Desarrollo de contenidos teóricos.

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

Índice de contenido

Introducción

El sistema operativo, o *SO* es como un director. Es el responsable de coordinar todos los componentes individuales del computador de forma que operan juntos siguiendo un solo plan. Cuando una orquesta está afinando, el colectivo de instrumentos produce una cacofonía de instrumentos discordantes. Pero cuando el director se hace cargo, el conjunto de instrumentos actúa como un todo para producir una secuencia de sonidos (esperamos que) agradables. El director establece el tempo de la música, da la entrada a cada instrumento según deben sonar, controla el volumen de cada

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

sector individual de la orquesta y demás. Del mismo modo, el SO asigna los recursos del computador a los diversos programas, sincroniza sus actividades individuales, y proporciona, generalmente, los mecanismos necesarios para que los programas se ejecuten en perfecta armonía.

Eficiencia y funcionalidad son conceptos clave para la utilidad de un sistema. La eficiencia fija un estándar de prestaciones para todo el software de un computador. Una de las razones más importantes para estudiar los sistemas operativos es aprender a extraer la máxima eficiencia de ellos. El SO también proporciona una gran cantidad de funciones que asisten a la ejecución de un programa. Un SO de altas prestaciones que proporcione poca funcionalidad fuerza a trabajar más a los programas de aplicación.

Los sistemas operativos se han ganado la reputación de ser el software más crítico en un sistema informático. Sólo a los programadores más hábiles y experimentados se les permite diseñar y modificar el sistema operativo de un computador.

Todos los sistemas operativos están diseñados bajo ciertas restricciones y circunstancias. Las decisiones de diseño suelen reflejarse en la interfaz del SO que se exporta para su uso por los programas de aplicación. El diseño podrá dar la impresión de tener discontinuidades, anormalidades u otras inconsistencias lógicas.

En este documento se explica qué es un sistema operativo y cómo evolucionó hasta su situación actual de desarrollo. En primer lugar se hablará acerca del entorno software, en conjunto, de modo que se pueda apreciar cómo encaja en él el SO. Después se tratarán los requisitos que deben cumplir los sistemas operativos modernos (abstracción y compartición) y de dónde provienen. Finalmente, se examinan las estrategias de SO más populares y cómo han influido sobre los servicios proporcionados por los sistemas operativos modernos.

COMPUTADORES Y SOFTWARE

Los sistemas informáticos constan de software y hardware que se combinan para proporcionar herramientas que resuelven problemas concretos. El software se diversifica según su finalidad. El software de aplicación trata de resolver un problema específico, o proporcionar herramientas genéricas para los usuarios finales. Por ejemplo, un software de aplicación para el control de inventario utiliza el computador para seguir la pista e informar a una compañía de su inventario, el software de correo electrónico permite que la gente se comunique, los programas de edición de documentos proporcionan una

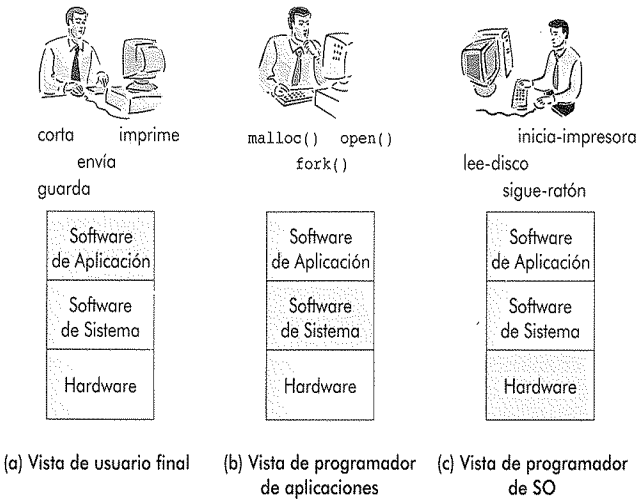
U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

herramienta para componer y editar documentos de texto, y los programas de hoja de cálculo permiten a los usuarios almacenar y manipular información para apoyar a la toma de decisiones.

FIGURA 1

Perspectivas del computador

Los usuarios finales, los programadores de aplicaciones y los programadores del SO utilizan diferentes programas de aplicación, el programador de aplicaciones usa el software del sistema para producir programas de aplicación, y el programador del SO utiliza el hardware para implementar software del sistema.



Últimamente, el coste de cualquier computador se justifica por el valor de su software de aplicación. Es decir, una persona o una compañía compra un computador para resolver problemas concretos de procesamiento de información según sus necesidades. Como se sugiere en la Figura 1(a), la visión del usuario final de un computador es la que le da el software de aplicación. Cualquier otro software o hardware es solo una parte del sobrecoste necesario para ejecutar el software de aplicación que resuelve el problema.

El software del sistema proporciona un entorno de programación general en el que los programadores pueden crear aplicaciones específicas para las necesidades de sus clientes. Este entorno se compone de herramientas de programación (tales como editores y compiladores) y abstracciones (tales como archivos y objetos). El programador de aplicaciones usa el software del sistema, que incluye al SO, para proporcionar un conjunto de aplicaciones a los usuarios finales (Figura 1(b)). Desde la perspectiva del programador de aplicaciones, el software del sistema es importante puesto que define el entorno en que pueden implementarse los programas para su envío al usuario final. Sin embargo, desde la perspectiva del usuario final, el software del sistema no parece más importante que, pongamos, la fuente de alimentación del hardware. *El hardware y el software existen para soportar la creación y el uso eficiente del software de aplicación*. Dada esta premisa, podemos decir que el software del sistema proporciona suficiente funcionalidad a los programadores de aplicaciones, y no supone ninguna barrera al usuario final.

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

Una parte importante de su función de facilitar las cosas es ofrecer *eficiencia*: El software del sistema trata de minimizar su propio uso de los recursos de la máquina (como tiempo de procesador y memoria), para maximizar el tiempo en que dichos recursos están disponibles para los programas de aplicación.

Podríamos minimizar el uso de los recursos, por parte del software del sistema, eliminando todo el software del sistema. Pero entonces cada programa tendría que proporcionar la funcionalidad que ofrece el software del sistema, ¡Imagínese si tuviera que implementar un sistema de archivos completo sólo porque no puede leer o escribir sobre un dispositivo de disco! Hoy en día sabemos que necesitamos software del sistema: La pregunta es ¿cuánto y con qué funciones? Esta pregunta es la base de la competencia entre los modelos de software: El software del sistema Macintosh proporciona un conjunto de herramientas de programación, los entornos de Microsoft proporcionan un con junto de herramientas diferente, Java tiene su propio conjunto de funciones software del sistema, y los sistemas UNIX/GNU-Linux ofrecen un modelo más de herramientas de programación.

El motivo original de la existencia del software del sistema fue proporcionar funciones que un programador pudiera usar para implementar software de aplicación. Con el paso del tiempo, apareció otro objetivo importante del software del sistema (en concreto en los sistemas operativos): Permitir que el software de aplicación pudiera compartir el hardware de una forma ordenada. Por ejemplo, un programa puede estar leyendo información de un disco, mientras que otro programa está calculando la raíz cuadrada de un número. Esta compartición incrementó las prestaciones del sistema al permitir que diferentes programas usaran diversas partes del hardware del computador simultáneamente, disminuyendo así el tiempo necesario para ejecutar un conjunto de programas e incrementando las prestaciones del sistema. Para garantizar que esta compartición se hace de modo seguro y eficiente, el software del sistema controla el uso de los componentes hardware por los diversos programas de aplicación en ejecución. Hablando en general, el sistema operativo es parte del software del sistema que gestiona el uso del hardware por el resto del software del sistema y todo el software de aplicación. Por ello, decimos que el SO es el software implementado “más cerca del hardware.” El programador del SO escribe software que controla el hardware (para implementar la compartición y la abstracción) proporcionando un entorno software utilizable por el programador de aplicaciones (véase la Figura 1(c)).

SOFTWARE DEL SISTEMA GENERAL

El software del sistema crea dos tipos de entornos: Uno que permite que

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

las personas interactúen con el computador, y un segundo que proporciona herramientas y subcomponentes empleados en los entornos de aplicación. La interfaz humano-máquina da soporte tanto a los usuarios finales como a los programadores proporcionándoles herramientas como escritorios electrónicos y editores de texto para gestionar la información. Los usuarios finales gestionan su correo, documentos e información numérica, mientras que los programadores gestionan su software. Un ejemplo que ilustre el caso de los programadores puede ser los entornos de programación de C y C++, se implementan importantes herramientas en las bibliotecas de tiempo de ejecución del **software del sistema** (a las que se accede usando diversos archivos . h), incluyendo:

- La biblioteca estándar de entrada/salida (I/O) proporciona procedimientos para realizar la entrada/salida con búfer, como `printf()` y `soanf()`, sobre un flujo de datos.
- La biblioteca matemática proporciona funciones como `sqrt ( )`, para calcular funciones matemáticas.
- Las bibliotecas gráficas proporcionan funciones como `drawCircle ( )`,para mostrar imágenes sobre un dispositivo *bitmap*.

Otro software del sistema implementa componentes lógicos del sistema. Mientras que las bibliotecas proporcionan conjuntos de funciones que pueden ser llamadas desde los programas de aplicación, estos componentes son elementos esenciales del entorno de computación. Aquí vemos algunos ejemplos de este tipo de componentes:

- Un **intérprete de línea de órdenes** (denominado usualmente **shell**) que es un programa basado en texto al que invoca el usuario para interactuar con el software del sistema. El usuario envía mandatos (p.e. `dir` a Windows y `ls` a UNIX/GNU-Linux), al intérprete que ordenan al software del sistema la realización de las acciones deseadas, tal como listar las entradas en un directorio. Los programas `sh`, `csh` y `bash` de UNIX/GNU-Linux, y el programa `cmd.exe` de Windows son ejemplos de intérpretes de líneas de órdenes.
- Un **sistema de ventanas** es un sistema software que proporciona un terminal virtual a un programa de aplicación. La ventana se califica como “virtual” porque el programador construye el software de aplicación usando funciones para leer y escribir en la ventana como si fuera un terminal, incluso aunque no haya un terminal físico único asociado a la ventana. El software del sistema aplica estas operaciones de terminal

virtual sobre una región física concreta de la pantalla. Después traduce las operaciones del software sobre el terminal virtual en las operaciones apropiadas sobre el terminal físico. Éste puede soportar varios terminales virtuales. Por ejemplo lo, el escritorio de Macintosh, el escritorio de Microsoft Windows y el escritorio de Gnome en GNU-Linux son sistemas de ventanas.

- Un **sistema de gestión de base de datos** (**DBMS**, *database management system*) puede usarse para almacenar información en los dispositivos de almacenamiento permanente del computador. El sistema de base de datos proporciona tipos abstractos de datos (denominados esquemas) y crea un nuevo software específico de la aplicación optimizado para la realización eficiente de consultas y actualizaciones de los datos según la definición del esquema. Cuanto más complejas y más numerosas sean las estructuras de datos que usa una aplicación, mayor es el beneficio que se extrae de usar un sistema de gestión de base de datos. Ejemplos de sistemas de gestión de base de datos son los sistemas de base de datos relacionales Oracle o MySQL.

La gente y las organizaciones compran computadores para resolver sus problemas de procesamiento de información. Por ejemplo, un negocio compra algunos computadores para tratar la información de contabilidad; una organización militar puede comprar un computador para calcular trayectorias balísticas; y cierto individuo puede comprarlo para jugar y “surfear” por Internet. Cada una de estas razones para comprar un equipo define un dominio de aplicación, o colección de problemas que pueden resolverse mediante un computador. En el dominio de aplicación de la contabilidad, los programas interesantes pueden crear facturas, mantener balances, y demás. En el dominio de las trayectorias balísticas, los programas deben resolver problemas relacionados con gobernar un misil. En el dominio de la computación personal, los programas deberían permitir dar soporte a programas de juego con muchos gráficos, edición de texto en línea y navegación web.

Ciertas partes del software del sistema, tales como las bibliotecas gráficas, pueden ser útiles en un dominio de aplicación pero no serlo en otro en absoluto, Otros software del sistemas, como las bases de datos relacionales, pretenden servir para un uso bastante general. Pueden soportar programas escritos para muchos dominios de aplicación diferentes. Con un pequeño coste adicional de complejidad, en el caso de las bases de datos, los DBMS pueden acomodarse a diferentes dominios Por ejemplo, el sistema de base de datos puede especializarse para dar soporte a un subdominio como el procesamiento de imagen de un sistema experto de inteligencia artificial, E incluso dentro del

software de base de datos para procesamiento de imágenes, a veces es preciso soportar aplicaciones más específicas. Por ejemplo, la base de datos de imágenes podría servir para soportar solamente imágenes topográficas monocromas.

¿Cómo se distingue un SO del resto del software del sistema? Aquí relacionamos algunas diferencias esenciales:

- El Un SO interactúa directamente con el hardware para proveer una interfaz que se emplea en el resto del software del sistema y en el software de aplicación.
- El Un SO de propósito general es independiente del dominio. Esto significa que puede usarse el mismo SO para dar soporte a una amplia gama de dominios de aplicación, tales como el software de gestión de inventarios, así como el software para calcular el flujo de fluidos sobre el ala de un avión.
- El programa de aplicación usa recursos abstractos provistos por el SO para determinar su interacción concreta con los componentes hardware.
- El SO permite que diferentes aplicaciones compartan los recursos hardware en virtud de su política de gestión de recursos.

La abstracción de recursos y la compartición son los dos aspectos clave de un sistema operativo. Veamos ahora, con más detalle, estos dos aspectos.

ABSTRACCIÓN DE RECURSOS

El software del sistema oculta (al usuario) los detalles de cómo opera la maquinaria subyacente. Esto implica que el usuario puede manejar un computador sin saber gran cosa sobre cómo funciona el hardware. Esta idea se extiende al programador de aplicaciones proporcionando modelos operacionales que son relativamente sencillos comparados con la interacción directa con el hardware. La técnica empleada es que el software del sistema provea un modelo abstracto de cómo funcionan los componentes del hardware. Esta estrategia de abstracción es frecuente en la vida diaria. Por ejemplo, para conducir un automóvil moderno no es preciso comprender los detalles de los motores, los frenos y el sistema de dirección. De hecho, en un automóvil con transmisión automática, ni siquiera hay que saber cambiar de marcha (o incluso aunque hubiera marchas). Esto es posible gracias a la abstracción de “interfaz de programación”. Durante el primer medio siglo de su existencia, los automóviles sólo disponían de cambio manual. Esto significaba que cualquiera que quisiera conducir un coche debería aprender a accionar un embrague y a

conocer los rudimentos de las diferentes marchas para cada rango de velocidades. Con la abstracción de cambio automático, un conductor sólo debe saber qué significan “P”, “D” y “R”, ya sean accionadas con una palanca o con un botón. El resto de marchas (punto muerto y reductora) no necesitan ser usadas. Hoy en día, los conductores pueden concentrarse en otras funciones de alto nivel de la conducción, en lugar del cambio, como guiar, frenar y cosas relacionadas como elegir la ruta, la velocidad, adelantar a otros conductores, y cosas así.

En un sistema informático, las abstracciones son útiles para eliminar los detalles tediosos de los que, de otro modo, debería preocuparse el programador. Sin una abstracción apropiada para imprimir caracteres en la pantalla (como la función `print`), imagine cuánto habría que aprender para imprimir “Hola, mundo” con la fuente Arial de 12 puntos en una ventana gráfica En lugar de aprender todos estos detalles, el programador C sólo aprende cómo usar `printf ( )` y la biblioteca `stdio`. El tiempo que el programador habría desperdiciado escribiendo código para formar caracteres en una pantalla ahora se puede usar para resolver el problema que tenemos entre las manos.

El lado oscuro de la abstracción es que mientras simplifica la forma en que el programador de aplicaciones controla el hardware, también puede limitar la flexibilidad con la que puede manipularse cierto hardware concreto. La generalidad tiene su precio en especificidad. Es decir, mientras ciertas operaciones se vuelven sencillas de realizar, otras operaciones se vuelven casi imposibles de lograr usando esta abstracción. Veamos cómo puede ocurrir esto meditando sobre el funcionamiento de un cajero automático.

Los cajeros automáticos permiten a la gente retirar cantidades de dinero variables de sus cuentas pulsando varios botones que indican la cantidad concreta de dinero a retirar. Algunos cajeros proveen una operación abstracta que permite al usuario retirar 50€ de la cuenta corriente pulsando un solo botón, Suponga que el cajero sólo permite operaciones abstractas para retirar 20€, 50€, 100€ o 200€ de su cuenta. No habría forma de extraer exactamente 30€. Él es más sencillo de usar, pero también menos flexible.

Los computadores se componen de componentes hardware muy diversos (forman parte de los recursos del sistema) que pueden ser usados por un programa de aplicación. Cada recurso concreto, como el disco duro, tiene una interfaz que define cómo el programador puede realizar cierta operación sobre él. Las abstracciones, sin embargo, permiten interfaces más sencillas sobre los recursos concretos. Por ejemplo, el escribir información a un archivo (una abstracción del disco) es mucho menos complejo que generar las operaciones

necesarias para escribir directamente sobre una unidad de almacenamiento de disco. Las abstracciones se implementan dentro del software del sistema. Las abstracciones de nivel más bajo (aquellas que tratan directamente con el hardware) están implementadas en el SO, mientras que las abstracciones de nivel superior, como un sistema de ventanas, se implementan en el software del sistema exterior al SO. Igual que las abstracciones de la transmisión de un automóvil o un cajero automático, el programador no tiene por qué aprender cada interfaz concreta con los recursos para utilizar el recurso. En su lugar, puede usarse la interfaz abstracta (que ignora los pequeños detalles de cómo opera el dispositivo). Entonces el programador se puede con centrar en las cuestiones de mayor nivel.

En muchos casos, varios recursos similares pueden abstraerse sobre una misma interfaz abstracta común de recurso. Por ejemplo, el software del sistema puede abstraer la unidad de disquete y la unidad de disco duro en una sola interfaz abstracta de disco, mientras que un programador de aplicaciones debe conocer el comportamiento general de los discos, no es necesario, ni deseable, que aprenda los detalles de la entrada/salida de disco. El programador únicamente necesita conocer cómo usar la abstracción de disco. En caso de conducir un automóvil, las abstracciones son tan comunes que se puede alquilar un coche que nunca se ha conducido antes y llevárselo inmediatamente. Las abstracciones utilizadas en los coches de alquiler son las mismas que las empleadas en su propio coche. Imagine el desastre que sería si las abstracciones de conducción de los automóviles no fueran comunes. Suponga que en algunos coches se girara el volante en la dirección contraria en la que se suele girar para obtener el efecto deseado.

Al diseñar software del sistema, se definen un conjunto de abstracciones que serán generalizables a través de varios recursos, siendo intuitivas para un programador a la par que adecuadas para uno o más dominios de aplicación. La abstracción de archivo para una operación de disco es un ejemplo de ello. Las buenas abstracciones serán fáciles de entender para cualquier programador, así como fáciles de usar, y permitirán al programador realizar fácilmente cualquier tipo de operación sobre los recursos empleados en el dominio.

Los programadores con orientación al objeto utilizan la abstracción a diversos niveles al trabajar con jerarquías de clases. Una clase base define las operaciones más abstractas sobre los objetos, y las subclases refinan las operaciones para miembros más específicos de la familia. La abstracción de manejador de disco del ejemplo de la caja adyacente muestra cómo puede utilizarse la abstracción a más de un nivel. Una vez que los componentes

hardware han sido simplificados mediante una interfaz, puede definirse un software del sistema para abstraer ese recurso en una interfaz de mayor nivel. El modelo básico de funcionamiento de disco mediante bloques se abstrae para proporcionar una operación de escritura de sectores, que de nuevo se generaliza para usar direcciones de bloques enteras. A continuación se tratan las direcciones enteras de los bloques a modo de lista de bloques relacionados que contienen una cadena lógica de bytes. Ahora puede verse que una razón obvia para referirse a los componentes hardware como “recursos” es permitir aplicar la abstracción de recurso a los componentes de los computadores (recursos físicos) y a los artefactos software implementados en el software del sistema (recursos abstractos).

Una Abstracción de Unidad de Disco

La idea bajo la abstracción de recurso puede examinarse mas de cerca considerando cómo pueden representarse las operaciones de salida hacia disco en diversos niveles de abstracción. El dispositivo se controla mediante operaciones software para copiar un bloque de información desde la memoria principal del computador hacia el búfer de memoria del dispositivo (véase la Figura 4(a)):

```
load(bloque, longitud, dispositivo);
```

para mover la cabeza de lectura/escritura a las zonas especificadas de la superficie del disco:

```
seek(dispositivo, pista);
```

y otras operaciones tales como escribir un bloque de datos desde el búfer hacia el dispositivo:

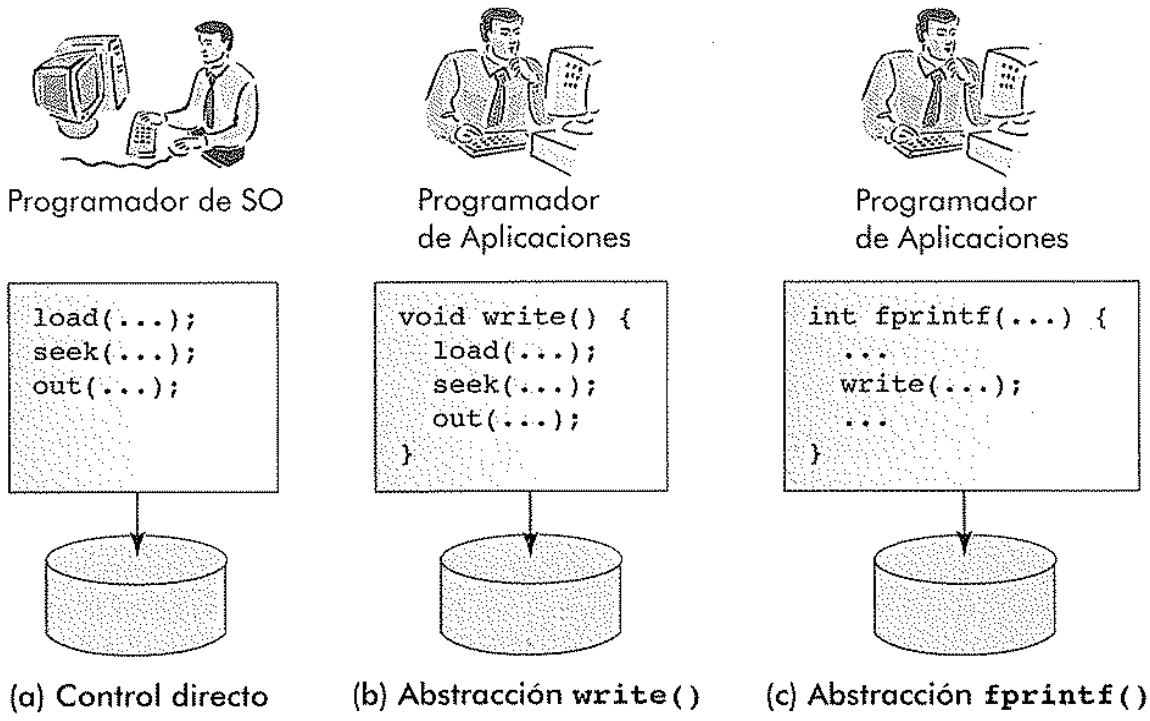
```
out(dispositivo, sector);
```

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

FIGURA.4

Abstracciones de disco

Aquí están las tres formas diferentes de escribir información en un disco. En (a) el software manipula directamente el hardware para seleccionar la dirección del bloque, y después lo escribe mediante la función **out()**. En (b), las instrucciones de la máquina se empaquetan en una función **write()** abstracta. También escribe un bloque de información sobre el disco pero es más sencillo de usar que (a). En (c), la función **write()** también escribe un bloque de información hacia el dispositivo, pero es más fácil de usar que (a) y (c). En (c), la función **write()** está abstraída en una función de biblioteca de tiempo de ejecución de C, **fprintf()**, que realiza E/S formateada con búfer.



Por ello se precisa de una serie de órdenes para escribir información desde un bloque de memoria primaria sobre el disco, tal como:

```
load(bloque, longitud, dispositivo);
seek(dispositivo, 236);
out(dispositivo, 9);
```

Una sencilla abstracción (véase la Figura 4(b)) serviría para empaquetar

estas órdenes, con cualquier otra orden precisa, en una orden write() que funcionara como

```
void write(char *bloque, int talla, int dispositivo, int pista,
int sector)
{
load(bloque, longitud, dispositivo);
seek(dispositivo, 236);
out(dispositivo, 9);
}
```

Las direcciones de los bloques de datos de un disco se especifican mediante un número de pista, como el 236 en la instrucción load, y un número de sector, como el 9 en la instrucción out. Una abstracción de nivel superior podría traducir cada especificación de bloque de modo que se usara una dirección entera no negativa en lugar de una dirección específica de un disco como la pista 236 de la función seek y el sector 9 en la función out. Esto permite que el programador ignore las direcciones físicas definidas por la tecnología del disco a favor de las direcciones lógicas aplicables a cualquier dispositivo de almacenamiento. Ahora, una operación de salida tal como

```
write(bloque, 100, dispositivo, 236, 9);
```

puede escribirse como

```
write(bloque, 100, dispositivo, 3788);
```

Un nivel de abstracción incluso más elevado proporciona al software una forma de tratar el disco como un almacén de archivos. Suponga que el sistema software provee una identificación de archivos, fileID, como abstracción de disco. Entonces una biblioteca, como la biblioteca C stdio, puede darnos una función para escribir una variable entera, datum (almacenada en un pequeño bloque de memoria), sobre el dispositivo como un desplazamiento implícito desde el principio del archivo. El programador usa entonces operaciones como

```
fprintf(fileID “%d”, datum);
```

para escribir información en el disco (vea la Figura 4(c)). Tal abstracción puede usarse también para implementar la misma abstracción para un dispositivo de cinta.

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

COMPARTICIÓN DE RECURSOS

Los computadores se distinguen por su velocidad de cálculo Un computador puede evaluar en microsegundos una expresión numérica que a un humano le podría llevar varios minutos. Esta disparidad de velocidad permite engañar a los humanos de forma que parezca que el computador estuviera ejecutando varios programas simultáneamente, incluso aunque realmente sólo esté ejecutándolos secuencialmente. El SO logra esta ilusión conmutando el hardware entre los programas a una velocidad muy elevada. Esto se parece a un escenario en el que un maestro ajedrecista sea capaz de jugar “simultáneamente” contra varios oponentes. En realidad, el ajedrecista juega contra ellos de modo secuencial atendiendo a cada oponente en momentos diferentes, y cambiando de uno a otro, mientras que el anterior está observando el estado actual del juego.

Los computadores algunas veces pueden soportar una auténtica operación simultánea. Por ejemplo, si un programa desea hacer cálculo numérico a la vez que otro desea leer de un dispositivo de disco, entonces el SO planifica el uso del hardware para que ambos programas se ejecuten a la vez. Esto es posible merced a que la unidad de procesamiento y el disco duro del computador son componentes físicamente diferentes que pueden ser usados a la vez.

En el estudio de los sistemas operativos, difuminamos la distinción entre la apariencia y la auténtica ocurrencia de operaciones simultáneas. Decimos que un sistema soporta ejecución concurrente (o exhibe **concurrencia**) cuando o bien parece que dos o más programas están ejecutándose simultáneamente, o realmente se están ejecutando a la vez. Si son auténticas operaciones simultáneas, decimos que los dos programas se **ejecutan en paralelo** (o simultáneamente).

La ejecución concurrente y paralela está relacionada con la noción de compartición de recursos: Para que los programas en ejecución sean concurrentes o paralelos, deben compartir el mismo computador. Un SO gestiona algunos de los recursos por compartición transparente entre las máquinas abstractas. Es decir, los usuarios y los programadores de aplicaciones no son conscientes de que los recursos están siendo compartidos. Un SO también permite compartición explícita entre programas en ejecución proporcionando mecanismos por los cuales los programadores de aplicaciones gestionan la forma en que se comparten los recursos de la máquina. Primeramente, describiremos la compartición transparente y después discutiremos la compartición explícita.

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

MÁQUINAS ABSTRACTAS Y COMPARTICIÓN TRANSPARENTE DE RECURSOS

La concurrencia es inherente al diseño de un SO y al modelo de operación usado por el programador de aplicaciones. Esto se hace más evidente cuando se piensa en el entorno de ejecución de programas que se ofrece al programador de aplicaciones y al usuario final: varias ejecuciones de programas cada una de las cuales aparentan tener su propio computador privado en el que ejecutarse. Esto se logra diseñando un SO que gestione cuidadosamente el procesador del computador, la memoria, los dispositivos y cualquier otro recurso abstracto que pueda compartirse entre los programas en ejecución, de forma que se presente al programador una abstracción de la máquina en sí (denominada **máquina abstracta**) (véase la Figura 5). Cada máquina abstracta es una “simulación” de un computador real: a cada programa se le da su propia máquina abstracta sobre la que ejecutarse. El SO implementará estas abstracciones compartiendo el hardware subyacente de forma que, idealmente, sea transparente al programador de aplicaciones. Un programa que esté siendo ejecutado por una máquina abstracta es lo que se denomina habitualmente un **proceso**.

Se emplean dos tipos de compartición para crear las máquinas abstractas: la compartición multiplexada en espacio y la compartición multiplexada en tiempo. En la compartición multiplexada en espacio, se divide el recurso en dos o más unidades distintas, y después se asigna a un proceso cada parte individual. Por ejemplo, un complejo de apartamentos está multiplexado en el espacio mediante la división de los edificios en un conjunto de apartamentos, siendo asignado cada uno de ellos a un ocupante diferente. Un autobús es otro ejemplo de compartición multiplexada en espacio, porque varios viajeros comparten el autobús, cada uno en su asiento respectivo. En los computadores, las máquinas abstractas (procesos) pueden multiplexar en espacio un recurso cuando el SO asigna el control exclusivo de las diferentes unidades de un recurso a la vez. La memoria principal y los discos son ejemplos de compartición de recursos multiplexada en el espacio.

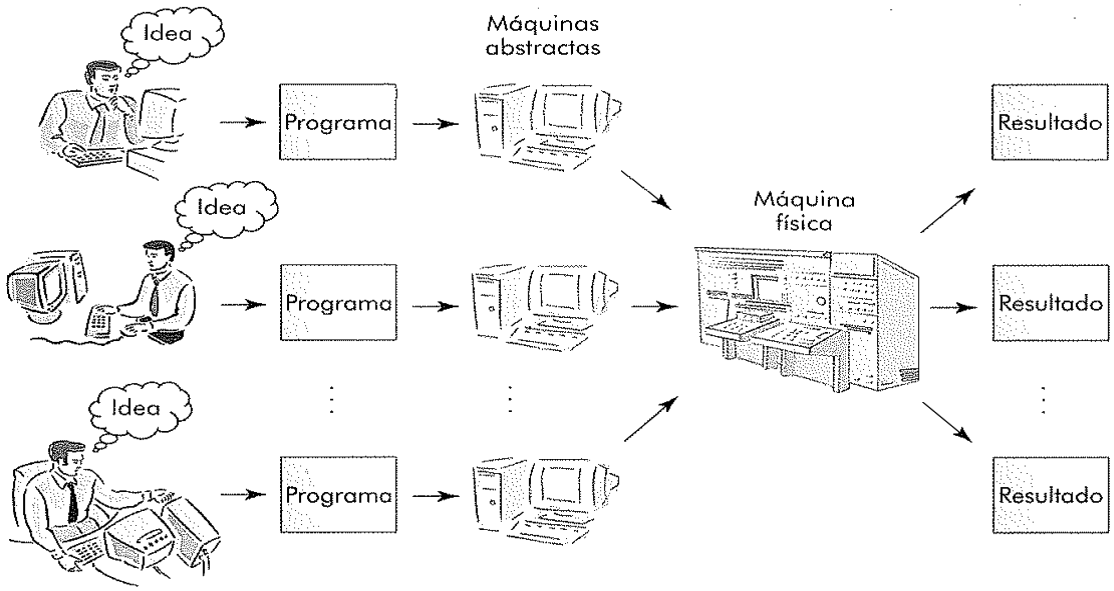
Desarrollo de contenidos teóricos.

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

FIGURA.5

Máquinas abstractas

Un SO puede proporcionar una máquina abstracta para el uso de un programador de aplicaciones creando una simulación de un computador. Al simular la máquina abstracta, el SO puede instruir a la máquina física para que simule varias máquinas abstractas al mismo tiempo.



En la compartición **por multiplexación en tiempo**, no se divide el recurso en unidades, sino que a cada proceso se le da el uso exclusivo del recurso durante un tiempo, y después lo usará otro proceso. Por ejemplo, una plaza de estacionamiento con estancia limitada en tiempo se comparte mediante la técnica de multiplexación en tiempo: un coche podrá tener el control exclusivo de la plaza de aparcamiento completa cada vez, pero tras cierto periodo de tiempo, el primer coche la abandona y un segundo toma el uso del espacio de aparcamiento. (Y sólo para complicar más las cosas, un negocio de aparcamiento público utiliza multiplexación en espacio para dividir el negocio en pequeños espacios de aparcamiento, y multiplexación en tiempo para compartir cada plaza individual de aparcamiento.) En un escenario de transporte urbano, un taxi es un ejemplo de compartición multiplexada en tiempo, puesto que cada ocupante utiliza el taxi exclusivamente, y después lo hace otro ocupante, después de que el primero lo abandone. En los sistemas informáticos, un proceso puede tener el control exclusivo de un recurso

completo durante un tiempo. Tras ese periodo de tiempo, el recurso se desasigna del proceso y es asignado a otro. La multiplexación en tiempo se emplea con el procesador del computador como recurso.

Los diferentes procesos usan sus máquinas abstractas concurrentemente puesto que el SO es capaz de asegurar que cada componente físico de la máquina subyacente es compartido bien en tiempo o bien en espacio. Por ejemplo, tres máquinas abstractas pueden multiplexarse en tiempo compartiendo el procesador, y mientras que un proceso de una máquina abstracta está leyendo del disco, otro proceso está leyendo de otro disco. Los tres tipos de procesos están usando tres partes del hardware diferentes multiplexadas en espacio.

La compartición del procesador de la máquina en el tiempo es un aspecto tan crucial en la implementación de las máquinas abstractas que se suele estudiar como un caso aparte de compartición de recursos. La máquina abstracta de cada uno de los procesos utiliza el procesador físico durante una fracción de segundo (denominada **franja de tiempo**, o *timeslice*), y después el SO multiplexa en tiempo el procesador hacia otra máquina abstracta diferente. Mientras tanto, la memoria del computador está siendo compartida por multiplexación en espacio. Esta técnica para compartir el procesador es tan importante (y ampliamente usada) que se identifica bajo el nombre de **multiprogramación**. En la Figura 6 se desarrolla esta idea intuitiva con una descripción informal. En esta figura, aparecen *N* máquinas abstractas diferentes (denominadas P_i, P_j,...P_k). El SO divide la memoria física en *N* bloques diferentes, y después asigna cada uno a una máquina abstracta. Cuando el programa de P_i se carga en su bloque de memoria, puede compartir el procesador usando compartición multiplexada en el tiempo. Durante cierto periodo de tiempo igual a *N* franjas de tiempo, cada P_i usa el procesador durante cada una de estas franjas, pero ocupa su bloque de memoria durante todo el tiempo

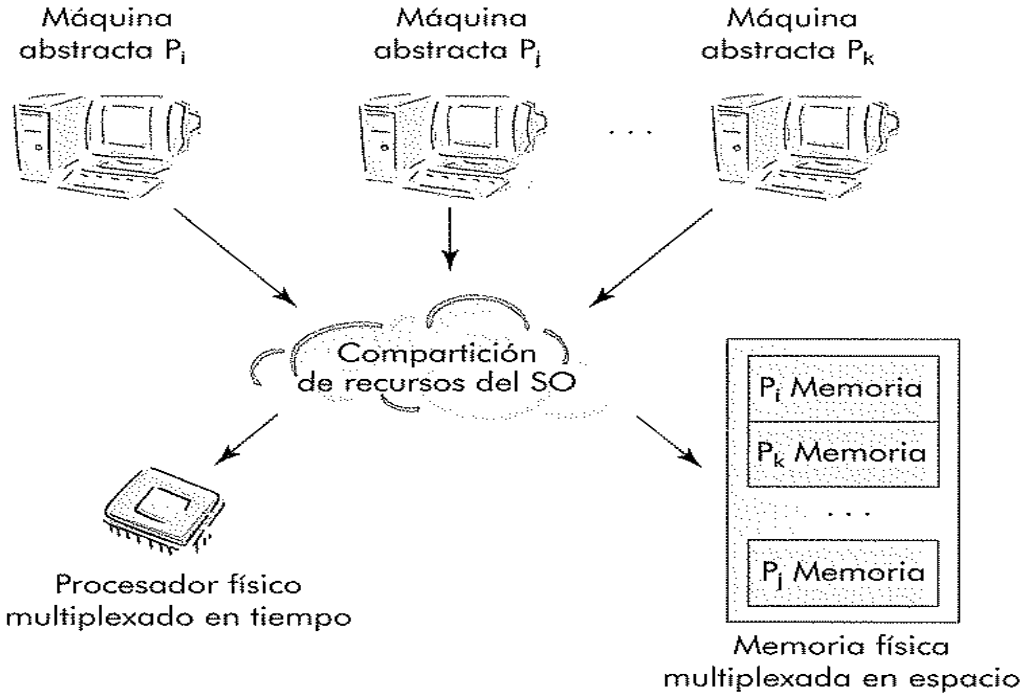
¿Mejora, el uso de la multiprogramación, las prestaciones del computador? *No puede mejorar las prestaciones para un solo proceso, pero puede mejorar las prestaciones conjuntas del sistema.* Veamos cómo funciona esta idea trasladada a un negocio de lavado de automóviles: hay tres coches en espera de ser limpiados. La operación completa precisa de lavar el coche, secarlo y aspirar la suciedad del interior (véase la Figura 7(a)). El coordinador de lavado de coches planifica el trabajo de modo que se lave el coche 1 mientras el coche 2 está siendo aspirado, mientras tanto, el coche 3 está esperando (véase la Figura 7(b)). Ahora que el coche 1 ha sido lavado podrá ser secado. Al mismo tiempo, el coche 2 será lavado y el coche 3 podrá ser

limpiado interiormente. Cuando haya sido secado el coche 1, podrá ser aspirado a la vez que el coche 2 es secado. Tan pronto como el coche 1 ha sido limpiado, y el coche 2 ha sido secado, se han terminado las operaciones con ellos. El único trabajo restante es secar el coche 3 (mientras las estaciones de aspirado y lavado permanecen ociosas). Observe que los coches 1 y 2 requieren ambos el mismo tiempo de operación que requerirían en un lavado “secuencial”, aunque cualquiera de ellos pueda pensar que ha sido el primero en ser atendido

FIGURA.6

Multiprogramación

La multiprogramación es la clave de la tecnología del SO para implementar varias máquinas abstractas. Se obtiene mediante la compartición multiplexada en espacio de la memoria entre cada máquina abstracta y la compartición por multiplexación en tiempo del procesador físico El SO coordina estas tareas al compartir.

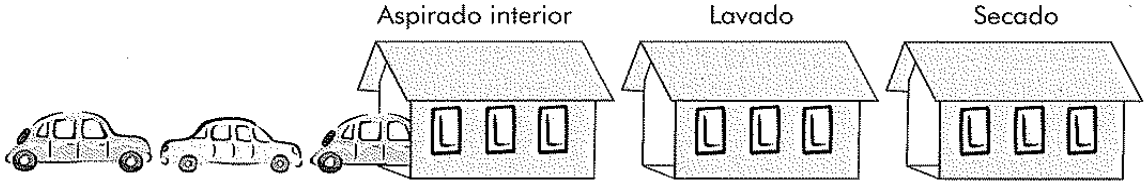


U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

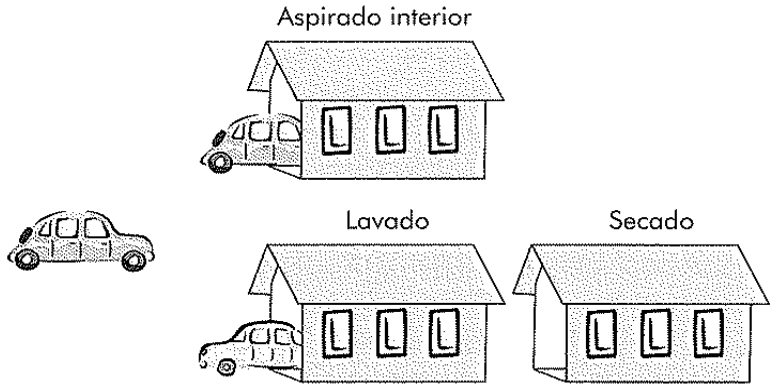
FIGURA.7

Abreviando el lavado de automóviles.

En (a), los coches pasan por cada etapa del procedimiento en el mismo orden. El tercer coche debe esperar porque los dos primeros sean limpiados en su interior antes de poder ser atendido. En (b), el tercer coche es aspirado en su interior a la par que se lava el segundo coche. El tercer coche tan sólo tiene que esperar a que el segundo coche sea aspirado antes de ser lavado en su interior.



(a) El lavado de coches secuencial



(b) El lavado de coches paralelo

El coche 3 requerirá un poquito más de tiempo, pero menos del que hubiera sido si hubiera tenido que esperar por el aspirado de los coches 1 y 2 antes de ser lavados. El sistema de lavado de coches requirió tan sólo 4 etapas para lavar 3 coches. Si hubiera limpiado cada uno de los tres coches con la misma secuencia aspirado-lavado-secado, habría requerido 5 etapas de tiempo.

Es posible aplicar la misma idea básica a los procesos. Aquí tenemos algunas características de la ejecución de procesos que podemos explotar para acelerar el sistema usando el paralelismo:

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

- En los computadores modernos, las operaciones de E/S llevan mucho más tiempo en completarse que las operaciones del procesador.
- El proceso P_i no necesita el procesador mientras está haciendo E/S (como cuando el usuario introduce información, depura un programa, y demás).
- Cada P_i gasta la mayoría de su tiempo usando los dispositivos de E/S del hardware (véase la Figura 8(a)).
- En un computador convencional, hay varios dispositivos pero sólo un procesador

Supongamos que el SO controla el uso del procesador de modo que cuando quiera que P_i trate con la E/S, los otros procesos, P_j, usen el procesador (Figura 8(b)). Entonces lograremos un auténtico paralelismo haciendo que los procesos usen simultáneamente diversas partes del computador.

En un sistema sin multiprogramación (como el lavado de coches secuencial), si *N* procesos tienen tiempos de ejecución t_1 t_2 ..,t_n entonces el tiempo total para servir a *N* usuarios sería

$$t_1 + t_2 ..+ t_n$$

Sabemos que la cantidad de tiempo mínima para ejecutar cualquier P_i es t_i dado que el proceso debe hacer un cómputo y un uso de la E/S conforme a su programa. Esto implica que si fuéramos capaces de planificar la ejecución de todos los procesos de modo que siempre hubiera uno capaz de utilizar un componente diferente del computador en cada momento, entonces la cantidad de tiempo que le llevaría al sistema ejecutarlos *N* procesos sería igual a la cantidad del tiempo para ejecutar el proceso más largo, cual es

$$\text{máximo}(t_1 + t_2 ..+ t_n)$$

Sólo podemos lograr este máximo incremento en la velocidad si las condiciones fueran “precisamente las adecuadas”. Sin embargo, podemos comprobar que el tiempo para ejecutar *N* procesos, *T*', usando un sistema multiprogramado debería ser como máximo

$$\text{máximo}(t_1 + t_2 ..+ t_n) <= T$$

Generalmente (jaunque no siemprel) también podemos tener

$$T <= (t_1 + t_2 ..+ t_n)$$

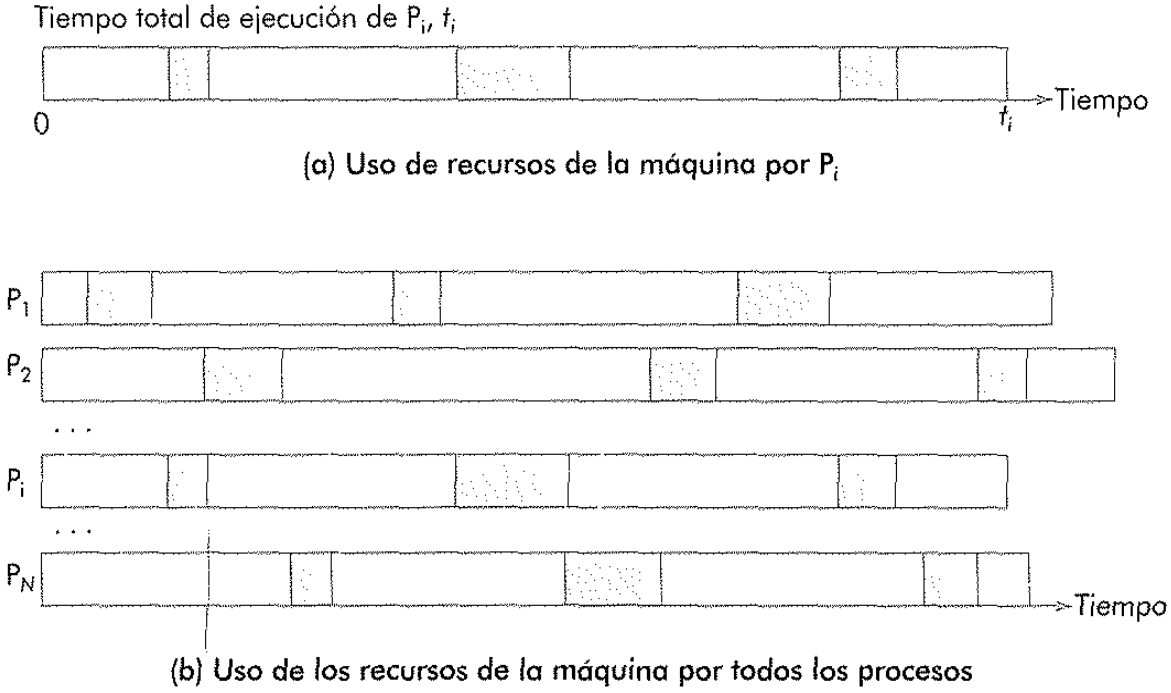
Hay muchas razones para que no tengamos la situación “precisamente adecuada”. Por ejemplo, los programas para esos procesos deben presentar el

balance correcto entre procesamiento y uso de E/S; debe existir un tiempo suficiente para que el SO determine la planificación; debe haber N—1 dispositivos en el sistema; los procesos deben usar todos N—1 dispositivos, y demás. Esto es por lo que tenemos que expresar el incremento de prestaciones como una relación en lugar de una afirmación exacta.

FIGURA.8

Prestaciones de la multiprogramación

En (a), vemos cómo P_i sólo usa el procesador durante tres cortas ráfagas de tiempo, intercaladas con operaciones de E/S. Cuando consideramos al procesador y la actividad de E/S de todos los N procesos en (b), vemos que podría ser factible coordinar su ejecución para que cada uno usara el procesador mientras el resto de los procesos están comprometidos en operaciones de E/S.



COMPARTICIÓN EXPLÍCITA DE RECURSOS

Los mecanismos de compartición explícita de recursos permiten que los procesos usen recursos comunes mediante su propia estrategia de

coordinación (en lugar de la estrategia de coordinación del SO). Por ejemplo, dos procesos podrían estar calculando cooperativamente la nómina mensual, de modo que precisan compartir un archivo que contiene el número de horas trabajadas de cada empleado. Hay dos aspectos importantes para la compartición explícita de recursos, independientemente de si se está multiplexando el tiempo o el espacio.

- El sistema debe poder aislar los accesos a los recursos según una política de asignación.
- El sistema debe poder permitir que los procesos compartan cooperativamente recursos cuando lo precisen.

El **aislamiento de recursos** se refiere a la obligación del SO de prevenir el acceso no autorizado a los recursos por una máquina abstracta cuando están ya asignados a otra máquina abstracta. Por ejemplo, un mecanismo de aislamiento de memoria permite que dos procesos puedan ser cargados a la vez en diferentes partes de la memoria, pero que ninguna de las dos máquinas abstractas tenga acceso al bloque de memoria usado por la otra. Análogamente, el mecanismo de aislamiento del procesador fuerza a las máquinas abstractas a compartir secuencialmente el procesador del sistema. Ningún proceso podrá cambiar o referenciar los contenidos de la memoria que están siendo usados por los otros procesos.

El aislamiento de recursos es obligatorio para la correcta operación de la mayoría de las máquinas abstractas. Sin embargo, en ocasiones donde dos o más procesos traten de cooperar en un problema común, el SO deberá permitir que las máquinas abstractas compartan explícitamente el acceso al recurso compartido cuando se precise. La compartición autorizada es deseable cuando, por ejemplo, un proceso quiere compartir un resultado de un cálculo con otro proceso; los dos procesos quieren compartir un bloque de memoria común donde pueda almacenarse la información compartida.

Al proporcionar un mecanismo de aislamiento de recursos en el SO, generalmente introducimos un nuevo problema. Suponga que el programador intenta que dos procesos compartan un recurso como un archivo. El SO debe incluir mecanismos para permitir que los dos procesos autoricen la compartición aun en presencia del mecanismo de aislamiento. Esto puede ser complicado, puesto que siempre existe la posibilidad de que un proceso malicioso trate de poder acceder a un recurso que ha sido asignado a otro proceso. Si el proceso malicioso consiguiera obtener acceso no autorizado a la memoria, por ejemplo, el segundo proceso sería incapaz de prevenir que el proceso malicioso copiara o sobrescribiera la información que se hubiera almacenado en su memoria.

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

El requisito para el aislamiento de recursos sugiere otra propiedad importante de los sistemas software y los sistemas operativos. Si un proceso confía en el aislamiento de recursos, entonces el sistema software deberá poder proporcionar esa funcionalidad. Suponga que la parte del software del sistema responsable de aislar los recursos no hace lo que se pretende que haga. Entonces podrá fallar el aislamiento de recursos. En el mundo real, esto sería similar a una situación en la que usted confiara en que la policía hiciera cumplir las leyes: la intención está clara, pero si la policía no pudiera hacer cumplir las leyes (da igual si la razón fuera la incompetencia o la corrupción), entonces el mecanismo de cumplimiento de la ley no seria muy útil. Se espera que el software del sistema haga cumplir el requisito dé aislamiento. Si fallara debido a errores de programación o algoritmos inadecuados, entonces no sería muy útil. Los sistemas operativos contemporáneos (en contraste con el software del sistema en general) se construye como software confiable, en el sentido de que se precisa que haga exactamente lo que se pretende para que todo el sistema se comporte correctamente, Las partes cruciales del software de aislamiento de recursos están en el sistema operativo para así proporcionar la máxima seguridad que sea posible.

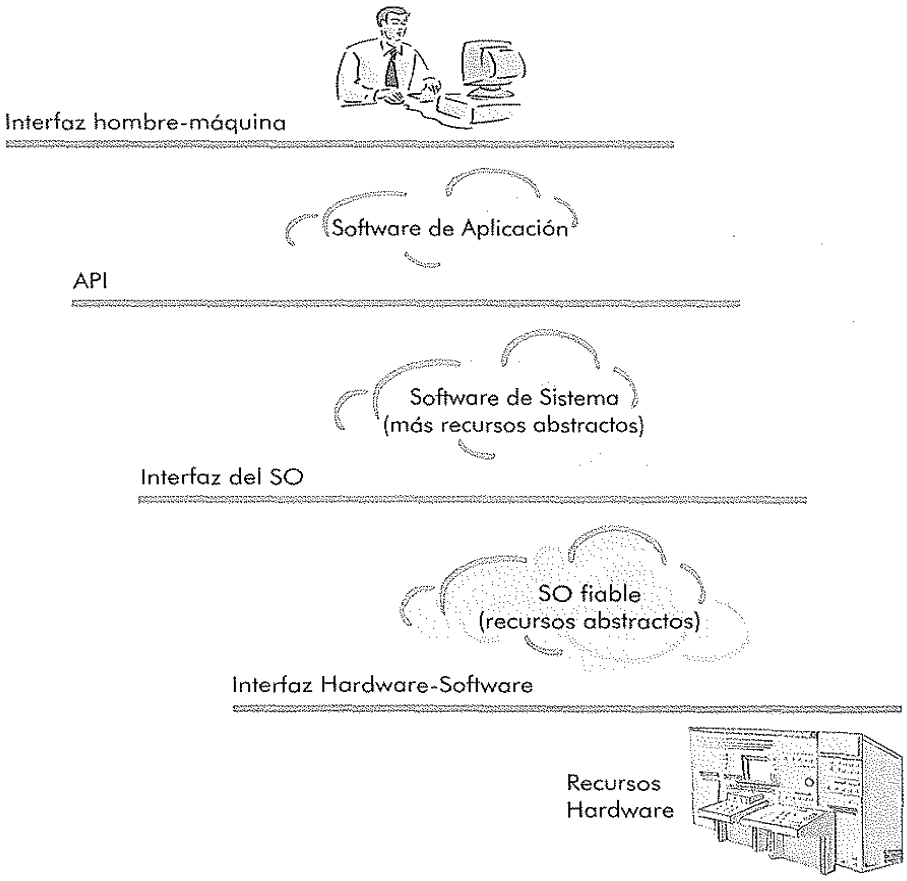
Resumiendo lo que ha aprendido sobre recursos compartidos, la Figura 9 muestra que el SO gestiona los recursos hardware físicos del computador (usando la interfaz software-hardware). El SO se diseña como software confiable que se responsabiliza de la correcta compartición y aislamiento de los recursos hardware. Las abstracciones del SO pueden manipularse usando la interfaz del SO (generalmente llamada **interfaz de llamadas al sistema**, o algo parecido). El software del sistema a parte del SO también puede implementar sus propios recursos abstractos y mecanismos de compartición de recursos (como las bases de datos y las ventanas). Este software del sistema no está implementado como software confiable. Su corrección depende de la confianza en el funcionamiento del SO. Todas las abstracciones software del sistema están accesibles a través de un API . Los programa dores de aplicaciones usan el API (application *programming interface*) del software del sistema para exportar la interfaz humano-máquina, que será usada por el usuario final En general, los programadores se refieren a cualquier interfaz software con el nombre de API (application *programming interface*), aunque la “interfaz de llamadas al sistema” suele referirse a la interfaz del SO. Microsoft ha bautizado su interfaz de llamadas al sistema Windows como “Win32 API” en lugar de interfaz de llamadas al sistema Windows

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

FIGURA 9

Software de Aplicación, Software del Sistema y el SO

Existe una jerarquía entre el software de aplicación, software del sistema y el SO. El SO usa la funcionalidad en la interfaz software-hardware para implementar la interfaz del SO. El software del sistema usa la interfaz del SO para exportar el API. Los programas de aplicación usan el API para crear software que implemente la interfaz humano-computador.



ESTRATEGIAS DE SISTEMAS OPERATIVOS

Durante la corta, y feliz historia de los sistemas operativos (los últimos 45 años, más o menos), se han usado diferentes estrategias para proporcionar los servicios del SO. En este caso, con estrategia nos referimos a las características generales de la máquina abstracta del programador. Por ejemplo, el que haya un número fijo de máquinas abstractas en el sistema, o

que puedan diseñarse máquinas abstractas especialmente para que los usuarios finales puedan interactuar con el software.

La estrategia favorita para un computador dado depende de criterios de negocio y de ingeniería tales como: ¿Quién usará el computador? ¿Es la interacción humana más importante que el ritmo al que se completan las tareas? ¿Habrá más de una persona (quizás controlando diversos procesos) usando el computador a la vez? ¿Será posible implementar una estrategia sin afectar de aburrimiento las prestaciones del sistema completo? Desde los primeros sistemas operativos, no han aparecido más de media docena de estrategias generales. Todas ellas utilizan alguna variante de la noción de máquina abstracta para resolver la abstracción y la compartición de recursos. Veamos muy breve mente las estrategias de SO más importantes.

Los primeros computadores dedicaban todo su tiempo a un único programa cada vez (sin multIprogramación, y esencialmente sin SO). Entonces, igual que ahora, las aplicaciones justificaban el coste de un sistema completo. Puesto que los computadores eran tan caros, sólo se usaban en tareas críticas como las aplicaciones de defensa nacional. Al programador se le daba acceso exclusivo a una máquina completa tanto para desarrollar como para depurar su programa. Cuando el programa estaba listo para funcionar, la máquina se le asignaba completamente al usuario fInal para que ejecutara dicho programa. Puesto que sólo había un programa en ejecución, no era necesario compartir recursos. La única finalidad del software del sistema era simplificar, mediante la abstracción, la programación de los dispositivos.

Hacia 1960, las presiones económicas y la tecnología del software evolucionaron hasta un punto en el que los usuarios esperaban la ejecución concurrente de varios programas en un solo computador; era necesaria una nueva estrategia de SO donde pudieran compartirse los recursos. Esto condujo a las ideas de máquina abstracta y multiprogramación que acabamos de ver. El resto del capitulo describe seis clases diferentes de sistemas operativos y de computadores que abordan de modo identificable estas ideas básicas.

- **Sistemas de procesamiento por lotes o tandas** (*batch*) de trabajos. Un trabajo es una secuencia predefinida de operaciones (tales como “compila un programa” o “ejecuta un programa”), programas y datos. El trabajo es autosuficiente en el sentido de que contiene todos los programas y datos precisos para la ejecución sin necesidad de intervención humana. Por ello, a los sistemas de procesamiento por lotes se les conoce por ser sistemas no interactivos. Los sistemas de procesamiento por lotes fueron los primeros en hacer uso de la multiprogramación. Esto permitía que el SO ejecutara unos pocos

trabajos (aunque generalmente no el trabajo completo) concurrentemente.

- Los **sistemas de tiempo compartido** (*timesharing*) soportan varios usuarios interactivamente. En lugar de que el usuario prepare con antelación un trabajo para su ejecución, el usuario establece una sesión interactiva con el computador y después proporciona operaciones, programas y datos según se precisen durante la sesión. Estos sistemas estimularon el desarrollo de mecanismos de multiprogramación más sofisticados que los de los sistemas de procesamiento por lotes. Por ejemplo, el SO necesario para dar soporte a varios procesos bajo el control de un único usuario. El tiempo compartido también condujo a la necesidad de que el SO proporcionara una respuesta adecuada en tiempo a los usuarios y enfocó la atención sobre la gestión de los recursos y los mecanismos de protección.

- Los **computadores personales y las estaciones de trabajo** fijaron la tendencia de la evolución desde un solo computador compartido entre varios usuarios hacia un entorno en el que la máquina completa estuviera dedicada a un solo usuario. En los sistemas de tiempo compartido, los tiempos de respuesta interactiva dependían del número de personas compartiendo la máquina, pero en estas máquinas dedicadas, los tiempos de ejecución son muy predecibles puesto que todos los procesos pertenecen a un solo usuario. Esta aproximación representa un cambio sustancial en la estrategia del SO porque se basa en la idea de que es más importante minimizar el tiempo de espera para la persona en lugar de tratar de maximizar la utilización del hardware. Además, las máquinas mono-usuario suelen estar multiprogramadas, de modo que el computador pueda realizar diferentes tareas (usando diferentes procesos) concurrentemente.

- Los **sistemas embebidos** (*embedded*) se concibieron originalmente para controlar “sistemas autónomos” tales como las presas hidroeléctricas, los satélites y los robots. En este tipo de aplicaciones, el SO debe garantizar tiempos de respuesta para ciertas tareas. Si el sistema no puede proporcionar el servicio deseado antes de cierto tiempo límite, se considerará que la aplicación ha cometido un fallo. Hoy en día, la tecnología está creciendo rápidamente en importancia dado el deseo de dar soporte a la computación multimedia (con estrategias de tiempos límite más flexibles que en los sistemas de tiempo real tradicionales).

- Los **computadores pequeños con capacidades de comunicación**

(**SCC**, *small, communication computers*), que incluyen los computadores móviles, y los computadores inalámbricos, son los representantes de la clase más reciente de computadores. Eje de este tipo de sistemas incluyen los aparatos Internet, los computadores de portafolio (tablet), aparatos de sobremesa (set-top boxes), teléfonos móviles, y asistentes personales digitales (**PDA**, *personal digital assistant*). Estas máquinas se construyen para ser pequeñas, portátiles, con capacidades de comunicación, y aun así poder soportar muchas de las aplicaciones del tipo de las que se ejecutan en los computadores portátiles y de sobremesa. Esto ha estimulado el desarrollo de un nuevo tipo de sistemas operativos con nuevas políticas de gestión de recursos, estrategias de gestión de energía, capacidad de almacenamiento limitado, y demás.

- La **tecnología de redes** ha evolucionado rápidamente desde 1980. Las configuraciones modernas de computadores usan redes de alta velocidad (incluyendo la Internet pública) para interconectar grupos de computadores personales, estaciones de trabajo, sistemas de proceso por lotes, sistemas de tiempo compartido, sistemas de tiempo real y pequeños computadores. Esto ha influido en la estrategia de los sistemas operativos dada la necesidad de gestionar recursos e información entre las máquinas conectadas en red.

SISTEMAS DE PROCESAMIENTO POR LOTES

Un sistema de procesamiento por lotes sirve trabajos individuales de una colección de trabajos predefinidos. Cada trabajo del lote (*job*) se especifica mediante una lista de órdenes predefinidas del SO (tales como “copia un archivo” o “imprime un archivo”) denominada especificación de control de trabajos. Una vez que el SO comienza a ejecutar un trabajo, procede con todas las órdenes de la lista en secuencia. Los usuarios no tienen ocasión de interactuar con un trabajo en ejecución.

En la década de 1960, los lotes de trabajos se introducían en la máquina en forma de un mazo de tarjetas perforadas. Hoy en día, las especificaciones de ejecución por lotes emplean archivos (como los guiones del shell de UNIX/GNU-Linux o los archivos `autoexec.bat` de Windows) para especificar la lista de operaciones. El SO lee la descripción completa del trabajo y lo dispone para su ejecución. Cuando están disponibles los recursos necesarios para el trabajo, el SO la ejecuta. Tras completar el trabajo, se imprimen los resultados y se devuelven al usuario.

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

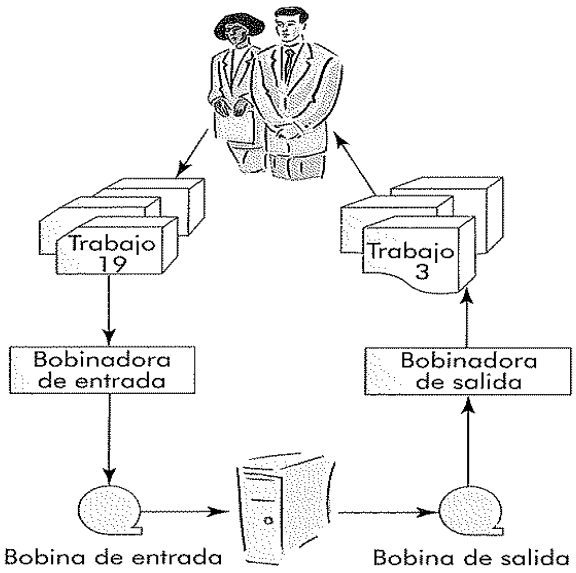
LA PERSPECTIVA DEL USUARIO

Desde el punto de vista del usuario, la especificación de control de trabajos define todas las peticiones necesarias para que el SO realice un trabajo de procesamiento de información. Por ejemplo, si la misión es producir las facturas mensuales de una compañía, el SO deberá ejecutar un conjunto de ejecuciones de programas para producir las facturas: un proceso para acumular las ventas de cada división, otro proceso para determinar las cantidades de las facturas, un tercer proceso para actualizar la información de los activos movilizables de la compañía, etc. Estos programas operan sobre la información contenida en los archivos en vez de la información proporcionada por un usuario interactivo, de modo que no necesita que un humano interactúe con el trabajo mientras está funcionando. Cada usuario prepara una tarea, y después las tareas se recolectan en un lote que se remite al computador (véase la Figura 10). Tras el cómputo del lote, se produce un lote de listados de salida (uno para cada tarea). El listado de salida se le entrega a cada usuario donde se refleja el resultado de cada tarea.

FIGURA 1O

Sistema de Procesamiento por Lotes

Un sistema de procesamiento por lotes maneja una tarea cada vez. Un mecanismo de recolección de trabajos (spooler) agrupa los trabajos en un lote, después los envía al computador. Según va completando los traba jos el computador, los escribe hacia la cola (spoo/) de salida. La cola de salida se imprime y los resultados de cada trabajo se remiten al usuario final.



Los sistemas operativos modernos no usan la estrategia de SO por lotes puro en la que los trabajos se copian de un dispositivo de entrada en una cola del sistema para su procesamiento. Ni siquiera la idea de trabajo no-interactivo es útil ya. En este caso, el usuario prepara una especificación de control de trabajo de una lista de operaciones de SO (que almacena en un archivo). El usuario dirige entonces al SO para que ejecute las operaciones del archivo haciendo que se ejecute el “trabajo” sin participación del usuario. La mayoría de las aplicaciones

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

contemporáneas se adecuan bien a este estilo de procesamiento por lotes, dado que no precisan de la interacción humana según se ejecutan. Por ejemplo, las facturas mensuales todavía se preparan de esta forma. Las otras aplicaciones por lotes incluyen la impresión de las nóminas, la actualización de los listines telefónicos, o la toma de datos de análisis sísmicos.

TECNOLOGÍA DE PROCESAMIENTO POR LOTES

En un sistema por lotes, un componente dedicado a reunir tandas de trabajos (spooler, bobinadora) lee cada trabajo y lo almacena en el lote de trabajos actuales. Los trabajos se guardaban en unas cintas magnéticas, y después la cinta se montaba en el computador principal. Entonces era cuando el computador principal leía los trabajos desde la cinta de entrada de lotes, los ejecutaba, y grababa los resultados en una bobina de salida. Después de que el trabajo se ejecutara en el computador principal, la bobina de salida se imprimía en otra máquina bobinadora. Según los sistemas se fueron haciendo más rápidos, las operaciones de recolección y de entrada y salida en cintas se fueron haciendo mediante subsistemas de E/S del computador principal, y el lote podía guardarse en disco en lugar de en cinta magnética.

El sistema operativo utiliza políticas para determinar el orden de ejecución de los trabajos. Una vez el SO ha seleccionado un trabajo del lote, le asigna un bloque de la memoria principal del computador (lo que se denomina planificación a medio plazo). Tras alojar la trabajo en la memoria, podrá comenzar a competir por el procesador. Cuando el procesador queda disponible, el planificador del procesador (también llamado planificador a corto plazo) selecciona un trabajo ya cargado en la memoria y le asigna el procesador.

Un trabajo puede usar el procesador tan sólo cuando está cargado en la memoria Cuando un trabajo completa su ejecución, su memoria se libera y se copia la salida del trabajo hacia un spool de salida para la ulterior impresión. En algunos sistemas de proceso por lotes, el SO fuerza la liberación de la memoria asignada a un trabajo; esto implica que el trabajo no podrá ya competir por el procesador Dichos sistemas se denominan sistemas con desalojo (el desalojo se usa también en los sistemas de tiempo compartido). La política de desalojo puede incluir liberar la memoria del trabajo y mover los contenidos hacia un disco si es que el trabajo ha sido particularmente pesado o también particularmente ligero para el procesador. En la primera estrategia, se penaliza un uso intensivo del procesador desalojándolo para que otros programas puedan tener más acceso al procesador. En la segunda, un uso liviano del procesador también puede invitar a su desalojo, pues no tendrá mucha

importancia. El criterio principal es liberar el uso de la memoria (como recurso) cuando un trabajo está ausente.

En tiempos en que el procesamiento por lotes era la estrategia predominante de SO, los computadores se utilizaban básicamente para tratar grandes cantidades de datos. El procesamiento de datos financieros resultó ser una aplicación computacional viable, animando al desarrollo de la tecnología de archivos. Los archivos son la abstracción natural de los dispositivos de almacenamiento de disco para los sistemas de procesamiento por lotes porque proporcionan una forma de controlar grupos de información similar (tales como registros de ficha de entrada y salida, datos de trayectorias, o registros personales). Al crear y refinar las abstracciones de archivos, los programadores no sólo disponían de una colección de información, sino que también se les liberaba de la obligación de conocer los detalles de E/S de disco (véase el ejemplo "*Una Abstracción de Unidad de Disco"* mas arriba).

Los sistemas de procesamiento por lotes proporcionaban un gran avance hacia la posibilidad de que diversos usuarios compartieran la máquina. Sin embargo, los sistemas por lotes multiprogramados impedían la interacción en tiempo real entre el usuario y el computador; las intenciones del usuario sólo se representaban mediante la especificación de control de trabajo. En los sistemas que comenzaban a hacerse sitio sobre los sistemas por lotes, el usuario era capaz de sentarse en la consola del sistema y depurar un programa. En los sistemas por lotes, los programas sólo podían ser depurados preparando un trabajo, remitiéndola al planificador de la cola (spooler), y esperando la ejecución del trabajo y los datos devueltos. Para empeorar las cosas, no se permitía que los usuarios introdujeran los trabajos por ellos mismos, ni siquiera que eliminaran la salida de la cola de la impresora. De hecho, el sistema de lotes podría estar ubicado en un punto geográficamente distante. No era raro que un programador profesional tuviera tan sólo dos oportunidades al día de introducir un trabajo en el flujo de lotes del sistema. ¡Compare ese modo de desarrollo con los entornos actuales en los que se puede recompilar y ejecutar un programa en segundos!

Archivos de Trabajos

Los sistemas operativos contemporáneos como UNIX/GNU-Linux y Windows soportan el procesamiento de archivos de trabajos. Incluso aunque estos sistemas operativos son sistemas interactivos de tiempo compartido,

un usuario puede preparar un archivo de trabajos con un conjunto de órdenes del SO y ejecutarlas sin intervención de usuario alguno. Los ejemplos más sencillos de esto son los archivos `config.sys` y `autoexec.bat` de DOS. Estos archivos contienen listas de órdenes a ejecutar para cuando se inicia el sistema en el computador.

La Figura 11 muestra un archivo de trabajos (un guión del shell) de un sistema UNIX. El archivó de trabajos puede ser ejecutado por un shell, ocasionando que cada línea del archivo sea tratada como una operación del SO. En el ejemplo, el primer paso es compilar un archivo denominado `menu.c`, produciendo un archivo reubicable denominado menu.o . La segunda línea del archivo de órdenes compila un archivo denominado `manejador.c` y enlaza su archivo objeto reubicable con `menu.o` en la librería C. La tercera línea ejecuta el archivo `manejador` producido por la ejecución del enlazador leyendo el archivo de entrada denominado `prueba_datos` y escribiendo el archivo de salida denominado `prueba_salida`. La cuarta línea imprime el archivo prueba_salida hacia una impresora denominada `laimpresora`. La quinta línea produce un archivo tar denominado `manejador_prueba.tar` que contiene el código fuente y la salida de prueba. La última línea en el archivo de órdenes codifica el archivo tar y guarda el resultado en un archivo denominado `manejador_prueba.encode`.

FIGURA.11
Un Archivo de Guión de Trabajos para Shell de un Sistema UNIX

Un guión de shell de UNIX es un archivo de trabajos. Contiene una serie de órdenes (6 órdenes en este ejemplo) que pueden ser leídas y ejecutadas por el shell sin intervención humana.

```
cc -g -c menu.c
cc -g -o manejador nanejador.c menu.o
manejador < prueba datos > prueba salida
lpr -PlaImpresora prueba_salida
tar -cvf manejadorprueba.tar menu.c manejador.c prueba_datos prueba_salida
uuencode manejador_prueba.tar manejador_prueba.tar > manejador_prueba.encode
```

SISTEMAS DE TIEMPO COMPARTIDO

Los sistemas de tiempo compartido comenzaron a popularizarse en la

década de 1970. Su objetivo era permitir que varios usuarios interaccionaran con el sistema informático a la vez, cada uno usando su propio teclado y pantalla. Esta estrategia fue el primer paso en ofrecer las variadas capacidades de procesamiento de información a la gente. Anteriormente, los computadores eran usados tan sólo por un pequeño número de especialistas.

Hubo cuatro sistemas que definieron inicialmente la estrategia de los SO de tiempo compartido:

- **CTSS (Compatible Time Sharing System**, Sistema Compatible de Tiempo Compartido). CTSS se desarrolló hacia mitad de la década de 1960 en el MIT. Fue el vehículo que soportó la primera investigación en algoritmos radicalmente nuevos para la planificación de la multiprogramación (radicales, comparados con los que había en aquellos entonces) y técnicas de gestión de memoria modernas.
- **Multics**. Multics reemplazó CTSS al poco de volverse éste operativo. Antes del desarrollo de Multics, los sistemas operativos solían ser poco fiables, y solían caer esporádicamente sin razón aparente. Multics se diseñó explícitamente para ser altamente fiable. Fue también, el sistema más avanzado (de su tiempo) en experimentación con memoria virtual, protección y seguridad.
- **Cal**. El sistema de tiempo compartido Cal se diseñó y se implementó por la misma época que CTSS y Multics. Este sistema se orientaba a la tecnología de compartición de tiempo, protección y seguridad, en general.
- **UNIX**. Algunos diseñadores de los laboratorios Bell de ATT, relacionados con Multics, deseaban disponer de un SO más simple para gestionar un pequeño computador de un laboratorio, de modo que desarrollaron UNIX hacia 1970. La primera contribución de UNIX fue la filosofía de diseño del SO: “lo pequeño es bello”. UNIX demostró la plausibilidad de crear un pequeño núcleo de SO con una funcionalidad mínima, pero capaz de soportar una clase amplia de servicios del SO ejecutándose como programas de aplicación. Mientras que CTSS, Multics, y Cal desaparecieron años más tarde, UNIX es aún una tecnología de sistema operativo importante (con muchas actualizaciones a lo largo de los años).

LA PERSPECTIVA DEL USUARIO

Los sistemas por lotes forzaban al usuario a planificar cuidadosamente cómo debía de ejecutarse el trabajo antes de remitirlo al computador Los

sistemas de tiempo compartido seguían la filosofía de que el usuario podía establecer una sesión con el sistema (es decir, de “entrar en/dentro del” sistema) y decidir después qué orden procesar justo en ese momento. Durante la ejecución, el usuario interactúa directamente con el SO y el proceso, aportando información en respuesta a los requerimientos de entrada de datos del programa y viendo el resultado directamente como consecuencia de las órdenes de escritura. Esto animaba a los usuarios a experimentar con la información, por ejemplo, para abordar problemas de apoyo a la decisión comprobando diferentes escenarios “qué pasaría si”. El computador puede aprovecharse para un conjunto completamente nuevo de dominios de procesamiento de información que previamente no eran factibles en los sistemas anteriores no interactivos.

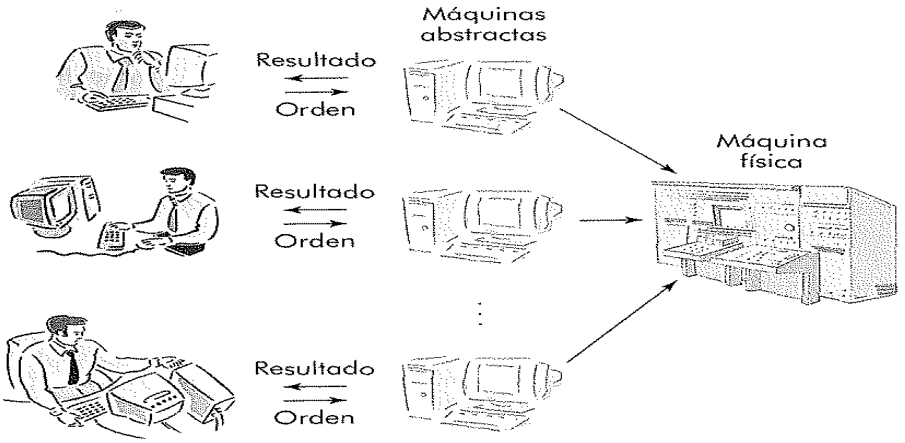
La descripción de las máquinas abstractas multiprogramadas de la Figura 5 también describe la perspectiva de usuario de un SO de tiempo compartido. Un sistema de tiempo compartido utiliza multiprogramación, pero también permite que los usuarios interactúen con los programas en ejecución (véase la Figura 12). Como en la multiprogramación, cada máquina abstracta es una simulación de un hardware real, pero un usuario interactúa con el computador tecleando una orden en una consola abstracta y viendo los resultados de vuelta desde la máquina tan pronto como la orden se completa.

Los sistemas de tiempo compartido se fijan en las políticas de implementación para compartir equitativamente el procesador. Esto permite que los usuarios vean la máquina como si tuvieran el control exclusivo de un computador comparativamente más lento (la máquina abstracta). Si es el caso de que el sistema de tiempo compartido no se encuentra sobrecargado, el tiempo relativo de respuesta será habitualmente tan pequeño que ningún usuario nota una ralentización comparable en las prestaciones del sistema.

FIGURA 12

Sistema de tiempo compartido

Los sistemas de tiempo compartido son sistemas multiprogramados que permiten que el usuario final interactúe con el computador entre dos órdenes cualesquiera del SO. Los sistemas de tiempo compartido son computadores interactivos.



U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

TECNOLOGÍA DE COMPARTICIÓN DE TIEMPO

Un SO de tiempo compartido usa la multiprogramación para dar soporte a varias máquinas abstractas. Sin embargo, las estrategias de planificación y asignación de memoria de los sistemas de tiempo compartido difieren significativamente de las usadas en los sistemas por lotes. Mientras que en los sistemas por lotes pretendían optimizar el número de trabajos procesados por hora, un sistema de tiempo compartido pretende proporcionar cantidades de equitativas de recursos de procesamiento y memoria a cada máquina abstracta. Un sistema de tiempo compartido usa políticas de asignación de recursos diferentes de los de un sistema por lotes.

Según fueron evolucionando los entornos de tiempo compartido, dieron lugar a que los diseñadores distinguieran entre la noción de trabajo y la de programa en ejecución, puesto que el usuario de un sistema de tiempo compartido podría desear, implícita o explícitamente, que dos programa diferentes se ejecutaran al mismo tiempo. Esta idea condujo directamente a la noción de proceso como cualquier “programa en ejecución”. El usuario de un sistema de tiempo compartido podría lanzar dos o más procesos en cada momento, mientras que un trabajo del lote ejecutaría sólo un programa en representación del trabajo global. Por ejemplo, en los entornos de tiempo compartido, un proceso podría estar traduciendo un documento de un formato a otro, y al mismo tiempo otro proceso estar leyendo un archivo de correo. Según fue evolucionando la compartición de tiempo, los procesos fueron denominándose **tareas** (task). Un sistema multiprogramado de tiempo compartido que soporta varios procesos por usuario suele denominarse un sistema **multitarea**.

Mientras que todas las máquinas multiprogramadas soportan varios usuarios, los sistemas de tiempo compartido resaltan la importancia de establecer barreras y salvaguardias entre los usuarios y sus procesos. Esto se debe en parte al hecho de que los sistemas de tiempo compartido permiten que coexistan muchos procesos a la vez, mientras que solo había unos pocos en un sistema por lotes. Sin barreras entre los procesos, un proceso podría dañar inadvertidamente la imagen en memoria de otro proceso. Las barreras

que se interponen para garantizar la protección de la memoria también hacen más difícil compartir información entre dos trabajos, puesto que ambos trabajos deben pasar por encima del esquema de protección de memoria. Las barreras de protección también se extienden por el sistema de archivos compartidos por los usuarios. En muchos casos, es deseable que un usuario pueda crear archivos que no puedan ser escritos por otros usuarios y en algunas ocasiones que nisiquiera puedan ser leídos. La protección y la seguridad se volvieron temas importantes en los primeros tiempos de la compartición de tiempo, incluso aunque estas cuestiones también se apliquen a los sistemas por lotes

El sistema de tiempo compartido UNIX

El sistema operativo UNIX fue presentado en 1974 al público en un artículo de investigación [Ritchie y Thompson, 1974]. UNIX fue desarrollado por dos investigadores de los laboratorios Bell de AT&T que se encontraban poco satisfechos con el SO de sus computadores de investigación. UNIX estableció dos nuevas tendencias en el diseño de SO: los anteriores sistemas operativos eran paquetes de software monstruosos (usualmente el paquete de software más voluminoso funcionando dentro del computador) y estaban escritos para una plataforma hardware concreta (el SO lo vendía el propio fabricante del computador). *Por el contrario, UNIX se pensó para ser pequeño, un SO sin lindezas que podía ser portado a cualquier computador pequeño*. La filosofia de UNIX era que el SO, denominado núcleo, proporcionaría la funcionalidad mínima esencial, y que cualquier funcionalidad nueva debiera ser añadida (como programa de usuario) sólo cuando se precisara. UNIX era revolucionario, y hacia 1980, ya se había convertido en el SO preferido por los programadores en los entornos de hardware de diversos fabricantes (universidades, investigación, laboratorios de investigación y organizaciones de desarrollo de software del sistemas).

Incluso aunque el núcleo de UNIX se podía portar sobre nuevas plataformas hardware sin tener que volver a desarrollar el SO por completo, existía una barrera diferente a la difusión general del SO. El código fuente era propiedad de los Laboratorios AT&T, y sólo podía usarse adquiriendo una licencia. El resto de organizaciones podían hacerse con una licencia (por ejemplo, para portar UNIX en su hardware preferido) pagando las tasas a AT&T Hacia 1980 muchas universidades y laboratorios de investigación habían ya obtenido el código fuente y se mantenían muy ocupados modificándolo para adaptarlo a sus propias necesidades, siendo el caso más

importante el de la Universidad de California en Berkeley bajo contrato de investigación con DARPA (*Defense Advanced Research Projects Agency*, Agencia de Proyectos de Investigación Avanzada para la Defensa). Los fabricantes de computadores comerciales también habían comenzado a usar el código fuente de UNIX para obtener su propia versión del sistema operativo UNIX.

Por 1985, había ya dos versiones principales de UNIX (funcionando en muchas plataformas hardware diferentes); la línea principal de versiones de los Bell Labs de AT&T (denominada UNIX Sistema V) y una versión alternativa de la Universidad de California en Berkeley (bautizada UNIX BSD). La versión para VAX de DEC a partir de UNIX BSD se bautizó UNIX BSD Versión 4, o simplemente UNIX BSD 4.x. Ambas versiones implementaban interfaces de llamada al sistema reconocibles como UNIX, aunque diferían entre ellas en algunos detalles. Había, sin embargo, diferencias sustanciales en la forma en que estaban implementados ambos núcleos de SO. La competición entre Sistema V y UNIX BSD fue muy activa, con lo que los programadores cambiaban de una versión a otra. Actualmente, los principales ponentes comerciales de UNIX BSD 4.x (Sun Microsystems) y AT&T han suscrito un acuerdo por el que estas dos versiones principales se unen en una versión común de UNIX (el sistema operativo de Sun, Solaris)

Mientras tanto, otros fabricantes de computadores patrocinaron una implementación alternativa de la interfaz de llamadas al sistema de UNIX Un suceso importante fue la formación de un comité para desarrollar una interfaz de llamadas al sistema de UNIX estandarizada: **POSIX.1**. (Esta interfaz de llamadas al sistema suele llamarse simplemente “POSIX,” aunque pueda llevar a confusiones, puesto que el comité POSIX también desarrolló varios otros API y sólo uno de esos estándares aborda la interfaz de llamadas al núcleo del sistema. En este documento consideraremos solamente POSIX.1, y usaremos con generalidad el término, más popular, aunque menos acertado, de “POSIX” para referirnos a la interfaz de llamadas al sistema POSIX. 1. Una vez establecido POSIX, los desarrolladores tuvieron la libertad de diseñar y desarrollar sus propios núcleos que proporcionaran la funcionalidad especificada en este API. Por ejemplo, en la Universidad Carnegie Mellon, un grupo de investigadores de SO dirigidos por Richard Rashid desarrollaron el sistema operativo Mach con una interfaz de llamadas al sistema de tipo POSIX/UNIX . Mach era una alternativa a las implementaciones de núcleos Sistema V y UNIX. Una versión de Mach se utilizó como base del núcleo de la Fundación para los Sistemas Abiertos (OSF, Open Systems Foundation) llamado OSF-1, que es la base del SO de Macintosh X. La tendencia continua y diversas implementaciones de UNIX de

código abierto se han vuelto populares (como Linux y FreeBSD) Finalmente parece que los desarrolladores de software están comenzando a usar estas implementaciones abiertas, y la barrera de código fuente con licencia de UNIX comienza a desaparecer.

El interprete de línea de ordenes (el shell Bourne) también estableció una tendencia importante en la forma en que los usuarios podían interaccionar con el SO Las ideas básicas que fueron desarrolladas e implementadas en el shell Bourne están actualmente universalizadas en las interfaces hombre-maquina.

UNIX fue creado tras los sistemas de investigación CTSS, Multics y Cal, por lo que se beneficio considerablemente de la exploración realizada en estos sistemas pioneros Se construyó sobre la base de soportar muchos procesos y sobre la necesidad de un alto grado de interacción con el usuario final. UNIX fue el banco de pruebas para refinar los conceptos fundamentales de SO de dispositivos reconfigurables, maquinas abstractas, seguridad y memoria virtual.

Hacia la mitad de la década de 1980, UNIX era reconocido como el SO de tiempo compartido dominante, así como un SO importante para las estaciones de trabajo.

COMPUTADORES PERSONALES Y ESTACIONES DE TRABAJO

El Apple II fine lanzado en abril de 1977 y el Computador Personal de IBM fue lanzado en agosto de 1981 Durante la siguiente década, el software de los sistemas personales fue diseñado para permitir que un usuario ejecutara un programa en cada momento (sin multiprogramación) Puesto que no había multiprogramación, no era necesario aislamiento de recursos o compartición El principal requisito del sistema software era proporcionar una abstracción del hardware. Apple proporcionaba un conjunto de funciones (que mas adelante se convirtió en el Toolbox -caja de herramientas-), y el PC de IBM daba su propio software de abstracción de dispositivos conocido como BIOS (Basic Input/Output System, Sistema Básico de E/S) Tanto Apple como IBM alojaron su software de abstracción en memoria de solo lectura (ROM, read only memory), de forma que siempre estuviera presente cuando se encendiera la máquina Es decir, una ROM es un dispositivo de interno memoria que presenta la propiedad de que la información se podía almacenar en él, y ésta permanecerá incluso después de apagar la máquina Estos computadores tempranos desaparecerían

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

hacia 1990, aunque la idea del software de abstracción BIOS de IBM aun se usa en los microprocesadores basados en el modelo de Intel

En 1982, Sun anunció su primer computador pequeño. Algunos otros fabricantes (como HP, Apollo y Three Rivers) también comenzaron a entregar estaciones de trabajo (*workstation*) Una **estación de trabajo** era bastante diferente de un computador personal (como el PC de IBM o el Apple II) Las estaciones de trabajo se configuraban con suficientes recursos como para que tuviera sentido la tecnología SO de tiempo compartido, especialmente la multiprogramación. Casi todos los computadores usaban al final alguna forma de UNIX

Por 1990, había tres facciones distintas: los computadores personales de Apple (el Macintosh fue anunciado en 1984), los computadores personales IBM y las estaciones de trabajo UNIX. La competencia fue feroz entre los dos grupos dedicados a los computadores personales, puesto que las estaciones de trabajo siempre fueron consideradas de otra categoría. Hacia 1995, el hardware de los computadores personales se había vuelto tan sofisticado que estas máquinas comenzaron a competir con las estaciones de trabajo. Mientras tanto Microsoft mejoró su oferta de SO con Windows NT y Windows 95. El terreno de juego se trasladó a los computadores personales compatibles IBM (IBM había vuelto su atención a otros tipos de computadores) y las estaciones de trabajo. Hoy en día, no puede apreciarse una diferencia importante entre un computador personal y una estación de trabajo. La mayoría de los principios y los conceptos descritos en este documento aparecen tanto en los computadores personales actuales como en las estaciones de trabajo.

LA PERSPECTIVA DEL USUARIO

Los computadores personales y las estaciones de trabajo nos han proporcionado una nueva libertad de cómputo, que ha cambiado la forma en que la gente percibe un computador. En lugar de verlos como un oneroso recurso corporativo, la gente comienza a verlos como una herramienta para hacer el trabajo del día a día, similar al teléfono, la máquina de escribir o la fotocopiadora. Según fueron evolucionando las herramientas de productividad personales como los procesadores de texto, los sistemas de autoedición, las hojas de cálculo y las bases de datos personales, el computador monousuario ha arraigado fuertemente en las empresas.

TECNOLOGÍA DE SO

La tendencia hacia los computadores de un único usuario comenzó con el

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

desarrollo de los computadores personales que pudieron situarse directamente en la oficina en lugar de una sala especial de computadores. Los minicomputadores, que comenzaron a aparecer en la década de 1970, fueron el primer ejemplo de dicha tecnología. Los primeros minicomputadores, como la PDP-8 de Digital Equipment y la Nova de Data General, eran muy asequibles y fáciles de instalar en cualquier lugar (en comparación con los computadores convencionales de la época, que precisaban aire acondicionado y alimentación de potencia específica). Los minicomputadores se hicieron muy populares en la década de 1980, pues podían usarse tanto como computadores personales (generalmente para el desarrollo de software), como de máquinas de tiempo compartido. La máquina de tiempo compartido PDP-11 de Digital Equipment fue la plataforma favorita para UNIX; bien como máquina personal de desarrollo de software, o bien como pequeña máquina de tiempo compartido. Finalmente, la PDP- 11 evolucionó en el popularísimo computador de tiempo compartido VAX; de Digital Equipment, probablemente el computador de tiempo compartido más ampliamente difundido en la década de 1980.

Por la época en que los minicomputadores se volvían más grandes, de esta misma tecnología surgieron máquinas más pequeñas (los microcomputadores). El elemento principal de un microcomputador es un procesador implementado en un único circuito integrado. En poco tiempo se construyeron microcomputadores como los basados en el Intel 8008 con procesamiento de 8 bits y velocidades de reloj de cerca de 1 millón de ciclos por segundo (1 MHz). Para comparar, los microcomputadores actuales emplean microprocesadores de 32 bits (o incluso 64 bits) con velocidades de reloj que exceden los 2.500 MHz (2,5 GHz). Tanto los computadores personales como las estaciones de trabajo se basan en microprocesadores.

El primer computador personal incluía un rudimentario sistema operativo (como el BIOS de IBM). Estos “sistemas operativos” basados en ROM proporcionaban unas pocas rutinas para controlar los dispositivos del computador personal. En poco tiempo, los sistemas basados en ROM se mejoraron con software de SO adicional, generalmente para gestionar archivos, que podía cargarse desde el disco sobre la **memoria de acceso arbitrario** (RAM, *random access memory*) de lectura/escritura. El sistema operativo más popular para computadores personales fue CP/M, que finalmente fue desplazado por el MS-DOS de Microsoft (o la versión de IBM, PC-DOS). Estos sistemas operativos extendían el software de abstracción de dispositivos proporcionando un sistema de archivos.

Los computadores personales con MS-DOS dominaron finalmente el resto de productos de sistema operativo del mercado. Hoy en día, muchos

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

computadores personales utilizan sistemas operativos descendientes del de Microsoft, Windows 95/98Me y Windows NT/2000/XP. La principal contribución de MS-DOS (a la tecnología del SO) fue la popularización de los computadores y su flexibili dad en la configuración de las diversas partes del SO en la fase de encendido.

El hardware de las estaciones de trabajo fue en origen más flexible y rápido que el de los computadores personales. Las estaciones de trabajo usaban hardware bastante similar al de los minicomputadores de la época (como la PDP-11). También incluían más recursos que los computadores personales. Por ejemplo, tenían más memoria, y más rápida, un procesador más veloz y potente, mayor capacidad de disco y dispositivos gráficos de mayor resolución por consola Dada la configuración de recursos, las estaciones de trabajo precisaban un SO más complejo que el MS-DOS para gestionar dichos recursos.

Aunque UNIX se diseñó como un sistema de tiempo compartido, su soporte a la multiprogramación, así como su extensibilidad satisfacía de un modo natural el entorno de las estaciones de trabajo, concretamente cuando la estación era usada para el desarrollo de software. UNIX creció con el mercado de estaciones (y minicomputadores). Según el mercado solicitaba soporte de gráficos, UNIX incorporaba medios para soportar gráficos de alta resolución. Análogamente, cuando los protocolos de red se volvieron importantes para la tecnología de las estaciones de trabajo, UNIX comen zó a soportar protocolos de red. Ahora cuando las estaciones y los computadores personales son indistinguibles, podemos usar un SO tanto de un computador personal como UNIX.

CONTRIBUCIONES A LA TECNOLOGÍA DE LOS SO MODERNOS

Los computadores personales y las estaciones estimularon grandemente el crecimiento del software del sistema para dar soporte a las herramientas de cómputo personales. Esta demanda, por otro lado, provocó la convergencia de los intereses de los desarrolladores de SO y los desarrolladores de interfaces hombre-máquina, por ejemplo, al crear interfaces efectivos del tipo apuntar-y-seleccionar. Los sistemas de ventanas OpenWindows/NeWS, de Sun, y X/Motif se encuentran profundamente enraizados en la tecnología (e implementación) del software del sistema. El interés en este tipo de máquinas también estimuló que los nuevos desarrollos de SO soportaran varias sesiones y terminales virtuales.

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

La familia de SO Windows de Microsoft

El primer SO de Microsoft fue MS-DOS. Su cometido era principalmente la abstracción de dispositivos, dado que los primeros computadores personales no se pensaron como computadores multiprogramados. Actualmente, la BIOS usada con los computadores basados en procesadores de Intel aún reflejan reminiscencias de la abstracción de dispositivos de MS-DOS. MS-DOS fue el SO dominante en los computadores personales hasta mediados de la década de 1990, cuando fue reemplazado paulatinamente por sistemas operativos más modernos (por lo general un SO de Microsoft más moderno).

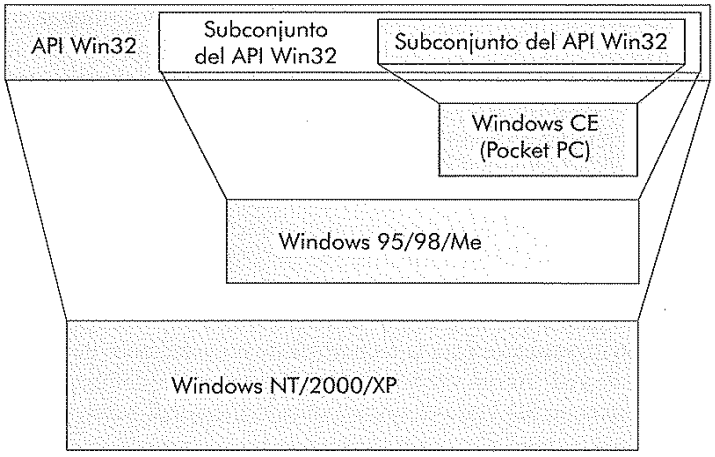
Los sistemas operativos contemporáneos de Microsoft (denominados sistemas operativos Windows) exportan su propio subconjunto de una única interfaz de llamadas al sistema, el API Win32 (véase la Figura 13). Este API es grande y dinámico. En el año 2000, el API Win32 contenía cerca de 2.000 funciones diferentes que abarcan desde la creación de un proceso hasta la consulta de una medida de prestaciones. Los diversos miembros de la familia de SO Windows implementan diversos subconjuntos del API Win32. Windows NT/2000/XP implementan todas las funciones del API, Windows 95/98/Me implementan unas tres cuartas partes de las funciones del API, y Windows CE (también conocido como Pocket PC) implementa sobre una cuarta parte de estas funciones..

FIGURA 13

La Familia de SO de Microsoft

Hay tres miembros diferentes en la familia de SO de Microsoft: Windows CE es el miembro más pequeño, y exporta la parte más reducida del API completo Win32 Windows 95/98/Me es un sistema mayor, que exporta cerca del 75% del total de funciones del API Win32. Windows NT/2000/XP es el miembro mayor de la familia e implementa el API Win3Z completo

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS



La razón de basar todos los API de SO en un solo API está relacionada con la portabilidad de aplicaciones entre los miembros de la familia. Si todos los sistemas operativos de Microsoft implementan diferentes subconjuntos anidados del mismo API, entonces un escritor de aplicaciones podría producir software de aplicación que trabaje sin cambios en cada versión de SO. Al hacer que los miembros de la familia Windows implementen subconjuntos anidados del API, la familia presenta compatibilidad hacia arriba. Por ejemplo, una aplicación escrita para Windows CE trabajaría sin modificación alguna sobre Windows Me y sobre Windows XP, y una aplicación escrita para Windows Me funcionaría en Windows XP. Adicionalmente, las mejoras a cualquiera de los productos de SO aún podrá proporcionar los mismos servicios (posiblemente de mejor calidad) mediante la misma e invariable interfaz. El coste de adoptar esta estrategia es la necesidad de establecer y mantener una definición concreta del API. Las implementaciones de las funciones del API Win32 de Win dows98/Me difieren poco de las implementaciones de las mismas funciones para Windows NT/2000/XP. Sin embargo, y dado que Windows CE está orientado a hardware como computadores de mano y aparatos de televisión, su variedad del API Win32 sí presenta diferencias significativas sobre la “tónica principal” del API.

Windows XP, 2000 y Windows NT

El desarrollo de Windows NT comenzó a finales de la década de 1980 y fue lanzado al uso público (como versión 3.1) en julio de 1993. La versión 4.0 fue lanzada en julio de 1996. La versión 5.0 se rebautizó como Windows 2000 y se lanzó en 2000. Windows XP usa la misma base de código de Windows NT, además de código de Windows 98; y fue lanzado en octubre de 2001.

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

Windows NT es el buque insignia de la familia de SO Windows. Su Ejecutivo Windows NT (Windows NT Executive), el SO, y los Subsistemas Win32 (software del sjstema complementario) implementan todas las funciones del API Win32. Es el miembro más complejo de la familia de SO

Microsoft. W95/98/ME

Windows 98 es un lanzamiento revisado de Windows 95, y Windows Me es un lanzamiento actualizado de Windows 98. Windows 95/98/Me fue el SO de trabajo hasta 2001. Desde esa fecha, muchos computadores personales comenzaron a utilizar Windows NT/2000/XP.

Windows 95/98/Me difiere de Windows NT en que implementa menos funciones del API Win32. Windows NT soporta un modelo de seguridad completo que no forma parte de Windows 95/98/Me. Muchas de las funciones extra son referencias a partes del sistema de seguridad del núcleo. El soporte de red en Windows 95/98/Me es también un subconjunto de la funcionalidad de Windows NT. La otra diferencia más importante está en la implementación de la memoria virtual. Windows NT permite que los programas de aplicación manipulen diversos parámetros que influyen en el comportamiento de la gestión de la memoria virtual, mientras que esto no es posible en Windows 95/98/Me.

Windows CE

Windows CE (Consumer Electronics, electrónica de consumo) es el miembro más pequeño de la familia. Fue desarrollado para tratar con el mercado emergente de pequeños computadores.

SISTEMAS EMBEBIDOS

Un computador embebido o empotrado (embedded) es un computador que funciona como un componente de otro sistema más complejo. Se dedica a dar soporte al sistema que lo aloja. Como ejemplos de sistemas embebidos tenemos los computadores que controlan las compuertas de un salto de agua, el control de proceso de refrigeración de un reactor nuclear, el guiado de un misil, el control de un terminal de punto de venta, o incluso la regulación del sistema de aspersión del jardín de una vivienda. Los sistemas embebidos han

sido un éxito comercial durante muchos años, pero alcanzaron nuevos niveles de popularidad con la introducción de los circuitos de gran. escala de integración en los años setenta y ochenta. Hoy, son un aspecto importante de la tecnología de los computadores.

LA PERSPECTIVA DEL USUARIO

Los sistemas embebidos suelen carecer de usuario humano. En su lugar, el “usuario” del sistema es un conjunto de sensores y actuadores. El motivo principal de la existencia de los sistemas embebidos es un razonamiento económico basado en los costes de implementación: Era más económico diseñar una parte del sistema formada por un componente electrónico, un computador y el software, que implementarlo sólo mediante un componente electrónico. Un buen ejemplo de ello son las controladoras de disco. Una controladora de disco puede implementarse por completo mediante hardware, o puede implementarse combinando cierto hardware controlado por un pequeño computador. Hoy en día, casi todas las controladoras de disco usan un sistema embebido. Un beneficio de esta aproximación es que el mismo hardware de la controladora puede servir para implementar diversas estrategias de control (tales como un SCSI o un IDE). También permite que el fabricante cambie el funcionamiento de la controladora tan sólo ¡poniendo un nuevo programa en el computador embebido dentro!

TECNOLOGÍA DE SO

Los requisitos de SO de un sistema embebido difieren dramáticamente de una aplicación tecnológica a otra puesto que el computador de control es sólo un componente más de un sistema mayor. Los desarrolladores de SO comerciales para sistemas embebidos centran su atención en la planificación del procesador (especialmente planificación de tiempo real), minimización de la cantidad de memoria y ciclos de procesador usados por el SO, y un diseño de SO que minimice el consumo de energía eléctrica en momentos de plena carga del software.

La computación de tiempo real se basa en la idea de que el “usuario” (un conjunto de sensores y actuadores hardware) requiere recibir ciertas garantías de tiempo de respuesta para ciertas tareas. Por ejemplo, un sensor detecta un incremento en la temperatura del núcleo del reactor, de modo que el sistema embebido debe enviar una señal a un actuador para permitir el enfriamiento del reactor en una pequeña, y fija, cantidad de tiempo. Esta restricción de tiempo real presenta dos retos problemáticos:

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

- Cómo garantizar que el tiempo de respuesta no exceda cierto valor máximo.
- Cómo lograr un tiempo de respuesta mínimo

La tecnología de los sistemas de tiempo real está condicionada por tales garantías de tiempos de respuesta. Muchas aplicaciones de tiempo real establecen un tiempo límite, o plazo, “flexible” (*soft*) en lugar de uno estricto. Los tiempos límite flexibles se parecen a los límites que se ponen a las tareas para casa: si el tiempo límite es estricto, no tiene sentido seguir trabajando en la asignación una vez se está fuera de plazo. Pero si el tiempo límite es flexible, se podrá obtener cierta puntuación al entregar la tarea tarde, por lo que debemos decidir si merece la pena completar la asignación aun siendo un poco tarde. En el mundo del tiempo real flexible, decimos que un SO realiza su mejor esfuerzo en llegar en plazo, pero si falla en llegar en el tiempo límite, el sistema debería seguir dando servicio en lugar de abandonar la petición de servicio a consecuencia del fallo.

A veces, un sistema embebido tiene un único propósito: ejecutar un único programa de aplicación. Como sólo suele haber una aplicación en el sistema embebido, puede no ser necesario implementar el aislamiento de recursos y la compartición entre procesos concurrentes. Por ello, el cometido principal del SO será proporcionar buenas abstracciones de recursos hardware. En este caso, el diseñador podría decidir implementar el gestor de recursos como parte de la aplicación. Esto permitirá evitar penalizaciones en las prestaciones, asociadas a las interacciones entre el programa de aplicación y el SO. Como resultado, reduce la cantidad de recursos de la máquina empleados por el SO, dejando más para la aplicación.

Los sistemas embebidos contemporáneos suelen dar cuenta de la cantidad de energía que consume el procesador en cada momento. Por ejemplo, si el sistema embebido forma parte de una unidad alimentada por baterías, a menos potencia que use el sistema embebido, más energía habrá disponible para el resto del sistema. Los sistemas embebidos actuales se construyen para funcionar apropiadamente aun cuando algunos dispositivos se hayan apagado espontáneamente durante un funcionamiento “normal.” Esto suele ocurrir cuando los discos, los visores, los sensores, o los actuadores están temporalmente apagados, aunque el sistema embebido deba seguir funcionando.

CONTRIBUCIONES A LA TECNOLOGÍA MODERNA DE SO

Las técnicas modernas para los sistemas embebidos suelen ceder en

generalidad de funcionamiento en virtud de una ganancia en eficiencia para garantizar que se cumplen las restricciones de tiempo real de los procesos. Los diseñadores de otros tipos de sistemas operativos suelen usar las técnicas empleadas en los sistemas embebidos cuando lo que interesa son las prestaciones en bruto o cierto tipo de procesamiento de tiempo real.

El núcleo de la tecnología de tiempo real también es útil para abordar las cuestiones de calidad de servicio (QoS, quality of service) de otros sistemas operativos. Por ejemplo, los procesos de aplicación pueden requerir que la información se entregue por la red en una cantidad prescrita de tiempo o con una desviación mínima en su tasa de entrega (“cortes” minimizados). Las soluciones a estos requisitos son complicadas, y son el tema de un esfuerzo considerable en los diseños de los sistemas operativos de tiempo real contemporáneos.

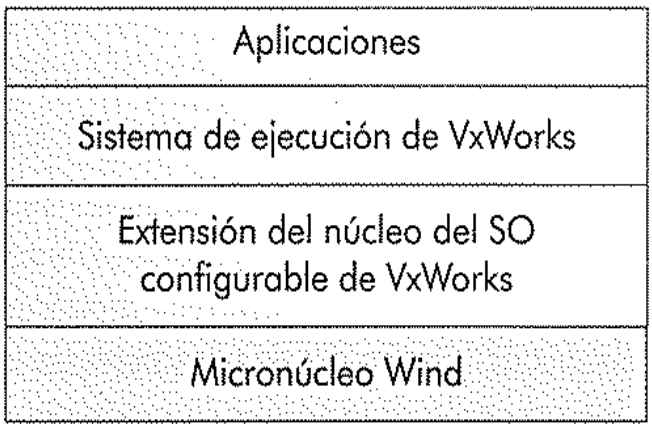
VxWorks

El entorno de ejecución y sistema operativo VxWorks de Wind Rivers goza de una merecida reputación entre los SO embebidos (véase http://www.windriver.com/) El componente de SO del producto es el *micronúcleo wind* con un núcleo de SO configurable (véase la Figura 14). El diseñador del sistema embebido decide qué partes del SO precisa para su sistema concreto, y configura el núcleo del SO para incluir sólo aqúellas funciones del SO que precisa.

FIGURA 14

Organización de VxWorks

VxWorks es un SO modular, basado en el micronúcleo wind. La Extensión Configurable del núcleo del SO proporciona software que utiliza el micronúcleo para implementar un SO operativo. La plataforma de ejecución y las aplicaciones funcionan sobre las extensiones del SO.



El micronúcleo maneja la multiprogramación, las interrupciones y la

planificación. Esta parte del SO permite implementar el soporte de tiempo real que precisa la capa de aplicación. El núcleo del SO aumenta la funcionalidad del micronúcleo proporcionando mecanismos opcionales para el paso de mensajes, la memoria compartida, el soporte de red, el soporte de gráficos, el soporte de Java, la sin cronización y demás. Esta extremada flexibilidad hace que el SO pueda ser configurado de modo que use muy poca memoria (Wind Rivers dice que el SO puede configurarse para funcionar en “unos pocos kilobytes de memoria”) aunque un sistema mínimo tal proporcionaría poca funcionalidad a las aplica ciones

COMPUTADORES PEQUEÑOS CON CAPACIDADES DE COMUNICACIÓN

Los avances en tecnología de chips y la ubicuidad de Internet han servido de catalizador para una revolución en los dispositivos de computación. La demanda de electrónica de consumo ha servido de revulsivo para el rápido desarrollo comercial de una clase completamente nueva de computadores pequeños con capacidades de comunicación (SCC, *small, comunicating computers*). Muchos de estos dispositivos se emplean para tareas muy concretas (desde aparatos MP3, a teléfonos móviles con Internet, a aparatos auxiliares a la TV que digitalizan y almacenan programas de televisión). Los computadores móviles (computadores pequeños con conexión de red inalámbrica) son una subclase importante de las SCC. Incluso aunque estos dispositivos puedan ser muy diversos, muchos de los conceptos del SO subyacente son los mismos. Sin embargo, existen marcadas diferencias en la filosofia de la implementación dada la escasez de ciertos recursos (por ejemplo, memoria, ancho de banda, y/o alimentación eléctrica) en una SCC.

LA PERSPECTIVA DEL USUARIO

El SO de un SCC es responsable de los mismos requisitos de cualquier otro SO (proporcionar abstracción hardware y gestionar la compartición). Las diferencias entre un SO de una SCC y un SO de una máquina mayor son:

- Los dispositivos SCC son diferentes de las máquinas convencionales, por lo que las abstracciones hardware de un SCC deben ser diferentes de las usadas en las máquinas convencionales.

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

- La implementación del SCC de diferentes abstracciones del hardware difiere de las implemen taciones de los SO convencionales debido a la relativa escasez de recursos del so:
- Una familia de SCC similares usará un SO similar, pero la restricción de recursos puede requerir que los diferentes miembros de la familia usen políticas de máquinas abstractas diferentes, que dependen de las peculiaridades del SCC físico. Por ejemplo, VxWorks puede usarse como SO de un SCC. El SO deberá soportar políticas y una configuración flexibles.
- Puesto que los SCC son la base de nuevos entornos de cómputo como los aparatos Internet, parece claro que el SO deba adaptarse a diferentes paradigmas de computación como es el caso de la rápida migración de los navegadores web hacia el SO y la ejecución interpretada.

TECNOLOGÍA DE SO

Los requisitos de SCC animan a la evolución de la tecnología del SO. Ha habido un firme paso hacia la computación basada en hilos, en los sistemas operativos modernos. La computación basada en hilos cambia varias premisas básicas del diseño del SO: las barreras entre los hilos en ejecución son diferentes a las barreras entre los procesos. Puesto que la computación basada en hilos usa menos recursos del sistema que la computación basada en procesos, las SCC se diseñan en torno a la computación basada en hilos en lugar de la basada en procesos.

El soporte de flujos multimedia es un aspecto importante en los modernos sistemas informáticos. Su soporte es casi obligado en una SCC. Esto ocurre porque las SCC contienen pocas o limitadas capacidad de almacenamiento. Un desafio tecnológico para el SO de las SCC es ajustar la filosofia de máquina abstracta de modo que el SO pueda proporcionar un soporte adecuado para la entrega de datos de tiempo real flexible.

Los sistemas operativos convencionales se diseñan en el entendimiento de que cuando las aplicaciones realizan peticiones conjuntas de más recursos de los disponibles en el computador, el SO ejerce una política fija del mejor esfuerzo (lo mejor posible) para asignar dichos recursos. La política se determina en el momento de diseño del SO, y es independiente del entorno en que se usa el computador. En los sistemas operativos para SCC, el sistema se queda corto con frecuencia. Sin embargo, sería inaceptable que el SO usara una política de asignación de recursos siguiendo el principio del mejor esfuerzo, puesto que creará una situación en la que el SO estaría trabajando en

cosas ajenas al cometido del computador. Los sistemas operativos futuros para SCC necesitarán ser rediseñados para hacer uso del conocimiento de la aplicación en que es siendo usados para establecer políticas de asignación de recursos concretas.

Las SCC operan en situaciones donde la gestión de los recursos es más desafiante que en los sistemas convencionales. Por ejemplo, una SCC en un computador móvil debería preservar la duración de la batería, puesto que la cantidad de tiempo que se puede mantener la operación del sistema es un factor limitador en la utilidad global del sistema. La estrategia de gestión de recursos debería poder tratar con dispositivos que se auto-apagan, y quizás incluso con estrategias que decidan qué dispositivos podrán apagarse (o ser configurados para consumir menos energía).

En un entorno de computación móvil, una conexión de red puede dejar de operar en ciertos momentos durante el funcionamiento de la máquina. Alternativamente, las características de la red pueden variar con el tiempo (a veces la señal de radio de red inalámbrica es fuerte y el ancho de banda es alto, mientras que otras veces, la señal es débil y el ancho de banda correspondiente es bajo).

Las estrategias de gestión de recursos para SCC necesitan poder adaptarse dinámicamente a la dis ponibilidad de los recursos.

Finalmente, las SCC no van a usarse en un entorno aislado. La computación moderna es el resultado de la conjunción de muchos negocios tradicionales: las telecomunicaciones, la difusión de contenidos de entretenimiento, la operación y el control de tiempo real, la entrega de información por Internet, y demás. Este es el resultado de un cambio en el clima de negocios en el mundo real, y la ampliación del uso de la información para el ciudadano de a pie. Cualquier computador operará en esta confluencia de negocios y tendrá que tratar con una amplia gama de protocolos de red, paradigmas de entrega de información en servidores de red, uso de caché de contenidos, y demás. Un SO para una SCC que pueda ser usada en el mundo de comercio emergente deberá poder tratar con toda la extensa variedad de protocolos y comportamientos.

La tecnología de SO para SCC está en sus primeros pasos (pero no tan al principio como para no poder estudiarla). Existen varios sistemas operativos propietarios diseñados para ser usados con los SCC:

- VxWorks puede usarse en un SCC.
- Windows Embedded de Microsoft y Windows CE (también conocido como

Pocket PC) se diseñaron especialmente para las SCC (véase http://www.microsoft.com/windows/embedded/)

- El SO Palm Pilot se diseñó especialmente para PDA, incluyendo la Palm Pilot, el Handspring Visor, la Sony CLIE, y otras (véase http://www.nalmos.com) Hay una gran base comercial para este SO, incluyendo un importante número de aplicaciones que usan su interfaz de máquina abstracta, y diversos sistemas de desarrollo.
- La Máquina Virtual Java (JVM, Java Virtual Machine) puede ser considerada un SO de SCC. Generalmente, la JVM se implementa sobre un SO que la aloja. Sin embargo, no hay ninguna especificación en contra de que pueda implementarse como un SO nativo.

Windows CE (Pocket PC)

Windows CE (también llamado Pocket PC cuando se usa con una PDA) se desarrolló a partir de dos proyectos de desarrollo de Microsoft que no culminaron en productos: **Microsoft-at-Work** y **Pulsar**. Ambos proyectos tenían su propio grupo de trabajo de SO, aunque ambos sistemas operativos tuvieran requisitos similares, y fue decisión de la empresa que se diseñara un nuevo SO que pudiera unir los requisitos de ambos proyectos. El SO original Windows CE era un SO orientado al objeto (OO), pero según se diseñaba el sistema OO, un pequeño grupo de diseñadores de CE crearon un núcleo de SO alternativo que implementaba un subconjunto del API Win32 (el “nuevo núcleo” o simplemente “nk”. El núcleo nk finalmente se convirtió en el SO Windows CE.

Los objetivos de Windows CE difieren considerablemente de aquellos que desembocaron en Windows NT. Por ejemplo, era preciso que Windows NT soportara interfaces de SO de legado; es decir, tenía que permitir la ejecución de aplicaciones MS-DOS, Win16, e incluso aplicaciones de OS/2 sobre Windows NT. Windows CE no tenía esos requisitos de legado puesto que estaba dirigido a un dominio de aplicación distinto de todos los sistemas operativos previos de Microsoft. Sin embargo, Windows CE se diseñó para ser implementado sobre un espectro muy diferente de plataformas. Como resultado, emplea una capa de abstracción de hardware denominada la capa de abstracción OEM (OAL, *OEM abstraction layer*).

La versión 1.0 de Windows CE se implementó para funcionar en un PC de mano. La versión 2 y siguientes son sistemas modulares configurables (como VxWorks) mediante una herramienta de construcción de plataformas

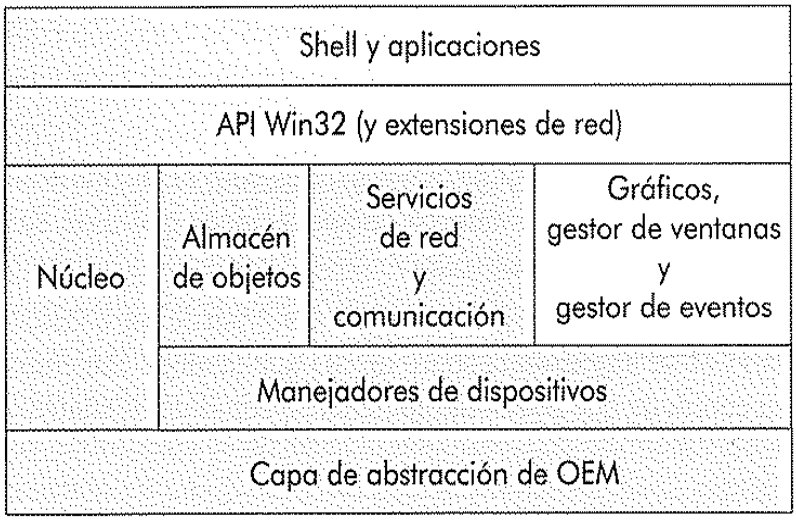
(Platform Builder). Es decir, en lugar de recompilar el SO para diferentes configuraciones de hardware, los diferentes conjuntos de componentes del SO pueden combinarse para crear diferentes versiones del SO que se enlazan para su uso en un PC de mano, un computador de automoción, videojuegos, aparatos para la televisión, y otros.

Normalmente se espera de un sistema operativo para dispositivos de consumo que proporcione el mismo nivel de gráficos y de red que el que tenemos en un sistema de sobremesa. Windows CE no soporta el conjunto completo de gráficos, gestión de ventanas, y funciones de red definidas en el API Win32 (es decir, que están implementadas en Windows NT). Esto simplifica enormemente el diseño de Windows CE.

FIGURA 15

Organización de Windows CE

Windows CE está construido sobre una capa de abstracción de OEM (*OEM Abstraction Layer*) de modo que sea fácil portar el SO completo desde una plataforma hardware a otra. El SO se compone de un núcleo, manejadores de dispositivos, un almacén de objetos, y servicios de red y de comunicaciones, Los gráficos, el gestor de ventanas y el gestor de eventos se diferencian lógicamente del resto del SO, pero forman parte del SO de Windows CE



Windows CE está diseñado para ser usado como un sistema embebido.

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

Las apli caciones de sistemas embebidos suelen requerir que el SO garantice que puede proporcionar ciertos tipos de servicio antes de determinado tiempo límite. Esto influye en el diseño de la gestión de interrupciones de Windows CE, y los diseños de manejadores de dispositivos y planificación de hilos de forma que posiblemente difieren de la aproximación tomada en Windows NT

REDES

La popularidad de los computadores personales y las redes ha desembocado en una alta demanda de sistemas que puedan realizar ciertos cómputos no triviales localmente, mientras usan la información almacenada en otro computador accesible por una red de alta velocidad. Hoy en día, el software del sistema está altamente focalizado en dar soporte al uso de computadores individuales interconectados mediante redes de área local y de área extensa. El aislamiento de recursos, la compartición y la abstracción de los recursos remotos presentan un nuevo desafio a los diseñadores de software y definen esencialmente lo último en investigación en esta área.

Hasta 1980, los computadores solían interconectarse mediante medios de comunicación punto a punto, serie, que operaban a velocidades de menos de 10 kilobits por segundo (kbps). Si un operador precisaba interconectar más de dos máquinas, entonces o bien se usaba una red de conexiones punto a punto completamente conectada o se empleaban máquinas interconectadas con una red con rutado. (En una red con rutado, existe un “camino” lógico entre cualquier par de máquinas. Cada máquina debe poder encaminar información hacia otras máquinas de modo que en conjunto implementan una red lógica que se comporta como si fuera completamente conectada.)

Las redes de área local (LAN, local area network) se convirtieron en una tecnología de comunicación económica hacia la misma época en que los computadores personales y las estaciones de trabajo comenzaban a evolucionar. En 1980, tanto Ethernet como la tecnología Token Ring (paso de testigo) proporcionaban una red completamente conectada capaz de transmitir de 10 a 16 megabits por segundo (Mbps) (tres órdenes de magnitud más rápido que una red punto a punto). Estas LAN permitieron la interconexión de máquinas pequeñas, tanto entre ellas, como con otras mayores, a una velocidad relativamente alta y a un precio razonable. El resultado fue una evolución en el modo en que se concebía el cálculo en todas las

organizaciones.

Desde 2000, la tecnología de redes inalámbrica ha emergido como una tecnología de comunicación a tener en cuenta. La tecnología inalámbrica depende de la disponibilidad de una banda de radio-frecuencias que no estaba siendo usada en aquel momento (2-5 GHz). Existen bocetos de estándares de comunicación para las dos tecnologías de LAN inalámbricas más populares sobre esta banda: el **IEEE 802.1 1b** (“WiFi”) y el **IEEE 802.15** (“Bluetooth”), Estas redes inalámbricas están pensadas para operar en un área fisica pequeña (una esfera de menos de 300 metros de radio), y para transmitir paquetes de información a 1 Mbps, Desde su introducción, han tenido un tremendo impacto en la naturaleza de los dispositivos de cómputo, concretamente sobre las SCC. Continúan apareciendo redes inalámbricas más modernas, más rápidas y más seguras. Por ejemplo, la **IEEE 802.1 1a** es una red parecida a la **IEEE 802.1 1b**, pero es varios órdenes de magnitud más rápida. La próxima genera ción de computadores y sistemas operativos se verá influida por la tecnología de redes inalámbricas.

La tecnología del software se ha visto estimulada por la presencia del hardware barato y el ancho de banda de red. Esto está conduciéndonos rápidamente a una nueva aproximación a la computación distribuida de gran escala y débilmente acoplada. Hoy en día, los servidores de disco en red, los servidores de archivos, los servidores de impresión, los servidores de bases de datos, los servidores de comunicaciones, y otros más forman parte habitual de las instalaciones habituales de LAN de 10-100Mbps. Existen redes incluso más veloces (1.000 Mbps) que se usan para conectar computadores de alta velocidad y subredes. La evolución de la computación en red ha desembocado en que los sistemas operativos evolucionen desde los sistemas de tiempo compartido y multiprogramados a sistemas con soporte de comunicación de red, estrategias de gestión de recursos distribuidos, nuevas estrategias de comunicación entre procesos, y nuevas estrategias de gestión de memoria.

En los últimos años, el último grito en aplicaciones proviene de la red: los navegadores, o visores, web para la Internet pública. La primera ola de este nuevo modelo de cómputo permitió que la gente usara los computadores personales para buscar y acceder a información de computadores ubicados por todo el mundo. Esto fue hecho posible gracias a los protocolos de red, especialmente el Protocolo Internet (IP, *Internet Protocol*), el Protocolo de Control de Transmisión (TCP, *Transmisión Control Protocol*), y el protocolo especialmente creado para la navegación web (HTTP). El dominio de los navegadores web y la disponibilidad de los contenidos en Internet está influyendo en la tecnología de los SO, aunque aún es muy pronto para predecir el efecto que tendrá a largo plazo.

U.D.	1 INTRODUCCION A LOS SISTEMAS OPERATIVOS
TEMA	1.2 ARQUITECTURA DE UN SISTEMA OPERATIVO; 1.3 FUNCIONES DE UN SISTEMA OPERATIVO; 1.4 TIPOS DE SISTEMAS OPERATIVOS

LA GÉNESIS DE LOS SISTEMAS OPERATIVOS MODERNOS

Los sistemas operativos modernos evolucionaron a partir de todos los sistemas discutidos en las secciones previas: sistemas por lotes, de tiempo compartido, de computadores personales y estaciones de trabajo, de los sistemas embebidos, de los computadores pequeños y también de las redes de computadores (véase la Figura 16). Éstos heredaron la tecnología de la multiprogramación de los sistemas por lotes y de tiempo compartido. Mientras que la protección y la seguridad comienzan su andadura en los sistemas por lotes, ambas se desarrollaron fuertemente en los entornos de tiempo compartido. La interacción hombre-máquina se ha convertido en una preocupación principal en los sistemas de tiempo compartido, y ésta continúa creciendo en importancia con las SCC. La tendencia se aceleró con la memoria dedicada y los procesadores ofrecidos con los computadores personales y las estaciones de trabajo. Los usuarios comenzaron a pedir ventanas y otras ayudas de tipo visual. El modelo de programación de red cliente-servidor (servidores de archivos, servidores de impresora, servidores de bases de datos, y demás) evolucionaron desde los sistemas que soportaban comunicaciones de red. Los sistemas embebidos han influido en la gestión de tiempo real, las aproximaciones de sincronización, la planificación y el trasiego de datos en los modernos sistemas operativos.

RESUMEN

Las compras de computadores se justifican por la funcionalidad que proporciona su software de aplicación. Si el software de aplicación no es efectivo, el sistema informático tampoco lo será. Los sistemas software y hardware son transparentes al usuario final (la persona que usa el software de aplicación para procesar información) por lo que éste no le atribuye un valor intrínseco. Más bien, están ahí para dar soporte al programador.

El software del sistema abstrae la interfaz hacia los recursos de modo que sean sencillos de usar para los programadores de aplicaciones. En concreto, el SO permite a las aplicaciones multiplexar en tiempo el procesador (y en espacio, la memoria) para crear una ilusión de ejecución paralela denominada multiprogramación. La existencia de la multiprogramación nos anima a pensar en un programa de aplicación como un proceso. También nos hace tener en cuenta las propiedades de la ejecución de procesos en paralelo, denominada ejecución concurrente. Concretamente, el SO gestiona los recursos de forma que puedan ser usados con exclusividad por un proceso o ser compartidos entre una comunidad de procesos cooperantes.

Los sistemas operativos han evolucionado desde los computadores

monousuarios a los sistemas por lotes multiprogramados, a los sistemas de tiempo compartido, a los computadores personales y estaciones de trabajo interconectados en red, a los sistemas embebidos y a las SCC. Los sistemas por lotes multiprogramados introdujeron la tecnología que permite soportar concurrencia entre trabajos. Los sistemas operativos de tiempo compartido extendieron la multiprogramación para que cada trabajo pudiera tener múltiples procesos en ejecución en representación del usuario en cada momento. Dada la proliferación de procesos en un sistema de tiempo compartido, la protección y la seguridad entre procesos y usuario se ha vuelto un tema crítico. Hoy en día, la protección y la seguridad han continuado creciendo en importancia debido a la ubicuidad de los computadores.

FIGURA 16

La Evolución de los Sistemas Operativos Modernos

Los sistemas operativos modernos han evolucionado desde los sistemas por lotes a los sistemas de tiempo compartido, los sistemas para computadores personales y estaciones de trabajo, los sistemas embebidos y las redes de computadores.