Repaso para Final
CSO

Índice

1. Procesos

1.1 Defina qué es un proceso y cuales son las estructuras de datos que lo representan

Un proceso es una **instancia en ejecución de un programa**, que incluye el codigo de programa, sus datos, recursos asignados (como memoria y archivos abiertos) y su estado de ejecución. Su ciclo de vida comprende desde que se solicita ejecutar hasta que termina.
Para poder ejecutarse, como minimo, incluye: seccion de codigo, seccion de datos, y stacks.

Las principales estructuras de datos que lo representan son:
PCB (info. asociada a cada proceso).
Tabla de procesos (contiene los PCB de todos los procesos en ejecucion).
Tabla de archivos abiertos (asociada a cada proceso, contiene los descriptores de los archivos que el proceso tiene abiertos).

1.2 a) Definir “contexto” y “espacio de direcciones” de un proceso. ¿Que info. se almacena en c/u?

b) ¿Donde se almacenan las estructuras mencionadas en 1.1 y cual es la razon de ello?

a)
El contexto de un proceso incluye toda la información que el SO necesita para administrarlo, la cpu para ejecutarlo correctamente. Son parte de este los registros de la CPU, la prioridad del proceso, las E/S pendientes en caso de haberlas, etc.
El espacio de direcciones de un proceso es el conjunto de direcciones de memoria que este ocupa (no incluye su PCB o tablas asociadas).
Un proceso en modo usuario solo puede acceder a su espacio de direcciones, pero en modo kernel puede acceder tanto a los espacios de direcciones de otros procesos como a estructuras internas.

b)
El PCB se almacena en el kernel, el motivo es que este necesita acceso directo y rápido a la información del proceso para gestionar su ejecución, cambios de contexto, y otros aspectos críticos. Al estar en el espacio del núcleo, se garantiza que esta información sea segura y no accesible por procesos de usuario, previniendo así posibles manipulaciones o errores

La tabla de procesos, que contiene los PCB de todos los procesos, también se almacena en el kernel. La razón es similar: el sistema operativo necesita acceder rápidamente a esta tabla para gestionar el estado de todos los procesos

La tabla de archivos abiertos es propia de cada proceso, que se almacena en su propio espacio de memoria del proceso, sin embargo, una referencia a estos descriptores y sus metadatos también está en el kernel, para que el sistema operativo pueda gestionar las operaciones de archivo

1.3 Cuál es el mecanismo utilizado que permite que un proceso sea expulsado de la cpu al finalizar su quantum en planificación RR. Indicar los pasos que se suceden y que componentes hw/sw intervienen en c/u.

El mecanismo utilizado se trata de utilizar un TIMER, que indica las unidades de CPU en las que el proceso se ejecutó. Cuando el mismo llega a 0, el proceso es expulsado.
Este TIMER puede ser global o local, y dos variantes respecto al valor inicial de este: Timer Fijo y Timer Variable.
Intervienen en este mecanismo distintos componentes:
HARDWARE:

TIMER: configura y genera la interrupción según este programada
Controlador de interrupciones: Detecta la interrupción y avisa a la CPU.
CPU.

SOFTWARE:

Planificador del SO: Asigna el quantum, selecciona los procesos y maneja el cambio de contexto.
Manejador de interrupciones: Guarda el estado del proceso actual y maneja la interrupción.
Kernel: Realiza el cambio de contexto entre procesos.

1.4 ARMADOS

1.4.1 Ud participa en la implementación de un nuevo SO, desarrollando el concepto “proceso” junto con el “Algoritmo de planificación RR”. ¿Cómo los implementaría/diseñaría? Tenga en cuenta las estructuras de datos necesarias, tareas que deberá realizar/implementar así como el apoyo del HW que se necesita. (Interrupciones, modos de ejecución, etc).

1. ***Estructuras de datos***
   - PCB. Así es como vamos a representar a cada proceso
   - Tabla de procesos. Una tabla global que mantiene una lista de todos los PCB de los procesos activos en el sistema. Esta tabla facilita el acceso rápido a los procesos por parte del sistema operativo.
   - Cola de listos. Una estructura de datos (lista enlazada, cola circular, etc.) que contiene los procesos listos para ejecutar, ordenados de acuerdo con la planificación Round Robin.
2. ***Tareas a implementar***
   - Inicialización de procesos

    - Crear procesos y asignar un PCB a cada uno.
    - Inicializar el Quantum y colocar el proceso en la cola de listos.
    - Cargar el PID en memoria y configurar el PC para apuntar a la primera instrucción.
- Planificación RR
    - Implementar un manejador de interrupciones que se dispare al agotar el Quantum
    - Desarrollar un algoritmo de planificación Round Robin que:
        a. Guarde el estado del estado actual en su PCB cuando se agote el quantum.
        b. Seleccione el próximo proceso de la cola de listos.
        c. Realice un cambio de contexto, restaurando los registros y el PC del siguiente proceso.
        d. Si el proceso anterior no ha terminado, colocarlo al final de la Ready Queue.
- Manejo de interrupciones
    - Implementar una rutina de manejo de interrupciones para capturar y manejar eventos de hardware (como la expiración de un quantum).
    - Configurar y gestionar el timer que genera interrupciones a intervalos regulares (c/ quantum).
- Cambio de contexto
    - Diseñar el kernel para soportar cambios de contexto eficientes, donde:
        a. El estado del proceso saliente se guarda en su PCB.
        b. El estado del próximo proceso se restaura desde su PCB.
        c. El kernel cambia el modo de ejecución a modo usuario para continuar con la ejecución del proceso.
- Manejo de modos de ejecución
    - Implementar los modos de ejecución Usuario y Kernel
        a. el primero es para ejecutar procesos de usuario con acceso limitado
        b. El segundo es para ejecutar código privilegiado.
    - Implementar mecanismos de protección para evitar que procesos en modo usuario accedan directamente a recursos críticos del sistema.

3. ***Apoyo del Hardware***
    - TIMER
        - Configurarlo para que genere interrupciones periódicas, luego las utilizara el quantum.
    - Controlador de Interrupciones
        - Necesario para manejar las señales del timer, notificando al CPU cuando es necesario cambiar de proceso.
    - Unidad de gestión de memoria (MMU)
        - Necesario para traducir las memorias virtuales a físicas, garantizando que los procesos estén aislados y no puedan interferir en la memoria de otros.
    - CPU
4. ***Pasos de ejecución***
    - Creación del proceso

        - El SO crea un proceso, inicializa su PCB, asigna memoria y lo coloca en la cola de listos.
    - Asignación del quantum
        - Se configura el temporizador para la duración del quantum.
    - Ejecutar proceso
        - El proceso se ejecuta hasta que agota su quantum o se bloquea esperando un recurso.
    - Interrupción y cambio de contexto
        - Si el quantum se agota, se genera una interrupción, se guarda el contexto, se selecciona el siguiente proceso, y se realiza el cambio de contexto.
    - Retorno al modo usuario
        - El proceso seleccionado continúa su ejecución, y el ciclo se repite.

1.5 Qué son las syscalls y cómo podrían implementarse. tenga en cuenta que apoyo debería brindar el HW para implementarlas.

Las **syscalls** (llamadas al sistema) son funciones proporcionadas por el sistema operativo que permiten a los programas de usuario solicitar servicios o recursos del kernel, como la manipulación de archivos, la gestión de procesos, la comunicación entre procesos, y otros servicios esenciales. Dado que el kernel opera en un modo privilegiado y tiene acceso a recursos críticos del sistema, las syscalls son el mecanismo seguro mediante el cual los programas en modo usuario pueden interactuar con el núcleo del sistema operativo.

Implementación de Syscalls

1. **Definición de Syscalls**:
    - El kernel define un conjunto de funciones que representan las syscalls, como read(), write(), fork(), etc.
    - Cada syscall tiene un número identificador único.
2. **Interfaz de Syscalls**:
    - Se proporciona una interfaz en las librerías del sistema (por ejemplo, la libc en Unix) que los programas de usuario utilizan para invocar las syscalls. Esta interfaz empaqueta los argumentos de la syscall y realiza una interrupción de software para pasar el control al kernel.
3. **Interrupción de Software (Trap)**:
    - Hardware (CPU): La invocación de una syscall se realiza mediante una instrucción especial, como INT en x86 o SVC en ARM, que genera una interrupción de software o trap. Esto cambia el modo de ejecución de usuario a kernel y transfiere el control al kernel.
    - **Modo Kernel**: El CPU cambia al modo kernel, permitiendo al sistema operativo ejecutar operaciones privilegiadas.
4. **Manejo de la Syscall**:
    - **Software (Kernel)**: El kernel maneja la interrupción, identifica la syscall mediante el número de syscall, y ejecuta la función correspondiente con los argumentos proporcionados.

    - El resultado de la syscall (si lo hay) se devuelve al espacio de usuario a través de registros o la pila.
5. **Retorno al Espacio de Usuario**:
    - **Hardware (CPU)**: Una vez completada la syscall, el kernel restablece el contexto del proceso de usuario y retorna el control al proceso, devolviendo el resultado de la syscall.

Apoyo del Hardware

- **Modos de Ejecución**: El CPU debe soportar modos de ejecución separados (usuario y kernel) para proteger el sistema operativo de operaciones no autorizadas por procesos de usuario.
- **Instrucción de Interrupción de Software**: El CPU necesita una instrucción que permita generar interrupciones de software (traps) para invocar syscalls.
- **Unidad de Gestión de Memoria (MMU)**: Asegura que el espacio de direcciones del usuario esté aislado del kernel, protegiendo la memoria del sistema operativo.

1.6 Kernel

Es un conjunto de módulos de software. Se ejecuta en el procesador como cualquier otro proceso.

Enfoque 1: Kernel como entidad independiente.

- Se ejecuta fuera de todo proceso.
- Cuando un proceso es interrumpido o realiza una syscall, el contexto de este se salva y el control de este se pasa al kernel.
- Tiene su propia región de memoria y su propio stack.
- Finalizada su actividad, le devuelve el control al proceso que corresponda.
- **NO** es un proceso. Se ejecuta como entidad independiente en modo privilegiado.

Enfoque 2: Kernel “dentro” del proceso.

- El código del kernel se encuentra dentro del espacio de direcciones de cada proceso.
- Se ejecuta en el mismo contexto que algún proceso de usuario.
- Se puede ver como una colección de rutinas que el proceso utiliza.
- Dentro de un proceso se encuentra el código del programa y el código de los módulos de SW del SO (kernel)
- Cada proceso tiene su propio stack (uno en modo usuario y otro en modo kernel)
- El código del kernel es compartido por todos los procesos.
- Cada interrupción es atendida en el contexto del proceso que se encontraba en ejecución pero en modo kernel.

2. Memoria

2.1 Memoria virtual con paginación por demanda ¿como funciona la técnica? Para ello tenga en cuenta: estructuras de datos necesarias y su objetivo, actividades desarrolladas por el SO, fallos de página y su resolución y apoyo del HW requerido.

Primero, para utilizar esta tecnica el HW debe soportar paginacion por demanda, se necesita un dispositivo de memoria secundaria que de apoyo para almacenar las porciones del proceso que no estan en memoria principal (*area de swap*), y el SO debe ser capaz de manejar el movimiento de las paginas entre la memoria principal y la memoria secundaria.
En esta tecnica, cada proceso tiene su tabla de paginas, cada entrada en esta tabla referencia al marco en el que se encuentra la pagina en memoria principal ademas de contar con bits de control (V indica si la pag esta en memoria, M indica si esta fue modificada). Para encontrar la direccion se cuenta con la MMU.

Cuando el proceso intenta utilizar una direccion de memoria que esta en una pagina que no se encuentra en memoria principal se genera un PAGE FAULT. El HW detecta esto y genera un trap al SO. El SO puede mandar al proceso a estado “blocked” hasta cargar la pagina necesaria ->
El SO busca un marco en la memoria y genera una op E/S al disco para copiar en dicho marco la pagina del proceso que se necesita utilizar.

2.2 Técnica de frecuencia de fallos de página (PF o FFP) Explique como funciona y cual es su objetivo. Como podría utilizarla el kernel para la detección de hiperpaginación (thrashing), y qué acciones se deberían tomar en caso de detectarla.

Decimos que el sistema esta en trashing cuando pasa mas tiempo paginando que ejecutando procesos. Como consecuencia tenemos una baja importante en el rendimiento del sistema. Esto se puede limitar con algoritmos de remplazo local, logrando que cuando un proceso entra en trashing no le robe frames a otros.
La tecnica PFF utiliza la taza de fallos de pagina para estimar si el proceso tiene un conjunto residente que representa adecuadamente al WS.
Para esto, se establecen limites superior e inferior de las PFF´s deseadas. Si se excede PFF_max se le asignan mas frames al proceso, si esta por debajo del PFF_min se le quitan.
Se puede llegar a suspender un proceso si no hay mas frames libres y todos los demas estan por encima de PFF_max.

2.3 Defina el concepto de paginación y paginación por demanda ¿que diferencias hay entre estas? ¿cuales son las estructuras de datos utilizadas para dar soporte a la paginación por demanda y quien las mantiene?
Enumere los pasos que se suceden cuando ocurre un PF y que componentes HW/SW intervienen en cada uno.

La paginacion es una técnica de gestión de memoria en la que el espacio de direcciones virtuales de un proceso se divide en bloques de tamaño fijo llamados páginas. Estas páginas se mapean a bloques físicos de memoria llamados marcos. La paginación permite que un proceso utilice más memoria virtual que la disponible físicamente, y facilita la protección y aislamiento de los procesos.
Paginación por Demanda es una variación de la paginación en la que las páginas no se cargan en la memoria física hasta que realmente se necesitan. En lugar de cargar todo el proceso en la memoria al inicio, el sistema operativo carga solo aquellas páginas que el proceso intenta acceder, generando un fallo de página si la página requerida no está en la memoria. Aqui se utiliza una pequeña caché de hardware que almacena las traducciones recientes de direcciones virtuales a físicas, para acelerar el acceso a la memoria, llamado Translation Lookaside Buffer (TLB), asi como tambien el area de swap donde se almacenan las páginas que no están en la memoria física, y la tabla de paginas.

PASOS PARA ABORDAR PF:

1. La MMU traduce la dirección virtual a una dirección física usando la tabla de páginas. Genera una interrupción de hardware para señalar el fallo de página.
2. El SO identifica cuál página causó el fallo, revisando la tabla de páginas y determina si está en el swap space o necesita ser cargada.
3. El SO selecciona una página existente en la memoria física para ser reemplazada, utilizando un algoritmo de reemplazo de páginas (por ejemplo, LRU).
4. El controlador de E/S del disco realiza la transferencia de la página desde el disco a la memoria física. El SO espera la finalización de la operación de E/S.
5. El SO actualiza la tabla de páginas para reflejar que la nueva página está en la memoria física, estableciendo el bit de presencia y asignando el marco de memoria.
6. El CPU reanuda la ejecución del proceso interrumpido. El proceso se reintenta, ahora que la página está en memoria.

3. Sistemas de archivos y buffer cache

3.1 Qué es un sistema de archivos y para qué sirve (objetivos). Tomando el sistema de archivos SystemV, explique qué acciones y estructuras son modificadas al crear un nuevo archivo. Y al modificarlo? y al eliminarlo? y al cambiar los permisos?

Un sistema de archivos es un conjunto de unidades de SW que proveen los servicios necesarios para la utilizacion de archivos. Facilita el acceso a los archivos por parte de las aplicaciones, y permite la abstraccion al programador, en cuanto al acceso de bajo nivel.

Los objetivos del SO en cuanto a archivos son:

- Cumplir con la gestion de datos
- Cumplir con las solicitudes de usuario
- Garantizar la integridad del contenido de los archivos (minimizando/eliminando la posiblidad de perder o destruir datos)
- Dar soporte de E/S a distintos dispositivos
- Brindar un conjunto de interfaces de E/S para tratamiento de archivos.

1. Crear un Nuevo Archivo

Estructuras Modificadas:

- **Directorio Contenedor**: El directorio donde se creará el archivo necesita ser actualizado para incluir una nueva entrada que apunte al nuevo inodo del archivo.
- **Inodo**: Un nuevo inodo se asigna para representar el archivo. El inodo contiene información sobre el archivo como el propietario, los permisos, la marca de tiempo, y los punteros a los bloques de datos.
- **Mapa de Inodos**: El sistema debe actualizar el mapa de inodos para marcar el inodo recién asignado como ocupado.
- **Bloques de Datos**: Aunque inicialmente un archivo vacío no necesita bloques de datos, el sistema puede asignar bloques si se escribe inmediatamente al archivo.

Acciones Realizadas:

1. **Asignación de un Inodo**: El sistema de archivos asigna un inodo libre.
2. **Actualización del Directorio**: Se crea una nueva entrada en el directorio que contiene el nombre del archivo y el número de inodo.
3. **Inicialización del Inodo**: El inodo se inicializa con la información del archivo (propietario, permisos, etc.).
4. **Asignación de Bloques**: Si se necesita, se asignan bloques de datos y el inodo se actualiza con los punteros a estos bloques.

2. Modificar un Archivo

Estructuras Modificadas:

- **Inodo**: Se actualiza para reflejar los cambios, como la modificación de la marca de tiempo, el tamaño del archivo y los punteros a los bloques de datos si se añaden o eliminan datos.
- **Bloques de Datos**: Los bloques correspondientes en el disco se escriben o reescriben para almacenar los nuevos datos.

Acciones Realizadas:

1. **Escritura de Datos**: Los datos son escritos en los bloques de datos correspondientes.
2. **Actualización del Inodo**: El inodo se actualiza con la nueva marca de tiempo, tamaño, y cualquier cambio en los punteros a los bloques de datos.
3. **Asignación o Liberación de Bloques**: Si el tamaño del archivo crece, se asignan nuevos bloques; si el tamaño decrece, los bloques sobrantes se liberan.

3. Eliminar un Archivo

Estructuras Modificadas:

- **Directorio Contenedor**: La entrada del directorio que apunta al inodo del archivo se elimina.
- **Inodo**: El inodo es liberado, marcándolo como disponible en el mapa de inodos.
- **Bloques de Datos**: Los bloques de datos que estaban asignados al archivo se liberan.

Acciones Realizadas:

1. **Eliminación de la Entrada del Directorio**: Se borra la entrada del directorio correspondiente al archivo.
2. **Liberación del Inodo**: El inodo asociado se libera.
3. **Liberación de Bloques**: Los bloques de datos ocupados por el archivo se liberan y el mapa de bloques se actualiza.
4. **Actualización del Mapa de Inodos**: Se marca el inodo como disponible.

4. Cambiar Permisos de un Archivo

Estructuras Modificadas:

- **Inodo**: El inodo del archivo es modificado para reflejar los nuevos permisos.

Acciones Realizadas:

1. **Modificación del Inodo**: Los bits de permisos en el inodo se actualizan según los nuevos valores de permisos.
2. **Actualización de la Marca de Tiempo**: Se puede actualizar la marca de tiempo del inodo si es necesario (en algunos sistemas Unix).

3.2 Cómo garantiza el diseño de un SO la independencia entre los sistemas de archivos y los dispositivos donde los mismos son utilizados.

1. Abstracción de Dispositivos

El sistema operativo introduce una capa de abstracción que oculta las diferencias específicas entre los distintos tipos de dispositivos de almacenamiento (discos duros, SSD, dispositivos de red, etc.) y presenta una interfaz uniforme para el manejo de archivos. Esta capa de abstracción permite que los sistemas de archivos interactúen con un conjunto común de operaciones, independientemente del hardware subyacente.

Los **controladores de dispositivos** son los encargados de traducir las operaciones genéricas del sistema de archivos (como leer, escribir, abrir, cerrar, etc.) en comandos específicos que el dispositivo puede entender. Así, el sistema de archivos no necesita conocer los detalles del dispositivo físico; simplemente realiza operaciones estándar que el controlador correspondiente se encarga de ejecutar correctamente.

2. Interfaz VFS (Virtual File System)

El Virtual File System (VFS) es una interfaz que permite que múltiples sistemas de archivos coexistan y funcionen en un mismo sistema operativo de manera uniforme. El VFS actúa como un intermediario entre las aplicaciones y los distintos sistemas de archivos, proporcionando una API común que es independiente del sistema de archivos subyacente.

El VFS permite que los programas trabajen con archivos y directorios sin preocuparse por el tipo de sistema de archivos en el que están almacenados, ya sea ext4, NTFS, FAT, o cualquier otro. El VFS maneja la traducción de las llamadas al sistema (syscalls) hacia el sistema de archivos específico, asegurando que las aplicaciones puedan acceder a los archivos de manera consistente, independientemente del dispositivo o sistema de archivos utilizado.

3. Puntos de Montaje

La independencia también se garantiza a través del concepto de **puntos de montaje**. En un sistema operativo Unix, por ejemplo, diferentes sistemas de archivos pueden ser montados en un único árbol de directorios. Esto significa que un directorio en un sistema de archivos existente puede convertirse en el punto de acceso a otro sistema de archivos, sin que las aplicaciones o usuarios necesiten saber en qué dispositivo físico se encuentran los datos.

Al utilizar puntos de montaje, el SO puede combinar diferentes sistemas de archivos en una única jerarquía, permitiendo que los usuarios y aplicaciones accedan a ellos de manera uniforme, sin importar en qué tipo de dispositivo están almacenados los datos.

4. Independencia del Hardware

Gracias a estas capas de abstracción y a la utilización de controladores, el sistema operativo puede operar sobre una amplia variedad de hardware sin necesidad de que los sistemas de archivos se adapten específicamente a cada dispositivo. La interacción con los dispositivos físicos se maneja en un nivel más bajo, permitiendo que los sistemas de archivos permanezcan independientes del hardware.

3.3 Defina el concepto de Buffer Cache junto con las estructuras de datos que utiliza para su funcionamiento.
Ejemplifique el funcionamiento del mismo cuando un proceso requiere acceder a un archivo de datos.

El concepto Buffer Caché hace referencia a Buffers en memoria principal para almacenar de manera temporal bloques de disco con el objetivo de minimizar la frecuencia de acceso a disco.
Cuando un proceso quiere acceder a un bloque de la caché hay dos alternativas:

- Se copia el bloque al espacio de direcciones del usuario -> no permite compartir el bloque.
- Se trabaja el bloque como memoria compartida -> permite acceso a varios procesos.
  -> Dicha área compartida debe ser limitada, con lo cuál debe existir un algoritmo de reemplazo.

Objetivo y estructura de un Buffer Cache

- Minimizar la frecuencia de acceso a disco.
- Estructura que se forma por buffers.
- El kernel asigna un espacio en la memoria durante la inicialización de dicha estructura.
- El buffer se compone de dos partes:
  - Header: contiene información del bloque, número del bloque, estado, relación con otros buffers, etc.
  - Buffer en sí: el lugar donde se almacena el bloque de disco traído a memoria.

Funcionamiento del buffer cache

- Cuando un proceso requiere el acceso a un archivo, utiliza su inodo para localizar los bloques de datos donde se halla el mismo.
- El requerimiento llega al buffer cache quien evalúa si puede satisfacer el requerimiento o si debe realizar la E/S.