SISTEMAS OPERATIVOS

ACTIVIDAD DE APRENDIZAJE 11
TAREA 6

Profesor:
VIOLETA DEL ROCIO BECERRA VELAZQUEZ

VERDUZCO ROSALES LUIS ENRIQUE
223992388
Ingenieria en Computación

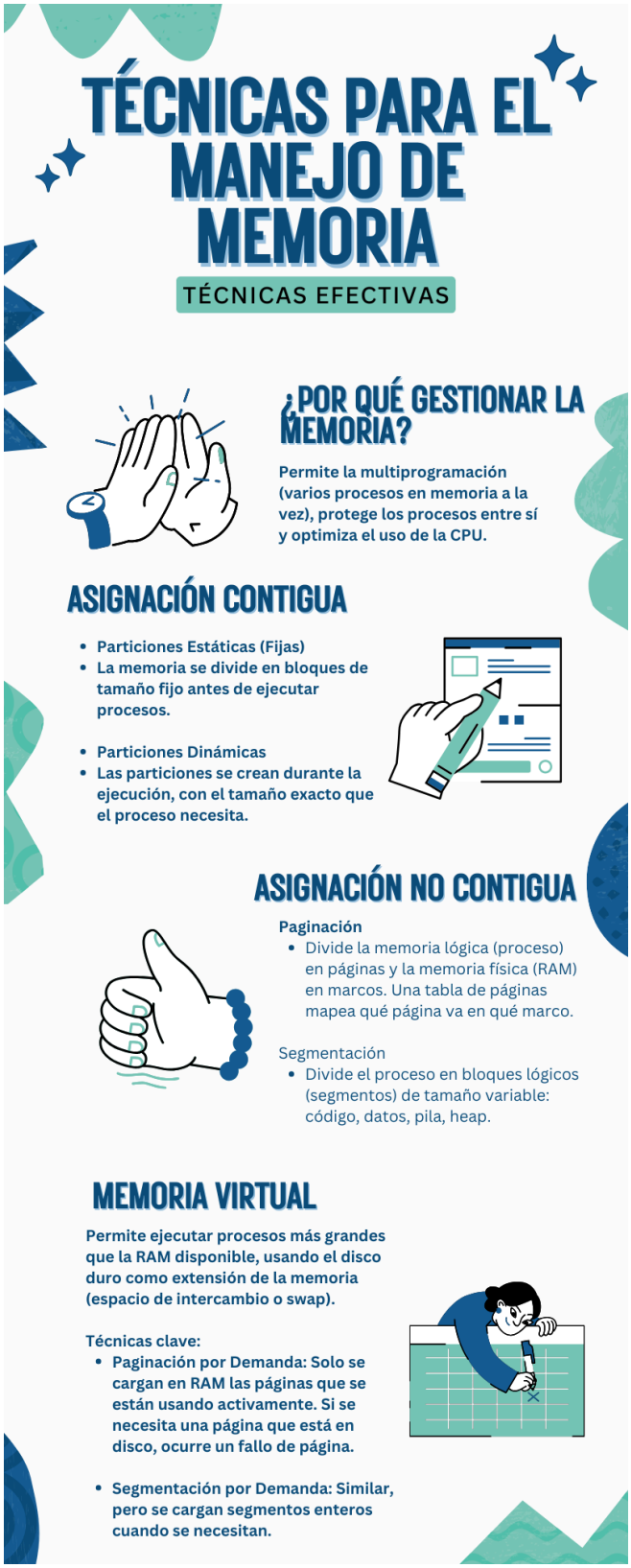
02/11/25

CUCEI DIVTIC D04

ÍNDICE

---

1. Infografía



2. ¿En qué consiste la paginación simple?

La paginación simple es una técnica de gestión de memoria no contigua que resuelve el problema de la fragmentación externa. Su funcionamiento se basa en dos conceptos:

1. Memoria Física (RAM): Se divide en bloques de tamaño fijo llamados marcos de página (frames).
2. Memoria Lógica (Proceso): Se divide en bloques del *mismo* tamaño fijo llamados páginas (pages).

Cuando un proceso necesita ejecutarse, sus páginas se cargan en cualquier marco de página que esté disponible en la RAM. No es necesario que los marcos estén uno al lado del otro.

Para rastrear dónde está cada página, el sistema operativo mantiene una tabla de páginas por cada proceso. Esta tabla traduce la dirección lógica (Nº de página) a la dirección física (Nº de marco) donde realmente reside.

- Ventaja: Elimina la fragmentación externa.
- Desventaja: Introduce fragmentación interna en la última página del proceso, si este no es un múltiplo exacto del tamaño de página.

3. ¿En qué consiste la Técnica de Particiones Estáticas?

También llamada particionamiento fijo, es una de las técnicas de asignación contigua más antiguas.

Consiste en dividir la memoria física (RAM) en un número fijo de particiones *antes* de que se ejecute cualquier proceso (generalmente al arrancar el sistema). Estas particiones pueden ser todas del mismo tamaño o de tamaños diferentes.

Funcionamiento:

1. Cuando un proceso llega, el sistema operativo busca una partición libre que sea lo suficientemente grande para albergarlo.
2. El proceso se carga en esa partición y la ocupa por completo, sin importar si el proceso es más pequeño que la partición.
- Desventaja Principal: Produce fragmentación interna severa. Si un proceso de 20 KB se carga en una partición fija de 64 KB, se desperdician 44 KB de RAM que no pueden ser usados por ningún otro proceso mientras el primero esté allí.

4. ¿En qué consiste la Técnica de Particiones Dinámicas?

Es la evolución de las particiones estáticas, diseñada para solucionar la fragmentación interna. En esta técnica de asignación contigua, las particiones no están predefinidas.

Funcionamiento:

1. Inicialmente, la memoria es un solo bloque grande y contiguo (un gran "agujero").
2. Cuando un proceso (de tamaño 'N') llega, el sistema operativo busca un agujero lo suficientemente grande.
3. El SO asigna *exactamente* 'N' bytes de ese agujero al proceso, creando una partición de tamaño dinámico.
4. El espacio restante del agujero (si lo hay) se convierte en un nuevo agujero más pequeño.
5. Cuando un proceso termina, su partición se libera y se fusiona con cualquier agujero adyacente.

- Ventaja: Elimina la fragmentación interna.
- Desventaja Principal: Con el tiempo, la memoria se llena de pequeños agujeros no utilizables entre las particiones ocupadas. Esto se llama fragmentación externa.

5. ¿En qué consiste la Memoria Virtual?

La Memoria Virtual es una técnica de gestión de memoria que permite al sistema operativo simular un espacio de memoria (RAM) mucho más grande del que realmente existe físicamente.

Logra esto separando la memoria lógica (la que ve el programador y el proceso) de la memoria física (la RAM real).

Utiliza el almacenamiento secundario (como un SSD o disco duro) como una extensión de la RAM. Este espacio en disco se denomina espacio de intercambio (*swap space* o archivo de paginación).

Beneficios:

- Programas más grandes: Permite ejecutar programas que, en su totalidad, no cabrían en la RAM.
- Mayor multiprogramación: Permite que más procesos residan en memoria (o parcialmente en memoria), mejorando el uso de la CPU.
- Abstracción: El programador no necesita preocuparse por cuánta RAM física hay disponible.

6. Describa el funcionamiento de paginación con memoria virtual

Esto se conoce comúnmente como paginación por demanda (*demand paging*). Es la implementación más común de la memoria virtual.

Funcionamiento:

1. Cuando se inicia un proceso, el sistema operativo *no* carga todas sus páginas en la RAM. Solo carga las páginas iniciales necesarias (o a veces, ninguna).

2. La tabla de páginas de cada proceso incluye un bit de validez/presencia (valid/invalid bit) para cada página.

    - Válido (1): La página está en un marco de RAM.
    - Inválido (0): La página no está en RAM (está en el disco, en el espacio de intercambio).

3. El CPU ejecuta el proceso. Cuando intenta acceder a una instrucción o dato en una página marcada como "Inválida", el hardware genera una interrupción especial llamada fallo de página (page fault).
4. El sistema operativo toma el control y maneja el fallo de página: a. Verifica si el acceso era válido (si la página pertenece al proceso). b. Busca la página requerida en el espacio de intercambio (disco). c. Busca un marco de página libre en la RAM. d. Si no hay marcos libres, ejecuta un algoritmo de reemplazo de páginas (como LRU o FIFO) para "desalojar" (victimizar) una página de la RAM (guardándola en disco si fue modificada). e. Carga la página requerida desde el disco al marco de RAM (ahora libre). f. Actualiza la tabla de páginas del proceso (marca el bit como "Válido" y asigna el nuevo número de marco). g. Reinicia la instrucción que causó el fallo.

Desde la perspectiva del proceso, todo esto es transparente; simplemente experimentó un pequeño retraso.

7. Describa el funcionamiento de Segmentación con memoria virtual

Esto se conoce como segmentación por demanda. El concepto es idéntico al de la paginación por demanda, pero aplicado a segmentos lógicos (código, datos, pila) en lugar de páginas de tamaño fijo.

Funcionamiento:

1. Cuando se inicia un proceso, sus segmentos (código, datos, etc.) se almacenan en el espacio de intercambio en disco. Ninguno se carga en RAM inicialmente.
2. La tabla de segmentos del proceso incluye un bit de presencia para cada segmento (similar al bit de validez en paginación).
3. Cuando el proceso intenta acceder a una dirección dentro de un segmento (ej. llamar a una función en el segmento de código), el hardware comprueba la tabla de segmentos.
4. Si el bit de presencia del segmento es "0" (no está en RAM), se genera un fallo de segmento (*segment fault*).
5. El sistema operativo maneja el fallo: a. Localiza el segmento completo en el disco. b. Busca un bloque de memoria *contiguo* en la RAM lo suficientemente grande para albergar el segmento completo (aquí reaparece el problema de la fragmentación externa). c. Carga el segmento completo desde el disco a ese bloque de RAM. d. Actualiza la tabla de segmentos (marca el bit como "Presente", y establece la dirección base y el límite). e. Reinicia la instrucción que causó el fallo.

En la práctica, los sistemas modernos (como x86) usan un esquema híbrido de segmentación con paginación. La segmentación proporciona la visión lógica y la protección, pero cada segmento se divide internamente en páginas, y es la paginación por demanda la que realmente maneja la memoria virtual.

8. ¿Cuáles son los elementos que conforman la tabla de páginas?

La tabla de páginas es la estructura de datos clave que traduce direcciones lógicas a físicas. Cada entrada en la tabla (una por cada página del proceso) contiene, como mínimo:

1. Número de Marco de Página (Frame Number): Es el elemento principal. Indica en qué marco de la RAM física se encuentra almacenada esa página.

   Además, incluye varios bits de control esenciales para la memoria virtual y la protección:

2. Bit de Presencia/Ausencia (Valid/Invalid Bit):

   - Indica si la página está actualmente en la RAM (presente) o si está en el disco (ausente). Es fundamental para gestionar los fallos de página.

3. Bit de Modificación (Dirty Bit):

   - Se activa si la página ha sido escrita (modificada) desde que se cargó en RAM. Si este bit está activo cuando la página va a ser reemplazada, el SO sabe que debe guardarla en disco; si no, simplemente puede descartarla (ya que la copia en disco está actualizada).

4. Bit de Referencia (Accessed Bit):

   - Se activa cada vez que la página es leída o escrita. El SO lo usa para implementar algoritmos de reemplazo de páginas (como LRU - Menos Usado Recientemente), para decidir qué página desalojar.

5. Bits de Protección (Read/Write/Execute):

   - Definen los permisos de acceso a esa página. Por ejemplo, una página del segmento de código puede estar marcada como "Solo Lectura" y "Ejecutable", mientras que una página de datos puede ser "Lectura/Escritura".

9. ¿Qué son los buffers, su importancia y su manejo?

- ¿Qué son?

Un buffer (o búfer) es una región de memoria (generalmente en la RAM) que se utiliza para almacenar datos temporalmente mientras se transfieren de un lugar a otro.
Actúan como un área de espera o "sala de amortiguación" entre dos dispositivos o procesos que operan a velocidades diferentes o que manejan datos en bloques (chunks) de tamaños distintos.

- Ejemplos comunes: El buffer del teclado (almacena las teclas que presionas), el buffer de un disco (almacena un bloque de datos leído del disco), o el buffer de un video en streaming (almacena los próximos segundos de video).

- Su Importancia

La importancia de los buffers radica en su capacidad para desacoplar procesos y dispositivos:

1. Manejo de Diferencias de Velocidad: Permiten que un componente rápido (como la CPU) envíe datos a un componente lento (como una impresora o un disco duro) sin tener que esperar. La CPU "vuelca" los datos en el buffer y sigue con otra tarea; el dispositivo lento los consume del buffer a su propio ritmo.

2. Manejo de Diferencias de Tamaño de Datos: Permiten que los datos se transfieran en bloques de diferentes tamaños. Por ejemplo, una aplicación de red puede enviar datos byte por byte, pero la tarjeta de red los acumula en un buffer hasta tener un paquete completo (ej. 1500 bytes) antes de enviarlo por el cable.

3. Suavizado de Flujos (Smoothing): En aplicaciones como el streaming de video o audio, el buffer almacena varios segundos de contenido por adelantado. Esto compensa las pequeñas interrupciones o variaciones ("jitter") en la velocidad de la red, evitando que la reproducción se corte.

- Su Manejo (Gestión)

El manejo de buffers es una tarea crítica del sistema operativo (a través de sus drivers o el kernel):

- Estructuras de Datos:

  La implementación más común de un buffer es una cola FIFO (First-In, First-Out), a menudo implementada como un buffer circular. Este usa dos punteros (uno de "escritura" o head y uno de "lectura" o tail) para gestionar el espacio de forma eficiente.

- Problemas Comunes:

    - Buffer Overflow (Desbordamiento): Ocurre cuando se intenta escribir datos en un buffer que ya está lleno. Esto no solo es un error, sino una de las vulnerabilidades de seguridad más explotadas (ya que los datos pueden sobrescribir otras áreas de memoria, como código ejecutable).

    - Buffer Underflow (Subdesbordamiento): Ocurre cuando se intenta leer datos de un buffer que está vacío.

- Sincronización: Cuando el buffer es compartido entre un productor de datos (ej. la red) y un consumidor (ej. la aplicación), el SO debe usar mecanismos de sincronización (como semáforos o mutex) para evitar que ambos intenten acceder al buffer al mismo tiempo y corrompan los datos.

Conclusión

En conclusión, aprendí que la gestión de memoria es una de las cosas más fundamentales y complejas de un sistema operativo, actuando como el pilar más importante ya que gracias a él existe la informática moderna. Su objetivo principal es resolver el conflicto principal entre la demanda (aparentemente sin fin) de memoria por parte de las aplicaciones y la oferta (limitada y finita) de RAM física.

Bibliografía

Mujica-Sequera, R. M. (2023, 3 de abril). TÉCNICAS DE GESTIÓN DE MEMORIA DE LAS COMPUTADORAS | DOCENTES 2.0. Docentes 2.0. Recuperado de: https://blog.docentes20.com/2023/04/%E2%9C%8Dtecnicas-de-gestion-de-memoria-de-las-computadoras-docentes-2-0/

AI-FutureSchool. (s.f.). Gestión de la memoria en programación. Recuperado de: https://www.ai-futureschool.com/es/programacion/gestion-de-la-memoria-en-programacion.php

Universidad de Almería, Departamento de Lenguajes y Computación. (c. 2003). TEMA 3. GESTIÓN DE MEMORIA [Diapositivas de clase]. Recuperado de: https://w3.ual.es/~acorral/DSO/Tema_3.pdf

Ruz Ortiz, J. J. (c. 2012). Tema 5: Organización de la memoria: memoria principal [Diapositivas de clase]. Facultad de Informática, Universidad Complutense de Madrid. Recuperado de: https://www.fdi.ucm.es/profesor/jjruz/web2/temas/ec5.pdf

Educatica. (s.f.). Gestión de memoria. Recuperado de: https://www.educatica.es/sistemas-operativos/principios-basicos/gestion-de-memoria/