Universidad de Costa Rica

Escuela de Ciencias de la Computación e Informática

CI-0122

Sistemas Operativos

Diseño de tarea programada

Alejandro Jiménez Rojas - C04079

Katherine Acosta Barquero - B70047

Docente:

Tracy Hernández

II ciclo 2024

Índice

INTRODUCCIÓN

La memoria virtual se encarga de mantener el espacio de direcciones accesibles para los procesos, además, el administrador de memoria posee dos vistas separadas del espacio de direcciones de un proceso, la lógica que describe las instrucciones que el sistema de memoria virtual ha recibido y la segunda vista física de cada espacio de direcciones, donde se visualizan las tablas de páginas de hardware del proceso (Silberschatz et al., 2018, p. 798) El presente diseño busca implementar un sistema de memoria virtual, donde se desarrollará un sistema que permita a varios procesos ejecutar instrucciones de manera concurrente, mientras que accede a un espacio de direcciones lógicas que se traducirán a direcciones físicas en la memoria principal.

DESARROLLO

La gestión de la memoria virtual es esencial en un sistema multiprocesador, ya que permite que varios procesos compartan la memoria de manera eficiente y segura. Para este proyecto, se utilizará una tabla de páginas que permitirá la traducción de direcciones lógicas a físicas, y se simulará un entorno donde los procesos pueden generar fallos de página y acceder a la memoria principal o secundaria según sea necesario.

Traducción de Direcciones

El sistema gestionará la memoria mediante una tabla de páginas por proceso, donde cada entrada en la tabla corresponde a una página de la memoria virtual. Cada proceso utilizará direcciones lógicas que serán traducidas a direcciones físicas mediante una unidad de gestión de memoria (MMU), la cual tomará la dirección lógica y la descompondrá en dos partes:

1. **Número de Página (Page Number):** Indica la página en la tabla de páginas que se está solicitando.
2. **Desplazamiento (Offset):** Especifica la posición exacta dentro de la página.

El MMU buscará en la tabla de páginas correspondiente para obtener el marco de memoria asociado y, junto con el desplazamiento, calculará la dirección física exacta. Si la página solicitada no está en la memoria principal, se producirá un fallo de página.

Estructura de la Tabla de Páginas

La tabla de páginas estará organizada como una estructura de datos donde cada entrada corresponde a una página de la memoria virtual. Cada entrada contendrá varios campos, entre ellos:

- **Bit de Validez (Válido/Inválido):** Indica si la página está actualmente cargada en la memoria principal. Un valor "inválido" indica que la página reside en el almacenamiento secundario y que un acceso generará un fallo de página.
- **Bit de Referencia:** Este bit se utilizará para ayudar en la implementación del algoritmo de reemplazo. Cada vez que se accede a una página, el bit de referencia se actualiza para indicar que la página ha sido utilizada recientemente.
- **Bit de Suciedad:** Indica si la página ha sido modificada desde que se cargó en la memoria principal. Si el bit de suciedad está activo, la página deberá escribirse de nuevo en el almacenamiento secundario si se necesita reemplazar.
- **Número de Marco de Memoria:** Contiene la dirección física del marco en el que se encuentra la página cuando está cargada en la memoria principal.

Manejo de Fallos de Página

Cuando un proceso intenta acceder a una dirección lógica cuya página no está en la memoria principal (bit de validez en "inválido"), se detecta un fallo de página, entonces el sistema deberá cargar la página solicitada desde el almacenamiento secundario (simulado) a la memoria principal. Este proceso implica los siguientes pasos:

1. **Suspensión Temporal del Proceso:** El proceso se suspende hasta que el fallo de página sea resuelto.
2. **Identificación de una Página para Reemplazar:** Si la memoria principal está llena, se seleccionará una página existente para ser reemplazada utilizando un algoritmo de reemplazo de páginas.
3. **Carga de la Página desde el Almacenamiento Secundario:** La página solicitada se copia desde el almacenamiento secundario a un marco de la memoria principal.
4. **Actualización de la Tabla de Páginas:** Se actualizará la tabla de páginas para reflejar la nueva ubicación de la página en la memoria principal. La entrada correspondiente se marcará como válida.
5. **Reinicio de la Instrucción Fallida:** El proceso que causó el fallo de página reiniciará la instrucción que generó el fallo.

El sistema debe gestionar múltiples procesos concurrentes, por lo que la frecuencia de fallos de página y la eficiencia del algoritmo de reemplazo son cruciales para el rendimiento general.

Algoritmo de Reemplazo de Páginas

Se ha decido que para implementar el algoritmo de reemplazo de páginas se usa Least Recently Used (LRU). Se mantendrá una lista o estructura adicional que registre el orden de uso de las páginas y cada vez que una página se acceda, se actualizará su posición en la lista para reflejar que ha sido usada recientemente.

Además, se ha optado por implementar el algoritmo de reemplazo de forma global debido a sus ventajas en la gestión de memoria y la mejora del rendimiento general del sistema en entornos de multiprocesamiento, ya que tiene una mayor eficacia en reducción de fallos de página, al utilizar una prioridad para las páginas utilizadas más recientes, permitiendo una mejor gestión de la memoria en multiprocesamiento.

Tamaño de la Memoria Principal

Para este proyecto, el tamaño de la memoria principal se ha fijado en 16 MB. Este valor se ha escogido para simular un entorno suficientemente complejo que permita la ejecución de varios procesos simultáneos y la observación de la gestión de la memoria. Aunque 16 MB puede parecer pequeño comparado con sistemas modernos, es suficiente para el propósito de simulación de este proyecto.

Tamaño de las páginas y marcos

Se ha elegido un tamaño de página y marco de 4 KB, lo cual es un estándar en la mayoría de los sistemas operativos modernos.

Diagramas UML y de clases

1. **Clase MemoriaVirtual:** Posee la tabla de páginas, marcos de memoria y la lógica del manejo de fallos en la página. Algunos de los métodos serán traducirDireccion(), manejarFalloPagina(), actualizarLRU().
2. **Clase Procesador:** Proceso independiente que solicita direcciones lógicas, procesa la traducción de direcciones y ejecuta instrucciones. Algunos de los métodos serán solicitarDireccion(), ejecutarInstruccion().
3. **Clase MMU:** Interfaz entre los procesadores y la memoria virtual, encargada de la traducción de direcciones y verificación de fallos de página. Algunos de los métodos serán buscarPagina(), asignarFrame().
4. **Clase TablaDePaginas**: Representa la tabla de páginas de un proceso para la traducción de direcciones. Algunos de los métodos serán obtenerFrame().
5. **Clase Frame:** Representa un marco de memoria física en el sistema. Algunos de los métodos serán asignarPagina().
6. **Clase EntradaPagina:** Representa una entrada en la tabla de páginas.
7. **Clase TLB:** Almacena una caché de traducciones de direcciones para acelerar el acceso a las páginas de memoria. Algunos de los métodos serán consultar() y actualizar().

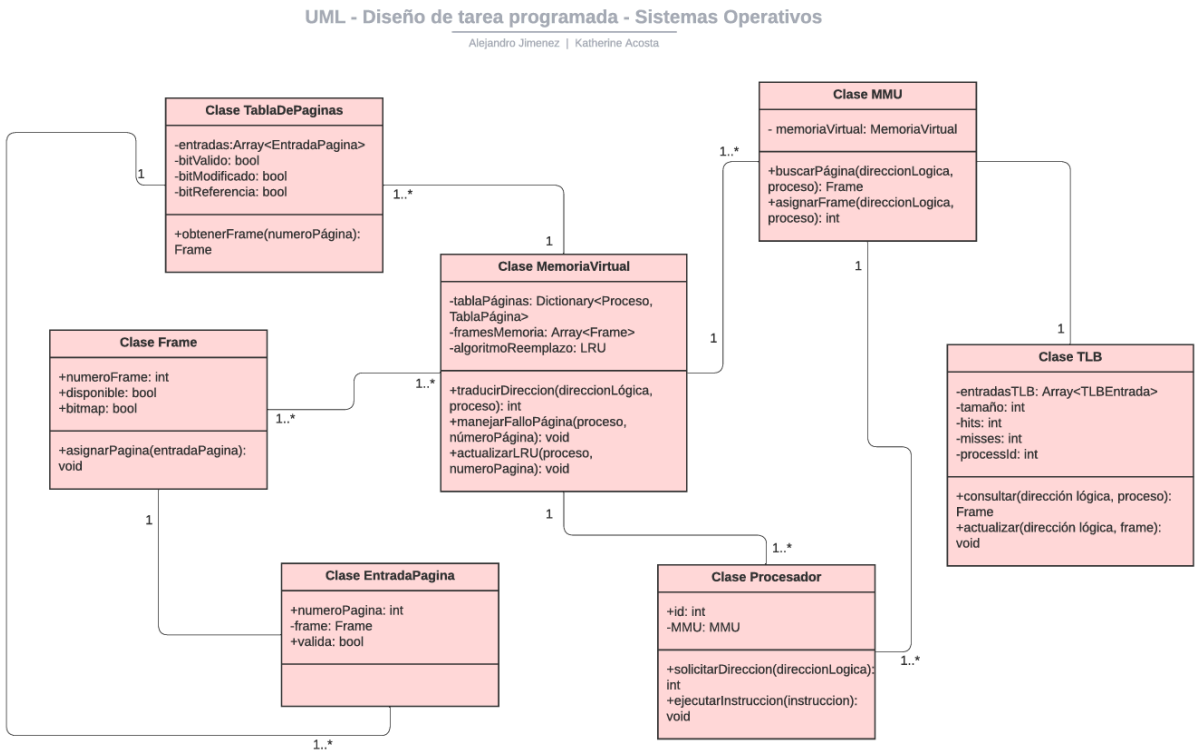

Figura 1: Diagrama de clases del Sistema de Memoria Virtual

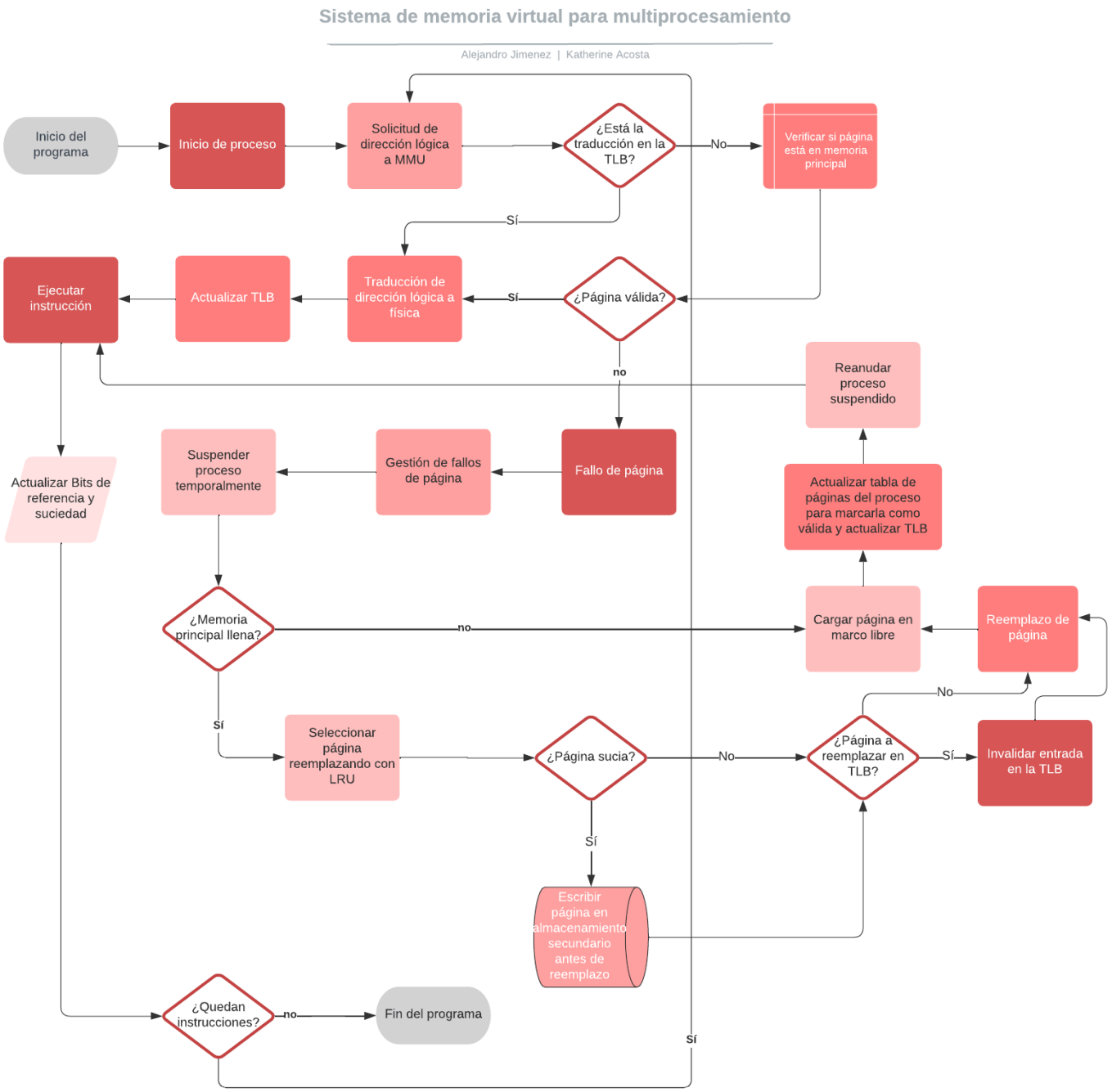

Figura 2: Diagrama de flujo del Sistema de Memoria Virtual

Procesos y concurrencia

Cada procesador se creará usando fork(), en un entorno de multiprocesamiento y los procesadores solicitan direcciones lógicas secuencialmente, cada solicitud es atendida por la MMU (unidad de gestión de memoria). En caso de que hayan fallos de página, la MMU se encargará de cargar la página requerida desde el funcionamiento secundario a un marco de memoria libre.

Para garantizar un acceso controlado a los recursos compartidos, se utilizarán mecanismos de sincronización como semáforos en el acceso a la tabla de páginas. De igual forma los marcos de la memoria física son un recurso compartido, por lo que el acceso a la estructura que los gestiona se controlará.

Por otro lado, el sistema va a permitir una cantidad máxima de 4 procesadores activos a la vez, para establecer un límite para optimizar el rendimiento. Este límite se implementará utilizando un contador de procesos activos que se actualizará de manera segura mediante semáforos.

También se plantea el posible uso de una cola de ejecución para gestionar las solicitudes de ejecución de los procesos. Cuando se alcanza el límite máximo de procesos activos, los procesos adicionales se colocarán en una cola de espera hasta que se libere un espacio. De esta forma, se asegura que el sistema no exceda la capacidad establecida.

Memoria asociativa

Para la TLB usa una memoria asociativa para acelerar el acceso a las traducciones de direcciones, para búsquedas rápidas por contenido y reduce la latencia. La TLB empleará el algoritmo LFU (Least Frequently Used) para reemplazar entradas. LFU mantendrá en la TLB las direcciones más utilizadas, para optimizar el rendimiento y reducir fallos.

Se registran hits y misses en la TLB, lo que permite evaluar la eficiencia del sistema, ya que una cantidad alta de hits indica un rendimiento óptimo mientras que los misses señalan que se requiere ajustar la configuración de la TLB para mejorar la eficiencia.

Manejo de instrucciones

El sistema soportará un subconjunto básico de instrucciones en lenguaje ensamblador siguiendo la sintaxis de x86_64, incluyendo operaciones aritméticas simples, saltos condicionales y no condicionales, así como ciclos. Entonces las instrucciones serían:

Instrucciones	**Explicación y justificación**	**Ejemplos**
ADD	Realiza una suma de dos valores en direcciones de memoria y almacena el resultado en una dirección específica. Es una operación fundamental para cálculos aritméticos.	ADD dest, src ADD 0x0010, 0x0020 (guarda el resultado en 0x0010)
SUB	Resta un valor de otro en direcciones de memoria y guarda el resultado en una dirección específica. Facilita operaciones y permite comparar indirectamente valores.	SUB dest, src SUB 0x0010, 0x0020 (almacena el resultado en 0x0010)
JMP	Realiza un salto ya sea condicional, o no condicional a una dirección y cambia el flujo de ejecución. Los saltos funcionan para implementar ciclos en el código.	JMP address JMP 0x0040

CMP	Compara dos valores en direcciones de memoria, establece banderas para el resultado. Permite realizar comparaciones para condicionar saltos.	CMP dest, src
		
		CMP 0x0010, 0x0020

LOOP	Decrementa un registro contador y realiza un salto si el valor del contador es mayor a cero. Facilita la implementación de ciclos.	LOOP address LOOP 0x0040

JUSTIFICACIÓN

La implementación de un sistema de memoria virtual permite que múltiples procesos compartan y administren la memoria sin interferencias, separando las direcciones lógicas y físicas. El sistema utilizará una unidad de gestión de memoria (MMU) y una tabla de páginas para realizar traducciones y gestionar el acceso de memoria. El algoritmo de reemplazo de páginas elegido es Least Recently Used (LRU) ya que ayuda a reducir los fallos de página al priorizar las páginas más recientemente usadas en la memoria principal.

El algoritmo de reemplazo de páginas se va a implementar de forma global para tener mayor flexibilidad en la asignación de memoria, y es que el reemplazo global permite a un proceso seleccionar un marco de cualquier conjunto de marcos en la memoria, independientemente del proceso al que esté asignado originalmente. Esto significa que un proceso puede tomar marcos de otros procesos si necesita más memoria, lo que permite que la distribución de los marcos se adapte dinámicamente a la demanda actual de cada proceso. También la implementación global permite un mayor rendimiento del sistema y throughput en comparación con la implementación individual (Silberschatz et al., 2018, p. 415).

Ahora bien, la combinación de un tamaño de memoria principal de 16 MB y un tamaño de página de 4 KB resulta en un total de 4096 páginas. Esta cantidad es manejable en términos de la administración de la tabla de páginas y permite observar los efectos de diferentes políticas de reemplazo de páginas sin que el sistema se sature rápidamente. Se establece en 4 KB el tamaño de las páginas y marcos para que cada página virtual coincida con un marco físico, facilite la traducción y minimice la fragmentación interna. También, en cuanto a la cantidad de procesadores activos a la vez, permite simular un entorno de varios procesos sin sobrecargar los recursos de memoria. Por otro lado, el sistema de memoria virtual está diseñado para gestionar fallos de página y asegurarse de que las páginas se carguen desde el almacenamiento secundario solo cuando sea necesario, optimizando el rendimiento.

Por estas razones, este diseño considera tanto la eficiencia de procesamiento como el control de concurrencia, garantizando que cada procesador activo gestione sus instrucciones en paralelo y solicite direcciones lógicas de manera independiente. Y, en cuanto a la escogencia de las instrucciones, se realizó de acuerdo a la necesidad de las operaciones aritméticas, comparaciones, saltos, y todo lo requerido para la ejecución de procesos y flujos controlados.

REFERENCIAS

GeeksforGeeks. (2024, octubre 3). *Page Fault Handling in Operating System*. GeeksforGeeks. https://www.geeksforgeeks.org/page-fault-handling-in-operating-system/

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating system concepts.