juanfrandm98
www.wuolah.com/student/juanfrandm98
2007

Tema-1.pdf

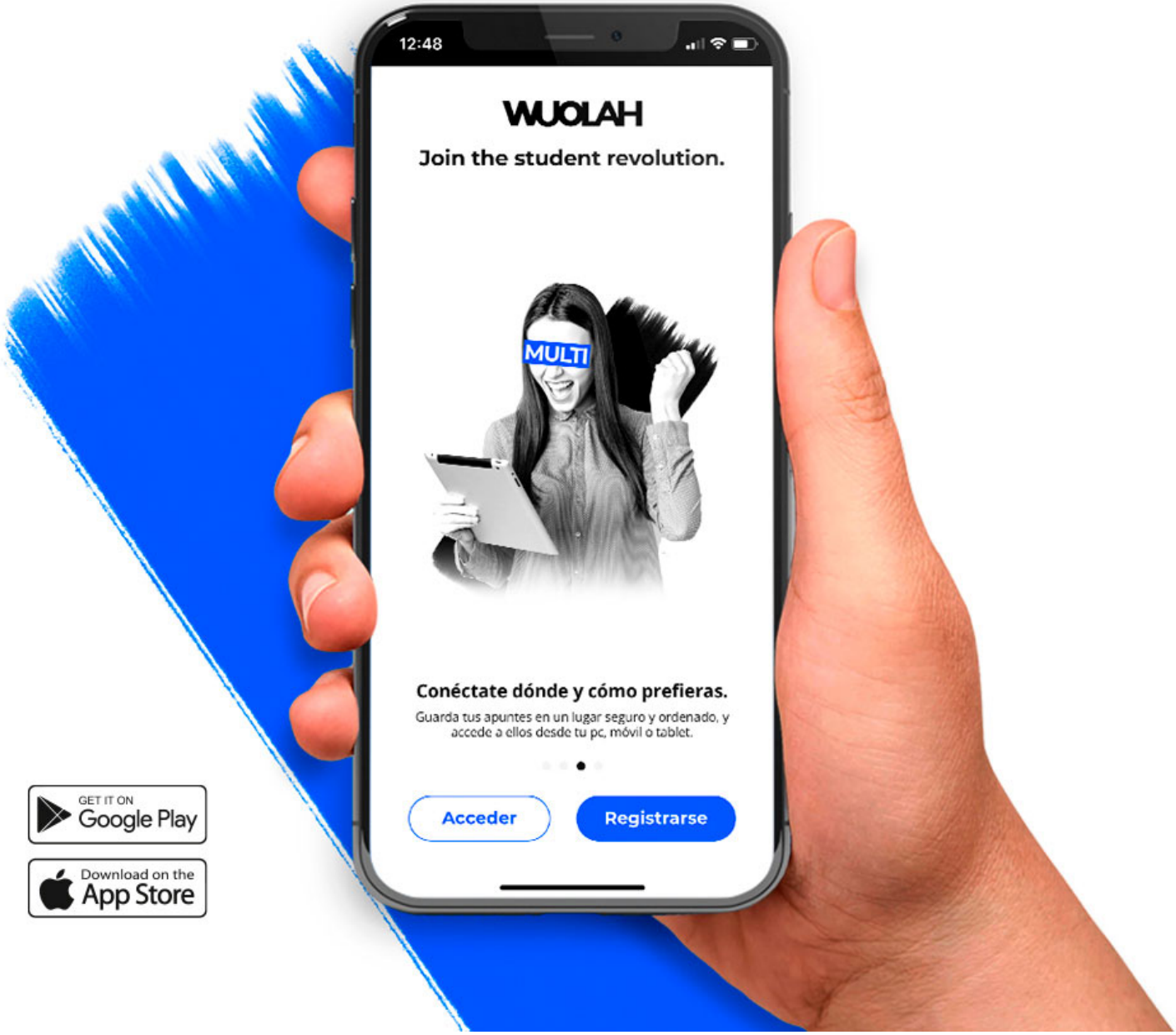
Apuntes de los Libros

2º Sistemas Operativos

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Exámenes, preguntas, apuntes.



GET IT ON Google Play

Download on the App Store

UNIVERSIDAD DE GRANADA

Sistemas Operativos

Tema 1. Estructuras de Sistemas Operativos

Juan Francisco Díaz Moreno

Curso 20/21

Estructura del sistema

A medida que se han añadido más características a los sistemas operativos y el hardware subyacente se ha vuelto más potente y versátil, han crecido el tamaño y la complejidad de los sistemas operativos.

El tamaño de los sistemas operativos junto a la complejidad de los problemas que deben resolver han generado cuatro problemas:

- Los sistemas operativos se entregan tarde de forma crónica mediante la creación de nuevos sistemas operativos y frecuentes actualizaciones.
- Los sistemas tienen fallos latentes que deben ser planteados y resueltos.
- El rendimiento no es frecuentemente el esperado.
- Es imposible construir un sistema operativo complejo que no sea vulnerable a una gran cantidad de ataques de seguridad, como virus, gusanos o accesos no autorizados.

Para solucionar la complejidad de los sistemas operativos y estos problemas, se ha puesto mucho énfasis en la estructura software del sistema operativo. El software debe ser modular para ayudar a organizar su desarrollo. Los módulos deben tener interfaces sencillas y bien definidas. Las interfaces entre módulos deben ser mínimas para poder modificar uno sin afectar al resto.

Para los sistemas operativos grandes no es suficiente la programación modular. La estructura jerárquica moderna separa las funciones en función de su escala de tiempo y nivel de abstracción. El sistema puede verse como una serie de niveles en la que cada uno realiza un subconjunto relacionado de funciones del sistema operativo. Cada nivel confía en los niveles inmediatamente inferiores para realizar funciones más primitivas sin conocer sus detalles; y proporciona servicios a la capa inmediatamente superior.

De forma general, las capas inferiores interaccionan directamente con el hardware, donde los eventos tienen una escala de tiempo ínfima; mientras que las capas superiores se comunican con el usuario en una escala de tiempo mucho más larga.

La aplicación de esta jerarquía de capas es distinta en cada sistema, pero un modelo general que no corresponde a ningún sistema operativo particular:

- Nivel 1 – Circuitos electrónicos: se tratan registros, celdas de memoria y puertas lógicas.
- Nivel 2 – Instrucciones del procesador: operaciones permitidas en el conjunto de instrucciones de lenguaje máquina, como adición, resta, carga o almacenamiento.
- Nivel 3 – Procedimientos: subrutinas y operaciones de llamada y retorno.
- Nivel 4 – Interrupciones: permiten al procesador guardar el contexto actual e invocar una subrutina de tratamiento de interrupciones.

Estos cuatro primeros niveles constituyen el hardware del procesador, aunque algunos elementos del sistema operativo empiezan a mostrarse. A partir del Nivel 5 corresponde con el sistema operativo propiamente dicho.

- Nivel 5 – Procesos primitivos: aparece la multiprogramación, lo que requiere la habilidad de suspender y continuar procesos, así como la cooperación y sincronización.

- Nivel 6 – Almacenamiento secundario local: funciones para posicionar las cabezas de lectura/escritura y la transferencia real de bloques. Delega al Nivel 5 la planificación de la operación y la notificación de finalización. Los niveles superiores calculan la dirección en el disco y la petición del bloque de datos apropiado.
- Nivel 7 – Memoria virtual: crea un espacio de direcciones lógicas para los procesos. Existen tres esquemas de uso común: páginas de tamaño fijo, segmentos de longitud variable o ambos a la vez. Cuando un bloque de memoria necesario no se encuentra en memoria principal, la lógica de este nivel requiere una transferencia desde el Nivel 6.

Hasta este punto, el sistema operativo trata con los recursos de un único procesador. A partir del Nivel 8, trata con objetos externos como periféricos o redes y computadores conectados a la red. Los objetos de estos niveles superiores son objetos lógicos con nombre, que pueden compartirse entre procesos.

- Nivel 8 – Comunicaciones entre procesos: una de las herramientas más potentes es la tubería o *pipe*, que es un canal lógico para el flujo de datos entre procesos definido por su salida en un proceso y su entrada en otro. Puede usarse para enlazar dispositivos externos o ficheros a procesos.
- Nivel 9 – Sistema de ficheros: da soporte al almacenamiento a largo plazo en ficheros con nombre. Los datos en el almacenamiento secundario se ven en términos de entidades abstractas y con longitud variable.
- Nivel 10 – Dispositivos: acceso a dispositivos externos con interfaces estándar.
- Nivel 11 – Directorios: mantiene la asociación entre los identificadores externos e internos de los recursos y objetos del sistema. El identificador externo es un nombre que puede utilizar una aplicación o usuario. El interno es una dirección de otro identificador que puede utilizarse por parte de los niveles inferiores para localizar y controlar un objeto.
- Nivel 12 – Procesos de usuario: da soporte a toda la información necesaria para la gestión ordenada de los procesos, incluyendo el espacio de direcciones virtuales de los procesos, lista de objetos y procesos con la cual puede interactuar y las restricciones de esta interacción, y parámetros pasados al proceso en la creación.
- Nivel 13 – Intérprete de mandatos: interfaz del sistema operativo para el usuario. Se denomina ***shell*** y acepta mandatos, los interpreta y crea y controla los procesos necesarios para su ejecución.

Desarrollos que han llevado a los sistemas operativos modernos

Los sistemas operativos han ido sufriendo una evolución gradual motivados principalmente por tres causas:

- Hardware: máquinas multiprocesador, los dispositivos de conexión de alta velocidad a la red y la gran variedad y tamaño creciente de los dispositivos de almacenamiento.
- Aplicaciones: multimedia, Internet y la computación cliente/servidor.
- Seguridad: acceso a Internet, ataques sofisticados, hacking.

La velocidad de cambio en las demandas de los sistemas operativos requiere nuevas formas de organización, entre las que destacan la arquitectura microkernel, multihilo, multiprocesamiento simétrico, sistemas operativos distribuidos y diseño orientado a objetos.

Hasta hace relativamente poco, la mayoría de los sistemas operativos estaban formados por un gran **núcleo monolítico** que proporcionaba la mayoría de las funcionalidades del sistema operativo. El núcleo se implementaba como un único proceso con todos los elementos compartiendo el mismo espacio de direcciones.

La **arquitectura micronúcleo** asigna unas pocas funciones esenciales al núcleo, incluyendo los espacios de almacenamiento, comunicación entre procesos (IPC) y la planificación básica. Ciertos procesos, denominados servidores, proporcionan otros servicios al sistema operativo, que ejecutan en modo usuario y son tratados como cualquier aplicación por el micronúcleo. Esta técnica simplifica la implementación, proporciona flexibilidad y se adapta perfectamente a un entorno distribuido.

Multithreading es una técnica en la que un proceso que ejecuta una aplicación se divide en una serie de hilos o *threads* que pueden ejecutar concurrentemente.

Un ***thread*** es una unidad de trabajo. Incluye el contexto del procesador (contador del programa y puntero de pila) y su propia área de datos para una pila, lo que posibilita el salto a subrutinas. Un hijo se ejecuta secuencialmente y se puede interrumpir para que el procesador pueda dar paso a otro hilo.

Un **proceso** es una colección de uno o más hilos y sus recursos de sistema asociados (memoria, código, datos, ficheros, dispositivos).

El *multithreading* es útil para las aplicaciones que llevan a cabo un número de tareas esencialmente independientes que no necesitan ser serializadas, como un servidor de base de datos que escucha y procesa numerosas peticiones de cliente. Intercambiar la ejecución entre hilos supone menos sobrecarga del procesador que intercambiar la ejecución entre procesos pesados.

El aumento del número de procesadores supone mayor eficiencia y fiabilidad. El **multiprocesamiento simétrico** (SMP) se puede definir como un sistema de computación aislado con múltiples procesadores que comparten las mismas utilidades de memoria principal y E/S interconectadas de forma interna, y que pueden realizar las mismas funciones.

El sistema operativo de un SMP planifica procesos o hilos a través de todos los procesadores, lo que supone ventajas potenciales frente a las arquitecturas monoprocesador:

- Rendimiento. Multiproceso y paralelización del trabajo.
- Disponibilidad. El fallo de un solo procesador no para la máquina, ya que el resto pueden seguir realizando el mismo trabajo.
- Crecimiento incremental. El rendimiento aumenta con procesadores adicionales.
- Escalado. Productos con diferente precio y características en función del número de procesadores.

Multithreading y SMP son frecuentemente analizados juntos ya que se complementan de manera eficiente, aunque cada una puede funcionar sin la otra correctamente.

Un **sistema operativo distribuido** proporciona la ilusión de un solo espacio de memoria principal y un solo espacio de memoria secundario. Los clústeres se están volviendo

más populares, aunque no están tan desarrollados como los sistemas monoprocesadores o SMP.

El **diseño orientado a objetos** introduce una disciplina al proceso de añadir extensiones modulares a un pequeño núcleo. Esto permite a los programadores personalizar un sistema operativo sin eliminar la integridad del sistema, y facilita el desarrollo de herramientas distribuidas y sistemas operativos distribuidos.

Descripción global de Microsoft Windows

Multitarea monousuario

Windows es un ejemplo de sistema operativo de microcomputadores. Su desarrollo surgió de la necesidad de explotar las capacidades de los microprocesadores de 32 bits.

Una de sus características más significativas es que a pesar de estar pensados para dar soporte a un único usuario interactivo, son sistemas operativos multitarea. La necesidad de multitarea surgió del incremento de velocidad y capacidad de los microprocesadores, junto al aumento de la computación cliente/servidor.

Arquitectura

La arquitectura modular que utiliza Windows desde su versión 2000 proporciona una considerable flexibilidad. Le permite ejecutarse en una gran variedad de plataformas hardware y da soporte a aplicaciones escritas para gran cantidad de sistemas operativos.

Como virtualmente en todos los sistemas operativos, Windows separa el software orientado a aplicación del software del sistema operativo. Esta segunda parte incluye el sistema ejecutivo, el núcleo y la capa de abstracción del hardware. El software que ejecuta en modo núcleo tiene acceso a los datos del sistema y al hardware. El resto, ejecutado en modo usuario, tiene acceso limitado a los datos del sistema.

Organización del Sistema Operativo

Windows tiene una arquitectura micronúcleo modificada. Es muy modular y cada función del sistema se gestiona mediante un único componente del sistema operativo. El resto del sistema operativo y las aplicaciones acceden a dicha función usando una interfaz estándar. Sólo puede accederse a datos clave del sistema mediante funciones apropiadas. Se puede borrar, actualizar o reemplazar cualquier módulo sin reescribir el resto del sistema o su API.

A diferencia de un sistema micronúcleo puro, Windows se configura de forma que muchas de las funciones del sistema externas al micronúcleo ejecutan en modo núcleo en

favor del rendimiento. Esto evita intercambios entre procesos o hilos, cambios de modo y uso de buffers de memoria extra.

Los componentes en modo núcleo son los siguientes:

- **Sistema ejecutivo**. Contiene los servicios básicos del sistema operativo, como la gestión de memoria, la gestión de procesos e hilos, seguridad, E/S y comunicación entre procesos.
- **Núcleo**. Componentes fundamentales del sistema operativo. Gestiona la planificación de hilos, intercambio de procesos, excepciones, interrupciones y la sincronización de multiprocesadores. A diferencia del resto del sistema ejecutivo y el nivel de usuario, no se ejecuta en hilos, por lo que es la única parte del sistema que no es expulsable o paginable.
- **Capa de abstracción de hardware (HAL**: *Hardware Abstraction Layer*). Permite la comunicación entre el hardware genérico y la plataforma específica. También entrega el soporte necesario para multiprocesamiento simétrico (SMP).
- **Controladores de dispositivo**. Incluye tanto sistemas de ficheros como controladores de dispositivos hardware que traducen funciones E/S de usuario en peticiones específicas a dispositivos hardware de E/S.
- **Gestión de ventanas y sistemas gráficos**. Funciones de la interfaz gráfica (GUI) como la gestión de ventanas, los controles de la interfaz de usuario y el dibujo.

El sistema ejecutivo de Windows incluye módulos para funciones del sistema específicas y proporciona una ASPI para software en modo usuario. Estos son sus módulos:

- **Gestor de E/S.** Permite a las aplicaciones acceder a los dispositivos de E/S. Envía las peticiones al controlador del dispositivo apropiado para su posterior procesamiento. Implementa todas las API de E/S de Windows y provee seguridad y nombrado para dispositivos y sistemas de ficheros.
- **Gestor de cache.** Mejora el rendimiento de la E/S basada en ficheros haciendo que los datos de disco referenciados recientemente residan en memoria principal para un acceso rápido y retardando las escrituras en disco a través del mantenimiento de las actualizaciones en memoria durante un periodo corto de tiempo antes de enviarlas al disco.
- **Gestor de *plug and play*.** Determina qué controladores se necesitan para los dispositivos y los carga.
- **Gestor de potencia.** Coordina la gestión de potencia entre dispositivos y se puede configurar para reducir el consumo de potencia hibernando el procesador.
- **Monitor de referencia de seguridad.** Asegura la validación de acceso y las reglas de generación de auditoría. La orientación a objetos de Windows proporciona una visión consistente y uniforme, por lo que utiliza las mismas rutinas de validación y comprobaciones de auditoría para todos los objetos protegidos.
- **Gestor de memoria virtual.** Proyecta direcciones virtuales del espacio de direcciones del proceso a las páginas físicas de memoria del computador.
- **Gestor de procesos e hilos.** Crea y borra los objetos y traza el comportamiento de los objetos proceso e hilo.
- **Gestor de configuración.** Es responsable de implementar y gestionar el registro del sistema, que es el repositorio para la configuración de varios parámetros a nivel de sistema global y por usuario.

- **Utilidad de llamada a procedimiento local (LPC:** *Local Procedure Call***).** Fuerza una relación cliente/servidor entre las aplicaciones y los subsistemas ejecutivos dentro de un único sistema de una manera similar a RPC.

Procesos en modo usuario

Hay cuatro tipos básicos de procesos en modo usuario en Windows:

- **Procesos de sistema especiales.** Servicios no proporcionados como parte del sistema operativo, como el proceso de inicio y el gestor de sesiones.
- **Procesos de servicio.** Servicios de Windows como el registro de eventos.
- **Subsistemas de entorno.** Expone los servicios nativos de Windows a las aplicaciones de usuario. Cada subsistema de entorno incluye bibliotecas de enlace dinámico (*Dynamic Link Libraries*, DLL) que convierten las llamadas de la aplicación a llamadas Windows.
- **Aplicaciones de usuario.** De cinco tipos: Win32, Posix, OS/2, Windows 3.1 o MS-DOS.

Windows está estructurado para soportar aplicaciones escritas para Windows 2000 y versiones posteriores. Sistemas como Windows 98 proporcionan soporte utilizando un solo sistema ejecutivo compacto a través de subsistemas de entorno protegidos, que son las partes de Windows que interactúan con el usuario final. Cada subsistema es un proceso separado que proporciona una interfaz de usuario gráfica o de línea de mandatos que define el aspecto del sistema operativo para un usuario, proporcionando su API.

Esto permite que las aplicaciones creadas para un entorno particular podrían ejecutarse sin ningún cambio en Windows, debido a que la interfaz del sistema operativo que ven es la misma para la que se han escrito.

El subsistema más importante es Win32.

Modelo Cliente/Servidor

Windows adopta el modelo cliente/servidor para el uso interno en un solo sistema. Cada subsistema de entorno y subsistema del servicio ejecutivo se implementa como uno o más procesos. Cada proceso espera la solicitud de un cliente por uno de sus servicios. El sistema ejecutivo encamina las peticiones de los clientes (aplicaciones u otros módulos del sistema operativo) al servidor apropiado y le devuelve la respuesta en forma de mensaje.

Las ventajas de esta arquitectura son:

- Simplifica el sistema ejecutivo. Se pueden construir diversos API sin conflictos ni duplicaciones en el sistema ejecutivo y se pueden añadir interfaces.
- Mejora la fiabilidad. Cada módulo de los servicios se ejecuta como un proceso independiente con su propia partición de memoria protegida, lo que hace que pueda fallar sin corromper el resto del sistema operativo. También evita que los clientes accedan directamente al hardware o a las zonas de memoria del sistema ejecutivo.

- Proporciona a las aplicaciones maneras uniformes de comunicarse con el sistema ejecutivo a través de los LPC sin restringir la flexibilidad.
- Proporciona una base adecuada para la computación distribuida, que suele utilizar este modelo con RPC.

Hilos y SMP

Características de Windows que dan soporte a los hilos y a los SMP:

- Las rutinas del sistema operativo se pueden ejecutar en cualquier procesador disponible, y diferentes rutinas se pueden ejecutar simultáneamente en diferentes procesadores.
- Permite el uso de múltiples hilos dentro de un único proceso, que se pueden ejecutar en diferentes procesadores simultáneamente.
- Los procesos de servidor pueden utilizar múltiples hilos para procesar más de una petición simultáneamente.
- Proporciona mecanismos para compartir datos y recursos entre procesos y capacidades flexibles de comunicación entre procesos.

Objetos de Windows

Windows se apoya en la orientación a objetos, lo que facilita la compartición de recursos y datos entre los procesos y la protección de recursos frente al acceso no autorizado. Los conceptos claves son:

- **Encapsulación.** Un objeto está compuesto por uno o más atributos y uno o más procedimientos (servicios). La única forma de acceder a los datos de un objeto es invocando a sus propios servicios, pudiendo así protegerse fácilmente.
- **Clases e instancias de objetos.** Una clase de objeto es una plantilla que lista los atributos y servicios del mismo y define sus características. El sistema operativo puede crear instancias de objetos cuando lo necesite.
- **Herencia.** Esta característica no es soportada a nivel de usuario, sino por alguna extensión dentro del sistema ejecutivo. Por ejemplo, los ficheros creados en un directorio comprimido también lo estarán.
- **Polimorfismo.** Internamente, Windows utiliza un conjunto común de funciones para manipular los objetos de cualquier tipo. Sin embargo, no es completamente polimórfico porque hay muchas API específicas para tipos de objetos específicos.

No todas las entidades de Windows son objetos. Los objetos se utilizan cuando los datos se usan en modo usuario y cuando el acceso a los datos es compartido o restringido. Windows crea y gestiona todos los tipos de objetos (como ficheros, procesos, semáforos…) a través del gestor de objetos.

Cada objeto dentro del sistema ejecutivo (objeto del núcleo) existe como un bloque de memoria gestionado y accesible únicamente por el núcleo. Las aplicaciones no pueden acceder

directamente a los datos de estos objetos, sino a través de un conjunto de funciones de manipulación soportadas por el sistema ejecutivo.

Los objetos pueden tener información de seguridad asociada en la forma de un descriptor de seguridad (SD), que se puede utilizar para restringir su acceso.

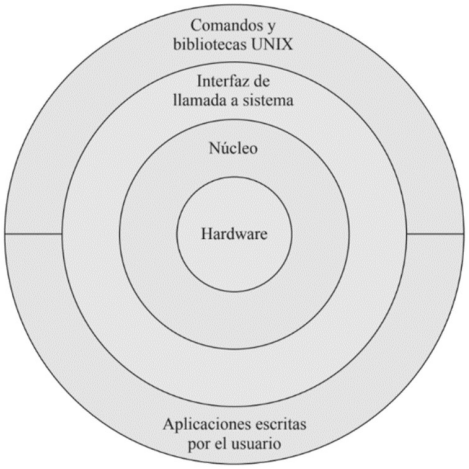
En Windows, los objetos pueden tener nombre o no. Cuando un proceso crea un objeto sin nombre, el gestor le devuelve un manejador, que es la única forma de referirse a él. Los objetos con nombre son referenciados por dicho nombre.

Existen dos categorías de objetos que gestiona el núcleo:

- **Objetos de control.** Utilizados para controlar las operaciones del núcleo en áreas que no corresponden a la planificación y sincronización:
    - **Llamada a procedimiento asíncrono:** rompe la ejecución de un hilo específico y provoca que se llame a un procedimiento en un modo de procesador especificado.
    - **Interrupción:** conecta un origen de interrupción a una rutina de servicio de interrupciones por medio de una entrada en una tabla IDT (*Interrupt Dispatch Table*). Cada procesador tiene una tabla IDT propia.
    - **Proceso:** representa el espacio de direcciones virtuales e información de control necesaria para la ejecución de un conjunto de objetos hilo. Contiene un puntero a un mapa de direcciones, una lista de hilos listos para ejecutar, una lista de hilos que pertenecen al proceso, el tiempo total acumulado para todos sus hilos y una prioridad base.
    - **Perfil:** mide la distribución de tiempo de ejecución dentro de un bloque de código, pudiendo medirse tanto el código de usuario como de sistema.
- **Objetos *dispatcher*.** Controla la activación y la sincronización de las operaciones del sistema.

Sistemas UNIX tradicionales

Descripción



El hardware subyacente es gestionado por el software del sistema operativo, que se denomina núcleo para destacar su aislamiento frente al usuario y las aplicaciones. Esto será conocido como UNIX. Viene equipado con un conjunto de servicios de usuario e interfaces que se consideran parte del sistema. Estos pueden agruparse en el *shell* y los componentes del compilador C (compilador, ensamblador y cargador). La capa externa está formada por las aplicaciones de usuario y la interfaz de usuario al compilador C.

Los programas de usuario pueden invocar los servicios del sistema operativo directamente o a través de programas de biblioteca. La interfaz de llamadas al sistema es la frontera con el usuario y permite que el software de alto nivel obtenga acceso a funciones específicas del núcleo. En el otro extremo, el sistema operativo contiene rutinas primitivas que interaccionan directamente con el hardware. Entre estas dos fronteras, el sistema se divide en dos partes principales, una encargada del control de procesos y otra de la gestión de ficheros y de la E/S.

El subsistema de control de procesos se encarga de la gestión de la memoria, la planificación y ejecución de los procesos, así como de la sincronización y la comunicación entre los procesos.

El subsistema de ficheros intercambia datos entre la memoria y los dispositivos externos tanto como flujos de caracteres como bloques. Para la transferencia orientada a bloques, se utiliza una técnica de cache de discos: entre el espacio de direccionamiento del usuario y el dispositivo externo se interpone un *buffer* de sistema en memoria principal.

Los sistemas UNIX tradicionales se diseñaron para ejecutar sobre un único procesador y carecen de capacidad para proteger sus estructuras de datos del acceso concurrente por parte de múltiples procesadores. Su núcleo no es muy versátil, soportando un único tipo de sistema de ficheros, una única política de planificación de procesos y un único formato de fichero ejecutable. No está diseñado para ser extensible y tiene pocas utilidades para la reutilización de código. El resultado es que, según se iban añadiendo nuevas características a varias versiones de UNIX, se tuvo que añadir mucho código, proporcionando un núcleo de gran tamaño no modular.

Sistemas UNIX modernos

En los sistemas UNIX modernos existe un pequeño núcleo de utilidades, escritas de forma modular, que proporciona funciones y servicios necesarios para procesos del sistema operativo.

System V Release 4 (SVR4)

Combina características de un gran número de sistemas operativos. Supuso una reescritura completa del núcleo del System V y produjo una implementación bien organizada, aunque compleja. Las nuevas características de esta versión incluyen soporte al procesamiento

en tiempo real, clases de planificación de procesos, estructuras de datos dinámicamente asignadas, gestión de la memoria virtual, sistemas de ficheros virtual y un núcleo explusivo.

SVR4 se diseñó para permitir el despliegue comercial de UNIX. Incorpora la mayoría de las características importantes de los sistemas UNIX y lo hace de forma integrada y comercialmente viable. Ejecuta en un gran rango de máquinas, desde microprocesadores de 32 bits hasta supercomputadores.

Solaris 9

Versión de UNIX basada en SVR4, cuya última versión proporciona todas las características del SVR4 más un conjunto de características avanzadas, como un núcleo multihilo completamente expulsivo con soporte completo para SMP, y una interfaz orientada a objetos para los sistemas de ficheros.

Linux

Estructura modular

La mayoría de los núcleos Linux son monolíticos. Linux acusa especialmente la dificultad de modificación del sistema, ya que su desarrollo es global y ha sido realizado por un grupo de programadores independientes asociados de forma difusa.

Linux no utiliza una técnica de micronúcleo, aunque logra muchas de las ventajas potenciales de esta técnica por su arquitectura modular particular. Linux está estructurado como una colección de **módulos cargables**, muchos de los cuales pueden cargarse y descargarse automáticamente bajo demanda.

Un módulo es un fichero cuyo código puede enlazarse y desenlazarse con el núcleo en tiempo real. Implementa algunas funciones específicas, como un sistema de ficheros o un controlador de dispositivo. No se ejecuta como un propio proceso o hilo, aunque puede crear los hilos del núcleo que necesite, sino que se ejecuta en modo núcleo en nombre del proceso actual.

Por tanto, aunque Linux se puede considerar monolítico, su estructura modular elimina algunas de las dificultades para desarrollar y evolucionar el núcleo.

Los módulos cargables de Linux tienen dos características importantes:

- **Enlace dinámico**. Un módulo de núcleo puede cargarse o descargarse y enlazarse o desenlazarse al núcleo mientras el núcleo está en memoria y ejecutándose.
- **Módulos apilables**. Los módulos se gestionan como una jerarquía, en la que actúan tanto como bibliotecas cuando son referidos desde la parte superior, como clientes cuando referencian a otros de la parte inferior.

El enlace dinámico facilita la configuración y reduce el uso de la memoria del núcleo. En Linux, un programa o un usuario puede cargar y descargar explícitamente módulos utilizando ***insmod*** y ***rmmod***. El núcleo detecta la necesidad de funciones particulares y puede cargar y descargar módulos cuando lo necesite.

Con módulos apilables se puede definir dependencia entre módulos, lo que tiene dos ventajas:

- El código común para un conjunto de módulos similares se puede mover a un único módulo, reduciendo la replicación.
- El núcleo puede asegurar que los módulos necesarios están presentes, impidiendo descargar un módulo del cual otros módulos que ejecutan dependen y cargando algunos módulos adicionalmente requeridos cuando se carga un nuevo módulo.

En Linux, cada módulo se define mediante dos tablas, la tabla de módulos y la tabla de símbolos. La tabla de módulos incluye los siguientes elementos:

- *next. Puntero al siguiente módulo, pues se organizan en una lista enlazada que comienza con un pseudomódulo.
- *name. Puntero al nombre del módulo.
- size. Tamaño del módulo en páginas de memoria.
- usecount. Contador del uso del módulo, que se incrementa cuando una operación relacionada con las funciones del módulo comienza y se decrementa cuando finaliza.
- flags. Opciones del módulo.
- nsyms. Número de símbolos exportados.
- ndeps. Número de módulos referenciados.
- *syms. Puntero a la tabla de símbolos del módulo.
- *deps. Puntero a la lista de módulos referenciados por el módulo.
- *refs. Puntero a la lista de módulos que usa este módulo.

La tabla de símbolos define aquellos símbolos controlados por este módulo que se utilizan en otros sitios.

Componentes del núcleo

El núcleo está compuesto por una colección de componentes que interaccionan, cada uno de los cuales se ejecutan en la CPU. Los principales componentes son:

- **Señales.** El núcleo utiliza las señales para llamar a un proceso, siendo estas algunas:

SIGHUP	Desconexión de un terminal	SIGPIPE	Tubería rota	SIGXCPU	Límite de CPU excedido
SIGQUIT	Finalización por teclado	SIGTERM	Terminación	SIGVTALRM	Reloj de alarma virtual
SIGTRAP	Traza	SIGCHLD	Cambio en el estado del hijo	SIGWINCH	Cambio de tamaño de una ventana
SIGBUS	Error de bus	SIGCONT	Continuar	SIGPWR	Fallo de potencia
SIGKILL	Señal para matar	SIGTSTP	Parada por teclado	SIGRTMIN	Primera señal de tiempo real
SIGSEGV	Violación de segmentación	SIGTTOU	Escritura de terminal	SIGRTMAX	Última señal de tiempo real

- **Llamadas al sistema.** Es la forma en la cual un proceso requiere un servicio de núcleo específico.

Relacionadas con el sistema de ficheros			
Close	Cierra un descriptor de fichero	Link	Construye un nuevo nombre para el fichero
Open	Abre y posiblemente crea un fichero o dispositivo	Read	Lee un descriptor de fichero
Write	Escribe a través de un descriptor de fichero		
Relacionadas con los procesos			
Execve	Ejecuta un programa	Exit	Termina el proceso que lo invoca
Getpid	Obtiene la identificación del proceso	Setuid	Establece la identidad del usuario del proceso actual
prtrace	Proporciona una forma por la cual un proceso padre puede observar y controlar la ejecución de otro proceso, examinar y cambiar su imagen de memoria y los registros		
Relacionadas con la planificación			
Sched_getparam	Establece los parámetros de planificación asociados con la política de planificación para el proceso identificado por su pid	Sched_get_priority_max	Devuelve el valor máximo de prioridad que se puede utilizar con el algoritmo de planificación identificado por la política
Sched_setscheduler	Establece tanto la política de planificación como los parámetros asociados al pid del proceso	Sched_rr_get_interval	Escribe en la estructura timespec apuntada por el parámetro de tiempo round robin para el proceso pid
Sched_yield	Un proceso puede abandonar el procesador voluntariamente sin necesidad de bloquearse a través de una llamada al sistema. El proceso entonces se moverá al final de la cola por su prioridad estática y un nuevo proceso se pondrá en ejecución		
Relacionadas con la comunicación entre procesos (IPC)			
Msgrcv	Se asigna una estructura de buffer de mensajes para recibir un mensaje. Entonces, la llamada al sistema lee un mensaje de la cola de mensajes especificada por msqid en el buffer de mensajes nuevamente creado	Semctl	Lleva a cabo la operación de control especificada por cmd en el conjunto de semáforos semid
semop	Lleva a cabo operaciones en determinados miembros del conjunto de semáforos semid	Shmat	Adjunta el segmento de memoria compartido identificado por shmid al segmento de datos del proceso que lo invoca
Shmctl	Permite al usuario recibir información sobre un segmento de memoria compartido, establecer el propietario, grupo y permisos de un segmento de memoria compartido o destruir un segmento		
Relacionadas con los *sockets* (red)			
Bind	Asigna la dirección IP local y puerto para un socket. Devuelve 0 en caso de éxito y -1 en caso de error	Connect	Establece una conexión entre el socket dado y el socket asociado remoto con sockaddr
Gethostname	Devuelve el nombre de máquina local	Send	Envía los bytes que tiene el buffer apuntado por *msg sobre el socket dado
Setsockopt	Envia las opciones sobre un socket		
Misceláneos			
create_module	Intenta crear una entrada del módulo cargable y reservar la memoria de núcleo que será necesaria para contener el módulo	Fsync	Copia todas las partes en memoria de un fichero a un disco y espera hasta que el dispositivo informa que todas las partes están en almacenamiento estable
Query_module	Solicita información relacionada con los módulos cargables desde el núcleo	Time	Devuelve el tiempo en segundos desde el 1 de enero de 1970
vhangup	Simula la suspensión del terminal actual. Esta llamada sirve para que otros usuarios puedan tener un terminal "limpio" en tiempo de inicio		

- **Procesos y planificador.** Crea, gestiona y planifica procesos.
- **Memoria virtual.** Asigna y gestiona la memoria virtual para los procesos.
- **Sistemas de ficheros.** Proporciona un espacio de nombres global y jerárquico para los ficheros, directorios y otros objetos relacionados con los ficheros. Además, proporciona las funciones del sistema de ficheros.
- **Protocolos de red.** Da soporte a la interfaz *socket* para los usuarios, utilizando la pila de protocolos TCP/IP.
- **Controladores de dispositivo tipo carácter.** Gestiona los dispositivos que requiere el núcleo para enviar o recibir datos un byte cada vez, como los terminales, los módems y las impresoras.
- **Controladores de dispositivo tipo bloque.** Gestiona dispositivos que leen y escriben datos en bloques, tal como varias formas de memoria secundaria.
- **Controladores de dispositivos de red.** Gestiona las tarjetas de interfaz de red y los puertos de comunicación que permiten las conexiones a la red, tales como los puentes y los encaminadores.
- ***Traps* y fallos.** Gestiona los *traps* y fallos generados por la CPU, como los fallos de memoria.

- **Memoria física.** Gestiona el conjunto de marcos de páginas de memoria real y asigna las páginas de memoria virtual.
- **Interrupciones.** Gestiona las interrupciones de los dispositivos periféricos.

Micronúcleos

Un micronúcleo es la pequeña parte central de un sistema operativo que proporciona las bases para las extensiones modulares.

Arquitectura micronúcleo

Los primeros sistemas operativos eran monolíticos, cualquier procedimiento podía llamar a cualquier otro. A medida que los sistemas iban aumentando en tamaño, esto se hacía insostenible, por lo que se desarrollaron sistemas operativos por capas, en los que las funciones se organizan jerárquicamente y sólo hay interacción entre las capas adyacentes. La mayor parte o todas las capas se ejecutan en modo núcleo.

Sin embargo, cada capa sigue poseyendo demasiada funcionalidad y grandes cambios en una capa pueden tener numerosos efectos en capas adyacentes. Esto dificulta la implementación de versiones y la seguridad, debido al número de interacciones entre capas.

Con el micronúcleo, solo las funciones absolutamente esenciales del sistema operativo están en el núcleo. El resto de servicios se construyen sobre él y se ejecutan en modo usuario. Muchos servicios que siempre habían formado parte del sistema operativo ahora son subsistemas externos que interactúan con el núcleo y entre ellos mismos, como los manejadores de dispositivos, servidores de archivo o gestores de memoria virtual.

Los componentes del sistema operativo externos al micronúcleo se implementan como servidores de procesos que interactúan entre ellos dos a dos por paso de mensajes a través del micronúcleo, que también funciona como una protección que previene el paso de mensajes a no ser que el intercambio esté permitido. De esta manera, existe una arquitectura cliente/servidor dentro de un solo ordenador.

Beneficios de una organización micronúcleo

- **Interfaces uniformes.** Los procesos no necesitan diferenciar entre servicios a nivel de núcleo y a nivel de usuario, porque todos los servicios se proporcionan a través de paso de mensajes.
- **Extensibilidad.** Permite agregar servicios, así como la realización de múltiples servicios en la misma área funcional. Cuando se añade una nueva característica, sólo el servidor relacionado necesita modificarse o añadirse, por lo que el impacto se reduce a un subconjunto del sistema y no requiere la reconstrucción del núcleo.

- **Flexibilidad.** Las características existentes pueden eliminarse para realizar una implementación más pequeña y eficiente.
- **Portabilidad.** Todo o gran parte del código específico del procesador está en el micronúcleo, por lo que los cambios necesarios para transferir el sistema a un nuevo procesador son menores y tienden a estar unidos en grupos lógicos.
- **Fiabilidad.** Un micronúcleo pequeño se puede verificar de forma rigurosa. El que sólo utilice un pequeño número de APIs hace más sencillo producir un código de calidad para los servicios del sistema operativo fuera del núcleo.
- **Soporte de sistemas distribuidos.** Si se configura un sistema distribuido, como un *cluster*, de tal forma que todos los procesos y servicios tengan identificadores únicos, habrá una sola imagen del sistema a nivel de micronúcleo. Un proceso puede enviar un mensaje sin saber en qué máquina reside el servicio pedido.
- **Sistema operativo orientado a objetos.** Los componentes son objetos con interfaces claramente definidas que pueden ser interconectadas para la realización de software a través de bloques de construcción.

Rendimiento del micronúcleo

La principal desventaja de los micronúcleos podría ser la del rendimiento, pues lleva más tiempo construir y enviar un mensaje a través del micronúcleo, y aceptar y decodificar la respuesta, que hacer una simple llamada a un servicio. Sin embargo, hay otros factores que hacen difícil generalizar esta desventaja.

Un primer enfoque fue hacer mayores los micronúcleos volviendo a introducir servicios críticos y manejadores en el sistema operativo para reducir el número de cambios de modo usuario-núcleo y el número de cambios de espacio de direcciones de proceso. Esto sacrifica la fortaleza del diseño del micronúcleo: mínimas interfaces, flexibilidad…

Otro enfoque consiste en hacer el micronúcleo más pequeño para eliminar las pérdidas de rendimiento a la vez que mejora la flexibilidad y fiabilidad.

Diseño del micronúcleo

Independientemente de su tamaño y funcionalidades, cualquier micronúcleo debe incluir aquellas funciones que dependen directamente del hardware y aquellas necesarias para mantener a los servidores y aplicaciones operando en modo usuario.

Gestión de memoria a bajo nivel

El micronúcleo tiene que controlar el concepto hardware de espacio de direcciones para hacer posible la implementación de protección a nivel de proceso.

La paginación y la gestión de la memoria virtual se pueden realizar de forma externa al núcleo. Cuando un hilo de la aplicación hace referencia a una página que no esta en memoria

principal, hay un fallo de página y la ejecución pasa al núcleo. El núcleo entonces manda un mensaje al proceso paginador indicando la página que ha sido referenciada y éste decide cargar esa pagina y localiza un marco de página para este propósito. El paginador y el núcleo deben interactuar para realizar las operaciones lógicas del paginador en memoria física. Una vez que la página está disponible, el paginador manda un mensaje a la aplicación a través del micronúcleo.

Esta técnica permite a un proceso fuera del núcleo proyectar archivos y bases de datos en espacios de direcciones de usuario sin llamar al núcleo. Las políticas de compartición de memoria específicas de la aplicación se pueden implementar fuera del núcleo.

Operaciones que un micronúcleo necesita para dar soporte a la paginación externa y a la gestión de memoria virtual:

- **Conceder (*Grant*).** El propietario de un espacio de direcciones (un proceso) puede conceder alguna de sus páginas a otro proceso. El núcleo borra estas páginas del espacio de memoria del otorgante y se las asigna al proceso especificado.
- **Proyectar (*Map*).** Un proceso puede proyectar cualquiera de sus páginas en el espacio de direcciones de otro proceso, de forma que ambos tienen acceso a las páginas. Esto genera memoria compartida entre procesos. El núcleo mantiene la asignación de estas páginas al propietario inicial, pero proporciona una asociación que permite el acceso de otros procesos.
- **Limpiar (*Flush*).** Un proceso puede reclamar cualquier página que fue concedida o asociada a otro proceso.

Para comenzar, el núcleo asigna toda la memoria física disponible como recursos a un proceso base del sistema. A medida que se crean los nuevos procesos, se pueden conceder o asociar algunas páginas del espacio de direcciones original a los nuevos procesos. Este esquema podría dar soporte a múltiples esquemas de memoria virtual simultáneamente.

Comunicación entre procesos (*Interprocess Communication*)

La forma básica de comunicación entre dos procesos o hilos en un sistema operativo con micronúcleo son los mensajes. Un mensaje incluye una cabecera que identifica a los procesos remitente y receptor y un cuerpo que contiene los datos, un puntero a un bloque de datos, o alguna información de control del proceso.

Podemos pensar que las IPC se fundamentan en puertos asociados a cada proceso. Un puerto es una cola de mensajes destinada a un proceso particular. Un proceso puede tener varios puertos. Asociada a cada puerto existe una lista que indica qué procesos pueden comunicarse con éste. Las identidades y funcionalidades de cada puerto se mantienen en el núcleo. Un proceso puede concederse nuevas funcionalidades mandando un mensaje al núcleo con las nuevas funcionalidades del puerto.

El paso de mensajes entre dos procesos que no tengan solapado el espacio de direcciones implica realizar una copia de memoria a memoria, y de esta forma está limitado por la velocidad de la memoria y no se incrementa con la del procesador.

Gestión de E/S e interrupciones

Con una arquitectura micronúcleo es posible manejar las interrupciones hardware como mensajes e incluir los puertos de E/S en los espacios de direcciones. El micronúcleo puede reconocer las interrupciones pero no las puede manejar, sino que genera un mensaje para el proceso a nivel de usuario que está actualmente asociado con dicha interrupción.

De esta forma, cuando se habilita una interrupción, se asigna un proceso de nivel de usuario a esa interrupción y el núcleo mantiene las asociaciones. La transformación de las interrupciones en mensajes las debe realizar el micronúcleo.

Se puede ver el hardware como un conjunto de hilos que tienen identificadores de hilo único y que mandan mensajes (únicamente con el identificador de hilo) a los hilos asociados en el espacio de usuario. El hilo receptor determina si el mensaje proviene de una interrupción y determina la interrupción específica. La estructura general de este código a nivel de usuario es la siguiente:

```
Hilo del dispositivo:
    Do
        waitFor(msg, remitente);
        if(remitente == mi_interrupcion_hardware){
            leer/escribir puertos E/S;
            reanudar interrupción hardware;
        } else …
    While(true);
```

Máquinas virtuales

La idea fundamental de las máquinas virtuales es la de abstraer el hardware del computador formando varios entornos de ejecución diferentes, creando así la ilusión de que cada entorno de ejecución está operando en su propia computadora privada, siendo esta una copia virtual de la computadora subyacente.

Para dar la ilusión de que cada máquina tiene su propio disco, las máquinas virtuales proporcionan discos virtuales (minidiscos) idénticos en todo salvo en tamaño. Cada minidisco tendrá asignadas tantas pistas de los discos físicos como necesite, siempre que la suma total de los tamaños de los minidiscos sea menor que el espacio de disco físicamente disponible.

Implementación

Una de las principales dificultades de la implementación de máquinas virtuales es que al igual que la máquina física tiene dos modos de ejecución, la máquina virtual debe tenerlos también. Esto implica que habrá un modo usuario virtual y un modo kernel virtual ejecutándose ambos en modo usuario físico.

Los cambios de modo deben funcionar igual que en los modos físicos. Cuando se hace una llamada al sistema por parte de un programa que se esté ejecutando en una maquina virtual en modo usuario virtual, se produce una transferencia al monitor de la maquina virtual

en la máquina real. Cuando el monitor de la máquina virtual obtiene el control, puede cambiar el contenido de los registros y del contador de programa para que la máquina virtual simule el efecto de la llamada al sistema. A continuación, puede reiniciar la máquina virtual, que ahora se encontrará en modo kernel virtual.

La principal diferencia es el tiempo: mientras que la E/S real puede tardar 100 ms, la E/S virtual puede llevar menos tiempo, puesto que no se pone en cola, o más tiempo, puesto que es interpretada. Además, la CPU se multiprograma entre muchas máquinas virtuales, relentizando aún más las máquinas virtuales.

Beneficios

Existe una protección completa de los diversos recursos del sistema, puesto que cada máquina virtual está completamente aislada de las demás.

Aunque no es posible la compartición directa de recursos entre máquinas virtuales, se han implementado dos métodos: la compartición de un minidisco o de una red de comunicaciones virtual.

La utilización de máquinas virtuales elimina gran parte del tiempo de desarrollo del sistema, pues llevar a cabo el desarrollo en máquinas virtuales evita la interrupción de la operación normal del sistema.

Ejemplos

VMware permite la creación de una serie de máquinas virtuales aisladas. Se ejecuta como una aplicación sobre un sistema operativo *host*, que es capaz de ejecutar varios sistemas operativos huéspedes concurrentemente. Cada máquina virtual tiene su propia CPU, memoria, unidades de disco, interfaces de red… virtuales.

JVM (*Java Virtual Machine*) es una especificación de una computadora abstracta que consta de un cargador de clases y de un intérprete de Java que ejecuta código intermedio. Tras cargar una clase, verifica que el código funcione correctamente y, si pasa la verificación, el intérprete lo ejecuta.

Gestiona automáticamente la memoria, llevando a cabo tareas de recolección de memoria en las que se reclama la memoria de objetos que no estén siendo usados.

JMV puede implementarse por software, encima de un sistema operativo *host* o como parte de un explorador web, o por hardware, en un chip diseñado para ejecutar programas Java.

Una técnica software más rápida emplea un compilador *just-in-time*, por el que la primera vez que se invoca un método Java, el código intermedio se convierte a lenguaje máquina nativo del sistema *host*. Estas operaciones se almacenan en caché para evitar la reinterpretación del código. Sin embargo, la utilización del chip Java es aún más rápida y no necesita ni el intérprete ni el compilador *just-in-time*.

Sistemas Operativos de Tiempo Real

La **computación en tiempo real** es aquella en la que la corrección del sistema depende no sólo del resultado lógico de la computación, sino también del momento en el que se producen los resultados.

En general, en un **sistema de tiempo real**, algunas de las tareas son tareas de tiempo real que tienen cierto grado de urgencia. Estas intentan controlar o reaccionar a eventos que ocurren en el mundo exterior, que ocurren en "tiempo real", por lo que deben ser capaces de mantener el ritmo de los eventos que les conciernen. De esta forma es posible asociar un plazo de tiempo límite con una tarea concreta, donde tal plazo especifica el instante de comienzo o de finalización.

Las tareas de tiempo real pueden ser clasificadas como duras o blandas. Una **tarea de tiempo real duro** es aquella que debe cumplir su plazo límite para evitar que se produzca un daño inaceptable o un error fatal. Una **tarea de tiempo real suave** tiene asociado un plazo límite deseable pero no obligatorio, por lo que sigue teniendo sentido planificar y completarla incluso con su plazo excedido.

Las tareas de tiempo real también pueden ser periódicas o aperiódicas. Una **tarea aperiódica** tiene un plazo en el cual debe finalizar o comenzar, o puede tener una restricción tanto de su instante de comienzo como de finalización. En el caso de las **tareas periódicas**, el requisito puede ser enunciado como "una vez por periodo T" o "exactamente T unidades a parte".

Características de los Sistemas Operativos de tiempo real

- Es **determinista** porque realiza operaciones en instantes de tiempo fijos predeterminados o dentro de intervalos de tiempo predeterminados. El determinismo se preocupa de cuánto tiempo tarda el sistema antes del reconocimiento de una interrupción.
  Cuando múltiples procesos compiten por recursos y tiempo de CPU, ningún sistema puede ser totalmente determinista. En un Sistema Operativo de tiempo real, las solicitudes de servicio de los procesos son dirigidas por eventos externos y temporizaciones. El grado en el que un sistema puede satisfacer de manera determinista las solicitudes depende, primero, de la velocidad a la que es capaz de responder a las interrupciones y, segundo, de si el sistema tiene capacidad suficiente para manejar todas las solicitudes dentro del tiempo requerido.
  Una medida útil de la capacidad de un sistema operativo de funcionar de manera determinista es el retardo máximo desde la llegada de una interrupción de un dispositivo de alta prioridad hasta que se comienza el servicio. En otros sistemas operativos, este retardo puede ser de decenas a cientos de milisegundos, pero en sistemas de tiempo real no puede superar el milisegundo.
- La **reactividad** se preocupa de cuánto tiempo tarda el sistema operativo en servir una interrupción previamente reconocida. Incluye los siguientes aspectos

- La cantidad de tiempo necesaria para manejar inicialmente la interrupción y comenzar a ejecutar la rutina de servicio de la interrupción (RSI). Si esta necesita un cambio de proceso será mayor que si puede ser ejecutada dentro del contexto del proceso actual.
- La cantidad de tiempo necesario para realizar la RSI, que depende de la plataforma hardware.
- El efecto del anidamiento de interrupciones (interrupciones interrumpidas por otras interrupciones).

El determinismo y la reactividad juntos conforman el tiempo de respuesta a eventos externos, lo que es crítico para los sistemas operativos de tiempo real.

- El **control de usuario** es generalmente mucho mayor en un sistema operativo de tiempo real que en sistemas ordinarios, en los que el usuario no tiene control sobre la planificación. En un sistema de tiempo real, el usuario debe ser capaz de distinguir entre tareas duras y suaves y de especificar prioridades relativas dentro de cada clase. También podría permitirle especificar características como el uso de paginación en los procesos, qué procesos deben residir siempre en memoria principal o qué algoritmos de transferencia a disco deben utilizarse.
- La **fiabilidad** es mucho mayor en un sistema operativo en tiempo real, ya que ha de responder y controlar eventos en tiempo real. La pérdida o degradación de sus prestaciones puede tener consecuencias catastróficas.
- La **operación de fallo suave** se refiere a la habilidad de un sistema de fallar de tal manera que se preserve tanta capacidad y datos como sea posible mientras continúa con la ejecución. Normalmente, el sistema notifica a usuarios o programas para que intenten acciones correctivas, y luego continúa operando con un nivel de servicio reducido. En caso de que haya que apagar la máquina, intentará mantener la consistencia de ficheros y datos.
  Un aspecto importante de estas operaciones es la **estabilidad**: en los casos en los que sea imposible cumplir los plazos de todas las tareas, el sistema cumplirá los plazos de las tareas más críticas sacrificando los de las menos críticas.

Para cumplir estos requisitos, los sistemas operativos de tiempo real implementan estas características:

- Cambio de proceso o hilo rápido.
- Pequeño tamaño (funcionalidades mínimas).
- Capacidad para responder rápidamente a interrupciones externas.
- Multitarea con herramientas para la comunicación entre procesos como semáforos, señales o eventos.
- Utilización de ficheros secuenciales especiales que pueden acumular datos a alta velocidad.
- Planificación expulsiva basada en prioridades.
- Minimización de los intervalos durante los cuales se deshabilitan las interrupciones.
- Primitivas para retardar tareas durante una cantidad dada de tiempo y para parar/retomar tareas.
- Alarmas y temporizaciones especiales.

El corazón de un sistema de tiempo real es el **planificador a corto plazo**. Lo importante es que todas las tareas de tiempo real duro se completen en su plazo y que tanteas tareas de tiempo real suave como sea posible también lo hagan.

La mayoría de los sistemas de tiempo real contemporáneos son incapaces de tratar directamente con los plazos límites, si no que se diseñan para ser tan sensibles como sea posible a las tareas de tiempo real de manera que, cuando se aproxime su plazo de tiempo, la tarea pueda ser planificada rápidamente.

Hay cuatro posibilidades:

- En un **planificador expulsivo de turno circular**, una tarea de tiempo real sería añadida a la cola de listos del siguiente turno para esperar su próxima rodaja de tiempo.
- En un **planificador no expulsivo dirigido por prioridad**, las tareas de tiempo real tendrán mayor prioridad, y cuando una esté lista será planificada tan pronto el proceso actual se bloquee o confluya.
- En un **planificador expulsivo dirigido por prioridad en puntos de expulsión**, utiliza las interrupciones basadas en el reloj para generar puntos de expulsión a intervalos regulares. En cada punto de expulsión, la tarea en ejecución será expulsada si hay esperando una tarea de mayor prioridad, pudiendo expulsar también tareas que son parte del núcleo del sistema operativo.
- En un **planificador expulsivo inmediato**, el sistema responde a una interrupción casi inmediatamente, a menos que el sistema esté en una sección bloqueada de código crítico.

Sistemas Operativos Distribuidos y de Red

En el **proceso de datos distribuido** (DDP) los procesadores, los datos y el procesamiento de datos pueden estar diseminados en una organización.

Soluciones distribuidas para la comunicación entre clientes y servidores:

- **Arquitectura de comunicaciones**: software que da soporte a un conjunto de computadores en red para aplicaciones distribuidas, como el correo electrónico o acceso a terminales remotos. Los computadores siguen siendo entidades independientes para los usuarios y las aplicaciones, que se deben comunicar entre sí por expreso deseo. Es posible tener una mezcla de computadores y sistemas operativos, siempre que todas las máquinas soporten la misma arquitectura de comunicaciones, siendo la más utilizada el conjunto de protocoles **TCP/IP**.
- **Sistema operativo de red**: existe una red de máquinas, normalmente de un solo usuario, y una o más máquinas servidoras que proporcionan servicios de red o aplicaciones, como almacenamiento de ficheros o gestión de impresión. El sistema operativo de red es un añadido a cada sistema operativo local que permite a las maquinas interactuar con los servidores. El usuario conoce la existencia de múltiples computadores y debe trabajar con ellos de forma explícita. Se suele utilizar una arquitectura de comunicaciones común para dar soporte a las aplicaciones de red.
- **Sistema operativo distribuido**: sistema operativo común compartido por una red de computadores. Para el usuario funciona como un sistema operativo centralizado, pero

le proporciona acceso transparente a los recursos de diversas máquinas. Puede depender de una arquitectura de comunicaciones para las funciones básicas de comunicación, pero normalmente se incorporan un conjunto de funciones de comunicación más sencillas para proporcionar mayor eficiencia.

La necesidad de una arquitectura de protocolos

Para enviar un fichero entre dos computadores, debe existir una ruta entre ambos, y se deben realizar las siguientes tareas:

1. El sistema emisor debe activar el enlace directo de comunicación o debe informar a la red de comunicaciones de la identidad del sistema destinatario.
2. El sistema emisor debe verificar que el sistema de destino está preparado para recibir datos.
3. La aplicación de transferencia de ficheros del sistema origen debe verificar que el programa de gestión de ficheros del sistema destino está preparado para aceptar y almacenar el fichero de ese usuario particular.
4. Si los formatos de los ficheros o las representaciones de datos en los sistemas son incompatibles, uno de los dos sistemas deberá ejecutar una función de traducción.

Un **protocolo** se utiliza para comunicar entidades de diferentes sistemas. Una **entidad** es cualquier cosa capaz de enviar y recibir información. Un **sistema** es un objeto físico que contiene una o más entidades.

Para que dos entidades se comuniquen con éxito, lo que se comunica, cómo se comunica y cuándo se comunica debe hacerse de acuerdo a unas convenciones entre las entidades involucradas, que se denominan **protocolos**, un conjunto de reglas que gobiernan el intercambio de datos entre dos entidades. Los elementos principales son:

- **Sintaxis**: formatos de datos y niveles de señales.
- **Semántica**: información de control para realizar coordinación y gestión de errores.
- **Temporización**: ajuste de velocidades y secuenciamiento.

Se puede elaborar una **arquitectura de protocolos** para dividir en módulos el proceso de cooperación entre sistemas:

El módulo de transferencia de ficheros contiene toda la lógica que es única a la aplicación de transferencia de ficheros, tal como la transmisión de claves, mandatos y registros de los ficheros. Se deben transmitir de forma fiable.

El modulo de servicios de comunicaciones se preocupa de que los requisitos de fiabilidad puedan utilizarse por múltiples aplicaciones. Se preocupa de que los sistemas de computación estén activos y listos para el intercambio de datos, y de llevar la cuenta de los datos que se están intercambiando para asegurar su entrega.

La lógica para tratar con la red se pone en el módulo de acceso a red. Si cambia la red en uso, sólo se ve afectado este módulo.

Multiprocesamiento simétrico

Arquitectura SMP

¿Cómo encaja la arquitectura SMP en los sistemas de procesamiento paralelo de Flynn?

- **SISD.** Un solo procesador ejecuta una única instrucción que opera sobre los datos almacenados en una sola memoria.
- **SIMD.** Una única instrucción de máquina controla la ejecución simultánea de un número de elementos de proceso, cada uno de los cuales tiene una memoria asociada.
- **MISD.** Se transmite una secuencia de datos a un conjunto de procesadores, cada uno de los cuales ejecuta una secuencia de instrucciones diferente.
- **MIMD.** Un conjunto de procesadores ejecuta simultáneamente diferentes secuencias de instrucciones en diferentes conjuntos de datos.

La arquitectura MIMD encaja con los ***cluster***, mientras que MISD encaja con los multiprocesadores de memoria compartida. Estos últimos se dividen en dos enfoques: maestro/esclavo y simétrico.

En la arquitectura **maestro/esclavo**, el núcleo del SO se ejecuta siempre en un determinado procesador, el resto de los procesadores ejecutan programas de usuario y utilidades del SO. El maestro es responsable de la planificación de procesos e hilos. Un proceso esclavo que necesite servicios debe enviar una petición al maestro y esperar a que realice el servicio. No hay conflictos porque un procesador tiene el control de toda la memoria y los recursos de E/S, pero tiene dos desventajas:

- Un fallo en el maestro echa abajo el sistema.
- El maestro puede convertirse en un cuello de botella.

En un **multiprocesador simétrico** (SMP), el núcleo puede ejecutar en cualquier procesador, y cada procesador realiza su propia planificación. El núcleo puede construirse como múltiples procesos/hilos, permitiendo la ejecución en paralelo.

El enfoque SMP complica el SO, ya que se debe asegurar que distintos procesadores no seleccionen el mismo proceso ni que se pierden procesos. Se debe sincronizar el uso de recursos.

Organización SMP

En un SMP, existen múltiples procesadores, cada uno de los cuales contiene su propia unidad de control, ALU y registros. Cada uno tiene acceso a memoria principal compartida y dispositivos de E/S a través de un bus de interconexión entre los procesadores, que pueden intercambiarse señales directamente. La memoria se organiza para que pueda haber accesos simultáneos a bloques separados.

En máquinas modernas, los procesadores suelen tener al menos un nivel de memoria cache privada, provocando problemas de coherencia solucionadas con técnicas hardware.

Consideraciones de diseño de SMP

Principales claves de diseño:

- **Procesos o hilos simultáneos concurrentes.** Las rutinas del núcleo necesitan ser reentrantes para permitir que varios procesadores ejecuten el mismo código del núcleo simultáneamente. Hay que impedir interbloqueos u operaciones inválidas.
- **Planificación.** Puede ser realizada por cualquier procesador, por lo que se deben evitar conflictos. También se pueden planificar hilos de un mismo proceso en distintos procesadores.
- **Sincronización.** Tanto de E/S como de espacios de direcciones compartidas. Pueden usarse, por ejemplo, cerrojos.
- **Gestión de memoria.**
- **Fiabilidad y tolerancia a fallos.**