Resumen tema 1 SO

1.1. Estructuras monolíticas, en capas, microkernel, y máquinas virtuales.
1.2. Sistemas operativos de propósito específico.

1.1. Estructura: Sistema monolítico

Un sistema operativo de este tipo no tiene una estructura clara y bien definida. Todos sus componentes se encuentran integrados en un único programa (el SO) que ejecuta en un único espacio de direcciones.

En este tipo de sistemas todas las funciones que ofrece el sistema operativo se ejecuta en un modo supervisor.

Estos sistemas operativos han surgido, normalmente, de sistemas operativos sencillos y pequeños a los que se les ha ido añadiendo un número mayor de funcionalidades. Esto les ha hecho evolucionar y crecer hasta convertirlos en programas grandes y complejos formados por muchas funciones situadas todas ellas en un mismo nivel.
Ejemplos claros de este tipo de sistemas son MS-DOS y UNIX.

Modelo simple de estructura de un SO monolítico

- Un programa principal que atiende a las llamadas de los programas de los usuarios.
- Un conjunto de procedimientos que hacen las llamadas al sistema.
- Un conjunto de procedimientos de más bajo nivel, que ayudan a la ejecución de los procedimientos de servicio.

Problemas:

- Complicado modificar el sistema operativo para añadir nuevas funcionalidades y servicios, por tanto difícil de mantener (Implica la modificación de un gran programa, compuesto por miles de líneas de código fuente y funciones, cada una de las cuales puede invocar a otras cuando así lo requiera).
- No se sigue el principio de ocultación de la información.
- No fiables. Un error en cualquier parte puede provocar caída del sistema.

Para solucionar estos problemas es necesario dotar de cierta estructura al sistema operativo.

1.1.1 Estructura: Sistema de capas

En un sistema por **capas**, el sistema operativo se organiza como una jerarquía de capas, donde cada capa ofrece una interfaz clara y bien definida a la capa superior y solamente utiliza los servicios que le ofrece la capa inferior.

Ventaja:

- **Modularidad** y la **ocultación de la información**. Una capa no necesita conocer como se ha implementado la capa sobre la que se construye, únicamente necesita conocer la interfaz que ofrece. Esto facilita enormemente la depuración y verificación del sistema, puesto que las capas se pueden ir construyendo y depurando por separado.

Problemas:

- Requiere una **planificación** muy **cuidadosa**, ya que es muy difícil organizar correctamente las capas.

- Cada capa posee demasiada funcionalidad y **grandes cambios** en una **capa** pueden tener **numerosos efectos**, muchos difíciles de seguir, en el código de las capas adyacentes (encima o debajo). Como resultado es difícil implementar versiones a medida del sistema operativo básico con algunas funciones añadidas o eliminadas.

- Difícil c**onstruir la seguridad** porque hay muchas interacciones entre capas adyacentes.

Este enfoque lo utilizo por primera vez el sistema operativo **THE**, un sistema operativo sencillo que estaba formado por seis capas. Otro ejemplo de sistema operativo diseñado por capas es el OS/2 , descendiente de MS-DOS.

1.1.2 Estructura: Microkernel o Micronúcleo

Núcleo o kernel:

- Tiende a ser residente en memoria.
- Tiende a ejecutarse en máximo nivel de privilegio (modo supervisor).
- Tiende a ejecutarse en alto nivel de inhibición de interrupciones.

Para implementar un microkernel, se estructura el SO eliminando todos los componentes no esenciales del núcleo y se implementan como programas de usuario sobre el micronúcleo.

La arquitectura del micronúcleo reemplaza la tradicional estructura vertical y estratificada en capas por una horizontal. Los componentes del sistema operativo externos al micronúcleo se implementan como servidores de procesos; interactúan entre ellos dos a dos, normalmente por paso de mensajes a través del micronúcleo.

El microkernel sólo debe contener información básica para crear y comunicar procesos. Esto último es clave. Debe proporcionar un mecanismo de comunicación entre procesos cliente y procesos servidor. La aplicación envía el mensaje de una solicitud (send) y permanece esperando la respuesta (receive). El Núcleo verifica este mensaje y lo manda al servidor. Por último, el servidor que está en espera realiza el servicio solicitado y devuelve el resultado. El micronúcleo también realiza una función de protección; previene el paso de mensajes a no ser que el intercambio esté permitido.

- Por ejemplo, si una aplicación quiere abrir un archivo, manda un mensaje al servidor del sistema de archivos.

Cada uno de los servidores puede mandar mensajes al resto de los servidores y puede invocar funciones primitivas del micronúcleo. Es decir, es una arquitectura cliente/servidor dentro de un solo computador.

Ventajas:

- **Interfaces uniformes** en las peticiones realizadas por un proceso. 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 nuevos servicios, así como la realización de múltiples servicios en la misma área funcional. Por ejemplo, puede haber múltiples organizaciones de archivos para disquetes; cada organización se puede implementar como un proceso a nivel de usuario más que tener múltiples servicios de archivos disponibles en el núcleo. De esta forma, los usuarios pueden elegir, de una variedad de servicios, el que mejor se adapte a sus necesidades.

- **Flexibilidad**: No sólo se pueden añadir nuevas características al sistema operativo, además las características existentes se pueden eliminar para realizar una implementación más pequeña y más eficiente.

- **Portabilidad**: En la arquitectura micronúcleo, todo o gran parte del código específico del procesador está en el micronúcleo. Por tanto, 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 interfaces de programación de aplicaciones (API) hace más sencillo producir un código de calidad para los servicios del sistema operativo fuera del núcleo

- **Soporte de sistemas distribuidos:** incluyendo clusters controlados por sistemas operativos distribuidos. Cuando se envía un mensaje desde un cliente hasta un proceso servidor, el mensaje debe incluir un identificador del servicio pedido. Si se configura un sistema distribuido (por ejemplo, un cluster) de tal forma que todos los procesos y servicios tengan identificadores únicos, entonces habrá una sola imagen del sistema a nivel de micronúcleo.

- **Soporte de sistemas operativos orientados a objetos (OOOS):** Un enfoque orientado a objetos puede servir para diseñar el micronúcleo y para desarrollar extensiones modulares para el sistema operativo.

Problemas:

- Peor rendimiento.

- Carga de procesamiento adicional por funciones del sistema.

Modelo cliente-Servidor

En este tipo de modelo, el enfoque consiste en implementar la mayor parte de los servicios y funciones del sistema operativo en procesos de usuario, dejando solo una pequeña parte del sistema operativo ejecutando en modo núcleo. A esta parte se le denomina micronúcleo y a los procesos que ejecutan el resto de funciones se les denomina servidores.

Microkernel vs Monolítico

La estructura **microkernel** es más flexible que la **monolítica**.
La estructura **microkernel** tiene peor rendimiento que la **monolítica**

1.1.3 Estructura: Máquinas Virtuales

Consiste en un monitor capaz de suministrar un número de versiones del hardware, es decir, máquinas virtuales. Cada copia es una réplica exacta del hardware, incluso con modo kernel y usuario, por lo que sobre cada una de ellas se puede instalar un Sistema Operativo. Cada petición de servicio es atendida por la copia del SO sobre la que se ejecuta. Cuando ese SO quiere acceder al hardware, se comunica con la máquina virtual como si se tratase de una real. Sin embargo, realmente se comunica con el monitor de máquina virtual, el único con acceso a la máquina real. Otra forma de construir una máquina virtual es aquel en que el monitor se ejecuta sobre un Sistema Operativo en lugar de sobre el hardware.

Ventajas:

- Añade **multiprogramación**, ya que cada máquina ejecuta sus procesos.
- Añade **mayor aislamiento**, ya que si se compromete la seguridad en un SO no se compromete en el resto.
- Si se genera una máquina estándar, **no hay** que **recompilar** para pasar de una máquina a otra.

Incovenientes:

- Añade una **sobrecarga computacional**, por lo que la ejecución puede ser más lenta.
- Son difíciles de implementar (dos modos del procesador también en MV).
- En algunos diseños se añade una capa (**exokernel**) que se ejecuta en modo privilegiado y se encarga de asignar recursos a las máquinas virtuales y garantizar que ninguna utilice los recursos de otra. Dada la importancia de este asunto, algunos procesadores incluyen mecanismos hardware para facilitar la construcción de máquinas virtuales (Intel Virtualization Technology).

1.2 SO de propósito específico: SO de Tiempo Real

La **computación de tiempo real:** puede definirse como 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.

Sistema de tiempo real: En general, en un sistema de tiempo real, algunas de las tareas son tareas de tiempo real, y éstas tienen cierto grado de urgencia. Tales tareas intentan controlar o reaccionar a eventos que tienen lugar en el mundo exterior. Dado que estos eventos ocurren en «tiempo real», una tarea de tiempo real debe ser capaz de mantener el ritmo de aquellos eventos que le conciernen. Así, normalmente es posible asociar un plazo de **tiempo límite** con una tarea concreta, donde tal plazo especifica el instante de comienzo o de finalización.

Tal tarea puede ser:

- **Tarea de tiempo real duro** es aquella que debe cumplir su plazo límite; de otro modo se producirá un daño inaceptable o error fatal en el sistema.

- **Tarea de tiempo real suave** tiene asociado un plazo límite deseable pero no obligatorio; sigue teniendo sentido planificar y completar la tarea incluso cuando su plazo límite ya haya vencido.

Problema:

- Un SO de este tipo debe poder ejecutar (**planificar**) todas las actividades del sistema con el fin de satisfacer todos los requisitos críticos de las mismas, incluidos los temporales.

Arquitecturas Multicomputador

1.2.1 SO de propósito específico: SO de Red

En esta configuración hay una **red de máquinas**, normalmente estaciones de trabajo de un solo usuario, y una o más máquinas servidoras. Éstas proporcionan servicios de red o aplicaciones, tales como almacenamiento de ficheros y gestión de impresión.

Cada computador tiene su **propio sistema operativo**. El sistema operativo de red es un añadido al sistema operativo local, que permite a las máquinas interactuar con los servidores.

El usuario conoce la **existencia de múltiples computadores** y debe trabajar con ellos de **forma explícita**.

Normalmente se utiliza una arquitectura de comunicaciones común para dar soporte a estas aplicaciones de red.

1.2.2 SO de propósito específico: SO’s Distribuidos

Es un sistema operativo común compartido por una red de computadores que no tiene memoria ni reloj común.

A los usuarios les parece un sistema operativo normal centralizado, pero les proporciona **acceso transparente a los recursos de diversas máquinas.**

Un sistema operativo distribuido 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.

Permiten un aumento de la **fiabilidad** del sistema. Si una parte falla, el resto puede seguir su ejecución, al menos parcialmente.

Permiten algún paralelismo, pero el aumento de velocidad no es el objetivo.

Arquitectura multiprocesador

1.2.3 SO de propósito específico: SO’s Paralelos

Son sistemas que tienen más de un procesador compartiendo el bus de la computadora, el reloj y en ocasiones la memoria y dispositivos periféricos.

Se les conoce también como sistemas fuertemente acoplados.

Características:

- Realizan más trabajo en menos tiempo.
- Ejecutan programas atendiendo de manera concurrente varios procesos de un mismo usuario.
- Proporcionan servicios de manera proporcional al nivel de hardware.
- Son tolerantes a fallas.
- Cada procesador cuenta con su propia memoria local Se mantienen copias de cada proceso.
- Permiten compartir de manera dinámica procesos y recursos entre los diferentes procesadores.

Dos tipos de multiprocesamiento:

- **Simétrico (SMP)** - cada procesador ejecuta una copia idéntica del SO, lo que proporciona un **buen rendimiento**
- **Asimétrico** (**ASMP**) - Un procesador maestro ejecuta el SO, los procesadores esclavos ejecutan procesos de usuario, que genera **peor escalabilidad**