puedes suponer que dichos nombres se refieren también a UNIX, aunque a un UNIX que no existe y que es una amalgama del comportamiento de los que existían cuando se escribieron dichos estándares.

3. ¿Cómo seguir este curso?

Para seguir este curso lo primero que has de hacer es buscar un sistema UNIX e instalarlo. Quizá ya lo tienes si utilizas Linux o un Mac. En caso contrario es muy fácil descargar Linux e instalarlo (sugerimos Ubuntu por su facilidad de instalación, no es que nos guste más). Conforme avances leyendo o atendiendo a las clases, utiliza tu UNIX para probar los programas, llamadas y comandos que vamos viendo. ¡Juega con ellos! Pocas cosas te resultarán tan rentables profesionalmente (¡Y económicamente!) como aprender **bien** a utilizar UNIX.

Cuando termines este curso podrás programar y utilizar cualquiera de ellos, ¡Y otros muchos!. Piensa que sistemas como Windows y otros que no son UNIX utilizan básicamente las mismas abstracciones y, con el tiempo, han ido incorporando las ideas de UNIX.

Si deseas que tu programa o los comandos que utilizas sean **portables**, se puedan utilizar en distintos sistemas UNIX, un buen consejo es que mantengas instalado al menos un sistema Linux y un OpenBSD y te asegures de que tus programas funcionan en ambos. Cuando lo hagan, es muy posible que funcionen correctamente en otros sistemas UNIX. En cualquier caso, las páginas de manual te informan de si las llamadas y comandos son específicos del sistema que usas o forman parte de algún estándar.

Referencias

1. The C programming language, 2nd. ed. Brian W. Kernighan,Dennis M. Ritchie. Prentice Hall. 1988.
2. The UNIX Programming Environment. Brian W. Kernighan,Rob Pike. Prentice-Hall. 1984.
3. Operating Systems Design and Implementation. Andrew S. Tanembaum. PRHALL. 2004.
4. Commentary on UNIX 6th Edition, with Source Code. John Lions. Peer-to-Peer Communications. 1996.
5. The Design and Implementation of the 4.4 BSD Operating System. Marshall Kirk McKusick,Keith Bostic,Michael J. Karels,John S. Quarterman. Addison-Wesley. 1996.
6. Advanced Programming in the Unix Environment. Stevens. Addison-Wesley. 1992.
7. The Practice of Programming. Brian W. Kernighan,Rob Pike. Addison-Wesley. 1999.