¿Qué es un proceso?
Un proceso es una entidad abstracta que representa la ejecución de un programa, se trata de un conjunto de instrucciones que se crea cuando un usuario solicita ejecutar un programa, desde ese momento se convierte en una entidad dinámica capaz de cambiar su estado, interactuar con otros procesos, consumir recursos y seguir una trayectoria de ejecución particular.
La estructura fundamental que necesita un proceso para que el kernel de un sistema operativo pueda utilizarlo es el bloque de control de proceso (PCB).
Esta almacena el identificador del proceso, el identificador del proceso padre, su estado actual, prioridad, las direcciones de memoria a las cuales puede acceder, valores de registro de la CPU como el contador de programa, entre otros.
Después es necesario un espacio de direcciones de memoria donde el proceso pueda almacenar secciones de código, secciones de datos, stacks, etc.
El kernel también necesita de colas de planificación donde almacenar las PCB de los procesos para poder determinar que procesos pueden esperar, ejecutar, matar o swappear.

En un ambiente multiprogramado ¿Qué estructuras de datos y funcionalidades se deberían agregar al kernel para implementar una planificación de la CPU Round Robin? ¿Se debe contar con apoyo de hardware?

Para implementar una planificación Round Robin en un ambiente multiprogramado, el kernel de un sistema operativo debe poseer una cola de planificación para procesos “Listos”, donde se mantengan los procesos preparados para ejecutar en la CPU.
Además, debe implementar un algoritmo o planificador apropiativo, es decir, que pueda quitar procesos de la ejecución de la CPU sin que estos hayan terminado su tarea, asignando la CPU a otro proceso, según el orden de la cola.
El algoritmo o planificador “Round Robin” asigna a cada proceso un quantum de tiempo (un periodo/intervalo de tiempo fijo) para que utilice la CPU. Requiere apoyo de hardware mediante un temporizador o reloj del sistema, que genere interrupciones periódicas cuando se cumple el quantum. La expulsión de un proceso ocurre por dos motivos principales: al finalizar su tiempo asignado, cambia del estado “Ejecutándose” al estado “Listo” y vuelve a la cola de planificación para procesos “Listos” para que pueda competir por la CPU; o bien, cuando se genera una interrupción por entrada y salida (por ejemplo, leer un archivo), el proceso voluntariamente abandona la CPU, pasa del estado “Ejecutándose” al estado “En espera” y, una vez que se cumpla la operación, vuelve al estado “Listo”, para competir en nuevamente por la CPU en la cola de planificación.

¿Cómo funciona la memoria virtual con paginación por demanda? Tener en cuenta estructuras de datos necesarias: para que se las utiliza y quien las mantiene, fallos de página, su resolución y apoyo del hardware requerido

La memoria virtual con paginación por demanda es una técnica que permite al sistema operativo ejecutar procesos sin cargar completamente su espacio de direcciones en la memoria principal. Solo se cargan las páginas necesarias (una página es una sección del proceso) en el momento en que se requieren.
El objetivo es evitar que todo el proceso esté cargado en la memoria principal todo el tiempo, permitiendo mantener mayor cantidad de procesos ejecutándose simultáneamente o ejecutar procesos que su tamaño es mayor al que la memoria principal puede soportar. En

este esquema, cada proceso tiene un conjunto residente o working set, que representa las páginas de su espacio de direcciones que se encuentran actualmente cargadas en memoria y que utiliza con mayor frecuencia. Mantener un working set adecuado ayuda a minimizar los fallos de página.
Cada proceso dispone de una tabla de páginas, mantenida por el sistema operativo, donde se registran las correspondencias entre direcciones virtuales/lógicas y marcos físicos de memoria (un marco es una sección de la memoria principal). Además, el sistema operativo mantiene una tabla de marcos libres, y el hardware incorpora una unidad de gestión de memoria (MMU) que utiliza una TLB (caché de traducciones) para acelerar el acceso a memoria.
Cuando una página de un proceso quiere acceder a una dirección virtual, la MMU consulta en la tabla de páginas el bit de validez asociado a esa página, si el bit de validez está en 0, significa que la página no está en la memoria principal y se necesita cargar desde el almacenamiento secundario, esto es lo que activa la demanda de la página, desencadenando un fallo de página.
El sistema operativo responde deteniendo temporalmente el proceso, eligiendo un marco libre o reemplazando uno existente según el algoritmo de remplazo de páginas que se utilice, cargando la página requerida desde el almacenamiento secundario. Una vez completada la transferencia, se actualiza la tabla de páginas, se marca el bit de validez en 1 y reanuda la ejecución del proceso.

¿Para qué sirven la técnica de frecuencia de fallos de página y técnica del working set? Expliquen como funcionan y compárelas, busque similitudes entre ellas. Indique como se pueden utilizar cada una para la detección de trashing (hiperpaginación)

El trashing es un problema que ocurre cuando el sistema gasta más tiempo atendiendo fallos de página (intercambiando páginas entre la memoria secundaria y la memoria principal) que utilizando la CPU para ejecutar procesos.
Para prevenir y solucionar este problema, el sistema operativo emplea dos técnicas: working set y frecuencia de fallos de página.
La técnica del working set actúa de manera proactiva. Define, para cada proceso, el conjunto de páginas que ha utilizado recientemente dentro de una ventana de tiempo determinada. Si la suma de los tamaños de los working sets de todos los procesos activos supera la cantidad de marcos disponibles en la memoria principal, el sistema está en riesgo de trashing, ya que la memoria es insuficiente para mantener los conjuntos residentes de todos los procesos. Para resolverlo, el sistema operativo reduce el grado de multiprogramación, liberando marcos para los procesos que permanecen activos.
La técnica de frecuencia de fallos de página actúa de manera reactiva. Supervisa la tasa de fallos de página de cada proceso: si esta frecuencia es consistentemente alta y supera un límite máximo, se interpreta que el proceso no tiene suficientes marcos asignados. La solución es aumentar su número de marcos o suspenderlo temporalmente si no hay más memoria disponible.
Ambas técnicas se basan en el principio de localidad, que establece que las direcciones de memoria referenciadas recientemente tienden a ser utilizadas nuevamente en un corto periodo de tiempo.
Ambas buscan evitar el trashing al intentar garantizar que cada proceso tenga en memoria la cantidad suficiente de marcos que necesita para su ejecución actual.

Ambas son técnicas que utiliza el sistema operativo para administrar de manera dinámica la cantidad de marcos asignados a un proceso.

Dado el llamado a la SysCall de lectura de un archivo f ya abierto, read(f, buff, count), enuncie el flujo que sigue para resolverla, considerando al menos la participación de: Subsistema de archivos (filesystem), buffer caché y driver de dispositivo

El llamado a la SysCall read(f, buff, count) para leer un archivo ya abierto sigue el siguiente flujo:

1. El proceso en modo usuario ejecuta la llamada a la función read, lo que provoca un cambio de contexto al modo kernel.
2. El subsistema de archivos (filesystem) recibe la solicitud y traduce el requerimiento de alto nivel en operaciones físicas sobre el dispositivo de almacenamiento.
3. A partir del descriptor de archivo se accede a las estructuras del sistema de archivos (por ejemplo, el i-nodo) para determinar qué bloques de disco contienen los datos solicitados.
4. Antes de acceder al disco, el kernel consulta el buffer caché para verificar si los bloques requeridos ya se encuentran en memoria.
    - Si los datos están en el buffer cache, se copian directamente al buffer del proceso, evitando el acceso al disco.
    - Si los datos no están en caché, se genera una solicitud de lectura al driver del dispositivo.
5. El driver de dispositivo traduce la solicitud en comandos específicos de E/S que el hardware entiende y los envía al controlador del dispositivo.
6. Cuando la lectura del bloque termina, el controlador genera una interrupción, que es atendida por el manejador de interrupciones del kernel.
7. Los datos leídos se almacenan en el buffer caché y luego se copian al espacio de usuario en el buffer indicado (buff).
8. Finalmente, se realiza un cambio de contexto de regreso al modo usuario, y la llamada a read() retorna la cantidad de bytes leídos.

Defina todo lo necesario para realizar las transiciones vistas en la teoria (new, ready running y waiting)

Para que un proceso pueda cambiar de estado entre los vistos en teoría (new, ready, running y waiting) es necesario que el sistema operativo tenga distintas colas de planificación, estas almacenaran las estructuras de bloques de control de proceso (PCB), estas estructuras de se encargan de almacenar toda la información necesaria para la ejecución de un proceso (identificador del proceso, identificador de su proceso padre, direcciones de memoria, su contexto, etc.).

En cada estado, estas colas de planificación se catalogan en long, medium y short. Es decir, para almacenar todos los procesos recientemente creados se almacenan sus PCB en una cola de planificación long, esto para poder almacenar la mayor cantidad de procesos creados, luego,

Durante el ciclo de vida de un proceso, el mismo puede atravesar distintos estados. Describa cada uno de ellos indicando que componentes (modulos) del Sistema Operativo intervienen y sobre que estructuras de datos se trabaja para implementar dicho modelo de estados.

Un proceso puede atravesar distintos estados durante su ciclo de vida:

- New: El proceso está siendo creado. El sistema operativo reserva los recursos iniciales (espacio de memoria, identificador del proceso, etc.) y crea su PCB (Bloque de control del proceso) una estructura de datos donde almacena todo lo necesario para el proceso (estado, contexto, registros, información de memoria, etc) esta será almacenada en la long term scheduler (una cola de planificación a largo plazo) donde el sistema operativo almacena todas las PCB de los procesos recién creados. Los módulos del sistema operativo que intervienen son el gestor de procesos que crea y administra las PCB y el gestor de memoria donde se reserva espacio.
- Ready: Cuando el proceso termina de instanciarse, se lo selecciona de la long term scheduler para entrar en la CPU, allí cambia de estado, está preparado para poder competir por el uso de la CPU. La PCB del proceso es insertada en el short term scheduler (una cola de planificación a corto plazo) donde espera su turno para usar la CPU.
- Running: Cuando la CPU este libre y el planificador elige al proceso del short term scheduler, es insertado en la CPU para ejecutar su tarea. Se cambia de estado y el proceso comienza a ejecutar sus instrucciones. Este proceso saldrá de la CPU si:
    - Termina su tarea donde su estado pasa a Terminated y muere.
    - Es expulsado por algún algoritmo de planificación apropiativa como Round Robin, donde la tarea del proceso no termina de ejecutarse, pero se lo extrae y se lo manda nuevamente al short term scheduler, donde su estado pasa a Ready, para darle espacio a otro proceso y mantener la multiprogramación.
    - Sucede una interrupción donde el proceso deba esperar a que se complete una operación de entrada y salida para continuar su ejecución, en este caso se pasa al estado waiting y la PCB del proceso es insertada en la medium term scheduler (una cola de planificación a corto plazo).
- Waiting: El proceso no puede continuar porque está esperando a que se complete una operación de entrada y salida, cuando termine la operación, el proceso saldrá de la medium term scheduler y será insertado nuevamente en el short term scheduler donde se cambiará el estado a Ready nuevamente
- Terminated: El proceso completo su ejecución o fue abortado por el sistema operativo, se elimina su PCB y libera los recursos asignados.

Uno de los grandes problemas que puede tener la técnica de paginación es que el tamaño de la tabla de páginas crezca considerablemente. Indique cuáles son las 3 formas que existen de organizar la tabla de páginas con el fin de minimizar el inconveniente mencionado.

Para que la técnica de paginación funcione, cada proceso necesita una tabla de páginas, donde se asocia cada página lógica del proceso con su marco físico en memoria.

Estas tablas se almacenan en memoria principal, cuando crecen considerablemente, desperdiciamos memoria que podría utilizarse para la ejecución procesos. Las tres maneras de minimizar este inconveniente son:

Tabla de un nivel: Es lo que genera el problema de consumo de memoria, se trata de una estructura de datos que almacena una entrada por cada página lógica posible. Es simple y ofrece un único acceso a memoria, por lo que es muy rápido, el problema es que consume demasiada memoria incluso si el proceso no utiliza todas las páginas.

Tablas multinivel: Se utilizan dos o más tablas, donde la tabla del primer nivel contiene punteros a tablas de segundo nivel, estas tienen la dirección lógica de la tabla . Esta técnica utiliza menos recursos, ya que crea tablas de segundo nivel si son necesarias, en el caso de que el proceso no utilice ciertas áreas de memoria, la estructura correspondiente no es creada, además las tablas de segundo nivel pueden ser almacenadas en memoria secundaria, lo que ahorra todavía más espacio. El único problema es que el acceso a una página requiere varias consultas sucesivas (una por nivel), lo cual aumenta el tiempo de traducción.

Tabla invertida (Hashing): En lugar de tener una tabla por proceso, se utiliza una sola tabla para todo el sistema para toda la memoria física. Cada entrada representa un marco físico, e indica que página lógica y que proceso se encuentra almacenada allí. Para localizar una página, el sistema operativo utiliza una función de hash que transforma el número de la página y el identificador del proceso en un índice de la tabla, esto reduce el tamaño de las tablas, pero incrementa el tiempo de búsqueda y la complejidad de la gestión.

¿Qué es el sistema de archivos? Describa y defina sus objetivos.

El sistema de manejo de archivos (file system) es un componente esencial del sistema operativo encargado de gestionar la organización, almacenamiento, acceso y persistencia de la información en los dispositivos de almacenamiento secundario (discos duras, SSD, memorias USB, etc).

El sistema de archivos provee una abstracción lógica sobre el hardware, permitiendo al usuario y a las aplicaciones trabajar con archivos y directorios mediante operaciones de alto nivel como open, read, write o close, sin necesidad de conocer los detalles físicos de como los datos se guardan, actualizan, recuperan o eliminan.

El sistema operativo delega las operaciones de bajo nivel a los controladores (drivers) de los dispositivos de almacenamiento.

Sus principales objetivos son:

- Organizar y estructurar los datos en archivos y directorios.
- Proveer persistencia, asegurando que la información se mantenga disponible incluso después de apagar el equipo.
- Facilitar el acceso eficiente y seguro a los datos.
- Controlar los permisos y la protección de los archivos frente a accesos indebidos.
- Gestionar el espacio de almacenamiento, asignando y liberando bloques de manera óptima.

El sistema de archivos actúa como intermediario entre las aplicaciones y los dispositivos físicos, permitiendo una gestión ordenada, persistente y segura de la información.

¿Qué características debe tener el hardware para que el sistema operativo pueda brindar a los procesos protección del uso de la CPU?

Para que el sistema operativo pueda brindar protección del uso de la CPU a los procesos, el hardware debe incorporar mecanismos que permitan controlar la ejecución, limitar los privilegios de los programas y asegurar la multiprogramación.

Modos de operación: el procesador debe permitir distinguir entre modo usuario y modo kernel. Para esto se utiliza el cambio de contexto entre modos. En el modo kernel, el kernel del sistema operativo se ejecuta en modo privilegiado y tiene acceso completo a todas las estructuras internas, como a la PCB de cualquier proceso y sus espacios de direcciones.

En el modo usuario, los procesos solo pueden acceder a su propio espacio de direcciones. El sistema operativo impone este límite como medida de seguridad y protección para evitar que un proceso interfiera con los espacios de direcciones y recursos de otro proceso.

Otros detalles que aseguran la protección del uso de la CPU son la utilización de mecanismos de interrupción por reloj como el utilizado para el algoritmo de planificación Round Robin, que permite medir el tiempo de ejecución de cada proceso para mantener una multiprogramación, es decir, que varios procesos tengan la oportunidad de tiempo de ejecución en la CPU.

¿Qué son las system calls (llamadas al sistema)? ¿Cómo podrían implementarse?

Las llamadas al sistema (system calls) son una interfaz controlada entre los programas de usuario y el sistema operativo, que permite a los procesos solicitar servicios que requieren privilegios del modo kernel, como operaciones de entrada/salida, manejo de archivos y gestión de memoria.

Las llamadas al sistema son el único medio seguro mediante el cual un proceso en modo usuario puede acceder a los recursos del sistema.

Cuando un proceso invoca una llamada al sistema, se produce una interrupción que detiene la ejecución del código de usuario y transfiere el control al kernel.

El sistema operativo realiza un cambio de modo y ejecuta la rutina correspondiente a esa llamada, con los permisos necesarios para acceder al hardware o a estructuras internas.

Una vez finalizada la operación, el control devuelve al proceso en modo usuario.

Para implementar una llamada al sistema mediante un mecanismo de interrupción controlada. El flujo es el siguiente:

1. El programa de usuario invoca una función de biblioteca como fork( ).
2. Esa función ejecuta una instrucción especial de hardware como syscall.
3. Esta instrucción genera una interrupción de software, que transfiere el control al manejador del kernel configurado para atender llamadas al sistema.

4. El hardware cambia automáticamente el modo de ejecución a modo kernel, protegiendo así las áreas de memoria y registros privilegiados.
5. El kernel identifica que llamadas se pidió (por un número de system call almacenado en un registro) y ejecuta la rutina correspondiente.
6. Finalmente, se devuelve el resultado al proceso y el hardware restaura el modo usuario, reanudando la ejecución normal.

El hardware necesario será modos de ejecución, instrucciones especiales, mecanismos de manejo de interrupciones y una unidad de gestión de memoria.

Explique que es la TBL y como sería una traducción de una dirección que provoca un TLB hit y otra que provoca un TLB miss con y sin page fault. En la explicación indiqué estructuras utilizadas y cambios sobre.

La memoria caché de traducción anticipada (TBL) es una memoria caché de traducciones ubicada en la unidad de gestión de memoria (MMU).
Su función es almacenar las traducciones más recientes de direcciones virtuales a físicas, para acelerar el proceso de acceso a memoria en sistemas con paginación. En estos sistemas, traducir una dirección virtual requiere consultar la tabla de páginas almacenada en la memoria principal, lo que implica uno o más accesos a memoria principal.
La TLB evita esos accesos adicionales al mantener una copia temporal de las traducciones usadas con mayor frecuencia.
Cuando la TLB se llena, utiliza una política de reemplazo para decidir qué entrada descartar.
Las estructuras involucradas son:

- TLB: memoria caché de alta velocidad que guarda traducciones de las páginas recientemente utilizadas.
- Tabla de páginas: estructura de datos almacenada en la memoria principal que contiene, por cada página, el número de marco y el bit de validez
- MMU: hardware que realiza automáticamente la traducción y gestiona los accesos a la TLB y la tabla de páginas.

Caso TLB hit:

1. La CPU genera una dirección virtual.
2. La MMU consulta la TLB con el número de página virtual.
3. La entrada es encontrada en la TLB.
4. El número de marco físico se obtiene directamente de la TLB.
5. La MMU combina el número de marco con el desplazamiento de la dirección virtual para formar la dirección física final.

Caso TLB miss sin page fault:

1. La CPU genera una dirección virtual.
2. La MMU consulta la TBL.
3. La entrada no es encontrada en la TLB, se produce la TLB miss.
4. La MMU accede a la tabla de páginas en la memoria principal utilizando el número de página como índice.
5. Al consultar la entrada, el bit de validez está en 1, por lo que la página está cargada en memoria, no hay page fault.
6. La MMU obtiene el número de marco físico y completa la traducción

7. La TLB se actualiza con la nueva entrada.
    a. En el caso de que la TLB este llena, utiliza una política de remplazo para decidir qué entrada remplazar.
    b. En el caso de que la TLB no este llena, se carga la entrada sin problema.

Caso TLB miss con page fault:

1. La CPU genera una dirección virtual.
2. La MMU consulta la TBL.
3. La entrada no se encuentra en la TLB, se produce un TLB miss.
4. La MMU accede a la tabla de páginas en la memoria principal utilizando el número de página como índice.
5. Al consultar la entrada, el bit de validez está en 0, por lo que la página no está cargada en memoria, hay page fault.
6. La MMU genera una interrupción por page fault que detiene la ejecución del proceso.
7. El sistema pasa a modo kernel, y el sistema operativo gestiona el fallo de página:
    a. Determina donde se encuentra la página en el almacenamiento secundario.
    b. Selecciona un marco libre en memoria principal; si no hay, aplica una política de reemplazo de página.
    c. Carga la página desde el disco al marco seleccionado mediante una operación de entrada y salida.
    d. Actualiza la tabla de páginas, marcando el bit de validez en 1 e indicando el número de marco fisico asignado.
8. La MMU actualiza la TLB, la nueva entrada.
    a. En el caso de que la TLB este llena, utiliza una política de remplazo para decidir qué entrada remplazar.
    b. En el caso de que la TLB no este llena, se carga la entrada sin problema.
9. El sistema operativo cambia a modo usuario y reanuda la ejecución del proceso.
10. Se reintenta la instrucción que causo el fallo; esta vez, la MMU encuentra la traducción en la TLB y la dirección se traduce correctamente.

MULTIPLES CHOICE (por las deudas)

PROCESOS

Todos los componentes de un sistema operativo se deben ejecutar en modo supervisor/privilegiado.

Falso, solo los componentes críticos del kernel (gestión de memoria, planificación, manejo de interrupciones, drivers, etc.) deben ejecutarse en modo supervisor/privilegiado. La idea de que exista un modo kernel y un modo usuario, es poder limitar que operaciones puede realizar un programa o proceso en el sistema operativo.

Un sistema operativo solo puede brindar servicios de administración de memoria y no gestiona dispositivos de entrada/salida

Falso, el sistema operativo brinda servicios de administración de memoria y también gestiona dispositivos de entrada/salida mediante controladores (drivers)

El PCB contiene información sobre el estado del proceso, pero no su ubicación de memoria.

Falso, la PCB de un proceso en particular posee información sobre el estado del proceso, direcciones de memoria donde está el proceso cargado, el contexto del proceso, etc.

En los algoritmos de planificación no apropiativos, los procesos se ejecutan hasta que abandonan la CPU por su propia cuenta.

Verdadero, los algoritmos de planificación no apropiativos permiten a los procesos utilizar la CPU hasta abandonarla por su propia cuenta, ya sea porque termino su tarea o porque entra en espera por una interrupción de entrada y salida.

El modo de ejecución es un atributo que los diseñadores del Kernel introducen en los procesos para limitar lo que los mismos pueden hacer.

Verdadero, los diseñadores del Kernel limitan el alcance de los procesos utilizando el modo de ejecución, ya sea usuario para el alcance normal o kernel para realizar operaciones extra. El modo no es un atributo del proceso propiamente dicho, sino un estado del procesador que define el nivel de privilegio.

Todo llamado a una System Call provocará un context switch

Falso, una system call provoca un cambio de modo usuario a modo kernel, pero no necesariamente un context switch.

La PCB se crea luego de que el programa correspondiente ya se cargó en la memoria principal para su ejecución.

Falso, la PCB se crea al iniciar el proceso, antes incluso de que se cargue totalmente en memoria principal.

El short term scheduller utiliza la cola de listos para determinar a que proceso le asignara la CPU.

Verdadero, el organizador short term scheduller utiliza una cola de listos para determinar que proceso asignara.

MEMORIA

El espacio de direcciones de un proceso incluye su PCB.

Falso, la PCB no forma parte del espacio de direcciones del proceso, su espacio de direcciones solo incluye secciones de código, bibliotecas, datos, etc. Las PCB se encuentran almacenadas en una zona de memoria del kernel, inaccesible en el modo usuario.

La MMU es el dispositivo de hardware encargado de la asignación de memoria principal.

Falso, la MMU se encarga de traducir direcciones lógicas a físicas utilizando la tabla de páginas. No se encarga de asignar memoria, de eso se encarga el Kernel, no el hardware.

La fragmentación interna ocurre cuando tanto en el esquema de particiones fijas como en el de particiones dinámicas.

Falso, la fragmentación interna se da principalmente en particiones fijas, mientras que en particiones dinámicas se produce fragmentación externa.

En la técnica de segmentación, el Kernel divide al proceso en porciones de igual tamaño denominadas segmentos.

Falso, en la técnica de segmentación, un proceso se divide en partes lógicas como el código, el stack y los datos. Cada parte constituye un segmento que contiene datos de un mismo tipo. Estas porciones pueden variar de tamaño, no van a ser fijas.

El beneficio de paginación sobre segmentación es que la primera no causa fragmentación.

Falso, la paginación puede generar fragmentación interna dentro de cada marco.

En la paginación por demanda, el Kernel coloca un 1 en el bit R cuando alguna dirección de la página correspondiente a ese bit es modificada por el proceso.

Falso, el bit que indica modificación es el bit M, mientras que el bit R indicada si la página fue accedida

En la técnica de Frecuencia de Fallo de Páginas, si se detecta que la tasa de fallos de uno de los procesos en ejecución supera la cota superior, se dice que el sistema se encuentra en Hiperpaginación (Trashing).

Verdadero, si se detecta que la tasa de fallos de uno de los procesos en ejecución supera la cota superior, el proceso se encuentra en Hiperpaginación, es decir, se pasa más tiempo resolviendo fallos de página que ejecutándose.

En la memoria virtual implementada con paginación por demanda, la utilización del área de swap permite que el proceso se pueda ejecutar tanto si la página requerida se encuentra cargada en la memoria principal o si está en dicho área de swap.

Verdadero, el área de swap permite que el proceso se ejecute aunque sus páginas estén parcialmente en memoria secundaria, esto es, gracias a la paginación por demanda.

Dado un sistema con paginación donde se utilizan tablas de páginas de un nivel, y dado dos procesos P1 Y P2. Si el espacio de direcciones de P1 tiene un tamaño de 150Mb y el de P2 de 244Mb, entonces la tabla de páginas de P2 será de un tamaño mayor a la de P1.

Verdadero, cuanto mayor es el espacio de direcciones, más entradas en la tabla de páginas, ya que una entrada representa una página lógica.

En un sistema con paginación por demanda, donde además el hardware utiliza TLB. Si se produce un TLB Miss, al intentar resolver una dirección de memoria, entonces también producira un Page Fault.

Falso, si se produce un TLB miss no necesariamente desencadena un page fault, para eso se verifica que el bit V sea 1, para ver si realmente la página se encuentra en la tabla de páginas. En caso de que el valor del bit sea 0, entonces si se produce el page fault.

ARCHIVOS

En el diseño de la Entrada/Salida en un kernel se busca lograr una forma uniforme de poder utilizar los diferentes dispositivos de hardware.

Verdadero, el kernel se abstrae de las características técnicas de cada dispositivo de hardware, esto es delegado a sus drivers.

Un filesystem es un componente de hardware que permite el uso de los dispositivos para implementar archivos sobre ellos.

Falso, el filesystem es un componente del kernel que utiliza archivos para poder almacenar información sobre dispositivos de hardware. Este, además, se encarga de mantener la información ordenada y operable.

En la implementación de filesystem, la utilización de clúster de gran tamaño podría causar una mayor fragmentación interna.

Verdadero, un clúster es la unidad mínima de asignación, si los clúster son muy grandes, los archivos pequeños desperdician espacio dentro de estos, por lo que se genera fragmentación interna.

Según la implementación de filesystem de SystemV vista en clase, cuando se modifica el contenido de un archivo A1, también se modificara la entrada del directorio donde se encuentra dicho archivo A1.

Falso, modificar el contenido de un archivo no altera su entrada en el directorio, porque esta solo contiene el nombre y número de inodo.
Solo se modificaría si se cambia el nombre, ubicación o i-nodo asociado.

En la técnica de buffer cache de systemV vista en clase y asumiendo que la función de hash es mod 5. Se sabe que el primer header en la free list contiene el bloque 33 y que un proceso requiere el bloque 45 que no se encuentra en buffer cache. Entonces, al cargar el bloque 45 en buffer caché el único cambio que se producirá en el header que contenía el bloque 33 (que es el que se utilizara para colocar el 45) es el de los 2 punteros a la free list, ya que dejara de estar libre.

Falso, no solo cambian los punteros de la free list, también se actualiza su posición en la hash list al cargar el nuevo bloque.

En el diseño de E/S, el servicio de “Caching” es brindado por el driver de cada dispositivo.

Falso, el servicio de caching es gestionado por el kernel, no por los drivers.

En unix systemV, para acceder al inodo del archivo /var/log/milog.txt, el kernel debera acceder previamente solo a los i-nodos de los directorios que participan del path (“/”, “/var” y “/var/log”)

Verdadero, para resolver la ruta completa, el kernel busca secuencialmente los i-nodos de los directorios del path hasta llegar al archivo final.

En la implementación de un filesystem, la técnica de asignación indexada podrá causar fragmentación externa.

Falso, la técnica de asignación indexada se utiliza para prevenir la fragmentación externa, ya que los bloques de un archivo pueden ubicarse en cualquier parte del disco.

El driver de un dispositivo es un hardware que se ejecuta en modo kernel.

Falso, el driver de un dispositivo es código escrito específicamente para el dispositivo que el kernel utiliza para poder manejarlo y utilizarlo.