Ejercicios-SO-Temas-1-y-2.pdf

user_4020263

Sistemas Operativos

2º Grado en Ingeniería Informática

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

1. **¿Cuáles podrían ser las principales razones para que un sistema operativo no sea diseñado de forma modular?**

1. Rendimiento eficiente:

- La comunicación entre componentes en un sistema monolítico es **directa y rápida** porque todos los módulos están en el mismo espacio de memoria. Esto permite que las funciones internas se llamen entre sí sin necesidad de pasar mensajes o utilizar intermediarios.
- La falta de capas o interfaces intermedias reduce el **overhead** de comunicación, aumentando la velocidad del sistema.

2. Menor overhead en llamadas al sistema:

- En un sistema monolítico, las llamadas al sistema (system calls) no requieren cambios de contexto o conmutación de modo (por ejemplo, de usuario a núcleo) que sí pueden necesitar en un sistema microkernel, donde cada servicio suele ejecutarse en su propio proceso.
- Esto implica que las operaciones críticas, como las de E/S y gestión de procesos, pueden realizarse con un **menor costo en tiempo**.

3. Integración y cohesión:

- Los sistemas monolíticos son **cohesivos** en el sentido de que los módulos internos están fuertemente integrados y tienen acceso directo a las mismas estructuras de datos globales y a otros recursos del sistema. Esto facilita una **coordinación más estrecha** entre los componentes, lo cual puede ser beneficioso en operaciones que involucran múltiples subsistemas (como gestión de memoria y E/S).

4. Facilidad de desarrollo inicial:

- Crear un sistema operativo monolítico suele ser más **rápido en términos de desarrollo inicial**, ya que todos los componentes están en un solo espacio de código, lo que reduce la complejidad de diseño en comparación con sistemas en capas o microkernels que requieren una arquitectura de comunicación entre módulos.
- Esto también facilita el **uso de bibliotecas comunes** y permite que los desarrolladores compartan y reutilicen código fácilmente dentro del mismo sistema.

5. Compatibilidad y soporte de hardware:

- Al tener todos los controladores y servicios en el mismo espacio de núcleo, los sistemas monolíticos pueden manejar el hardware de manera más **directa y eficiente**. No dependen de servicios adicionales o niveles de abstracción para interactuar con los dispositivos.
- Esto ha hecho que sistemas como Linux y UNIX sean populares por su **compatibilidad con una amplia gama de hardware**.

6. Facilidad de depuración:

- Debido a su diseño unificado, las estructuras monolíticas permiten una **depuración más directa** en el entorno de desarrollo, ya que no tienen tantas interfaces de comunicación intermodular como los microkernels. Los problemas de rendimiento o errores son más fáciles de rastrear dentro de un solo espacio de direcciones.

2. ¿Cuáles son los principales problemas de una arquitectura monolítica?

La arquitectura monolítica, aunque tiene sus ventajas, presenta varios problemas que pueden limitar la flexibilidad, escalabilidad y mantenibilidad de un sistema operativo. Algunos de los principales problemas incluyen:

1. **Dificultad de Mantenimiento**: Debido a que todos los componentes están integrados en un solo bloque de código, realizar cambios o actualizaciones en una parte del sistema puede afectar otras partes, lo que dificulta el mantenimiento.
2. **Escalabilidad Limitada**: Los sistemas monolíticos pueden ser difíciles de escalar, ya que todos los componentes están interdependientes. A medida que el sistema crece, la complejidad aumenta, lo que puede llevar a un rendimiento deficiente.
3. **Riesgo de Fallos**: Un fallo en un componente puede causar que todo el sistema se caiga. Esto significa que un error en una parte del código puede comprometer la estabilidad y la disponibilidad de todo el sistema operativo.
4. **Despliegue y Actualización**: Actualizar o desplegar un sistema monolítico puede ser complicado y arriesgado, ya que a menudo se debe volver a compilar y desplegar todo el sistema, en lugar de poder actualizar módulos individuales.
5. **Dificultad en la Integración de Nuevas Tecnologías**: La incorporación de nuevas tecnologías o componentes puede ser más difícil en un sistema monolítico, ya que se requiere una comprensión profunda de la arquitectura existente y puede ser necesario modificar gran parte del sistema para integrar nuevas funcionalidades.
6. **Menor flexibilidad**: La arquitectura monolítica tiende a ser menos flexible frente a cambios en los requisitos del usuario o el entorno. Adaptar el sistema a nuevas necesidades puede ser un proceso complicado y costoso.
7. **Desarrollo Colaborativo**: En un entorno de desarrollo colaborativo, trabajar en un sistema monolítico puede ser problemático, ya que los desarrolladores deben lidiar con una base de código grande y compleja, lo que puede generar conflictos y complicar la colaboración.
8. **Dificultad para realizar pruebas**: Probar un sistema monolítico puede ser complicado, ya que las pruebas deben tener en cuenta todas las interacciones entre componentes. Esto puede hacer que las pruebas sean más extensas y difíciles de gestionar.
9. **Dependencias Fuertes**: En una arquitectura monolítica, los componentes a menudo tienen dependencias fuertes entre sí, lo que puede dificultar la sustitución o actualización de un componente sin afectar a otros.
10. **Limitaciones en el Uso de Recursos**: Un sistema monolítico puede tener un uso ineficiente de recursos, ya que todos los componentes deben ser cargados en memoria, incluso si solo se necesita un subconjunto para tareas específicas.

3. ¿Cómo interactúan los programas de usuario y los servicios del sistema operativo con una arquitectura basada en microkernel?

1. Definición del Microkernel

Un microkernel es una arquitectura de sistema operativo que incluye solo las funcionalidades más esenciales en el núcleo, como la gestión de memoria, la gestión de procesos y la comunicación entre procesos (IPC). Otros servicios del sistema, como controladores de dispositivos, sistemas de archivos y otros servicios de red, se ejecutan como procesos de usuario en espacio de usuario.

2. Componentes Clave

- **Microkernel**: Proporciona las funciones básicas y críticas del sistema operativo. Se encarga de la comunicación entre procesos y de la gestión de recursos mínimos necesarios.
- **Servidores de Usuario**: Estos son servicios que se ejecutan en el espacio de usuario y que ofrecen funcionalidades más avanzadas, como controladores de dispositivos, sistemas de archivos y otros servicios del sistema operativo. Cada uno de estos servicios funciona como un proceso independiente.
- **Programas de Usuario**: Son las aplicaciones que los usuarios ejecutan, que pueden hacer uso de los servicios proporcionados por los servidores.

3. Interacción entre Programas de Usuario y Servicios del Sistema Operativo

1. **Comunicación a Través de IPC (Inter-Process Communication)**:
    - La interacción entre los programas de usuario y los servicios del sistema operativo se realiza principalmente a través de mecanismos de comunicación entre procesos. Estos mecanismos incluyen:
        - **Llamadas de sistema**: Los programas de usuario utilizan llamadas de sistema para solicitar servicios del sistema operativo. Estas llamadas se envían al microkernel.
        - **Mensajería**: Los procesos se comunican mediante el envío de mensajes. Cuando un programa de usuario necesita acceder a un servicio (por ejemplo, un controlador de dispositivo), envía un mensaje al servidor correspondiente.
        - **Pipes y Colas de Mensajes**: Se pueden usar para la comunicación asíncrona entre procesos.
2. **Manejo de Llamadas de Sistema**:
    - Cuando un programa de usuario necesita realizar una operación que requiere acceso al sistema (como la lectura de un archivo), realiza una llamada de sistema. Esta llamada es procesada por el microkernel, que a su vez delega la tarea al servidor correspondiente.
    - Por ejemplo, si un programa de usuario quiere leer un archivo, envía una solicitud al microkernel, que luego comunica con el servidor del sistema de archivos para que realice la operación.
3. **Aislamiento y Seguridad**:

- La arquitectura de microkernel proporciona un mejor aislamiento entre los diferentes servicios. Si un servicio falla, no afecta directamente al microkernel o a otros servicios, lo que mejora la estabilidad del sistema.
- Además, el aislamiento también mejora la seguridad, ya que los servicios se ejecutan en el espacio de usuario y tienen menos privilegios que el microkernel.

4. **Gestión de Recursos**:
   - El microkernel es responsable de la gestión básica de recursos como la memoria y la CPU. Cuando un servidor de usuario necesita un recurso, envía una solicitud al microkernel, que se encarga de asignar el recurso adecuado.
5. **Flexibilidad y Extensibilidad**:
   - Al tener servicios en el espacio de usuario, es más fácil agregar, actualizar o modificar servicios sin necesidad de realizar cambios en el microkernel. Esto permite una mayor flexibilidad y adaptabilidad a nuevas necesidades.

4. ¿Cuáles son las ventajas y desventajas de la arquitectura microkernel?

Ventajas de la Arquitectura Microkernel

1. **Modularidad**:
   - La separación de funcionalidades en diferentes módulos permite una mejor organización y gestión del código. Cada componente se puede desarrollar, probar y mantener de forma independiente.
2. **Estabilidad y Robustez**:
   - Los fallos en un servicio de usuario no afectan directamente al microkernel ni a otros servicios. Esto mejora la estabilidad del sistema, ya que los problemas en un componente no causan la caída del sistema operativo completo.
3. **Flexibilidad y Extensibilidad**:
   - Se pueden agregar, actualizar o eliminar servicios sin necesidad de modificar el microkernel. Esto facilita la adaptación a nuevos requisitos o tecnologías.
4. **Seguridad Mejorada**:
   - Al ejecutar muchos servicios en el espacio de usuario con privilegios limitados, se reduce la superficie de ataque. Un fallo en un servicio de usuario no compromete el núcleo del sistema.
5. **Portabilidad**:
   - La arquitectura de microkernel facilita la portabilidad a diferentes plataformas de hardware, ya que el núcleo puede mantenerse constante mientras que los servidores pueden ser adaptados a diferentes entornos

Desventajas de la Arquitectura Microkernel

1. **Rendimiento**:
   - La sobrecarga de comunicación entre procesos puede afectar el rendimiento. La necesidad de interacciones frecuentes entre el microkernel y los servicios puede resultar en una mayor latencia y un uso ineficiente de recursos.
2. **Complejidad en la Implementación**:

    - El diseño y la implementación de un microkernel pueden ser más complejos que un núcleo monolítico. La gestión de la comunicación entre procesos y la sincronización puede presentar desafíos adicionales.
3. **Desarrollo y Mantenimiento**:
    - Aunque la modularidad facilita el desarrollo, la gestión de múltiples procesos y la comunicación entre ellos puede complicar el mantenimiento del sistema.
4. **Menor Eficiencia en Acceso a Recursos**:
    - Las llamadas de sistema pueden requerir múltiples pasos y transferencias de mensajes, lo que puede resultar en un acceso menos eficiente a recursos en comparación con un sistema monolítico.
5. **Curva de Aprendizaje**:
    - Los desarrolladores pueden necesitar un tiempo considerable para familiarizarse con la arquitectura y las prácticas de desarrollo en un entorno de microkernel, lo que puede ser una barrera en equipos menos experimentados.
6. **Dependencia de IPC**:
    - La dependencia en los mecanismos de comunicación entre procesos puede limitar la capacidad de algunos sistemas para operar eficientemente en entornos con alta demanda de rendimiento.

5. Describa el objetivo principal de las máquinas virtuales, tanto desde el punto de vista de un diseñador de sistemas operativos como de un usuario. Ventajas y desventajas de las máquinas virtuales.

Desde el Punto de Vista del Diseñador de Sistemas Operativos:

El objetivo principal de las máquinas virtuales (VMs) es **aislar y simular entornos de hardware** para ejecutar múltiples sistemas operativos o aplicaciones de manera independiente en una sola máquina física. Esto permite probar, desarrollar y depurar sistemas operativos sin afectar el sistema anfitrión ni otros entornos virtuales.

Desde el Punto de Vista del Usuario:

Para los usuarios, el objetivo de las máquinas virtuales es **ejecutar distintos sistemas operativos o aplicaciones en un solo equipo** de forma segura y aislada. Esto es útil para correr software que requiere diferentes entornos, probar configuraciones, o trabajar en un sistema operativo distinto al de su máquina principal.

Ventajas de las Máquinas Virtuales

1. **Aislamiento**: Los sistemas operativos y aplicaciones dentro de cada VM están completamente aislados, protegiendo al sistema anfitrión y a otras VMs de fallos o problemas.
2. **Optimización de Recursos**: Permiten ejecutar varios entornos en una sola máquina física, aprovechando mejor el hardware.
3. **Flexibilidad y Portabilidad**: Las VMs pueden moverse y copiarse fácilmente entre diferentes servidores, facilitando la gestión y el despliegue.

1/6

Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

ING BANK NV se encuentra adherido al Sistema de Garantía de Depósitos Holandés con una garantía de hasta 100.000 euros por depositante. Consulta más información en ing.es

Que te den **10 € para gastar** es una fantasía. ING lo hace realidad.

Abre la **Cuenta NoCuenta** con el código **WUOLAH10**, haz tu primer pago y llévate 10 €.

Quiero el cash

Consulta condiciones **aquí**

do your thing

4. **Seguridad**: Los entornos aislados minimizan el riesgo de ataques y problemas de seguridad, ya que cada VM actúa como una entidad separada.

Desventajas de las Máquinas Virtuales

1. **Rendimiento**: La emulación y virtualización de hardware pueden causar una sobrecarga, haciendo que las VMs sean menos eficientes que un sistema operativo directamente instalado.
2. **Consumo de Recursos**: Cada VM consume recursos del sistema anfitrión, y correr muchas VMs simultáneamente puede reducir el rendimiento general.
3. **Complejidad de Configuración y Gestión**: La administración de múltiples entornos virtuales puede ser compleja y requerir conocimientos especializados.
4. **Dependencia de Software de Virtualización**: Las VMs dependen de un hipervisor o software de virtualización, lo cual puede ser una capa adicional de fallo o incompatibilidad.

6 ¿Dónde es más compleja una llamada al sistema, en un sistema monolítico o por capas?

Por lo general en un sistema monolítico las llamadas al sistema son más sencillas puesto que al estar todo integrado en un único núcleo, se podría acceder más fácilmente a los recursos solicitados. En cambio en uno por capas una llamada que se haga desde una capa muy superior puede requerir de varias llamadas a las capas inferiores ralentizando mucho la respuesta y disminuyendo así la eficiencia. Otro problema de las llamadas por capas es que cada vez que se realice una es necesario un cambio de contexto lo que podría producir un overhead y quitar tiempo útil a la cpu.

7 Los contextos abarcan todas las actividades que realiza el núcleo ¿Cuáles son los diferentes contextos que ejecuta un procesador?
Hay 5 tipos de contexto:

-Usuario: Es en este modo donde se ejecutan los programas y procesos creados por el usuario, el cual cuenta con acceso limitado a algunas partes del núcleo, dándole seguridad al núcleo y evitando problemas de accesos descontrolados a zonas donde no se debería.

-Kernel: En este tipo se tienen todos los permisos y accesos, ya que se establece directamente con el núcleo, lo que lo hace más peligroso ya que puede modificar cualquier cosa del sistema.

-Interrupción: Este modo permite al sistema operativo poder gestionar las llamadas que se hagan desde otro programa o de un proceso del sistema, y así poder atenderlos de forma inmediata. Este contexto cuenta con una muy alta prioridad.

-Proceso: Cada proceso cuenta con su propio contexto donde se guardaran los datos las instrucciones y posición actual dentro del programa que se está ejecutando.

-Hilos: Similar al anterior solo q los hilos únicamente cuentan con un pc y los registros que utilizan.

8 ¿Cómo consigue Linux evitar problemas de rendimiento que tienen los sistemas operativos microkernel, y aproximarse a la características de tener en memoria el núcleo mínimo posible y ser extensible a la vez? Enumere las ventajas adicionales que se obtienen con la propuesta de Linux al respecto.

Linux como tal no cuenta con una implementación microkernel sino que cuenta con un núcleo con las nociones básicas de gestión de recursos, gestión de procesos y comunicación entre procesos(IPC), cómo sería un sistema monolítico sino que además tiene módulos con distintas funcionalidades que serán cargadas en tiempo de ejecución, lo que quiere decir que en el momento en el que se requiera algún función de cualquiera de esos módulos únicamente será necesario cargarlo, sin necesidad de volver a compilarlo ni de reiniciar el sistema.
Gracias a esto cuenta con las ventajas de un sistema monolítico al tener las funciones más básicas a prueba de errores y de accesos erróneos pero a su vez evita los problemas relacionados con el tiempo de ejecución.

9. Describa las tareas típicas de las funciones que se realizan cuando se inicia y finaliza la utilización de un módulo.

-El primer paso consiste en que el sistema operativo le asigna memoria a ese módulo pasándolo del disco a memoria para poder enlazarlo (lo carga). Luego se hacen las configuraciones necesarias para que pueda ser utilizado correctamente, después el núcleo se encarga de gestionar las interrupciones o operaciones de E/S e inicializar los buffer o listas de control necesarias para la gestión del módulo. Por último informa en caso de que haya habido algún error en alguna inicialización.

-Una vez se quiere desacoplar ese módulo, se debe ejecutar una llamada de salida, liberar los recursos que se hayan solicitado, destruir las estructuras de datos utilizadas en la creación del módulo y por último se ejecuta una llamada indicando que ha terminado de desacoplarse para que el núcleo no intente acceder a esos recursos.

12. Ponga un ejemplo práctico de utilidad del concepto Namespaces que implementa Linux.

Los namespace sirven para aislar y segmentar recursos específicos del sistema para separarlos de ciertos aspectos del entorno.

Cuando se crea un contenedor en Docker, se necesita que este tenga un entorno aislado, donde pueda tener:

- Su propio sistema de archivos (aislado del host y de otros contenedores).
- Su propio espacio de procesos (de modo que sus procesos no sean visibles en el host).
- Su propio espacio de red (con su propia IP y configuración de red).

Esto se logra gracias a algunos namespace.

13 ¿Cuál es la principal ventaja de Namespaces en comparación con entornos virtualizados tales como KVM y VMWare?

La principal ventajas que tienen los namespace frente a los entornos virtualizados es su eficiencia en la gestión de recursos, esto producido principalmente debido a que los namespace crean los sistemas virtualizados compartiendo el mismo kernel por lo que no requiere una copia del SO. No es necesario simular por completo un sistema sino que carga solo lo más básico y necesario relacionado con la gestión de archivos y ejecución de programas y además no tiene ningún componente por medio sino que accede directamente al núcleo.

14. ¿De qué forma se pueden establecer Namespaces?

El primer uso generalmente se hace mediante llamadas al sistema, también se pueden usar herramientas de contenedores como docker o portman que mediante los namespace crean contenedores que aíslan y gestionan procesos.

15. ¿Cuál es el objetivo de los Control Groups en Linux en cuanto a virtualización?

El principal objetivo es controlar el uso de recursos que se le asignan a cada sistema virtualizado para asegurar un reparto equitativo y mejorar la estabilidad y el rendimiento de la virtualización, aislando los recursos que cada contenedor utiliza para evitar que interfieran entre sí garantizando la seguridad del sistema. También ofrecen un seguimiento detallado para que el usuario sepa los recursos que se están utilizando (monitoreo) .Además asignan también la prioridad de cada contenedor según los recursos que utilice y la demanda de ese contenedor.

16 ¿Es necesario que lo último que haga todo proceso antes de finalizar sea una llamada al sistema para finalizar? ¿Sigue siendo esto cierto en sistemas monoprogramados?
No no es estrictamente necesario ya que en los sistemas modernos cuando un proceso termina, el propio sistema operativo se encarga de liberar los recursos utilizados hasta el momento. Sin embargo en algunos casos más antiguos podrían no liberarse esos recursos produciendo que muchos recursos estén ocupados sin realmente ser utilizados.
Por lo general este tipo de sistema (monoprogramado) suele estar menos preparado para la liberación de recursos de forma autónoma, por lo que sí sería necesario una llamada al sistema existente indicando la finalización del proceso.

Cuando un proceso se bloquea, ¿deberá encargarse él directamente de cambiar el valor de su estado en el descriptor de proceso o PCB?
No, ningún proceso debería de tener que cambiar su propio estado en el PCB, sino que eso se encargará el propio SO que es el que tiene acceso directo a los recursos y al estado actual de los procesos. Si cada programa pudiera controlar sus accesos al PCB sería un descontrol y debe de mantener una estructura más organizada.

¿Qué debería hacer el planificador a corto plazo cuando es invocado pero no hay ningún proceso en la cola de ejecutables?

Por lo general el SO entrará en un tiempo de espera, donde la CPU se mantiene ociosa ya que no puede llevar a cabo ninguna acción o incluso según la implementación podría entrar en un modo de ahorro. Puede también aprovecharse esos tiempo de “descanso” para realizar actividades de mantenimiento o procesos en segundo plano que no se suelen tener en cuenta al no ser tan urgentes.

17 ¿Qué algoritmos de planificación quedan descartados para ser utilizados en sistemas de tiempo compartido?

-FCFS (primero en llegar, primero en ser ejecutado), este no sería factible ya que de primeras podría llegar un proceso muy largo que monopoliza el uso del sistema haciendo que los tiempos medios de espera sean muy altos.
-Por prioridad sin desalojo, no serían muy factibles puesto que si el primer proceso es muy largo pese a tener poca prioridad seguramente se ejecute antes que muchos que podrían llegar después con una prioridad mayor e incluso los procesos de baja prioridad podrían quedar de forma indefinida en espera si siguen llegando proceso con mayor prioridad
-SJF (los procesos más cortos primero), Esto podría producir que procesos muy importantes al ser de mayor tiempo de consumo nunca puedan llegar a ejecutarse al no parar de crear procesos más cortos.
-Algoritmos apropiativos pero que no repartan el tiempo de uso de la CPU de forma equitativa.

18 La representación gráfica del cociente (tiempo _en_ cola ejecutables + tiempo de CPU) / tiempo de CPU frente a tiempo de CPU suele mostrar valores muy altos para ráfagas muy cortas en casi todos los algoritmos de asignación de CPU ¿Por qué?

Seria debido a que al ser rafagas muy cortas estarian continuamente cambiandose, lo que supone un tiempo para guardar los contextos a esto hay que añadirle que seguramente haya muchos procesos en marcha por lo que una vez termina su corta ejecución debe esperar a pasar de nuevo todos los procesos lo que supone mucho tiempo de espera.

19 El modelo cliente-servidor es muy utilizado en sistemas distribuidos, ¿puede utilizarse este modelo en sistemas de un único computador?

Si, un ejemplo práctico que se utiliza en la actualidad es en un programa de usuario donde el usuario (cliente) proporciona la interfaz gráfica que se comunica con un componente externo (servidor).

33. ¿Cuál es el problema que se plantea en Linux cuando un proceso no realiza la llamada al sistema wait para cada uno de sus procesos hijos que han terminado su ejecución? ¿Qué efecto puede producir esto en el sistema?
pasarían a estar en modo zombie, sin embargo hay un proceso que los recoge (Init) que los acoge como hijos y pasaría a ser su padre, lo que significa que la señal sigchld la mandaron al padre Init y dejan de estar en modo zombie (se limpia todo menos el PCB) puesto que Init hace wait periódicamente eliminando esos procesos zombie.