SISTEMAS OPERATIVOS – PARCIAL 3

GESTIÓN DE LA MEMORIA

MEMORIA

En un sistema monoprogramado, la memoria se divide en dos partes: una parte para el S.O. y otra parte para el programa actualmente en ejecución.

En un sistema multiprogramado, la parte de usuario se debe subdividir para poder acomodar varios procesos.

El S.O. es el encargado de la tarea de subdivisión y a esta tarea se le denomina **gestión de memoria**. Una gestión de memoria efectiva es vital en un sistema multiprogramado.

REQUISITOS DE LA GESTIÓN DE LA MEMORIA

Estos son los requisitos que la gestión de memoria debe satisfacer:

- Reubicación.
- Protección.
- Compartición.
- Organización lógica.
- Organización física.

REUBICACIÓN

En un sistema multiprogramado, la memoria principal disponible, se reparte generalmente entre varios procesos.

Normalmente, el programador no sabe qué programas residirán en memoria principal en tiempo de ejecución de su programa.

También es bueno poder intercambiar procesos en la memoria principal para maximizar la utilización del procesador.

Una vez que un programa se ha llevado a disco, sería bastante limitante tener que colocarlo en la misma región de memoria principal donde se hallaba anteriormente, cuando éste se trae de nuevo a memoria. Por el contrario, podría ser necesario **reubicar** el proceso a un área de memoria diferente.

PROTECCIÓN

Cada proceso debe protegerse contra interferencias no deseadas por parte de otros procesos, sean accidentales o intencionadas.

Por tanto, los programas de otros procesos no deben ser capaces de referenciar sin permiso posiciones de memoria de un proceso, tanto en modo lectura como en escritura.

Los requisitos de protección de memoria deben ser satisfechos por el procesador (hardware) en lugar de por el S.O. (software).

COMPARTICIÓN

Cualquier mecanismo de protección debe tener la flexibilidad de permitir a varios procesos acceder a la misma porción de memoria principal.

Esto significa que el sistema de gestión de memoria debe permitir el acceso controlado a áreas de memoria compartidas sin comprometer la protección esencial.

ORGANIZACIÓN LÓGICA

La memoria principal de un computador se organiza como un espacio de almacenamiento lineal o unidimensional, compuesto por una secuencia de bytes o palabras.

Esta organización es similar en la memoria secundaria pero no se corresponde con la forma en la cual los programas se construyen normalmente.

La mayoría de los programas se organiza en módulos, algunos de los cuales no se pueden modificar (lectura y ejecución) y algunos de los cuales contienen datos que se pueden modificar.

Si el S.O. y el hardware pueden tratar de forma efectiva los programas de usuarios y los datos en la forma de módulos de algún tipo, se pueden lograr grandes ventajas:

- Los módulos se pueden escribir y compilar independientemente.
- Con poca sobrecarga adicional, se puede proporcionar diferentes grados de protección a los módulos.
- Es posible introducir mecanismos por los cuales los módulos se pueden compartir entre los procesos.

La herramienta que más adecuadamente satisface este requisito es la **segmentación**.

ORGANIZACIÓN FÍSICA

La memoria del computador se organiza en al menos dos niveles: **memoria principal** y **memoria secundaria**.

La memoria principal proporciona un acceso rápido a un coste relativamente alto, además, es volátil (no proporciona almacenamiento permanente).

La memoria secundaria es más lenta, más barata y normalmente no es volátil.

Por ello, la memoria principal se utiliza para programas y datos que se encuentran en uso.

- La memoria principal disponible para un programa más sus datos podría ser insuficiente. En este caso, el programador debe utilizar una técnica conocida como **superposición** (*overlaying*), en la cual los programas y los datos se organizan de tal forma que se puede asignar la misma región de memoria a varios módulos, con un programa principal responsable para intercambiar los módulos entre disco y memoria según las necesidades. Este tipo de programación malgasta tiempo del programador.

- En un entorno multiprogramado, el programador no conoce en tiempo de codificación cuánto espacio estará disponible o dónde se localizará dicho espacio.

PARTICIONAMIENTO DE LA MEMORIA

La operación principal de la gestión de memoria es traer los procesos a la memoria principal para que el procesador los pueda ejecutar.

Esto requiere del uso de un esquema sofisticado llamado **memoria virtual**. Antes, veremos técnicas más sencillas que no usan memoria virtual:

PARTICIONAMIENTO FIJO

El sistema operativo ocupa una porción fija de la memoria principal. El resto de la memoria principal está disponible para múltiples procesos.

Este esquema más simple para gestionar la memoria disponible es repartirla en **regiones con límites fijos**.

Tamaños de partición

Hay dos alternativas para el particionamiento fijo:

- Particiones del **mismo tamaño**.
- Particiones de **distinto tamaño**.

En el caso de las particiones del mismo tamaño, cualquier proceso cuyo tamaño sea menor o igual que el tamaño de partición, puede cargarse en cualquier partición disponible. Si todas las particiones están llenas, y no hay ningún proceso en estado de 'Listo' o 'Ejecutando', el sistema operativo puede mandar un *swap* a un proceso de cualquiera de las particiones y cargar otro proceso.

Pero existen dos dificultades para el caso de particiones del mismo tamaño:

- Un programa podría ser demasiado grande y no caber en una partición. Esto haría que el programador diseñase el programa mediante overlays, de forma que solo se necesite una porción del programa en memoria principal en un momento determinado. Cuando se necesite un módulo que no esté presente, el programa debe cargar dicho módulo en la partición del programa, superponiéndolo (overlaying) a cualquier programa o datos que haya allí.
- La utilización de la memoria principal es ineficiente. Cualquier programa sin importar lo pequeño que sea, ocupa una partición entera. Al fenómeno en el que sobra espacio porque el bloque de datos cargado es menor que la partición se le llama **fragmentación interna**.

Ambos problemas se pueden mejorar, aunque no resolver, usando particiones de tamaño diferente.

Algoritmo de ubicación

Con el caso de particiones del mismo tamaño, la ubicación de procesos en memoria es trivial, en cuanto hay un hueco, se mete en él un proceso.

Con el caso de particiones de distinto tamaño, hay dos formas posibles de asignar los procesos a las particiones:

- Asignar a cada proceso a la partición más pequeña dentro de la cual cabe. Para ello, se necesita una cola de planificación para cada partición. Con esta técnica se minimiza la memoria malgastada dentro de una partición.
- El problema de esta anterior técnica es que algunas particiones quedarían vacías porque habría procesos esperando en colas de un tamaño más similar al suyo. Una solución es emplear una sola cola para todos los procesos. En el momento de cargar un proceso en memoria principal, se selecciona la partición más pequeña disponible que puede albergar dicho proceso.

Estas son las desventajas que presenta el particionamiento físico:

- El número de particiones especificadas en tiempo de generación del sistema limita el número de procesos activos (no suspendidos) del sistema.
- Como los tamaños de las particiones se establecen en tiempo de generación del sistema, los entornos donde el requisito de almacenamiento principal de todos los trabajos se conoce de antemano, son eficientes, pero en la mayoría de los casos, esto no es así, por lo que es una técnica ineficiente.

PARTICIONAMIENTO DINÁMICO

En particionamiento dinámico, las particiones son de longitud y número variable.

Cuando se lleva un proceso a memoria principal, se le asigna exactamente tanta memoria como requiera y no más.

Con este método, se puede llegar a situaciones en las que existan muchos huecos pequeños en la memoria. Según pasa el tiempo, la memoria se fragmenta cada vez más, y la utilización de la memoria decrementa. Este fenómeno se conoce como **fragmentación externa**.

Una técnica para eliminar la fragmentación externa es la **compactación**. De vez en cuando, el sistema operativo, desplaza los procesos en memoria, de forma que se encuentren contiguos y toda la memoria libre se encuentre en un solo bloque.

La desventaja de la compactación es que es un procedimiento que consume tiempo y malgasta tiempo de procesador.

La compactación requiere de la capacidad de **reubicación dinámica**. Es decir, debe ser posible mover un programa desde una región a otra en la memoria principal, sin invalidar las referencias de la memoria de cada programa.

Algoritmo de ubicación

A la hora de cargar o intercambiar un proceso a la memoria principal, y siempre que haya más de un bloque de memoria libre de suficiente tamaño, el sistema operativo debe decidir qué bloque libre asignar. Existen tres algoritmos para realizar esta tarea:

- **Mejor ajuste**: escoge el bloque más cercano en tamaño a la petición.
- **Primer ajuste**: comienza a analizar la memoria desde el principio y escoge el primer bloque disponible que sea suficientemente grande.
- **Siguiente ajuste**: comienza a analizar la memoria desde la última colocación y elige el siguiente bloque disponible que sea suficientemente grande.

El algoritmo de primer ajuste, es más sencillo y además, normalmente es el mejor y el más rápido

El algoritmo de siguiente ajuste, tiende a producir resultados ligeramente peores que el algoritmo de primer ajuste, además, mediante el uso de este algoritmo se suele necesitar más frecuentemente el uso de compactación.

El algoritmo de mejor ajuste suele ser normalmente el que peores resultados da, además, también necesita de compactaciones, con más frecuencia que el resto de los algoritmos.

Algoritmo de reemplazamiento

Imaginémonos el supuesto de que la memoria está ocupada por procesos, y todos se encuentran bloqueados.

Para evitar que el procesador se quede ocioso, el sistema operativo intercambiará alguno de los procesos entre la memoria principal y el disco.

El sistema operativo es el que debe escoger qué proceso reemplazar.

SISTEMA BUDDY

Tanto el particionamiento fijo como el dinámico, tienen desventajas. El particionamiento fijo limita el número de procesos activos y puede utilizar el espacio ineficientemente si existe un mal ajuste entre los tamaños de partición disponibles y los tamaños de los procesos.

Un esquema de particionamiento dinámico es más complejo de mantener e incluye la sobrecarga de la compactación.

En un sistema *buddy*, los bloques de memoria disponibles son de tamaño 2^K, con $L \leq K \leq U$ donde 2^L es el bloque de tamaño más pequeño asignado y 2^U es el bloque de mayor tamaño asignado.

El espacio completo disponible se trata como un único bloque de tamaño 2^U. Si se realiza una petición de tamaño S con $2^{U-1} \leq S \leq 2^U$, se asigna el bloque entero. En otro caso, el bloque se divide en dos bloques *buddy* iguales de tamaño 2^{U-1}.

REUBICACIÓN

En un esquema de particionamiento fijo, se espera que un proceso se asigne siempre a la misma partición. Cuando el proceso se carga por primera vez, todas las referencias de la memoria relativas del código, se reemplazan por direcciones de la memoria principal absolutas, determinadas por la dirección base del proceso cargado.

En el caso de particiones de igual tamaño y en el caso de una única cola de procesos para particiones de distinto tamaño, un proceso puede ocupar distintas particiones durante el transcurso de su ciclo de vida. Cuando la imagen de un proceso se crea por primera vez, se carga en la misma partición de memoria principal. Más adelante, el proceso podría llevarse a disco y al traerse de vuelta a la memoria principal, podría asignarse a una partición distinta de la última vez. Lo mismo ocurre con el caso del particionamiento dinámico. Por tanto, las ubicaciones (de las instrucciones y los datos) referenciadas por un proceso no son fijas. Cambiarán cada vez que un proceso se intercambia o se desplaza.

Para resolver este problema planteado, se realiza una distinción entre varios tipos de direcciones.

Una **dirección lógica** es una referencia a una ubicación de memoria independiente de la asignación actual de datos a la memoria. Se debe llevar a cabo una traducción a una dirección física antes de que se alcance el acceso a la memoria.

Una **dirección relativa** es un ejemplo particular de dirección lógica, en el que la dirección se expresa como una ubicación relativa a algún punto conocido, normalmente un valor en un registro del procesador.

Una **dirección física** o dirección absoluta es una ubicación real de la memoria principal.

Normalmente todas las referencias de memoria de los procesos cargados son relativas al origen del programa. Por tanto, se necesita un mecanismo hardware para traducir las direcciones relativas a direcciones físicas de la memoria principal, en tiempo de ejecución de la instrucción que contiene dicha referencia.

PAGINACIÓN

Tanto las particiones de tamaño fijo como variable, son ineficientes en el uso de la memoria. Los primeros provocan fragmentación interna, los segundos, fragmentación externa.

Supongamos ahora que la memoria principal se divide en porciones de tamaño fijo relativamente pequeñas, y que cada proceso también se divide en porciones pequeñas del mismo tamaño fijo. A dichas porciones del proceso, conocidas como **páginas**, se les asigna porciones disponibles de memoria, conocidas como **marcos**.

En este sistema no existe la fragmentación externa y la interna se da tan solo en la última página de cada proceso.

Si no hay suficientes marcos contiguos disponibles como para cargar un nuevo proceso, no significa que el sistema operativo no pueda cargar el nuevo proceso. Para ello, utilizaremos de nuevo el concepto de dirección lógica.

El sistema operativo mantiene una **tabla de páginas** por cada proceso. La tabla de páginas muestra la ubicación muestra la ubicación del marco por cada página del proceso. Dentro del programa, cada dirección lógica está formada por un número de página y un desplazamiento dentro de la página.

En la paginación, el procesador debe conocer cómo acceder a la tabla de páginas del proceso actual. Presentado como una dirección lógica (número de página, desplazamiento), el procesador utiliza la tabla de páginas para producir una dirección física (número de marco, desplazamiento).

Podemos ver que la paginación simple, es similar al particionamiento fijo. Las diferencias son que, con la paginación, las particiones son bastante pequeñas, un programa podría ocupar más de una partición y dichas particiones no necesitan ser contiguas.

Si el tamaño de página y, por tanto, también el tamaño de marco, son potencias de 2, podemos ver que la dirección relativa y la dirección lógica son lo mismo.

Las consecuencias de utilizar un tamaño de página que es potencia de 2 son dobles:

- El esquema de direccionamiento lógico es transparente al programador, al ensamblador y al montador. Cada dirección lógica (número de página, desplazamiento) de un programa es idéntica a su dirección relativa (se define como referencia al origen del programa).

- Es relativamente sencillo implementar una función que ejecute el hardware para llevar a cabo dinámicamente la traducción de direcciones en tiempo de ejecución. Podemos verlo a continuación:

Consideramos una dirección de 'n + m' bits, donde los n bits de la izquierda corresponden al número de página, y los m bits de la derecha corresponden al desplazamiento dentro de esa página. Se necesita llevar a cabo los siguientes pasos para la traducción de direcciones:

- Extraer el número de página como los n bits de la izquierda de la dirección lógica.
- Utilizar el número de página como un índice a tabla de páginas del proceso para encontrar el número de marco, k.
- La dirección física inicial del marco es $k \cdot 2^m$, y la dirección física del byte referenciado es dicho número más el desplazamiento.

SEGMENTACIÓN

Un programa de usuario se puede subdividir utilizando segmentación, en la cual el programa y sus datos asociados se dividen en un número de **segmentos**.

No es necesario que todos los programas sean de la misma longitud, aunque hay una longitud máxima de segmento.

Una dirección lógica está compuesta de dos partes: número de segmento y un desplazamiento.

Debido al uso de segmentos de distinto tamaño, la segmentación es similar al particionamiento dinámico. La diferencia con el particionamiento dinámico es que con la segmentación un programa podría ocupar más de una partición, y estas particiones no tienen que ser contiguas necesariamente. La segmentación elimina la fragmentación interna, pero sufre de fragmentación externa. Sin embargo, debido a que el proceso se divide en varias piezas más pequeñas, la fragmentación externa debería ser menor.

Mientras que la paginación es invisible al programador, la segmentación es normalmente visible y se proporciona como una utilidad para organizar programas y datos. Normalmente, el programador o compilador asignará programas y datos a diferentes segmentos. Para los propósitos de la programación modular, los programas o datos se pueden dividir posteriormente en múltiples segmentos.

Otra consecuencia de utilizar segmentos de distinto tamaño es que no existe una relación simple entre direcciones lógicas y direcciones físicas.

Un esquema de segmentación sencillo hace uso de una tabla de segmentos por cada proceso y una lista de bloques libres de memoria principal.

Cada entrada de la tabla de segmentos tendría que proporcionar la dirección inicial de la memoria principal del correspondiente segmento y la longitud del segmento, para asegurar que no se utilizan direcciones no válidas. Cuando un proceso entra en ejecución, la dirección de su tabla de segmentos se carga en un registro especial utilizado por el hardware de gestión de la memoria.

Si tenemos una dirección de n + m bits, con n bits para el número de segmento y m bits para el desplazamiento. Necesitamos llevar a cabo los siguientes pasos para la traducción de direcciones:

- Extraer el número de segmento como los n bits de la izquierda de la dirección lógica.
- Utilizar el número de segmento como un índice a la tabla de segmentos del proceso para encontrar la dirección física inicial del segmento.
- Comparar el desplazamiento, expresado como los m bits de la derecha, y la longitud del segmento. Si el desplazamiento es mayor o igual que la longitud, la dirección no es válida.
- La dirección física deseada es la suma de la dirección física inicial y el desplazamiento.

En resumen, con la segmentación simple, un proceso se divide en un conjunto de segmentos que no tienen que ser del mismo tamaño. Cuando un proceso se trae a memoria, todos sus segmentos se cargan en regiones de memoria disponibles, y se crea la tabla de segmentos.