Resumen Administración de Memoria

Tomás Felipe Melli

Noviembre 2024

Índice

1 Introducción

Estuvimos hablando sobre procesos y políticas de scheduling pero *qué pasa con la memoria en todo este caos ?*
Para poder llevar a cabo todo lo visto, hay que establecer *pautas de organización de memoria*, el subsistema del SO responsable es el **Handler de Memoria**. Este debe ser capaz de : **manejar el espacio libre/ocupado**, **asignar y liberar memoria**, **controlar el swapping** (esto es, el proceso de mover datos entre la memoria principal (RAM) y un espacio de almacenamiento secundario, típicamente un disco duro o un SSD, llamado **archivo de intercambio (swap file) o partición de intercambio (swap partition)**).
Cuando un proceso está bloqueado y ponemos a ejecutar otro *qué pasa con la memoria ?* Una forma es swapping. Es lento (ya que requiere grabar un proceso y leer otro) ... los dejamos en memoria entonces ... pero supongamos que en cierto momento sí los tenemos que bajar a disco, y cuando les corresponda correr de nuevo, el espacio de memoria que les toque no sea el mismo. *Por qué esto deriva en un problema ?* Esto es un problema porque **cada proceso tiene direcciones de memoria asociadas para guardar datos y variables**. Entonces, si se va de ese lugar y después vuelve a uno diferente, se pierde la consistencia de los datos.

2 Protección

La idea es resguardar la información de un proceso para que otros no puedan acceder sin los permisos adecuados. *La solución es* **Segmentación**. Es decir, espacios de memoria propios de cada proceso. Con un registro de segmento sabemos a cuál hacen referencia las direcciones. Facilita la protección y permite que cada segmento crezca sin cambiar el programa y facilita el trabajo de bibliotecas compartidas (cada una en su segmento).
Existen problemas de fragmentación y swapping y es por ello que se combina con **paginación**

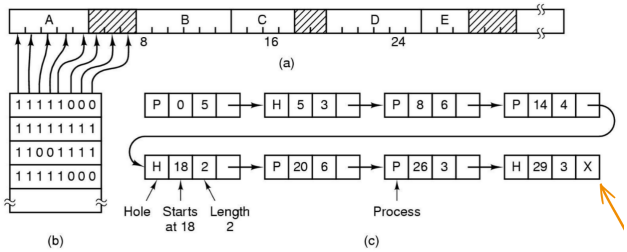
3 Fragmentación

La **Fragmentación** es la forma en la que se distribuye el espacio de memoria disponible a medida que se asigna y se libera durante la ejecución de procesos.
Puede resultar en un problema por ejemplo, para un proceso que necesita un bloque contigüo de memoria, tal vez tiene el espacio pero no contigüo. Una posible solución sería compactar (REUBICAR) pero resulta muy costoso.

3.1 Existen dos tipos

- **Fragmentación Interna** : cuando hay muchos bloques libres pero son pequeños y están dispersos.
- **Fragmentación Externa** : ocurre cuando hay mucho desperdicio de espacio dentro de los propios bloques.

3.2 Representación



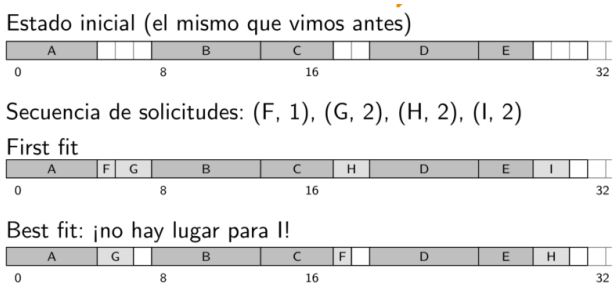
Tenemos el **BitMap** que es una tabla de bits en donde cada bit representa el estado de un bloque de memoria. Este bitmap ayuda a *rastrear qué bloques de memoria están ocupados y cuáles no.* Encontrar

bloques *requiere de una barrida lineal*. No se usa mucho.
Tenemos también una **Lista Enlazada** donde *cada nodo representa a un proceso o bloque libre* (en este caso figura tamaño y límites). La liberación toma tiempo constante (manipular punteros más chequear si los nodos previos eran espacio libre) también asignar ...

3.3 Asignación

Existen diferentes algortimos para asignar memoria.

1. **First-Fit** : el primer bloque donde entre, asigno. Es rápido pero tiende a fragmentar la memoria, partiendo bloques grandes.
2. **Best-Fit** : donde entre más justo. Es más lento y no es mejor. Llena la memoria de bloquecitos inservibles. Busca el bloque más pequeño de al menos el tamaño solicitado. Existe una variación de este que se llama **Quick-Fit**, en el que hay una lista de bloques libres de los tamaños más frecuentemente solicitados.
3. **Buddy-System** : organiza la memoria en bloques de tamaño variabe mediante la división y combinación de bloques de memoria que son potencias de 2. Veamos un ejemplo : si se necesita un bloque de 16KB y sólo hay un bloque de 32KB disponible, el bloque de 32KB se divide en 2 de 16KB. Uno de los bloques se asigna al proceso, y el otro, se mantiene libre.



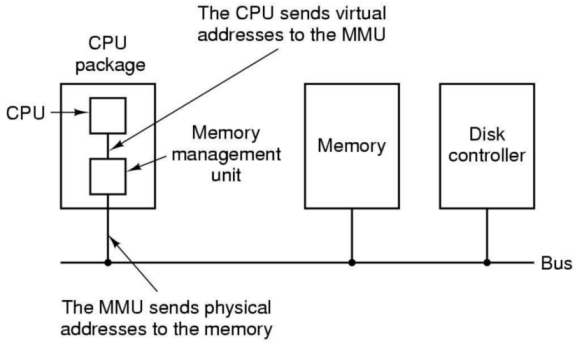
4 Qué pasa si hacemos un Fork() ?

AL crear un nuevo proceso, *copiamos las páginas del proceso padre ?*
La estrategia más común es **Copy-On-Write** que consiste en **utilizar las mismas páginas** hasta que alguno de los dos escribe. En ese momento, se duplican y cada uno tiene su copia independiente. La *Administración de la Memoria* se puede abordar desde distintos puntos :

- **Sistema Operativo** : memoria del kernel y de los procesos
- **Lenguaje de Programación** de 3 maneras diferentes :
  - *Explícita* : Bibliotecas (malloc() en libc) o Primitivas (new C++).
  - *Implícita* : no controlada por el programa (Haskell) o parcialmente (Swift).
  - *Híbrida* : creación explícita y liberación implícita (JAVA)
  - *Integrada* : Android, iOS

5 Por qué la Reubicación podría presentar a problemas ?

La reubicación es **la acción de ajustar las direcciones de memoria internas del programa para que se correspondan con su ubicación real en la memoria**. Dicho esto, si no logramos relacional al proceso con las direcciones que utiliza, no encontraría ni sus datos ni su código.
Qué solución tiene este problema ?
Swapping + Virtualización del espacio de direcciones. En otras palabras, **Memoria Virtual**. Si recordamos de orga 2, esto es labor del HW, la *MMU (Memory Management Unit)*

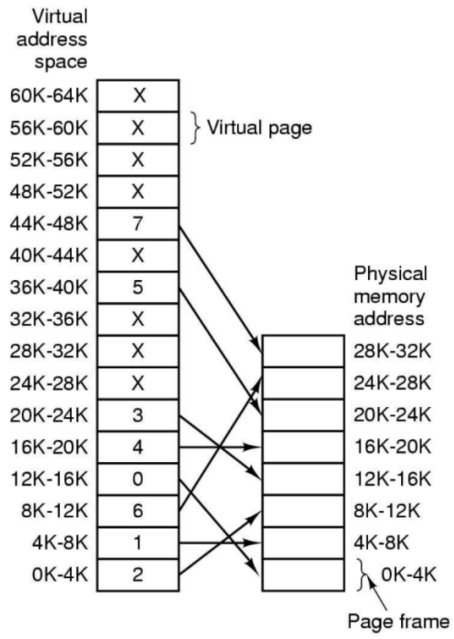


5.1 Qué pasaría si no tuviésemos memoria virtual ?

El espacio de direcciones se reduce al tamaño de la memoria física. Por tanto, obtener una celda es poner la dire en el bus y listo.
Y con memoria virtual ?
EL espacio de direcciones es la suma del tamaño de la memoria física + SWAP. Es decir, el tamaño de la RAM + el espacio de disco. La MMU se encarga de la traducción de $VIRTUAL \rightarrow FÍSICA$ para obtener el dato.

5.2 Si hay Paginación ?

El espacio de Memoria Virtual est divido en **páginas** y el espacio de Memoria Física en **Page Frames**. Miremos este ejemplo:



Existen páginas especiales
Read-Only No-Swappeables por razones de seguridad.
Qué pasa cuando el sistema operativo captura un Page Fault ?
Cuando una página no está en memoria, la MMU manda un Page Fault y entonces el sistema operativo debe decidir **qué página desalojar** de memoria y subir la que corresponde. Cuál sacar genera un problema de rendimiento.
El **Thrashing** es una consecuencia indeseable que ocurre cuando la memoria no alcanza y hay mucha

competencia entre los procesos para usarla. El SO se la pasa cambiando páginas de memoria a disco ida y vuelta.
Más info ...
Como el **Page Fault** es una interrupción, el Kernel la captura. Se guarda el IP y otros registros en la pila. Llama a la Rutina de atención. Averigüa a qué dirección virtual se estaba buscando (esta queda usualmente en un registro). Se chequean , la validez de la dire y los permisos. Si no puede, se mata al proceso en UNIX y se envía la senñal **segmentation violation**. Caso contrario, se selecciona un *page frame* libre y sino se libera mediante una política de desaolojo. Si la página estaba *dirty*, se baja a disco. Es decir, el proceso del kernel que maneja I/O debe ser suspendido, generando así un cambio de contexto y permitiendo que otros se ejecuten. La página estará BUSY para evitar su uso. Cuando se notifica al SO que se bajó correctamente la página a disco, comienza otra operación I/O, pero ahora para cargar la que se buscaba subir, se deja ejecutar a otros procesos. Una vez cargada, se emite una interrupción que indica que hay que actualizar la *page table*. Se recomienza la instrucción que generó *page fault* (recordar que tenemos el IP en el STACK entre otras cosas). Se devuelve el control al proceso y cuando el scheduler le de tiempo y reintente la instrucción la página, ya la va a tener.