Gestión de Memoria

Carrera: Licenciatura en Sistemas de Información

Materia: Sistemas Operativos

Año: 2019

Profesores: Aguiar Osvaldo

Rapallini Ulises

Alumnos: Curtoni Juan

Gonzalez Exequiel

Soria Lucas

1. En un **sistema monoprogramado**, la memoria se divide en dos partes: una parte para el sistema operativo (monitor residente, núcleo) y una parte para el programa actualmente en ejecución.
   En un **sistema multiprogramado**, la parte de «usuario» de la memoria se debe subdividir posteriormente para acomodar múltiples procesos.

2. El sistema operativo es el encargado de la tarea de subdivisión, y a esta tarea se le denomina **gestión de la memoria**.

3. La gestión de la memoria debe satisfacer cinco requisitos :
   - **Reubicación**: Una vez que un programa se ha llevado al 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 la memoria. Por el contrario, podría ser necesario reubicar el proceso a un área de memoria diferente. Por tanto, no se puede conocer de forma anticipada dónde se va a colocar un programa y se debe permitir que los programas se puedan mover en la memoria principal, debido al intercambio o swap. El sistema operativo necesitará conocer la ubicación de la información de control, del proceso y de la pila de ejecución, así como el punto de entrada que utilizará el proceso para iniciar la ejecución. Las instrucciones de salto contienen una dirección para referenciar la instrucción que se va a ejecutar a continuación. Las instrucciones de referencia de los datos contienen la dirección de byte o palabra de datos referenciados. De alguna forma, el hardware del procesador y el software del sistema operativo deben poder traducir las referencias de memoria encontradas en el código del programa en direcciones de memoria físicas, que reflejan la ubicación actual del programa en la memoria principal.

   - **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 escritura. Por un lado, lograr los requisitos de la reubicación incrementa la dificultad de satisfacer los requisitos de protección. Todas las referencias de memoria generadas por un proceso deben comprobarse en tiempo de ejecución para poder asegurar que se refieren sólo al espacio de memoria asignado a dicho proceso. Afortunadamente, se verá que los mecanismos que dan soporte a la reasignación también dan soporte al requisito de protección.

   - **Compartición**: Cualquier mecanismo de protección debe tener la flexibilidad de permitir a varios procesos acceder a la misma porción de memoria principal. Por ejemplo, si varios programas están ejecutando el mismo programa, es ventajoso permitir que cada proceso pueda acceder a la misma copia del programa en lugar de tener su propia copia separada.

- **Organización Lógica:** Si el sistema operativo y el hardware del computador pueden tratar de forma efectiva los programas de usuarios y los datos en la forma de módulos de algún tipo, entonces se pueden lograr varias ventajas:
  - Los módulos se pueden escribir y compilar independientemente, con todas las referencias de un módulo desde otro resueltas por el sistema en tiempo de ejecución.
  - Con una sobrecarga adicional modesta, se puede proporcionar diferentes grados de protección a los módulos (sólo lectura, sólo ejecución).
  - Es posible introducir mecanismos por los cuales los módulos se pueden compartir entre los procesos. La ventaja de proporcionar compartición a nivel de módulo es que se corresponde con la forma en la que el usuario ve el problema, y por tanto es fácil para éste especificar la compartición deseada.

- **Organización Física:** La memoria del computador se organiza en al menos dos niveles, conocidos como memoria principal y memoria secundaria. La memoria principal proporciona acceso rápido a un coste relativamente alto. Adicionalmente, la memoria principal es volátil; es decir, no proporciona almacenamiento permanente. La memoria secundaria es más lenta y más barata que la memoria principal y normalmente no es volátil. Por tanto, la memoria secundaria de larga capacidad puede proporcionar almacenamiento para programas y datos a largo plazo, mientras que una memoria principal más pequeña contiene programas y datos actualmente en uso.

4. **Particionamiento fijo:** La memoria principal se divide en particiones estáticas en tiempo de generación del sistema. Un proceso se puede cargar en una partición con igual o superior tamaño. Virtudes: Sencilla de implementar, poca sobrecarga para el sistema operativo. Defectos: Uso ineficiente de la memoria, debido a la fragmentación interna; debe fijarse el número máximo de procesos activos. Existen dos alternativas para el particionamiento fijo: Una posibilidad consiste en hacer uso de particiones del mismo tamaño. En este caso, cualquier proceso cuyo tamaño es 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 Listo o ejecutando, el sistema operativo puede mandar a swap un proceso de cualquiera de las particiones y cargar otro proceso, de forma que el procesador tenga trabajo que realizar.

5. Existen dos dificultades con el uso de las particiones fijas del mismo tamaño:
   - Un programa podría ser demasiado grande para caber en una partición. En este caso, el programador debe diseñar el programa con el uso de overlays, de forma que sólo se necesite una porción del programa en memoria principal en un momento determinado. Cuando se necesita un módulo que no está presente, el programa de usuario 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 extremadamente ineficiente. Cualquier programa, sin importar lo pequeño que sea, ocupa una partición

entera. En el ejemplo, podría haber un programa cuya longitud es menor que 2 Mbytes; ocuparía una partición de 8 Mbytes cuando se lleva a la memoria. Este fenómeno, en el cual hay espacio interno malgastado, se conoce con el nombre de fragmentación interna.

6. **Algoritmo de ubicación en particiones fijas:** la ubicación de los procesos en memoria es trivial. En cuanto haya una partición disponible, un proceso se carga en dicha partición. Debido a que todas las particiones son del mismo tamaño, no importa qué partición se utiliza. Si todas las particiones se encuentran ocupadas por procesos que no están listos para ejecutar, entonces uno de dichos procesos debe llevarse a disco para dejar espacio para un nuevo proceso.

   En particiones de diferente tamaño: Hay dos formas posibles de asignar los procesos a las particiones. La forma más sencilla consiste en asignar cada proceso a la partición más pequeña dentro de la cual cabe. En este caso, se necesita una cola de planificación para cada partición, que mantenga procesos en disco destinados a dicha partición. Aunque esta técnica parece óptima desde el punto de vista de una partición individual, no es óptima desde el punto de vista del sistema completo. Por ejemplo, se considera un caso en el que no haya procesos con un tamaño entre 12 y 16M en un determinado instante de tiempo. En este caso, la partición de 16M quedará sin utilizarse, incluso aunque se puede asignar dicha partición a algunos procesos más pequeños. Por tanto, una técnica óptima sería emplear una única cola para todos los procesos. En el momento de cargar un proceso en la memoria principal, se selecciona la partición más pequeña disponible que puede albergar dicho proceso. Si todas las particiones están ocupadas, se debe llevar a cabo una decisión para enviar a *swap* a algún proceso. Tiene preferencia a la hora de ser expulsado a disco el proceso que ocupe la partición más pequeña que pueda albergar al proceso entrante.

7. Con **particionamiento dinámico**, las particiones son de longitud y número variable. Cuando se lleva un proceso a la memoria principal, se le asigna exactamente tanta memoria como requiera y no más. El método comienza correctamente, pero finalmente lleva a una situación en la cual existen muchos huecos pequeños en la memoria. A medida que pasa el tiempo, la memoria se fragmenta cada vez más y la utilización de la memoria se decrementa. Este fenómeno se conoce como fragmentación externa, indicando que la memoria que es externa a todas las particiones se fragmenta de forma incremental. 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 de este modo toda la memoria libre se encontrará unida en un bloque. La desventaja de la compactación es el hecho de que se trata de un procedimiento que consume tiempo y malgasta tiempo de procesador.

8. 9. **Algoritmo de ubicación**: se utiliza para decidir cómo asignar la memoria a los procesos (cómo eliminar los huecos). 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. Tres algoritmos de colocación que pueden considerarse son **mejor-ajuste** (best-fit), **primer-ajuste**(first-fit) y **siguiente-ajuste** (next-fit). Todos, por supuesto, están limitados a escoger entre los bloques libres de la memoria principal que son iguales o más grandes que el proceso que va a llevarse a la memoria.

- **Mejor-ajuste**: escoge el bloque más cercano en tamaño a la petición. A pesar de su nombre, su comportamiento normalmente es el peor. Debido a que el algoritmo busca el bloque más pequeño que satisfaga la petición, garantiza que el fragmento que quede sea lo más pequeño posible, el resultado es que la memoria principal se queda rápidamente con bloques demasiado pequeños para satisfacer las peticiones de asignación de la memoria. Por tanto, la compactación de la memoria se debe realizar más frecuentemente.
- **Primer-ajuste**: comienza a analizar la memoria desde el principio y escoge el primer bloque disponible que sea suficientemente grande. No es sólo el más sencillo, sino que normalmente es también el mejor y más rápido. Puede dejar el final del espacio de almacenamiento con pequeñas particiones libres que necesitan buscarse en cada paso del primer ajuste siguiente.
- **Siguiente-ajuste**: comienza a analizar la memoria desde la última colocación y elige el siguiente bloque disponible que sea suficientemente grande. Tiende a producir resultados ligeramente peores que el primer-ajuste. llevará más frecuentemente a una asignación de un bloque libre al final de la memoria. El resultado es que el bloque más grande de memoria libre, que normalmente aparece al final del espacio de la memoria, se divide rápidamente en pequeños fragmentos. Por tanto, en el caso del algoritmo siguiente-ajuste se puede requerir más frecuentemente la compactación.

Cuál de estas técnicas es mejor depende de la secuencia exacta de intercambio de procesos y del tamaño de dichos procesos.

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

11. Tanto las particiones de tamaño fijo como variable son ineficientes en el uso de la memoria; los primeros provocan fragmentación interna, los últimos fragmentación externa. Supóngase, sin embargo, que la memoria principal se divide en porciones de tamaño fijo relativamente pequeños, 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, o marcos de páginas. No existe fragmentación externa.
    Con la paginación simple, la memoria principal se divide en muchos marcos pequeños de igual tamaño. Cada proceso se divide en páginas de igual tamaño; los procesos más pequeños requieren menos páginas, los procesos mayores requieren más. Cuando un proceso se trae a la memoria, todas sus páginas se cargan en los marcos disponibles, y

se establece una tabla de páginas. Esta técnica resuelve muchos de los problemas inherentes en el particionamiento.

12. 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 se requiere que todos los programas sean de la misma longitud, aunque hay una longitud máxima de segmento. Como en el caso de la paginación, una dirección lógica utilizando segmentación está compuesta por dos partes, en este caso un número de segmento y un desplazamiento. Debido al uso de segmentos de distinto tamaño, la segmentación es similar al particionamiento dinámico. En la ausencia de un esquema de overlays o el uso de la memoria virtual, se necesitaría que todos los segmentos de un programa se cargaran en la memoria para su ejecución. La diferencia, comparada con el particionamiento dinámico, es que con la segmentación un programa podría ocupar más de una partición, y estas particiones no necesitan ser contiguas.
    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.

13. Referencias:

	Primer ajuste
	Mejor ajuste
	Siguiente ajuste
	Peor ajuste