



Sistemas Operativos 2

Unidad 1: Administración de memoria

Conceptos generales de la administración de memoria

René Ornelis
Primer semestre 2023

Contenido

Introducción	3
Objetivo general de la administración de memoria	3
Jerarquía de dispositivos de memoria	3
Funciones de la administración de memoria.....	4
Relocalización	4
Compartición y protección	4
Asignación de memoria.....	4

ADMINISTRACIÓN DE MEMORIA

"los programas se expanden con el fin de llenar la memoria"

Introducción

En todos los sistemas operativos y la gran mayoría de aplicaciones, es necesario el manejo de memoria dinámica, por lo que se hace necesario entender, cómo el sistema operativo realiza dicho manejo, para tener mejores criterios respecto al funcionamiento de las estructuras de datos. Además, todos los procesos a ejecutarse necesitan primero subirse a memoria.

El sistema operativo es responsable de administrar la **memoria real** de la computadora y presentar a los procesos un espacio de **memoria lógico** en el cual pueda trabajar con la garantía de que otro proceso no va a interferir en su memoria ni que dicho proceso interferirá en la memoria de otro.

Objetivo general de la administración de memoria

Lograr un balance óptimo entre la usabilidad y el rendimiento del sistema

Jerarquía de dispositivos de memoria

En la terminología de computación cuando se habla de memoria no solo se refiere a la RAM, sino también a una serie de dispositivos que almacenan información y que debe administrar el sistema operativo. Para realizar esta administración, primero se clasifican los diferentes dispositivos de memoria, según sus características en una jerarquía tal como se muestra en la Figura 1. Esta jerarquía consta de los siguientes niveles:

1. **Memoria interna o volátil (inboard):** Es la memoria principal y se caracteriza por ser el dispositivo de memoria de mayor velocidad y usualmente los más caro, que está integrado en la tarjeta madre. Entre estos dispositivos están los registros del procesador, la memoria caché y la memoria RAM.
2. **Memoria externa o persistente (offboard):** Es el tipo de dispositivo de almacenamiento que tiene la característica de la persistencia y no está integrado en la tarjeta madre. Son más lentos que la memoria interna, pero más baratos. Ejemplo de estos dispositivos son los discos duros y discos ópticos.

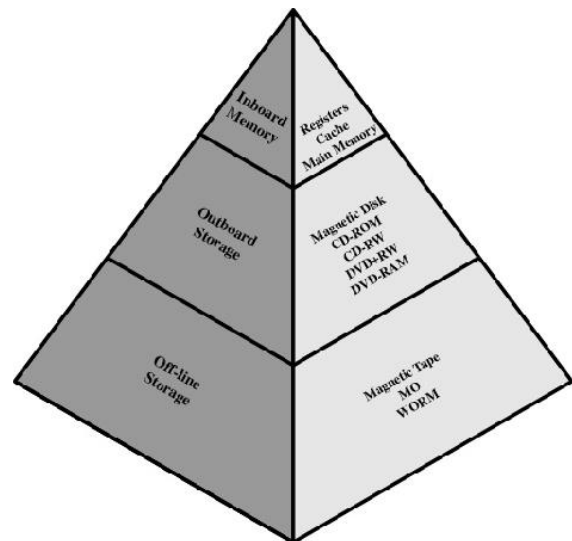


Figura 1: Jerarquía de dispositivos de memoria

3. **Memoria fuera de línea o extraíble (offline):** Está conformado por dispositivos como las cintas magnéticas, discos magnético-ópticos y discos de una escritura (Write Once, Read Many), que se caracterizan por su alta capacidad de almacenamiento (menor precio), pero una mayor lentitud en las operaciones de lectura y escritura.

Funciones de la administración de memoria

Para lograr el objetivo general de balance del rendimiento entre usabilidad y rendimiento, el subsistema de administración de memoria del sistema operativo debe cumplir con tres funciones:

- Relocalización
- protección y compartición
- asignación

Relocalización

Es la habilidad del sistema operativo para colocar la memoria lógica de un proceso en diferentes partes de la memoria real, según las necesidades del sistema operativo, sin que el proceso se vea afectado ni el programador deba tomar medidas al respecto.

Compartición y protección

Cada proceso necesita la garantía de que ningún otro proceso va a afectar su memoria y de la misma forma, dicho proceso no puede afectar la memoria de otros. El sistema operativo debe proporcionar esta garantía a través de la función de **protección**.

Adicionalmente, los procesos tienen diferentes necesidades de colaboración con otros procesos por lo que necesitan definir áreas comunes de código (código reentrante) y de datos (áreas críticas). El sistema operativo también debe proveer los mecanismos para que los procesos puedan compartir explícitamente su memoria, según sus necesidades. Esta debe ser la única forma en que un proceso pueda acceder la memoria de otro proceso.

Como veremos más adelante, los mecanismos para proveer esta función han evolucionado desde los registros límite, hasta la paginación y segmentación.

Asignación de memoria

Todo proceso, luego de iniciarse, requerirá de inmediato memoria dinámica. De hecho, el sistema operativo asignará memoria al proceso como parte del proceso de carga en memoria: espacio para su código y para sus datos estáticos.

También, eventualmente, los procesos indicarán al sistema operativo que ya no utilizarán la memoria solicitada, por lo que el sistema operativo dispondrá de esta para otros procesos o para el mismo sistema operativo. En último caso, al finalizar un proceso, el sistema operativo tomará toda la memoria asignada a dicho proceso como una liberación.

Estas operaciones de asignación y liberación de memoria las debe proveer el sistema operativo de forma que garantice los principios de relocalización y protección.



Sistemas Operativos 2

Unidad 1: Administración de memoria

Paginación y segmentación

René Ornelis
Primer semestre 2023

Contenido

Evolución de los mecanismos de memoria	4
Carga estática.....	4
Carga dinámica: registros de relocalización/límite	4
Paginación.....	5
Enlace estático y dinámico.....	7
Segmentación.....	9
Sistemas híbridos	11
Paginación multinivel	11

Índice de figuras

Figura 1: Traducción de memoria lógica a real con registros de relocalización.....	4
Figura 2: Paginación	6
Figura 3: Ejemplo de paginación	6
Figura 4: Programas con librerías	7
Figura 5: Compilación con enlace estático	7
Figura 6: Compilación con enlace dinámico.....	8
Figura 7: Tabla de páginas para librerías dinámicas.....	9
Figura 8: Segmentación	10
Figura 9: Memoria con fragmentación externa.....	10
Figura 10: Memoria compactada	10
Figura 11: Sistema híbrido de segmentación/paginación.....	11
Figura 12: Paginación multinivel.....	11

Paginación y segmentación

La paginación y la segmentación son los mecanismos actuales de los sistemas operativos por medio de los cuales se cumple con las funciones de relocalización, protección y compartición.

Evolución de los mecanismos de memoria

En el desarrollo de los sistemas operativos, el hardware necesario para que este pudiera cumplir con las funciones de relocalización ha evolucionado en las siguientes fases:

- Carga estática
- Carga dinámica: Registros de relocalización/límite
- Paginación

Carga estática

Inicialmente, los sistemas operativos no contaban con mecanismos de protección de memoria por ser sistemas monousuarios y no existía ningún hardware especial para esto. En estos sistemas se utilizó la **carga estática** de programas, lo cual consiste en que el programador debía conocer con antelación la posición de memoria en la que el programa sería cargado a memoria para poder hacer *referencias absolutas* de memoria, tanto para sus datos (lecturas de memoria) como para su código (saltos y saltos condicionales). Un ejemplo tardío de esto es el sistema DOS, el cual tenía los ejecutables de tipo .COM que acompañaban al sistema como programas especiales para línea de comandos. Estos programas .COM solo funcionaban con la versión específica de DOS ya que estaban programados para iniciar en una determinada posición de memoria según la memoria del sistema operativo. Al cambiar de versión de sistema operativo, cambiaba la memoria utilizada por el sistema operativo, por lo que la posición inicial de memoria disponible para los programas era diferentes y los programas de otra versión ya no funcionaban.

Carga dinámica: registros de relocalización/límite

Esto cambió con la carga dinámica de programas, la cual ya no necesitaba que estos trabajaran en una dirección específica de memoria porque utilizaban referencias relativas de memoria, las cuales estaban basadas en el concepto general de registro de relocalización (o registro base) y registro límite, los cuales funcionaban en el Memory Management Unit (MMU) tal como se muestra en la Figura 1.

Con este mecanismo, al acceder una dirección lógica (P_i) el MMU verifica:

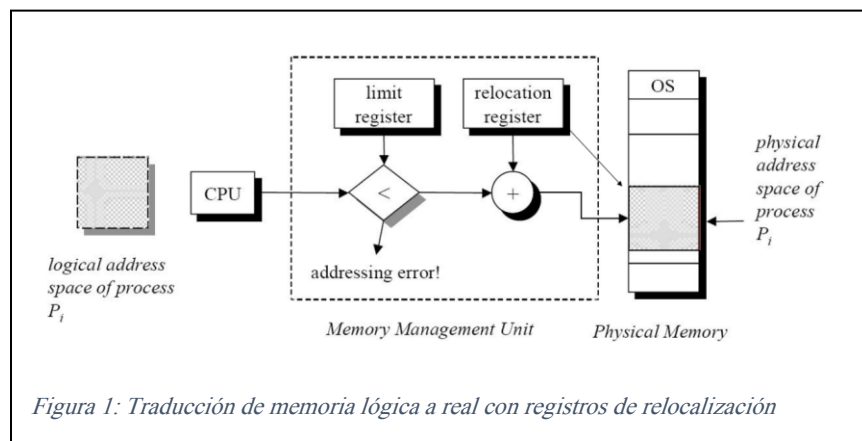


Figura 1: Traducción de memoria lógica a real con registros de relocalización

1. Que la dirección esté dentro del rango de memoria asignado al proceso, es decir que el proceso no esté accediendo a memoria más allá de su área de memoria asignada.
2. Si la dirección de memoria está dentro del rango de memoria asignada al proceso, entonces la dirección lógica se suma al contenido del registro relocalizador para formar la dirección física a acceder.

Tener en cuenta que el registro límite y del registro relocalizador son parte del MMU y su contenido será determinado por los valores del proceso en ejecución, es decir, al momento de que a un proceso se le

otorga el uso del procesador, junto con todos sus registros y estado en general, debe ser restaurado el valor de estos registros.

Por ejemplo, una implementación en los primeros procesadores de Intel, la relocalización se realizaba a través de varios registros base, como el DX, CS, SS, etc. de forma que las instrucciones

`mov ax, [FF]`

`jump [5]`

equivalen a:

`mov ax, DX:[FF]`

`jump CS:[5]`

De esta forma el programador no se preocupa de las direcciones absolutas de memoria, sino solo de las direcciones relativas.

Paginación

La paginación es el mecanismo actual por medio del cual los sistemas operativos y los procesadores cumplen con la función de relocalización, de protección y compartición de las páginas de memoria. La paginación consiste en dividir la memoria en partes iguales, cada una de ellas denominada página o marcos de página. Cada una de estas páginas es la unidad mínima de asignación de memoria para cada proceso y cualquier petición que realice un proceso para adquirir nueva memoria, implica que el sistema le asignará tantas páginas como sea necesario para cumplir con el requerimiento. Dado que los requerimientos no siempre van a coincidir con un múltiplo del tamaño de las páginas es posible que dentro de alguna página no se aproveche el espacio al máximo. Por ejemplo: si un sistema tiene una un tamaño de página de 10kb y un proceso pide un bloque de memoria de 45Kb, el sistema le asignará un total de 5 páginas lo que implica que en la última página habrá un desperdicio de 5Kb, el cual el proceso ni ningún otro proceso del sistema podrán utilizarlo. A este fenómeno se le conoce como **fragmentación interna**.

Fragmentación interna: Es desperdicio de memoria que existe dentro de un bloque de K, que se entrega a un proceso que solicitó un bloque de tamaño N y $K < N$. Existe un desperdicio de memoria $N-K$, *dentro* de bloque.

Tal como se puede apreciar en la Figura 2, el mecanismo de paginación consiste de:

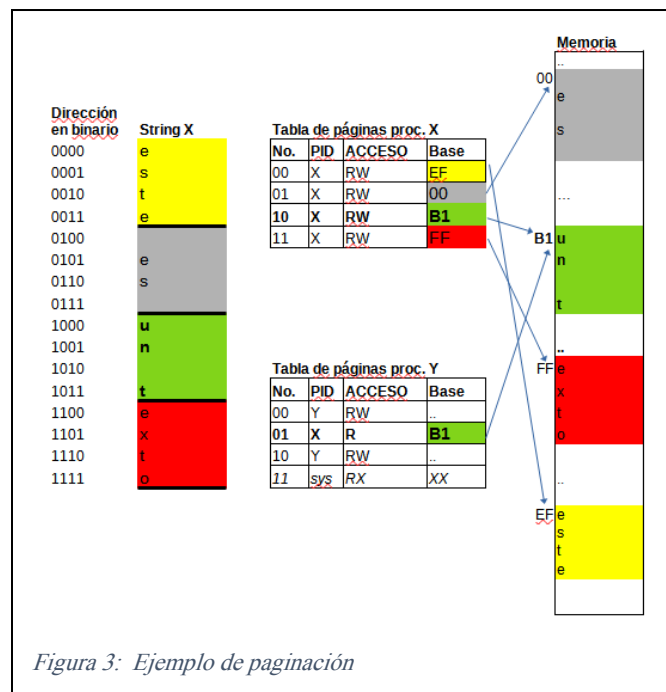
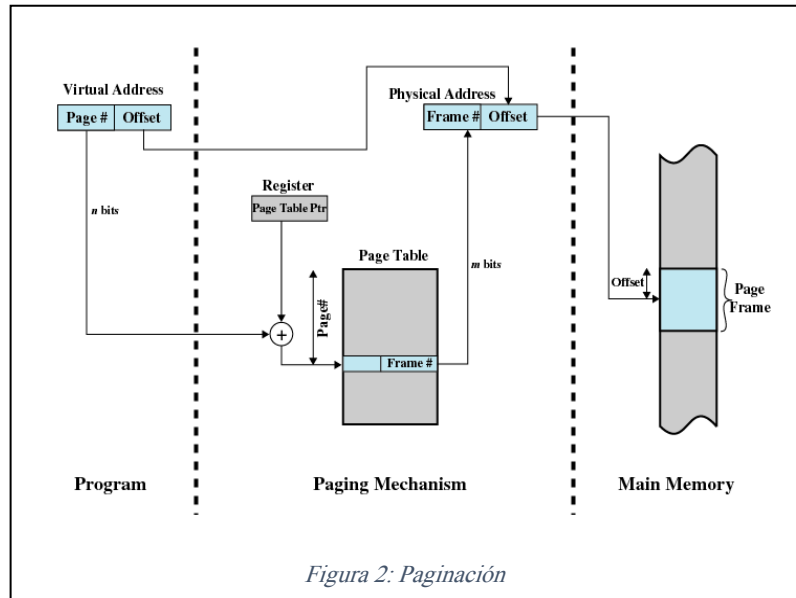
1. Direcciones lógicas o virtuales, las cuales están divididas en dos partes: el número de página (*Page #*) y el desplazamiento (*Offset*) dentro de esa página
2. Tabla de páginas (*Page Table*) que contiene la información de cada página, principalmente la dirección base de esa página (*Frame #*) o la dirección en memoria real donde está ubicada la página solicitada.
3. Registro apuntador a la tabla de páginas (*Page Table Pointer*) que contiene la dirección de la tabla de páginas del proceso en ejecución. En el momento de un cambio de contexto (cuando a un proceso se le asigna el uso del procesador) parte de la información que se tiene que restaurar del proceso es su Page Table Pointer, junto con todos registros de estado del proceso, como el Instruction Pointer, Data Segment, Stack Segment, etc.

El proceso de **traducción de direcciones** para trasladar una dirección lógica (o virtual) a una dirección física, involucra los siguientes pasos:

1. En el momento que el proceso acceda a esta dirección lógica el MMU o la unidad de memoria administración de memoria, toma la parte de número de página (*Page #*) la busca en una tabla de páginas (*Page Table*).
2. Si la página existe, se toma la dirección base (*Frame #*) y se le agrega la parte del desplazamiento (*Offset*) de la dirección lógica para formar la dirección real.

Al iniciar el sistema operativo se puede configurar el MMU para determinar cuál va a ser nuestro tamaño de página, lo cual debe elegir cuidadosamente para mantener un

balance adecuado entre el tamaño de la página y el tamaño de la tabla de páginas. Si tenemos páginas muy grandes la fragmentación interna va a ser mayor y por lo tanto va a haber mayor desperdicio memoria, por el otro lado si tenemos páginas muy pequeñas el número de páginas crecerá, lo cual implica que nuestras tablas de páginas también crecerán, lo cual tendrá un impacto en la búsqueda de páginas al momento de acceder.



Consideremos el ejemplo de la Figura 3. Supongamos un sistema que tiene capacidad de direccionamiento de cuatro bits y un tamaño de palabra de 8 bits. Usaremos en la dirección lógica 2 bits para el tamaño de página y 2 bits para el desplazamiento; esto significa que este sistema teórico tiene un máximo de 16 direcciones. Si el proceso necesita un string con contenido “*este es un texto*”, la representación en memoria de la tabla de páginas va a tener un total de cuatro páginas, en la gráfica numeradas de 00 a la 11 (0 a 3 en decimal). La información de la tabla de páginas incluye el *process id (PID)*, el cual es la identificación del proceso propietario de esa página y los accesos que el proceso en ejecución tienen sobre dicha página. En este caso vemos que todas las entradas de la tabla de páginas tienen por propietario al proceso X, con acceso de lectura y escritura.

También en la tabla de páginas del proceso X está la dirección base de cada uno de los marcos de páginas que corresponden a la

posición de memoria de cada dirección lógica.

De esta forma, lo que para el proceso es una secuencia de direcciones que va de las 00 a las 1111 (16 en decimal), en memoria física cada una de las páginas puede estar en cualquier orden de forma independiente. Para aplicar la relocación, cuando el sistema mueva una de estas páginas, a cualquier otra posición de memoria, lo único que tiene que hacer es actualizar la entrada correspondiente en la tabla de páginas, sin que esto afecte el direccionamiento lógico del proceso. Es decir, el proceso sigue viendo las mismas direcciones lógicas sin ningún cambio, aunque la página física cambie de lugar.

Supongamos también que existe un proceso Y, el cual debe utilizar parte de la memoria del proceso X. Para esto, el proceso X debe compartir explícitamente una página para que otro proceso pueda accederla. En este caso ejemplificamos que se va a compartir la página 10 del proceso X, por lo que en la tabla de páginas de proceso Y, se agregará una entrada en la tabla de páginas para acceder a la página del proceso X. Notemos dos cosas:

1. La dirección lógica con la que el proceso Y va a acceder a la página compartida del proceso X no es la misma. Son dos entradas independientes, porque la asignación en la tabla de páginas depende del estado en que esta estaba antes de dicha asignación.
2. La entrada de la tabla de páginas del proceso Y que corresponde a la página compartida, indica que pertenece al proceso X y que el proceso Y tiene solo acceso de lectura a dicha página.

Enlace estático y dinámico

Otro ejemplo en el cual podemos apreciar el uso de la paginación es el caso del enlace dinámico de las aplicaciones. Al momento de generar ejecutables, existen 2 formas de generar un ejecutable en cualquier compilador: uno es **enlace estático** y otro es **enlace dinámico**. El enlace estático se refiere a que un ejecutable contiene en sí mismo todas las librerías y todo el código necesario por el cual puedes ejecutarse por completo. La forma de distribución es simplemente entregar el archivo ejecutable (programa.exe) y el usuario lo puede correr directamente. En el caso del enlace dinámico hay un ejecutable y una serie de archivos o librerías las cuales se enlazan dinámicamente a los programas conforme los van necesitando.

Supongamos el caso de dos programas que utilizan un conjunto común de librerías, tal como se muestra en la Figura 4, tenemos dos programas X y Y, los cuales utilizan un conjunto de librerías en común (librerías A, B y C), las cuales tienen diferentes funcionalidades y se pueden integrar a diferentes programas. En este ejemplo, vemos que el programa X utiliza la librería A y la librería C, por su lado el programa Y utiliza la librería A y la librería B.

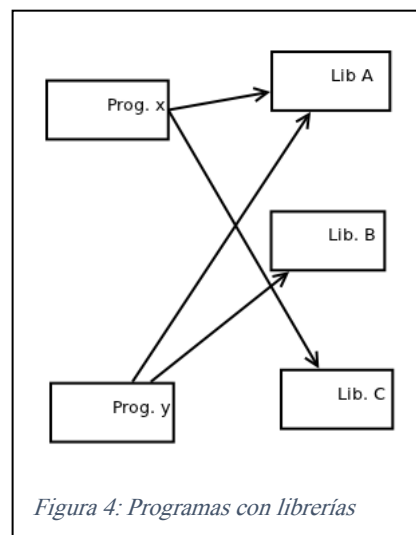


Figura 4: Programas con librerías

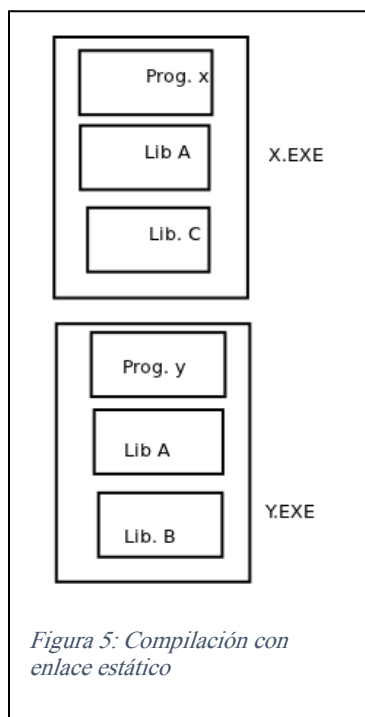


Figura 5: Compilación con enlace estático

En la Figura 5 podemos ver cómo estarían conformados los programas X y Y si estos fueran compilados de forma estática. Vemos que el programa X contiene dentro de su archivo (X.exe) el contenido específico del programa X, el contenido de la librería A y el contenido de la librería C. Por su parte el programa Y tiene dentro de su archivo (Y.exe) el contenido específico del programa Y, la librería A y la librería B.

Si el usuario corre ambos programas simultáneamente la librería A está cargada 2 veces en memoria: una vez dentro del programa X y otra dentro del programa Y, resultando en un desperdicio de memoria. Claro que en un sistema se utilizan muchos más que dos programas y tres librerías, por lo que el enlace estático implica un desperdicio de memoria y tiempo de carga de programas.

Para contrarrestar estas desventajas, se inventó el enlace dinámico que consiste en el uso compartido de librerías a través de varios programas como se ve en la Figura 6, en la cual podemos apreciar que se producen los siguientes archivos:

- Programa **X.exe**, el cual contiene el código específico y único del programa x
- Programa **Y.exe**, el cual contiene sólo el código específico del programa Y.

- **A.dll, B.dll y C.dll** los cuales contienen cada uno el código de las librerías correspondientes.

Al momento de ejecutarse el programa X, invoca al sistema operativo para indicar que necesita la librería A y, suponiendo que no hay ningún otro programa cargado con anterioridad, el sistema operativo determinará que la librería A no está en memoria y por lo tanto lo cargará y le compartirá a X la dirección de memoria de la librería A. Luego se repite el proceso para la librería C, después de los cual el programa X se puede ejecutar en su totalidad, ya que está en memoria todo lo que necesita.

Por su lado, el programa Y utiliza la librería A y la librería B. Al momento de cargar el programa Y, indicará al sistema operativo que necesita la librería A (A.dll) y el sistema determinará que este archivo ya está cargada en memoria, por lo que ya no la cargará de nuevo sino que sólo compartirá con el proceso Y la dirección de memoria para que pueda ejecutarla. Luego el programa Y indicará que necesita la librería B, se cargará en memoria y se le compartirá la dirección al programa Y.

De esta forma optimizamos la memoria, porque no habrá doble (o múltiple) carga del código de una librería utilizada por varios programas.

Una de las desventajas de utilizar múltiples librerías es de que se complica el mantenimiento de estas, dado que se tiene que mantener una compatibilidad “hacia atrás” de forma que las nuevas versiones deben ser compatibles, en interfaz y en funcionamiento, con las versiones anteriores para que programas que utilizan dicha librería no se vean afectados por el cambio de librería.

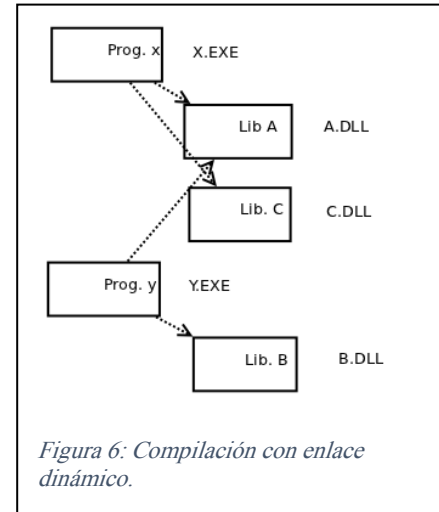
Los compiladores proporcionan diferentes facilidades para los programadores que permiten realizar el enlace dinámico. Esto varía según el compilador, pero en general se puede resumir en el siguiente código:

```
/* se define la función y se indica en qué librería dinámica se encuentra */
int a1 (int a, X c) ; extern "a.dll" ;
....
/* se utiliza la función definida como si fuera una función definida localmente */
int z = a1 (0, null) ;
....
```

Esta facilidad, común en los compiladores, nos permiten ahorrarnos la interfaz directa con el sistema operativo, para lo cual transforman el código anterior en el siguiente:

```
/* variable tipo función para definición de parámetros */
typedef int tf(int a, X c) ;
....
/* se solicita cargar la librería */
int handle = loadLibrary ("a.dll") ;

/* se obtiene la dirección de la función dentro de la librería */
tf a1 = getProc (handle, "a1") ;
...
/* se utiliza la función */
int z = a1(0, null) ;
....
/* se solicita descargar la función cuando ya no se utilizará */
unloadLibrary (handle) ;
..
```



Aquí, el programa X tiene que cargar manualmente la librería **a.dll** y obtener la dirección de memoria de la función que nos interesa dentro de esa librería (“a1.dll”), y después de asignar esa dirección podemos iniciar el uso de dicha función. Posteriormente hay que indicar al sistema operativo que descargue esa librería porque no la volveremos a utilizar.

Descargar una librería (*unloadLibrary*) no significa necesariamente que el sistema operativo la va a eliminar de memoria, sino que tiene en cuenta el conteo de procesos que la están usando. En nuestro ejemplo: cuando el proceso X solicita a la librería A, el sistema la carga a memoria y pone el contador de esta librería en 1; cuando el programa Y solicita la misma librería, el sistema se da cuenta de que ya está cargada y sólo incrementa su contador a 2. De forma inversa, si el programa X decide que ya no va a utilizar la librería A, puede dar la instrucción de descargarla, pero el sistema no lo va a eliminar de memoria, sino solo disminuye su contador a 1, lo que indica que aún hay un proceso que la está utilizando. Cuando el programa Y finalice e indique que se descargue la librería A, entonces el contador llegará a cero, por lo cual el sistema ya sabe que puede eliminarla de memoria sin afectar ningún proceso.

En la Figura 7, se muestra cómo podrían configurarse las tablas de páginas de cada uno de los procesos compartiendo las páginas que contienen las librerías. Notemos que en la tabla de páginas de cada proceso existen entradas, tanto para el código propio, como entradas para las páginas donde se encuentran las librerías. En el caso del proceso X, vemos que el marco de página **M10** contiene la página específica de este, que no la está compartiendo con ningún proceso. También contiene el marco de página **M1** que contiene el código de la librería A y el marco de página **M20** que contiene el código de la librería C. Vemos en estas dos últimas entradas que el propietario de la **M1** y **M20** es el sistema, y el proceso X tiene acceso de lectura y ejecución en estas páginas.

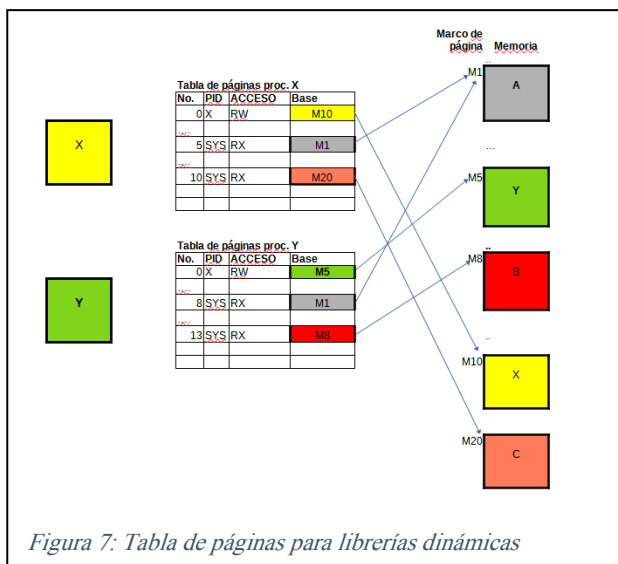


Figura 7: Tabla de páginas para librerías dinámicas

De la misma forma, el proceso Y tiene la página **M5** la cual es propia de este proceso y también tiene entradas para las librerías A y B que corresponden a las direcciones **M1** y **M8**, respectivamente. Notar las mismas características: son propiedad del sistema y el proceso Y solo tiene acceso para leer y ejecutar.

Segmentación

Vemos que la paginación cumple con funciones de relocalización, compartición y protección, pero adolece del problema de la **fragmentación interna**. Adicionalmente, los programadores ven la memoria como un conjunto de bloques semánticamente relacionados, es decir se ve el bloque de datos, el bloque de código, el bloque para la pila, etcétera.

La segmentación surge como otro esquema de organización de la memoria, para superar los problemas de la paginación. Consiste básicamente del mismo proceso que la paginación, pero con una diferencia importante: el tamaño. Vimos que la paginación trabaja con páginas del mismo tamaño, lo cual nos lleva a la fragmentación interna, por lo cual se definió en la segmentación como bloques de memoria semánticamente relacionados de tamaño variable. Es decir, cada segmento es de diferente tamaño, lo cual nos elimina el problema de la fragmentación interna, ya que se entrega a cada proceso la cantidad exacta de memoria que está requiriendo.

El mecanismo por el cual funcionan la segmentación se muestra en la Figura 8. Aquí podemos ver que se tiene:

1. Una dirección lógica o dirección virtual, compuesta de un número de segmento (*Seg #*) y desplazamiento (*Offset*)

2. La tabla de segmentos, similar a la tabla de páginas que básicamente almacena la misma información, como la dirección base del segmento. También incluye el tamaño del segmento.
3. El registro apuntador a la tabla de segmentos (*Segment Table Pointer*), el cual contiene la dirección de la tabla de segmentos del proceso, similar al Page Table Pointer de la paginación.

El mecanismo de traducción de direcciones en la segmentación involucra los siguientes pasos:

1. El MMU toma el número del segmento (*Seg #*), de la dirección lógica, y lo busca en la tabla de segmentos.
2. Con la información del tamaño en la tabla de segmentos el MMU compara que el desplazamiento de la dirección lógica (*Offset*) no sobrepase el tamaño de dicho segmento.
3. Si todo es correcto, a la dirección base de la tabla de segmentos (*Base*) se le agrega el desplazamiento, lo cual forma la dirección física de la memoria a acceder.

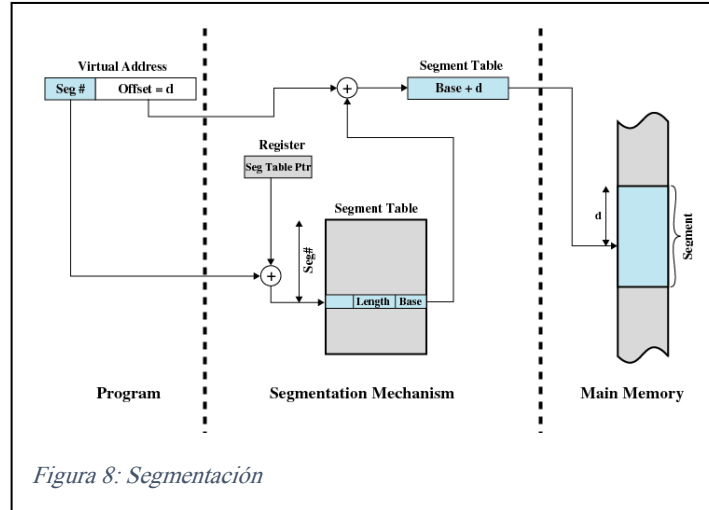


Figura 8: Segmentación

Como vemos, el mecanismo de paginación y segmentación funcionan con los mismos principios de la paginación, con la única diferencia del control del tamaño de los bloques de memoria. Sin embargo, aunque la segmentación resuelve el problema de la fragmentación interna, introduce el problema de la **fragmentación externa**.

Fragmentación externa: Es cuando en la memoria existen pequeños bloques de memoria libre, intercalados entre los bloques utilizados por procesos, de modo que, aunque la memoria disponible total sea de tamaño N (la suma de todos los bloques libres), el sistema no puede atender requerimientos de dicho tamaño, y aún tamaños menores, dado que está distribuido a lo largo de toda la memoria.

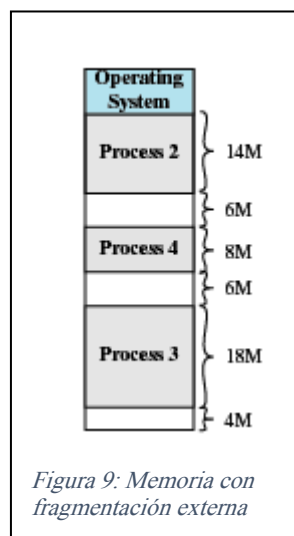


Figura 9: Memoria con fragmentación externa

Tal como se ilustra en la Figura 9, tenemos la memoria de los procesos 2, 3 y 4 los cuales ocupan bloques de tamaño 14Mb, 18Mb y 18Mb respectivamente. Asimismo, vemos que hay tres bloques libres, dos de 6Mb y uno de 4Mb, que hace un total de 16 Mb libres. Si un proceso requiere un bloque de 7Mb, no podrá ser atendido por el sistema dado que no hay ningún bloque libre que pueda satisfacer este requerimiento.

Para solucionarlo se debe recurrir a la compactación de memoria la cual consiste en mover todos los bloques ocupados hacia un extremo de la memoria y dejar del otro extremo, un solo bloque libre con el total de memoria disponible tal como se muestra en la Figura 10.

Aunque la compactación soluciona el problema de la fragmentación externa, hay que tomar en cuenta que mientras se realiza este procedimiento los procesos del usuario y del

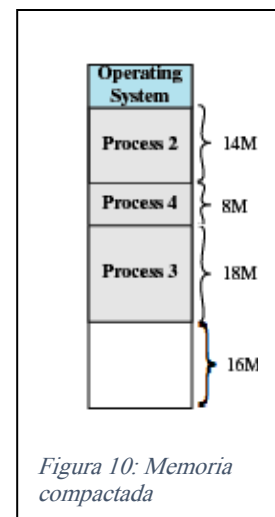


Figura 10: Memoria compactada

sistema deben detenerse hasta finalizar relocalización de todas las páginas. Esto implica un impacto directo al rendimiento general del sistema.

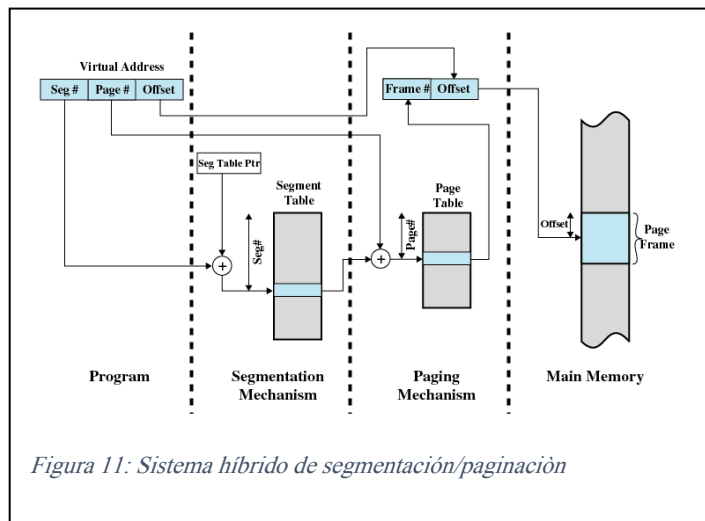
Sistemas híbridos

Como veremos en el estudio de la memoria virtual, adicional a la fragmentación externa, los sistemas de segmentación introducen la complejidad de la variabilidad de los tamaños de los segmentos que complican

el control y la eficiencia de estos. Por los tanto, surgen los sistemas híbridos, los cuales intentan tomar lo mejor de dos mundos y proveen un primer nivel de segmentos los cuales y en un segundo nivel paginación. De esta forma el programador solicita segmentos de memoria y transparentemente estos se dividen en páginas lo cual facilitará el manejo posterior de la memoria virtual. Aunque posiblemente haya fragmentación interna, esta se limita a la última página de cada segmento.

Notemos en la Figura 11 que la dirección lógica (*Virtual Address*) ahora se conforma de tres partes:

1. El número de segmento (*Seg #*)
2. El número de página (*Page #*)
3. El desplazamiento (*Offset*).

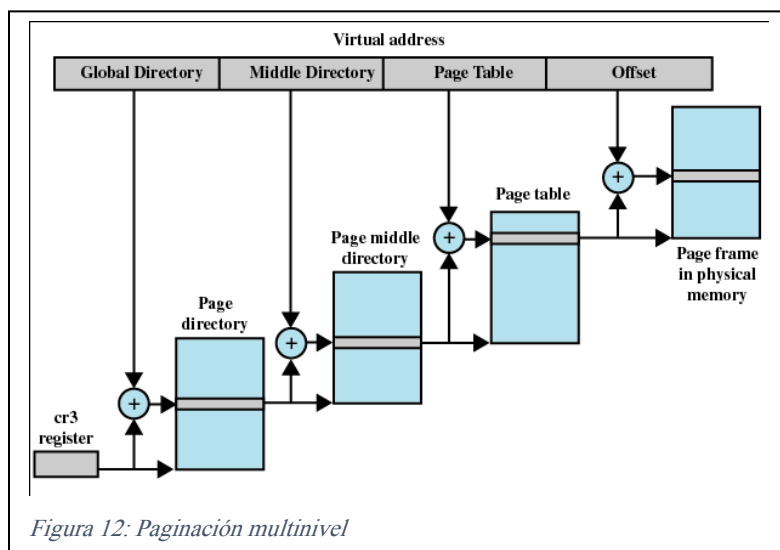


El proceso de traducción de direcciones es similar a los esquemas de un nivel:

1. Se toma el número de segmento de la dirección lógica se busca en la tabla de segmentos. En esta configuración, la tabla de segmentos ya no tiene la dirección de memoria del segmento, sino la dirección a la tabla de páginas de ese segmento.
2. En la tabla de página del segmento, se busca la segunda parte de la dirección lógica, el número de página. Esto nos dará la dirección base de esta página
3. Se suma el desplazamiento, que es la tercera parte de la dirección lógica, a la dirección base que se obtuvo de la tabla de páginas, lo cual nos da la dirección física.

Paginación multinivel

Como se puede en la Figura 12, el concepto de 2 niveles que se utiliza en la segmentación/ paginación, se puede expandir al concepto de paginación multinivel, la cual consiste de una jerarquía de tablas de páginas y al final un desplazamiento dentro de la página, lo cual puede ser más eficiente si los procesos no utilizan todo el rango disponibles de páginas y solo utilizan una fracción de las mismas, ya que la creación de las tablas de páginas se hace dinámicamente, según se van necesitando las páginas.





Sistemas Operativos 2

Unidad 1: Administración de memoria

Asignación y liberación

René Ornelis
Primer semestre 2023

Contenido

Operaciones sobre el manejo de la memoria	3
Particiones fijas	5
Particiones variables	6
Métodos de Ajuste secuencial (Sequential fit)	7
Proceso general	7
Asignación de un bloque de tamaño n	7
Efecto del ordenamiento	9
Apuntador Vagabundo	9
Liberación	9
Primer ajuste	10
Mejor ajuste	10
Peor ajuste	10
Óptimo ajuste	11
Sistemas compañero	11
Compactación	15
Colección de basura	16

Asignación y liberación de memoria

Operaciones sobre el manejo de la memoria

El sistema operativo, al realizar el manejo de memoria, realiza las siguientes operaciones sobre la misma:

- **Asignación de memoria:** entregar a los procesos bloques de memoria solicitados con distintos tamaños
- **Liberación de memoria:** recibir de los procesos los bloques de memoria que éstos ya no utilizarán y dejarlos como parte de la memoria libre que podrá ser utilizada por otros procesos.
- **Compactación:** cuando se produce *fragmentación externa*, todos los bloques de memoria se mueven hacia un extremo de la memoria, de modo que queda sólo un bloque libre al final
- **Recolección de basura:** en algunos lenguajes de programación como java, no existen operaciones de liberación de memoria, por lo que al momento de existir una solicitud de asignación de memoria y no existe disponible, el sistema deberá liberar los bloques que no están siendo utilizados, para liberar memoria y atender las solicitudes pendientes.

Estas operaciones pueden ser realizadas a nivel de sistema operativo o a nivel de aplicación. Para requerimientos de bloques pequeños, si se invoca al sistema operativo, provoca *fragmentación interna*. Por lo que los compiladores normalmente generan un segmento de HEAP para manejar internamente (a nivel de librerías de usuario) el uso de la memoria dinámica. Esta es la operación del sistema operativo en el cual un proceso solicita un bloque de memoria de tamaño K y el sistema le debe proporcionar la dirección de un bloque de memoria para uso del proceso. Dicho bloque debe tener al menos el tamaño solicitado (K).

Para realizar esta operación, anteriormente se utilizaban métodos que utilizaban bloques de *tamaño uniforme*, los cuales consistían en dividir la memoria en un número de bloques, todos del mismo tamaño. Cada requerimiento de memoria, se realizaba en término de número de bloques, en vez de tamaño en bytes (Kbytes, etc).

Estas políticas, aunque son simples de implementar, rápidamente cayeron en desuso, debido a que la programación de los procesos se complicaba, ya que, si se tienen bloques de tamaño N, y se necesita un bloque de memoria 2N, el programador debía tener en mente que al solicitar dos bloques, en muchos casos no había garantía que los bloques entregados sean continuos, y aunque así fuera, siempre existía el problema de tener que calcular los requerimientos en términos de bloques y no de bytes.

Más determinante que las complicaciones de programación, fue la *fragmentación interna*

FRAGMENTACIÓN INTERNA: Es desperdicio de memoria que existe dentro de un bloque de tamaño K , que se entrega a un proceso que solicitó un bloque de tamaño N y $K < N$. Existe un desperdicio de memoria *dentro* de bloque de tamaño $N-K$

Actualmente, los sistemas utilizan métodos de bloques de *tamaño variable*, los cuales dividen la memoria en bloques cuyo tamaño se define conforme los procesos requieren y liberan memoria, garantizando que cada bloque atendido, sea uno sólo de memoria contigua.

Estos métodos resultan mucho más eficientes en términos de utilización de la memoria, sin embargo, varían en complejidad, rendimiento, manejo de la fragmentación interna y *fragmentación externa*.

FRAGMENTACIÓN EXTERNA: Es cuando en la memoria existen pequeños bloques de memoria libre, intercalados entre los bloques utilizados por procesos, de modo que, aunque la memoria total sea de tamaño N (la suma de todos los bloques libres), el sistema no puede atender requerimientos de dicho tamaño, y aún tamaños menores, dado que está distribuido a lo largo de toda la memoria.

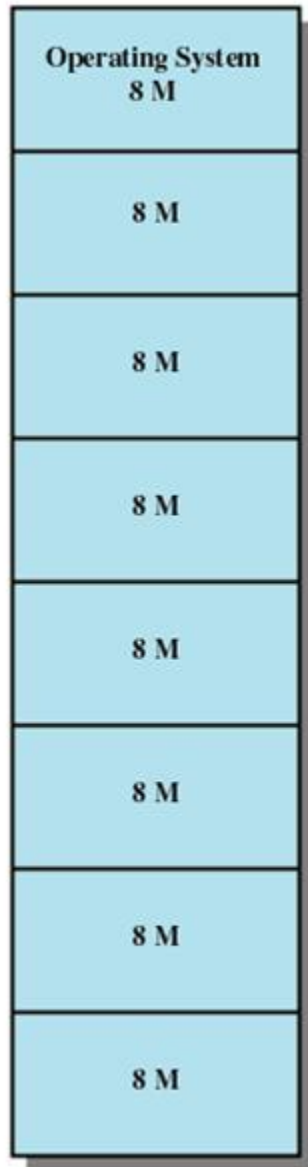
La solución a este problema es aplicar la compactación y/o recolección de basura.

En los métodos actuales, de tamaño variable, al liberar memoria, si el bloque liberado es adyacente a uno que estaba libre, estos bloques se unen para formar un bloque más grande, de tamaño igual a la suma de los tamaños de los bloques mezclados. De este modo, al liberarse toda la memoria, sólo existirá un bloque libre, del tamaño de la memoria total.

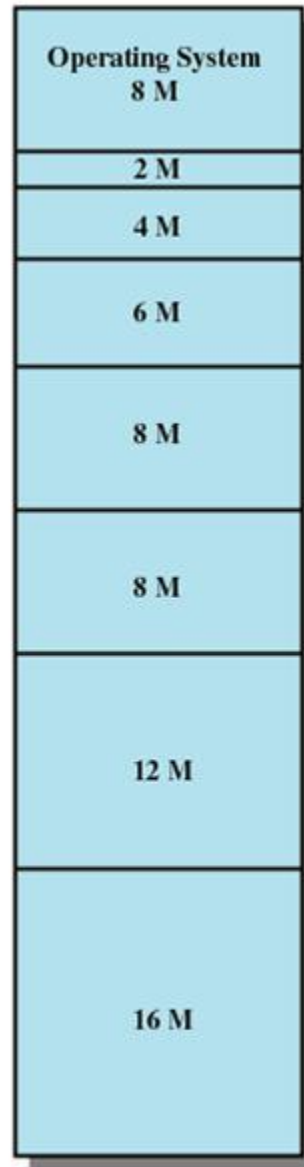
Los métodos que estudiaremos son:

- Ajuste secuencial
- Sistemas compañero

Particiones fijas



(a) Equal-size partitions



(b) Unequal-size partitions

- fácil programación de la asignación, ya que, al ser todas las particiones iguales, no importa que partición se le asigna al proceso
- Si un proceso no cabe en las particiones disponibles, el programador debe preparar el programa para trabajar sobreposiciones (overlay) y usar más de una partición
- alta fragmentación interna
- Variaciones:
 - Colas por tamaño de la partición: se crean varios tamaños de partición y cada proceso será asignado a la partición más pequeña que pueda contenerlo.

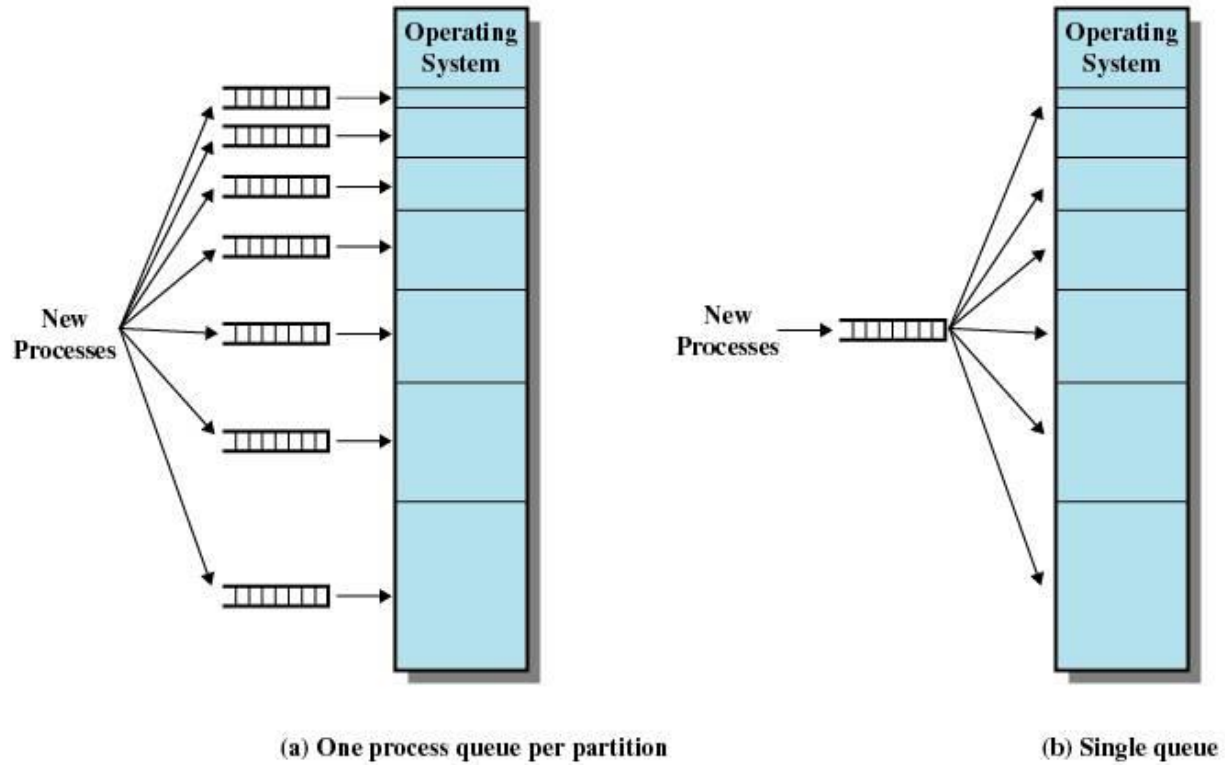


Figure 7.3 Memory Assignment for Fixed Partitioning

Particiones variables

- Las particiones son de tamaño variable
- Las particiones son creadas dinámicamente, según sea requerido por los procesos
- Se produce fragmentación externa
- Se necesita usar compactación periódicamente

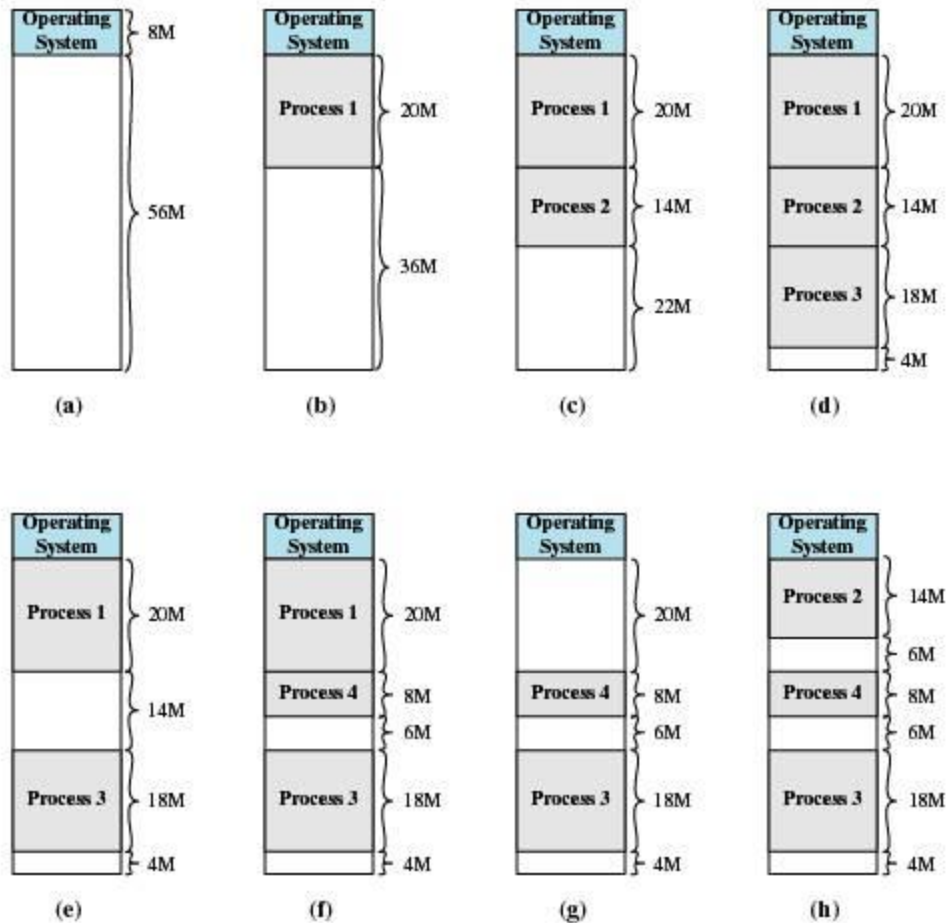


Figure 7.4 The Effect of Dynamic Partitioning

Métodos de Ajuste secuencial (Sequential fit)

Este es un conjunto de métodos que se basa en una lista de bloques libres mantenida dentro de la memoria disponible.

Proceso general

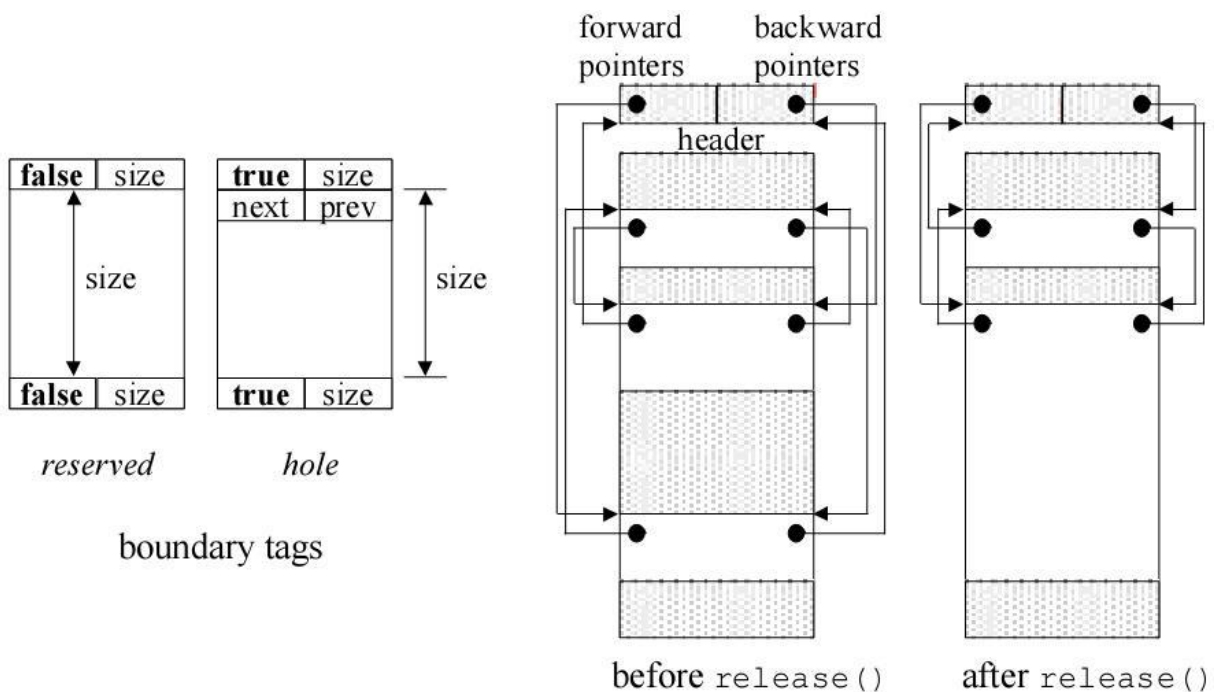
Asignación de un bloque de tamaño n

- Se recorre la lista de bloques libres en busca de un bloque de tamaño k , *según alguna política*, donde $k \geq n$.
- se marca RESERVADO y se desenlaza de la lista de libres.
- Si $k > n$, se puede decidir si la fragmentación será interna o externa (no se divide o se divide).
- Si se divide se enlaza el bloque restante como un nuevo bloque libre en la lista.

El criterio para dividir un bloque puede ser un parámetro que indique el máximo de fragmentación interna. Este parámetro no puede ser menor que el tamaño de las banderas y apuntadores necesarios.

Las políticas que se pueden aplicar para seleccionar el bloque a trabajar pueden ser:

- Primer ajuste
- Mejor ajuste
- Peor ajuste
- Óptimo ajuste



- **Primer ajuste:** se toma el primer bloque tal que pueda cumplir con el requerimiento. En el peor caso se recorre toda la lista. Desventaja: tiende a producir más fragmentación externa. Ventaja: normalmente es más rápido.
- **Mejor ajuste:** buscando eliminar la fragmentación externa, se busca el bloque menor que cumpla con el requerimiento. Desventaja: siempre hay que recorrer toda la lista
- **Peor ajuste:** se busca eliminar al máximo la fragmentación interna, buscando siempre el bloque más grande y dividiéndolo. Y además elimina gran parte de la fragmentación externa del mejor ajuste. Desventaja: Se tiene que recorrer toda la lista.
- **Ajuste óptimo:** En muestra de la lista se hace un mejor ajuste (N/e), luego sobre el resto, se hace un primer ajuste, de un bloque mejor (de menor tamaño que cumpla con el requerimiento) que el encontrado en la primera parte.

Otra optimización: mantener la lista ordenada por tamaño.

Efecto del ordenamiento

Cualquiera de estas políticas, a excepción del Ajuste Óptimo (ya que está diseñado para la lista sin un orden), se les puede agregar la variante de mantener la lista de bloques libres ordenada por el tamaño. Ésto tendría los siguientes efectos:

- **Primer Ajuste:** reduce el tiempo de búsqueda, ordenado ascendentemente, no se tiene que recorrer toda la lista para determinar que no existe un bloque adecuado, y en promedio se recorre la mitad para satisfacer los requerimientos, ya que la búsqueda de un bloque de tamaño n implica búsqueda de todos los bloques de tamaño $k < n$. Si esta ordenado descendientemente, el efecto es el mismo que el peor ajuste, y se toma sólo el primer bloque
- **Mejor ajuste:** tiene el mismo comportamiento que el primer ajuste con orden en la lista. Ya no se tiene que recorrer por completo la lista.
- **Peor Ajuste:** Si se mantiene la lista ordenada descendientemente, en TODAS las asignaciones de memoria, sólo se necesita examinar el primer nodo de la lista para determinar, y si el tamaño k del primer bloque es mayor o igual n , se atiende el requerimiento con dicho bloque, en caso contrario se determina que no se puede atender la solicitud. Lo cual incrementa el rendimiento de la asignación de memoria, en forma inmejorable.

Sin embargo, el mantener una lista ordenada, implica un retardo en el procesamiento de las liberaciones de memoria, dado que al combinar bloques adyacentes e insertarlos en la lista, implica realizar una búsqueda para determinar el lugar adecuado del bloque combinado.

Apuntador Vagabundo

Otra variante a cualquiera de las políticas originales consiste en mantener circulando la cabeza de la lista a lo largo de la misma, según algún criterio, como colocar redireccionar la cabeza de la lista en el nodo anterior (el último de la lista, dado que es doblemente encadenada), o colocar la cabeza en el siguiente nodo del que se está entregando al proceso solicitante.

Los efectos de esta variante son:

- evita que la fragmentación externa se acumule en un solo punto
- los bloques divididos son alcanzados “a la vuelta” y son los de mayor duración. Esto implica que aumenta la probabilidad de la combinación.

Liberación

- Se marca el bloque como LIBRE
- Mientras uno de los vecinos (izquierdo o derecho) estén libres
 - El bloque se combinan con sus vecinos libres para formar un sólo bloque, cuyo tamaño será la suma del tamaño de los bloques combinados

Este proceso permite que cuando se liberen todos los bloques ocupados, sólo existirá un bloque de memoria, cuyo tamaño será el de toda la memoria disponible.

Primer ajuste

Esta política consiste en retorna el primer bloque de la lista de tamaño k , donde $k \geq n$.

Ventajas:

- Resulta bastante simple de implementar
- es relativamente rápido en su desempeño, dado que las búsquedas en la lista, en promedio no se busca en toda la lista

Desventajas:

- Produce un alto índice de fragmentación externa, dado que los bloques tienden a dividirse constantemente

Mejor ajuste

Se toma el más pequeño de los bloques, cuyo tamaño $k \geq n$

Ventajas:

- salva los bloques de $>$ tamaño para pedidos grandes
- Reduce la fragmentación interna

Desventajas:

- Implica que en cada requerimiento se realiza un búsqueda en toda la lista
- tiende a fragmentar externamente

Peor ajuste

Consiste en tomar el bloque más grande de la lista para entregar al usuario, lo cual implica que en la gran mayoría de casos habrá una división del bloque.

Ventajas:

- reduce la fragmentación interna al máximo

Desventajas:

- Implica que en cada requerimiento se realiza un búsqueda en toda la lista
- tiende a fragmentar externamente

Óptimo ajuste

Esta política consiste en buscar en dividir la búsqueda en dos fases:

- sobre una muestra de los bloques libres, se aplica la política del mejor ajuste
- sobre el resto de la lista después de la muestra, se toma el primer bloque que sea mejor que el mejor de la muestra

El tamaño de la muestra, se toma como una muestra estadística y según algunos autores puede ser N/e (e =constante de Euler).

Ventajas:

- El resultado de esta política produce una eficiencia intermedia entre primer ajuste (más rápido y el mejor ajuste (menos fragmentación interna y externa)
- en el mejor de los casos no se tiene que recorrer la lista completa, y se devuelve un bloque que es un buen aproximado al mejor ajuste, y en el peor de los casos, se tiene que recorrer toda la lista, el resultado es igual al del mejor ajuste

Desventajas:

- se debe conocer el número de bloques libres
- Aumenta la fragmentación interna respecto al mejor ajuste

Sistemas compañero

Los sistemas compañero, al contrario de los métodos de ajuste secuencial, no se basan en listas, sino en cálculos de direcciones y divisiones por pares para manejar los bloques de memoria, por lo que el rendimiento es mejor que el Ajuste Secuencial, dado que no existen búsquedas en listas, las cuales son reemplazadas por cálculos matemáticos.

Los sistemas compañeros binarios, están basados en la propiedad de las direcciones de memoria, las cuales al final (como todo número en la computadora) es una potencia de dos. Este método, Al haber una solicitud de tamaño n , se procede a dividir el bloque de memoria por la mitad, hasta que se tenga un bloque de tamaño k , tal que $k \geq n$ y $k/2 < n$,

En este método, para satisfacer una solicitud, toma un bloque libre y lo divide sucesivamente en dos bloques compañeros, hasta que se tiene el menor bloque de tamaño k , que pueda satisfacer la solicitud de tamaño n ($k \geq n$).

Bloques compañero: son dos bloques adyacentes, resultado inmediato de la división de un bloque mayor, que por lo tanto son del mismo tamaño
--

Al momento de realizarse una liberación de un bloque, procede a combinarlo con su bloque compañero, si este está libre. Obsérvese que dos bloques adyacentes libres, no necesariamente son compañeros, por lo que la liberación permite que existan dos bloques libres adyacentes sin

combinar, si es que éstos no cumplen con la definición de bloque compañero, lo cual puede resultar en una deficiencia en la utilización de la memoria.

La asignación se realiza de la siguiente manera:

- El espacio de memoria se trata como un bloque de 2^U
- Para un requerimiento de tamaño s es requerido, tal que $2^{u-1} \leq s \leq 2^u$, entonces se entrega dicho bloque, de lo contrario, se divide el bloque en dos y se continua el proceso hasta que un bloque cumpla con $2^{k-1} \leq s \leq 2^k$

Ejemplo:

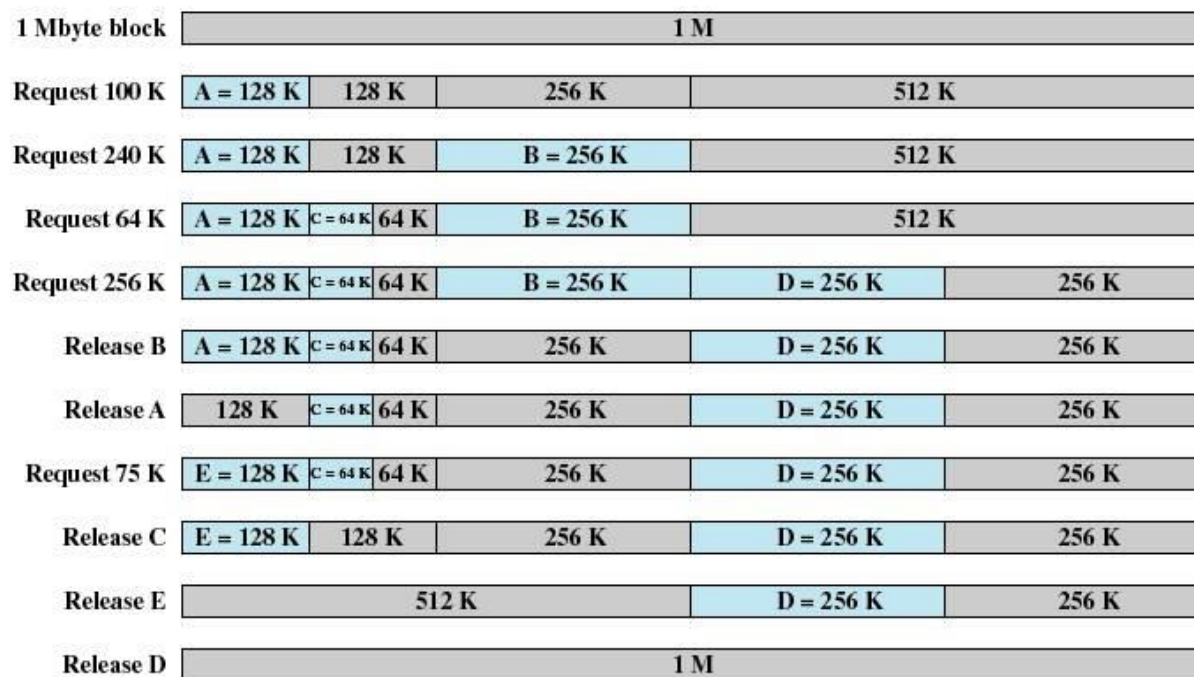


Figure 7.6 Example of Buddy System

Una ventaja es que no se necesitan listas para recorrer los bloques, sino que puede usarse sólo aritmética binaria (xor) de las direcciones y tamaños. Por ejemplo supongamos un bloque total de memoria de 64 bytes, y las siguientes direcciones:

Bloque	A	B	C	D
Tamaño del bloque (decimal/hexadecimal/binario)	16/10/0010000	08/08/0001000	08/08/0001000	32/20/0100000
Dirección (decimal/hexadecimal/binario)	00/00/0000000	16/0F/0010000	24/18/0011000	32/20/0100000
Dirección del bloque compañero (decimal/hexadecimal/binario)	16/10/0010000	24/18/0011000	16/10/0010000	00/00/0000000

Podemos notar que se cumple que:

$$\text{Dir. Bloque compa\~nero} = \text{Dir. bloque } \mathbf{xor} \text{ Tam.Bloque}$$

Dado que un bloque compa\~nero es un bloque *del mismo tama\~no*, resultado de la divisi3n de un bloque mayor, podemos calcular la direcci3n del compa\~nero de cualquier bloque, y si dicho compa\~nero es del mismo tama\~no, entonces s\~i es su compa\~nero. Puede ser que la direcci3n calculada no corresponda a un bloque del mismo tama\~no, debido a una divisi3n posterior, por lo tanto, ya no ser\~a su compa\~nero.

En la tabla anterior:

1. Vemos que la direcci3n del compa\~nero de A es la direcci3n 10h, que corresponde a B. En este caso A y BC, fueron compa\~neros, pero luego BC se dividi3 en B y C, por lo tanto, A y B ya no son compa\~neros, dado que no tienen el mismo tama\~no, ni proceden de la misma divisi3n.
2. El mismo caso sucede con el compa\~nero de D, que es la direcci3n 00h, correspondiente a A.
3. S3lo en el caso de B y C, se cumple con la definici3n de compa\~neros y el c\~alculo de la direcci3n de compa\~nero, corresponde en ambos casos..

También se puede representar por medio de un árbol binario, de la siguiente forma:

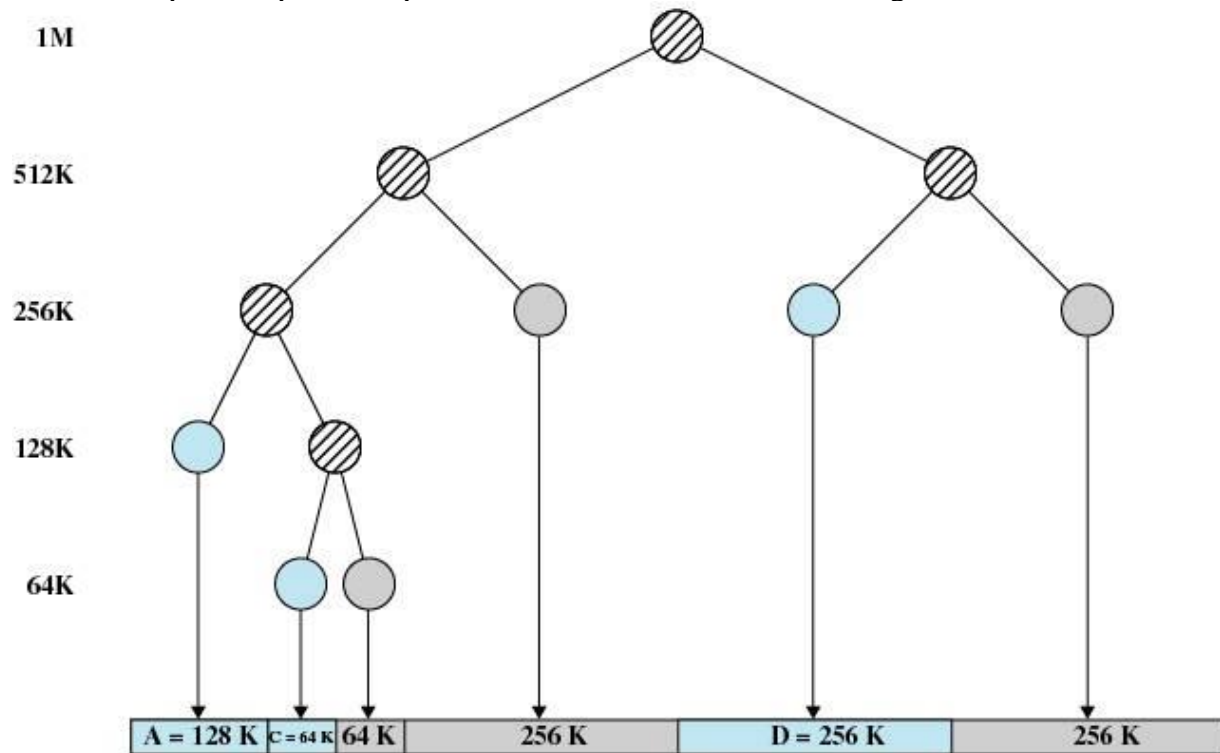


Figure 7.7 Tree Representation of Buddy System

En esta representación, cada nodo puede representar dos casos:

1. Es un bloque dividido, por lo que es un nodo interno del árbol.
2. Corresponde a un bloque directo, por lo que es un nodo hoja, el cual puede estar siendo utilizado o está libre.

El recorrido de este árbol facilita la operatoria y el cálculo en los siguientes aspectos:

1. Cada nivel del árbol representa una división entre dos, por lo que el cálculo del tamaño del bloque es directo
2. Al liberar un bloque, si el bloque hermano está libre, entonces procede la unión con el compañero, y el nodo padre se convierte en un nodo hoja.

Ventajas:

- El cálculo direcciones de memoria, para determinar los bloques compañeros es directo ya que está basado en números binarios

Desventajas:

- Aumenta la fragmentación interna, dado que el tamaño de cada bloque es una potencia de 2
- No siempre se pueden unir los bloques libres adjuntos
- Una variedad de fragmentación externa, que consiste en que no se puede utilizar toda la memoria aunque esté disponible

Fibonacci:

Este sistema, no realiza la división de los bloques por la mitad, sino que se basa en la serie de Fibonacci (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 ..) para realizar la división y combinación de bloques.

Por ejemplo: si un bloque de memoria es de 55 y se solicita un bloque de 10, el bloque de 55 se dividirá en un bloque de 21 y uno de 34, luego el de 21 se dividirá en uno de 8 y otro de 13 y se entregará este último al proceso solicitante

Ventajas:

- Dado que el tamaño de los bloques está determinado por la serie de Fibonacci en vez de potencias de dos, se reduce la fragmentación interna

Desventajas:

- El cálculo de los bloques compañeros se complica, dado que se tiene que calcular constantemente los predecesores y sucesores en la serie de Fibonacci.

Compactación

Esta operación sobre la memoria se utiliza para eliminar la fragmentación externa. Básicamente consiste en mover hacia un extremo de la memoria todos los bloques ocupados, dejando en el otro extremo un sólo bloque libre, cuyo tamaño es la suma de los tamaños de los bloques libres existentes antes de la compactación.

Esta operación supone un acceso exclusivo a la memoria, por lo que todos los procesos quedarán bloqueados hasta terminar la compactación, lo cual tiene un impacto directo en el rendimiento del sistema. Esto obliga a realizar un balance entre rendimiento y espacio libre: si se realiza frecuentemente, habrá más memoria disponible, pero el costo es una degradación del rendimiento y viceversa. Para que pueda realizarse la compactación es necesario que el sistema tenga la capacidad de **relocalización**.

Puede utilizarse hasta el último minuto, es decir cuando se pide memoria y no hay a causa de la fragmentación externa, se realiza automáticamente la compactación.

Colección de basura

La colección de basura se utiliza en aquellos lenguajes donde no se obliga al programador a liberar la memoria utilizada, sino que el programa, automáticamente, recupera la memoria no utilizada cuando existe un requerimiento de memoria que no puede ser completado.

El algoritmo básico de la colección de basura es la siguiente:

- marcar todos los bloques de memoria como libres, ya sea que estén usados o no
- para todas las variables, marcar su bloque de memoria como ocupado
- los bloques de memoria que queden marcados como libres, se enlazan en la lista de libres.

Otra estrategia utilizada, es llevar contadores por área de memoria. Cada vez que se asigna un área de memoria a un apuntador, se incrementa el contador de la región, y al designar, se decrementa. De esta forma la colección de basura se reduce a eliminar las regiones de memoria que tienen su contador a cero.



Sistemas Operativos 2

Unidad 1: Administración de memoria

Memoria virtual

René Ornelis
Primer semestre 2023

Contenido

1	Conceptos generales.....	4
1.1	Mecanismo general de reemplazo de páginas	5
1.2	Componentes del manejo de la memoria virtual	6
2	Estrategias de reposición.....	6
2.1	Al azar	7
2.2	PEPS: primero en entrar primero en salir.....	7
2.3	Segunda oportunidad.....	8
2.4	Menos recientemente usado	9
2.5	Menos frecuentemente usado	9
2.6	No usado recientemente	10
3	Estrategias de búsquedas.....	10
3.1	Por demanda.....	11
3.2	Anticipada	11
4	Estrategia de alcance:.....	11
4.1	Alcance local	12
4.2	Alcance global.....	12
5	Control de la hiperpaginación.....	12
5.1	Por conjunto de trabajo	12
5.2	Frecuencia de fallas de páginas.....	12

Figuras

Figura 1: Mecanismo de paginación con memoria virtual.....	5
Figura 2: Flujograma del proceso de reemplazo de página	6
Figura 3: Reemplazo por segunda oportunidad	8
Figura 4: Estado de la cola después del reemplazo.....	8

Memoria virtual

1 Conceptos generales

Vimos en el capítulo anterior que el sistema operativo necesita administrar la memoria con un balance de eficiencia y usabilidad para el usuario. Esto es un objetivo bastante difícil de conseguir considerando que los procesos tienden a consumir cada vez más memoria y muchas veces piden memoria al sistema operativo que probablemente nunca utilizarán. Entonces si el sistema operativo sólo trabajará con la memoria que está disponible en RAM en muy poco tiempo o con muy pocos procesos llegará a quedarse sin memoria lo que no permitirá al usuario cargar más procesos, lo cual reduce la usabilidad del sistema ante los ojos del usuario.

Debido a esto se inventó la memoria virtual la cual básicamente consiste en el intercambio continuo de páginas de memoria entre la memoria principal y el disco duro, con el fin de permitir que el sistema muestre más memoria de la que realmente tiene y permita trabajar más procesos de los que cabrían en la memoria principal.

Esto también se tiene que hacer con precisión y cuidado del balance entre el rendimiento y la usabilidad, debido a que, si se abusa de este proceso, es decir hay muchos accesos a disco para bajar páginas o subir páginas de memoria, el impacto en el rendimiento del sistema será sensible porque el rendimiento del disco es mucho más lento que el acceso a memoria RAM.

En esta unidad estudiaremos las diferentes técnicas y las desventajas de cada una de ellas para el manejo eficiente de la memoria virtual en los sistemas operativos.

Memoria virtual: Capacidad del sistema de permitir a los procesos utilizar más memoria que la disponible en RAM, a través de utilizar memoria secundaria (discos duros) para almacenar parte de la memoria en RAM, con énfasis en conservar un balance entre la usabilidad y rendimiento.

Intercambio (swap): Es el hecho de grabar en disco memoria que estaba en RAM y viceversa.

Fallo de página:

Es el evento que un proceso accede una página que no está en memoria y se tiene que buscar en el disco duro (provocar un intercambio).

Hiperpaginación:

Exceso de fallos de página.

Desperdicio (trashing):

Es cuando el procesador pasa más tiempo decidiendo qué páginas intercambiar y esperando respuesta del disco, que ejecutando procesos. Es una consecuencia directa de la hiperpaginación

Localidad temporal y espacial:

Localidad temporal: la probabilidad condicional de que suceda de nuevo un evento dado que sucedió hace poco tiempo, es inversamente proporcional al tiempo transcurrido.

Localidad espacial: la probabilidad condicional de que suceda de nuevo un evento dado que sucedió cerca, es inversamente proporcional al espacio recorrido.

1.1 Mecanismo general de reemplazo de páginas

Como podemos apreciar en la figura el mecanismo de la memoria virtual es básicamente el mismo que vimos en la paginación, con la única diferencia de que se tiene que obtener en la página de tablas, además de todos los datos que vimos, una bandera que indique si esa página está en memoria o está en disco duro.

En caso de que la bandera indique que está en memoria el proceso continúa tal como lo vimos en la unidad anterior con un acceso directo a memoria. De lo contrario se invoca una interrupción al sistema operativo indicando un fallo de página, lo cual significa que la memoria que se está intentando acceder no se encuentra en memoria y está en algún lugar del disco duro, por lo que el sistema operativo se debe proceder así:

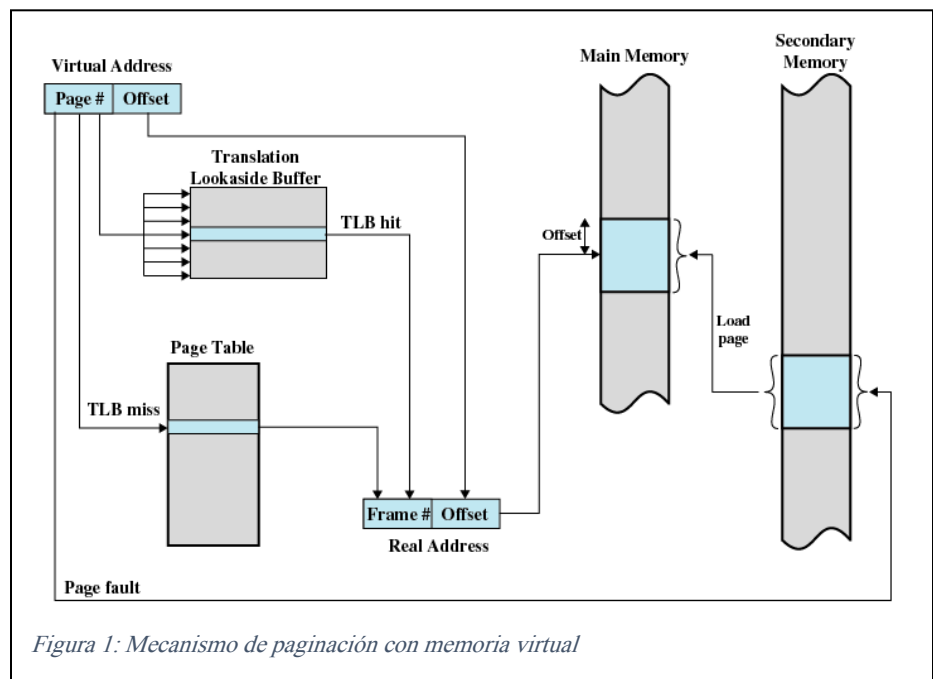


Figura 1: Mecanismo de paginación con memoria virtual

1. buscar un lugar en la memoria principal, es decir un marco de página que no se esté utilizando y colocar allí la página que está en disco duro.
2. Puede suceder que en el área de los marcos de página no existan ningún marco libre donde pueda colocar la memoria que se debe de traer del disco, en cuyo caso se debe elegir una de las páginas que está en memoria, bajo alguna **política de reemplazo**, y bajarla disco duro para dar lugar a la página que se está intentando acceder y provocó el fallo de página.
3. Una vez reestablecida la página, se actualiza la página de tablas con la bandera de que está en memoria y la dirección en memoria donde se cargó.

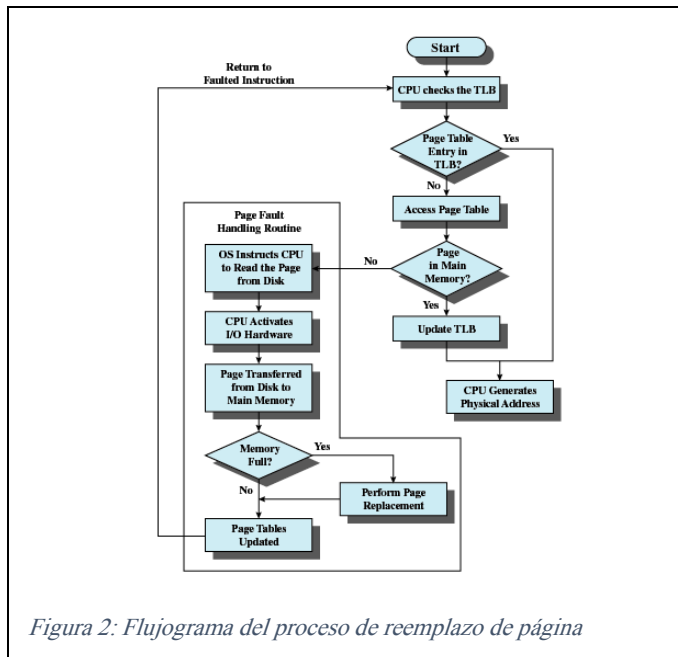


Figura 2: Flujograma del proceso de reemplazo de página

4. Se reinicia el proceso de acceder a la dirección virtual sólo que esta vez ya estamos seguros de que esa página está en memoria por lo cual se hará por el proceso normal de paginación

Esto se ilustra en el flujograma de la Figura 2, en la cual se ilustra la lógica que sigue el MMU y el sistema operativo para atender un fallo de página.

1.2 Componentes del manejo de la memoria virtual

Como vimos, hay varios elementos estratégicos involucrados en el

manejo de la memoria virtual por parte del sistema operativo estos son:

1. **Estrategia de reposición:** Es la política para decidir qué página en RAM es elegida para bajar a disco, cuando sea necesario subir una página de disco. Es decir se produjo un fallo de página y es necesario hacer un intercambio.
2. **Estrategia de búsqueda:** Es la política que decide cuándo subir a RAM, páginas que están en disco.
3. **Estrategia de alcance:** Es la política que indica el grupo de páginas que se considerarán al momento de una reposición o intercambio.

esto lo veremos a detalle en la siguiente sección

2 Estrategias de reposición

Una vez invocado el sistema operativo para solucionar un fallo de página, mencionamos que es posible que en los marcos de página no hay ningún marco libre para poder colocar la página que se necesita desde el disco duro. Por lo tanto, el sistema deberá elegir una de las páginas que están en memoria para descargarla a disco y así hacer el espacio necesario para colocar la página que nos provocó el fallo de página. Cualquier política que se implemente debe buscar alcanzar el principio de optimización:

Principio de optimización: Debe reemplazarse la página que no se va a utilizar durante el tiempo más largo, es decir el que tiene la menor tasa de fallas de páginas de todos los algoritmos.

Por supuesto que este principio es sólo una meta que se busca en cualquier política de reemplazo, ya que es imposible saber el futuro y por lo tanto no podemos predecir cuándo se va a

volver a utilizar una página, pero el objetivo final es minimizar la hiperpaginación del procesador para que el rendimiento total del sistema sea aceptable.

Entre las diferentes estrategias de reposición vamos a ver las siguientes:

1. Al azar
2. Primero en entrar primero en salir
3. Segunda oportunidad
4. Menos recientemente usado
5. Menos frecuentemente usado
6. No usado recientemente

2.1 Al azar

Esta política de reemplazo en realidad no es una política utilizada por ningún sistema operativo sino una implementación de referencia para comparar estadísticas con otras estrategias. La implementación de esta política pues resulta bastante simple, aunque en las mismas comparaciones empíricas del rendimiento de una política resulta muy ineficiente en el objetivo de evitar la hiperpaginación.

2.2 PEPS: primero en entrar primero en salir

En este esquema el sistema operativo conserva una lista de las páginas, según cómo se fueron cargando en memoria, ya sea por creación de la página o porque se trasladó del disco duro hacia memoria principal. De esta forma la primera página en la lista es la página que lleva más tiempo en memoria.

La estrategia de PEPS consiste en que al momento de decidir qué páginas se reemplazará, se toma la primera página de la lista, lo que resulta en un esquema fácil de implementar y eficiente en el proceso de decisión. Es decir, en el momento en que el sistema operativo está atendiendo un fallo de página donde necesita decidir, lo más rápidamente posible, la página elegible para descargar a disco.

Sin embargo, este esquema tiene una desventaja que consiste en que las páginas de uso constante normalmente son enviadas a disco y reemplazadas y restauradas desde disco con mucha frecuencia, porque no se toma en cuenta la cantidad de uso de una página sólo el tiempo en que fue cargada a memoria. Adicionalmente, existe un fenómeno llamado la Anomalía PEPS o anomalía Belady (por su descubridor), que consiste en que si a un proceso se le asignan más marcos de página para que trabaje, dada una cierta secuencia de accesos a páginas, se producen más fallos de página, lo cual es contrario al sentido común ya que se supone que si a un proceso se le asigna más memoria, debería de producir menos fallos de página.

Como ejercicio considere un proceso que tiene 3 marcos de página para su uso. Si este proceso accede la siguiente secuencia de página: A, B, C, D, A, B, E, A, B, C, D, E. sucederá que si le agregamos un marco de página más, es esta misma secuencia de páginas, provocará más fallos de página. Es decir, con 3 marcos de página disponible va a producir menos fallos de página a que si tuviera cuatro marcos de página disponibles.

2.3 Segunda oportunidad

Esta estrategia de reemplazo es derivada de la estrategia PEPS con el fin de eliminar el problema de la anomalía Belady. Requiere de una implementación de hardware muy específica que es el bit de referencia (bit R) el cual consiste en que el MMU activa un bit en la entrada de la tabla de páginas cada vez que la página respectiva es utilizada.

Tal como se aprecia en la Figura 3, esta estrategia consiste en una lista circular de páginas referenciadas. Se tiene un apuntador vagabundo hacia la siguiente página que le corresponde según el orden PEPS, el cual avanza cada vez que se produzca un fallo de página.

Al momento de decidir un reemplazo se analiza la primera de la fila y si tiene el bit de referencia en 1, éste se cambia a 0 y se examina la siguiente página.

Por ejemplo: consideremos Figura 3, la cual representa el estado de la cola circular antes del fallo de página, donde el apuntador a la primera página de la lista está señalando la página 45. Suponiendo que se necesita restaurar del disco la página 727, el sistema analizará la página 45 y determinará que su bit de referencia está activado por lo tanto sólo cambia el bit de

referencia a cero y continúan analizando la siguiente página qué es la 191. Esta página también tiene encendido subir de referencia por lo que repite el proceso y analiza la página 556. Dado que esta página tiene su bit de referencia en cero entonces es la elegida para ser reemplazada y en ese marco de página se coloca la página 727 y el apuntador se mueve para la página 13 para estar listo para el próximo fallo de página.

Como podemos apreciar esta política es similar a PEPS en simplicidad y eficiencia, Además de eliminar el problema de la anomalía Belady.

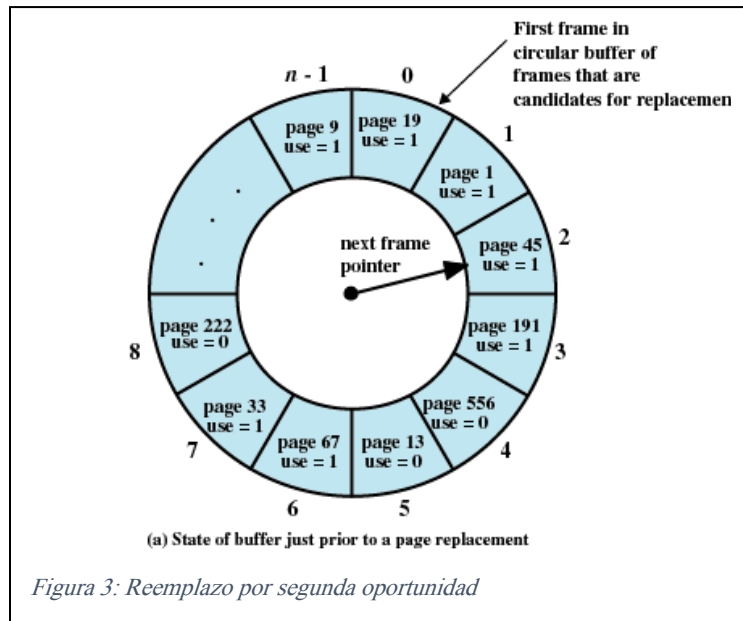


Figura 3: Reemplazo por segunda oportunidad

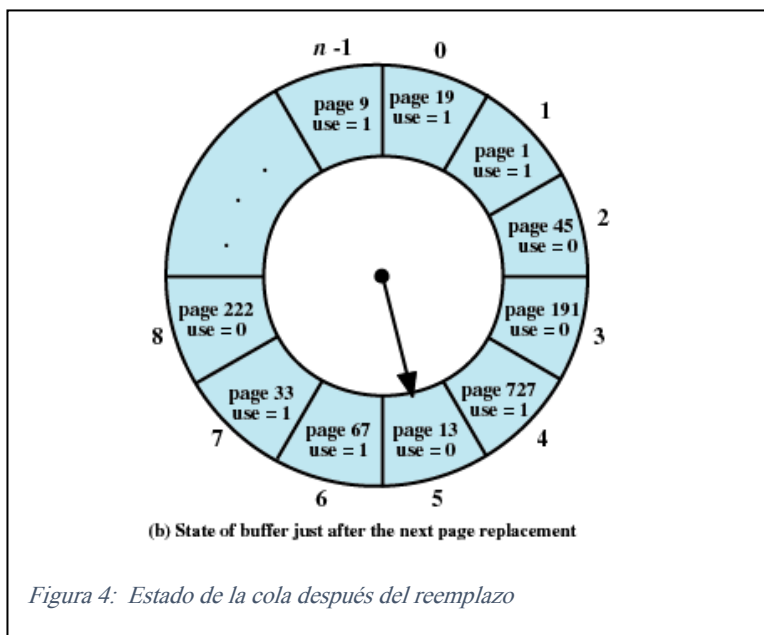


Figura 4: Estado de la cola después del reemplazo

Sin embargo, para su implementación se requiere de la capacidad del hardware para activar el bit de referencia de cada página cada vez que se utiliza.

2.4 Menos recientemente usado

En esta política el hardware ayuda en poner una marca de tiempo (la hora) la entrada de la tabla de páginas cada vez que la página correspondiente es accedida. De esta forma, la estrategia consiste en buscar cuál de todas las páginas tiene la menor marca de tiempo es decir la página que no ha sido accedida por más tiempo. Por concepto de localidad temporal es la que menos probabilidades tiene de ser utilizada en el futuro.

La desventaja de este esquema es que cada vez que necesite realizar un reemplazo, se necesita revisar todas las páginas para decidir la que se reemplazará, por lo cual se deben de recurrir a optimizaciones como tener una lista ordenada por tiempo de todas las páginas, de forma que al momento de decidir se toma la primera de la lista. Esto implica que el mantenimiento ordenado de la lista debe de realizarse en un proceso de fondo para no interferir en el rendimiento del proceso de acceso a las páginas.

Otra desventaja de este esquema es que las páginas de configuración o de datos globales de un proceso suelen ser las primeras en ser cargada esa memoria y ser accedidas a lo largo del proceso, lo que provoca que estas páginas sean descargadas a disco para luego ser vueltas a restaurar, provocando un acceso innecesario a disco. También sucede que algunas páginas pueden ser accedidas sólo una vez y van a permanecer un largo tiempo en memoria hasta que sean las más antiguas de la lista. Para contrarrestar esto se utiliza se utiliza un proceso que periódicamente inserta un bit a la izquierda de cada marca de tiempo (se hace un shift a la derecha con cero en cada marca de tiempo) lo que tiene el efecto de dividir entre 2 el valor de la entrada de páginas y si una página es accedida una vez el valor de su marca de tiempo disminuirá rápidamente produciendo un envejecimiento prematuro que la hará elegible más pronto para un reemplazo.

2.5 Menos frecuentemente usado

En esta política interesa la intensidad de uso de una página, es decir que en vez de mantener una marca de tiempo en la entrada de la tabla de páginas se mantiene un contador de acceso de esta forma se reemplaza la página cuyo contador de acceso tiene el menor valor.

Al igual que la política de menos recientemente utilizado, elegir a la página qué se reemplazará requiere recorrer toda la lista de páginas para determinar la que tiene el menor valor. Igualmente se puede aplicar la misma optimización de mantener la lista de páginas ordenada ascendentemente por frecuencia de uso de forma que al momento de elegir se tome la primera de la lista.

Se tiene la limitante de que existe un número definido de bits en los contadores lo cual al momento de llegar al máximo valor no se le podrá agregar más, lo que conlleva al problema de una memoria eterna, es decir, si una página fue utilizada mucho, posiblemente al inicio del proceso, pero ya no se volvió a utilizar se conservará bastante tiempo en memoria y posiblemente nunca se descargue a disco. Para esto también se puede hacer un esquema similar de envejecimiento de las páginas con proceso de fondo que:

1. Inserta a la izquierda del contador el valor del bit de referencia (SHR o desplazamiento a la derecha), de forma que si el bit está encendido se le inserta a la izquierda del contador un 1, en caso contrario se insertará un 0
2. Se reinicia el bit de referencia a 0 periódicamente

De esta forma una página que fue anteriormente utilizada con mucha frecuencia, eventualmente llegara a tener el valor cero, dado que periódicamente se le inserto un cero a la izquierda (bit de referencia).

2.6 No usado recientemente

Los esquemas anteriores del menos frecuentemente usado y menos recientemente usado tienen la desventaja de qué se debe mantener una lista de las páginas y se debe hacer un recorrido completo de dicha lista o mantenerla ordenada con el correspondiente costo en rendimiento para poder determinar la página reemplazar. Este esquema busca eliminar ese problema con el uso de 2 bits de hardware:

- Bit de referenciado (bit R): se activa cada vez que la página es leída
- Bit de modificación (bit M): se activa cada vez que la página es modificada por el proceso

Adicionalmente también se cuenta con que el bit R es reiniciado periódicamente para todas las páginas. De esta forma se forman cuatro clases de páginas según estén activos los R y M, que se enumeran en la siguiente tabla:

M	R	Descripción
0	0	páginas no leídas ni modificada
1	0	página no leída y modificada
0	1	página leída y no modificada
1	1	página leída y modificada

Notamos que la clase 10 es posible debido al reinicio periódico del R, lo que significa que una página de clase 10 antes era una página de clase 11 cuyo bit R fue reiniciado porque no fue leída desde la última vez que se corrió el proceso de reinicio de bits.

La combinación de los valores del bit R y M se toman como los dígitos de un numero binario que corresponderían a los valores 0, 1, 2 y 3, por lo que, en esta estrategia, recorre las páginas y reemplaza la primera que encuentre de menor valor, es decir una página de clase 00. aunque en el peor caso, si no hubiera páginas de clase 00 tendría que recorrer todas las páginas para elegir una de las de menor clase, en promedio sólo se recorrerá la cuarta parte de las páginas para elegir una página de clase 00.

3 Estrategias de búsquedas

La otra estrategia que forma parte de la administración de la memoria virtual es la estrategia de búsqueda que define cuándo se debe de restaurar una página de disco hacia la memoria, Y básicamente existen 2:

- Por demanda
- Anticipada

3.1 Por demanda

La búsqueda por demanda es la más simple, ya que se busca una página cada vez que se produce un fallo de página, pero no utiliza ninguno de los conceptos de localización.

Idealmente el sistema operativo debería de buscar sincronizar el tamaño de página con el tamaño de bloque de disco óptimo, pero esto no siempre es posible, por lo que leer página por página no será óptimo.

tamaño de bloques de disco óptimo: es la cantidad de datos X tal que leer del disco una cantidad menor que x se tarda el mismo tiempo y a partir de dicha cantidad el tiempo de lectura/escritura es proporcional a la cantidad de datos leída.

Por ejemplo: leer un byte del disco nos llevará un tiempo X , y leer 100 bytes también nos llevará el mismo tiempo, al igual que leer 32 k. Sin embargo, a partir de 33Kb el disco duro tardará $2X$, de tiempo entonces nuestro tamaño de bloque de disco óptimo es de 32 k.

3.2 Anticipada

Se sube a memoria un conjunto de páginas que el proceso muy probablemente utilizará

Conjunto de trabajo: conjunto de páginas que un proceso accedió en un determinado período de tiempo (los que tengan encendido el bit R)

- Se examinan las X referencias a páginas más recientes de un proceso, las cuales componen su conjunto de trabajo
- Implementación por medio del bit R, interrupciones periódicas que limpien este bit. De esta forma las páginas de un proceso que estén con $R=1$ son las que pertenecen al conjunto de trabajo.
- Se trata de mantener en memoria el conjunto de trabajo de todos los procesos:
- Al inicio, se ponen en los marcos de página la memoria que el proceso solicite dentro de ciertos límites predeterminados
- Al suspender un proceso, por cualquier razón, se conserva la información de cuál es su conjunto de trabajo, y sus páginas se bajan a disco.
- Al despertarse el proceso, se sube a memoria las páginas que conforman su conjunto de trabajo.

4 Estrategia de alcance:

Define la forma en que se asignarán marcos de páginas a los procesos, y la definición del conjunto de páginas candidatas a intercambio, cuando se necesite. Se definen dos posibles estrategias de alcance:

- Alcance local
- Alcance global

4.1 Alcance local

Se asigna a cada proceso un conjunto de marcos de páginas, en el que trabajará. Cuando un proceso produce un fallo de página, la página a ser reemplazada es elegida entre las páginas en los marcos asignados al proceso.

Ventajas: Sólo se tiene que buscar en la tabla de páginas de un proceso

Desventaja: Un proceso muy intenso en memoria, provocará una tasa alta de fallos de páginas, por lo incurrirá en hiperpaginación.

El número de páginas a asignar a cada proceso (m marcos y n procesos) puede ser de dos formas:

- Uniforme: a cada proceso se le asignan m/n marcos de página
- Proporcional: a cada proceso se le asignan marcos de página en proporción al tamaño de la imagen del ejecutable

4.2 Alcance global

Los procesos no tienen asignados un número determinado de marcos de página, y al momento de provocar un fallo, el sistema puede elegir reemplazar una página que pertenece a un proceso diferente al que provocó el fallo de página.

Puede provocar que la hiperpaginación de un proceso, lleve a otros procesos a elevar su tasa de fallos de página.

5 Control de la hiperpaginación

5.1 Por conjunto de trabajo

- Si se tienen el tamaño del conjunto de trabajo CJ_i para un proceso i , entonces $\text{Demanda} = \text{SUM}(CJ_i)$.
- Si la Demanda llega a ser mayor que el número de páginas disponibles, entonces se producirá hiperpaginación, por lo que se elige un proceso y se suspenden, y sus marcos de páginas son asignadas a otros procesos

5.2 Frecuencia de fallas de páginas

- Se establecen límites inferior y superior para la tasa de fallas de páginas
- Si la tasa de fallas de página es menor del límite inferior, entonces el proceso tiene muchos marcos asignados, por lo que se le quita uno y se asigna al proceso con mayor tasa de fallas de página

- Si la tasa de fallas de página es mayor del límite superior, entonces el proceso tiene pocos marcos asignados, por lo que se le da un marco más, a costa del proceso con menor tasa de fallos de página. Si no hay marcos disponibles, entonces el proceso se suspende.



Sistemas Operativos 2

Unidad 2: Administración de dispositivos de E/S

Conceptos generales de la administración de dispositivos

René Ornelis
Primer semestre 2023

Contenido

1	Introducción	4
2	Caracterización de los dispositivos de E/S	5
2.1	Componentes generales de los dispositivos	5
2.2	Clasificación de dispositivos según el uso	5
2.3	Clasificación de dispositivos según la transferencia de información.....	6
2.4	Clasificación de dispositivos según su direccionamiento	6

Índice de figuras

Figura 1: Velocidad de respuesta de diferentes tipos de dispositivos.....	4
Figura 2: Dispositivos en Linux.....	6
Figura 3: Puertos asignados por dispositivo en el procesador Intel.....	7

ADMINISTRACIÓN DE DISPOSITIVOS

'''

1 Introducción

Los dispositivos que debe manejar el sistema operativo incluyen:

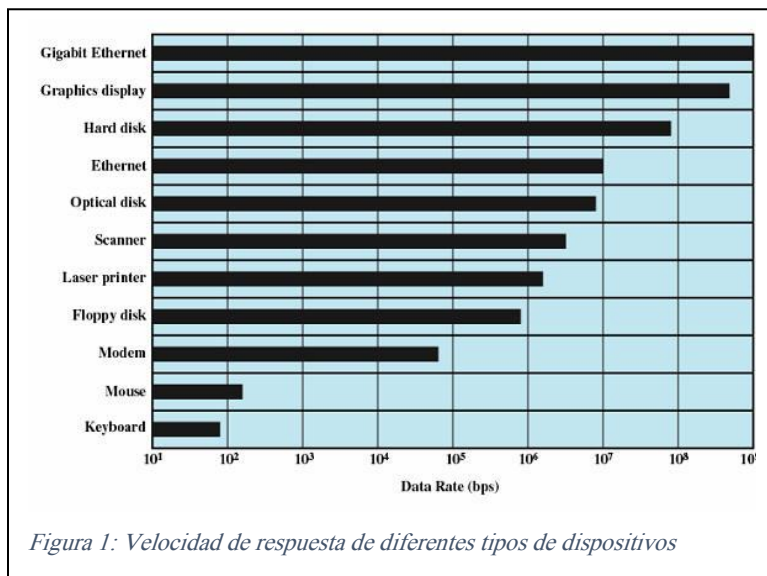
- Teclado
- Monitor
- Dispositivos de almacenamiento: discos, cintas, disquetes, etc.
- Impresoras
- Red
- MODEM
- Ratón
- etc.

Cómo se puede ver, se abarca una gran *variedad de tipos de dispositivos*, la cual está cambiando constantemente. Esto complica el diseño del SO dado que éste tiene que establecer una interfaz entre las aplicaciones y el equipo, lo más simple y uniforme posible. Es decir, la interfaz debe ser la misma para todos los dispositivos a nivel de aplicación (**principio de independencia de dispositivos**).

La amplia variedad de dispositivos hace difícil la aplicación del principio de independencia de dispositivos, dado que existen *diferencias radicales* entre los dispositivos, en los siguientes aspectos:

- Velocidad de los datos
- Aplicaciones
- Complejidad del control
- Unidad de transferencia: se puede tratar por bloques o por caracteres
- Representación de los datos: cada dispositivo puede utilizar diferentes sistemas de codificación
- Condiciones de error

Los dispositivos son mucho más lentos que el procesador y la memoria, ya que estos se miden en Ghz y nanosegundos respectivamente, mientras que los dispositivos están en el orden de milisegundos.



Esta diferencia y el deseo de aplicaciones interactivas hace que la E/S sea el **cuello de botella**, por lo que el sistema operativo debe proporcionar un sistema de E/S que logre los siguientes objetivos:

- Interfaz sencilla y fácil de utilizar
- Optimizar la E/S
- Permitir conectar cualquier dispositivo sin tener que remodelar el sistema E/S ni el sistema operativo

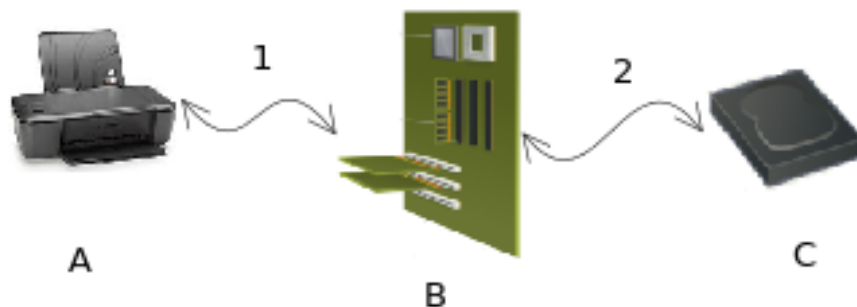
2 Caracterización de los dispositivos de E/S

Con el objetivo de mejorar la abstracción que tenemos de los dispositivos, se definen los componentes generales de todos los dispositivos y las diferentes clasificaciones existentes de los tipos de dispositivos. Aunque estas abstracciones y clasificaciones no son del todo absolutas, por la amplia variedad de dispositivos existentes y porque hay dispositivos que no caben en alguna de las clasificaciones, es un ejercicio forzado para una referencia y un manejo generalizado, en la medida de lo posible.

2.1 Componentes generales de los dispositivos

Los dispositivos constan de tres componentes:

- **Periférico, o componente mecánica:** es el dispositivo en sí
- **Controlador, o componente electrónica:** es el **adaptador** que interactúa entre el dispositivo y el procesador, a través del bus del sistema.
- **Driver, o componente lógica:** es la parte de software que convierte los comandos generales del sistema operativos, en instrucciones específicas para el dispositivo



2.2 Clasificación de dispositivos según el uso

Según su función, los dispositivos se pueden clasificar en:

- **Dispositivos de interfaz de usuarios:** todo lo que ayuda a la comunicación hombre-máquina. Se podría decir que esta tal vez es de las clasificaciones más importantes desde

la percepción del usuario, porque nos da la utilidad de una computadora, ya una computadora no sirve si no podemos obtener información de ella o si no podemos ingresarle información. Dentro de esta clasificación de dispositivos entran obviamente el monitor, el teclado, el mouse, etc. y cualquier dispositivo que nos permita interactuar entre la máquina y el ser humano, es decir un dispositivo que nos permita dar un comando o información a la computadora, un dispositivo que nos da información de la computadora.

- **Dispositivos de almacenamiento:** todo lo que puede almacenar información de forma persistente o temporal. Aquí se incluyen dispositivos como la memoria RAM, discos duros, CD, DVD, blu ray, cualquier medio óptico-magnéticos.
- **Dispositivos de comunicaciones:** Todo lo que ayuda a la comunicación máquina-máquina. Aquí clasificamos a las tarjetas de red, puertos de rayos infrarrojos, etc.

2.3 Clasificación de dispositivos según la transferencia de información

Esta clasificación es la más antigua, que se creó con los sistemas de Unix, en la cual los dispositivos se dividen en dispositivos de bloques y dispositivos de caracteres.

- **Dispositivos de bloques:** Son los que reciben/entregan información en bloques. De estos dispositivos se lee/escribe una determinada cantidad de información a la vez con *direccionamiento*. El direccionamiento es la capacidad que podamos escribir información en cierta posición dentro del dispositivo, y dicha información puede ser recuperada a través de la misma dirección. Por ejemplo: en un disco duro usualmente mandamos a leer/escribir un bloque de memoria, se especifica la dirección de cilindro-cabeza-sector.
- **Dispositivos de caracteres:** Son los que reciben/entregan información un flujo de caracteres que no son direccionables y no tienen funciones de localización. Por ejemplo:

teclado, monitor, puertos. Por ejemplo: el teclado uno envía comandos a la computadora como una secuencia de teclas, los cuales no pueden ser recuperados posteriormente.

Como ejemplo, observemos el contenido del directorio `/dev/` en un Linux o Unix, donde se muestra la lista de todos los dispositivos. Tal como podemos ver en la Figura 2, los dispositivos de carácter tienen el identificador `c` al inicio, y los dispositivos de bloque

```
> ls -l /dev
total 0
crw-r--r-- 1 root root 10, 235 Jun 28 10:25 autofs
drwxr-xr-x 2 root root 380 Jun 28 10:25 block
drwxr-xr-x 2 root root 140 Jun 28 10:25 bsg
drwxr-xr-x 3 root root 60 Jun 28 10:25 bus
lrwxrwxrwx 1 root root 3 Jun 28 10:25 cdrom -> sr0
drwxr-xr-x 2 root root 120 Jun 28 10:25 centos
drwxr-xr-x 2 root root 3040 Jun 28 10:25 char
crw--w---- 1 root tty 5, 1 Jun 28 10:25 console
lrwxrwxrwx 1 root root 11 Jun 28 10:25 core -> /proc/kcore
drwxr-xr-x 3 root root 60 Jun 28 10:25 cpu
cW----- 1 root root 10, 126 Jun 28 10:25 cpu_dma_latency
cW----- 1 root root 10, 203 Jun 28 10:25 cuse
drwxr-xr-x 7 root root 140 Jun 28 10:25 disk
bW-rw---- 1 root disk 253, 0 Jun 28 10:25 dm-0
bW-rw---- 1 root disk 253, 1 Jun 28 10:25 dm-1
bW-rw---- 1 root disk 253, 2 Jun 28 10:25 dm-2
bW-rw---- 1 root disk 253, 3 Jun 28 10:25 dm-3
drwxr-xr-x 2 root root 60 Jun 28 10:25 dma_heap
```

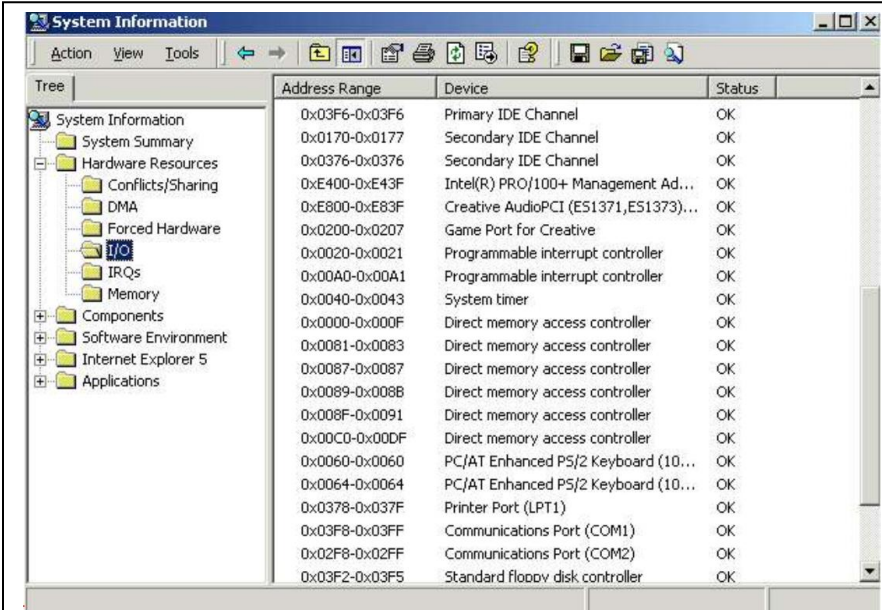
Figura 2: Dispositivos en Linux

se distinguen por el identificador `b`.

2.4 Clasificación de dispositivos según su direccionamiento

El procesador, en general, no interactúa directamente con el dispositivo, sino con los adaptadores. La interfaz entre el procesador y el adaptador puede ser a través de:

- **Proyección en memoria:** Se asigna a cada controlador un rango de direcciones a través de las cuáles se accede a sus registros y se programa con instrucciones de acceso a memoria. Se reserva una zona de memoria física para asignar a controladores de E/S. Aquí las operaciones de entrada y salida son lecturas y escrituras a memoria, con las instrucciones existentes. Esta interfaz simplificar la labor del sistema operativo, ya que solo debe mantener el mapeo de dirección de memoria para cada dispositivo. Por su parte, el controlador o componente electrónico escanea constantemente la memoria asignada y en cuanto haya información escrita, por ejemplo un comando y sus parámetros, la interpreta y envía el comando correspondiente al dispositivo. Actualmente, el ejemplo más evidente de este tipo de dispositivo es el monitor, el cual utiliza la memoria de video, que se lee y se proyecta en pantalla.
- **Proyección en puertos:** Al controlador se le asigna un puerto de E/S, una interrupción de hardware y se programa con operaciones de E/S (*portin* y *portout*) para indicar que dispositivo se quiere manipular y cómo. Por ejemplo: en la Figura 3 se muestran los puertos asignados a los diferentes dispositivos en el procesador Intel.



Tree	Address Range	Device	Status
System Information	0x03F6-0x03F6	Primary IDE Channel	OK
System Summary	0x0170-0x0177	Secondary IDE Channel	OK
Hardware Resources	0x0376-0x0376	Secondary IDE Channel	OK
Conflicts/Sharing	0xE400-0xE43F	Intel(R) PRO/100+ Management Ad...	OK
DMA	0xE800-0xE83F	Creative AudioPCI (ES1371,ES1373)...	OK
Forced Hardware	0x0200-0x0207	Game Port for Creative	OK
I/O	0x0020-0x0021	Programmable interrupt controller	OK
IRQs	0x00A0-0x00A1	Programmable interrupt controller	OK
Memory	0x0040-0x0043	System timer	OK
Components	0x0000-0x000F	Direct memory access controller	OK
Software Environment	0x0081-0x0083	Direct memory access controller	OK
Internet Explorer 5	0x0087-0x0087	Direct memory access controller	OK
Applications	0x0089-0x008B	Direct memory access controller	OK
	0x008F-0x0091	Direct memory access controller	OK
	0x00C0-0x00DF	Direct memory access controller	OK
	0x0060-0x0060	PC/AT Enhanced PS/2 Keyboard (10...	OK
	0x0064-0x0064	PC/AT Enhanced PS/2 Keyboard (10...	OK
	0x0378-0x037F	Printer Port (LPT1)	OK
	0x03F8-0x03FF	Communications Port (COM1)	OK
	0x02F8-0x02FF	Communications Port (COM2)	OK
	0x03F2-0x03F5	Standard floppy disk controller	OK

Figura 3: Puertos asignados por dispositivo en el procesador Intel.



Sistemas Operativos 2

Unidad 2: Administración de dispositivos de E/S

Organización de la entrada salida

René Ornelis
Primer semestre 2023

Contenido

1	Introducción	4
2	E/S programada (<i>polling</i>).....	4
3	E/S dirigida por interrupciones	4
4	E/S con acceso directo a memoria	6
5	Configuraciones de DMA	6

Índice de figuras

Figura 1: Asignación de interrupciones	5
Figura 2: Flujo de acceso por interrupciones	5
Figura 3: Acceso por DMA.....	6
Figura 4: DMA de un solo bus.....	6
Figura 5: DMA con dispositivos integrados	7
Figura 6: DMA con doble bus	7

Organización de la entrada/salida

1 Introducción

La otra parte importante de la administración de dispositivos es la organización o la forma interactuar entre el procesador y el dispositivo. Recordemos que el objetivo de la administración de dispositivos es tener un balance entre el rendimiento del sistema y la usabilidad, por lo que es necesario que el sistema de administración de dispositivos sea eficiente, ya que los dispositivos son mucho más lentos que las operaciones de memoria, por lo que la forma en que el procesador va a leer y escribir de los dispositivos es determinante para la eficiencia.

La interacción entre el procesador y los controladores de los dispositivos puede ser de tres formas:

- E/S programada
- E/S dirigida por interrupciones
- E/S directa a memoria (DMA)

2 E/S programada (*polling*)

El procesador monitorea constantemente los dispositivos para determinar si están listos o contienen información disponible para procesar.

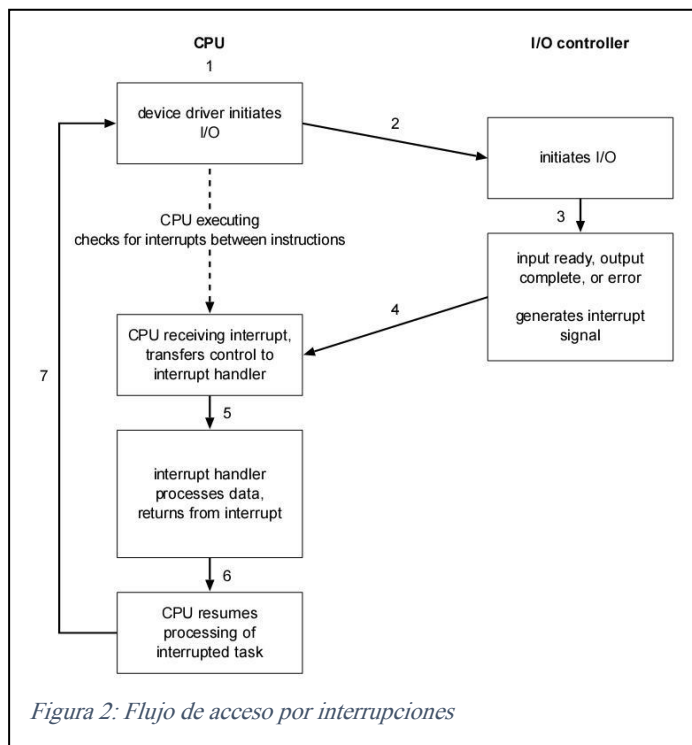
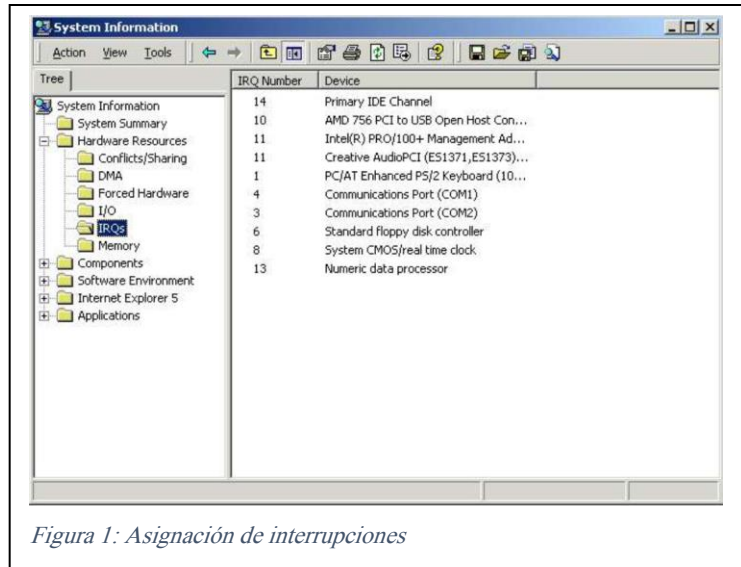
Esta organización es la utilizada en los primeros sistemas operativos, pero se discontinuó rápidamente ya que no era eficiente cuando se necesita el sistema para multitarea. Actualmente, este esquema es utilizada en sistemas muy especializados, por ejemplo: en los procesos industriales el sistema operativo está especializado en monitorear con algún sensor el funcionamiento de equipos, y notificar en caso de fallas o mal funcionamiento, como en el control de un sistema de aire acondicionado. También es el caso de los automóviles y sus computadoras de control, que un sistema operativo que, dicho de forma simplificada, está dedicado a leer los sensores, actualizar el tablero del piloto y tomar acciones, como detener el auto, en caso de condiciones críticas como cuando el motor empieza a calentarse más allá de los parámetros aceptables.

En resumen, este esquema puede resultar simple y eficiente si los dispositivos son de uso constante, o el procesador no tiene otras tareas que realizar.

3 E/S dirigida por interrupciones

La entrada y salida programada no es suficiente para los sistemas de uso general como las computadoras de escritorio o los celulares, por lo que rápidamente fue reemplazada por la E/S dirigida por interrupción. Esta organización consiste en la siguiente interacción entre el sistema y los dispositivos:

1. Se asigna a cada dispositivo un número de interrupción, tal como se aprecia en la
2. Cuando el dispositivo necesita la atención del sistema operativo, por ejemplo cuando el disco duro finaliza una escritura, genera una interrupción, según el número de interrupción asignado.
3. El sistema operativo transfiere el control de la interrupción al driver asignado
4. El driver transfiere la información desde/hacia el dispositivo hacia la memoria principal.
5. El driver decide si ya se finalizó la operación completa (que puede involucrar varias lecturas/escrituras). Si este es el caso envía una señal al proceso que solicitó la operación, si no, envía al dispositivo otra orden de lectura/escritura.
6. El procesador retoma sus operaciones y atiende otros procesos mientras el dispositivo realiza su operación.



Esta lógica se ilustra en la Figura 2.

De esta forma, se logra concurrencia entre las operaciones del procesador y las del dispositivo, ya que el procesador no está chequeando constantemente en el estado del dispositivo para determinar si la operación de entrada/salida finalizó.

Es imperativo que la rutina de interrupción que atiende a los dispositivos sean muy eficientes ya que durante su ejecución detiene al procesador, por lo que las tareas dentro de esta rutina deben reducirse al mínimo, como transferir la información a memoria y notificar al proceso interesado. Las demás tareas necesarias para que la información sea procesada por completo, por ejemplo mostrar en pantalla la información recibida del dispositivo, debe ser realizada por otros

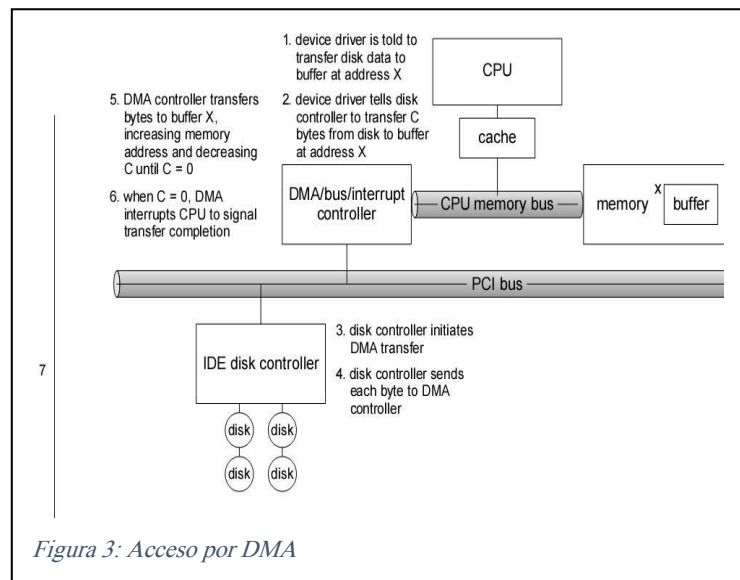
proceso, fuera de la rutina de servicio de interrupción.

4 E/S con acceso directo a memoria

El mecanismo de interrupciones puede funcionar muy bien para dispositivos de caracteres o de bloques pequeños. Sin embargo, para dispositivos de bloques grandes esto retardaría el tiempo de CPU en leer los datos del adaptador y pasarlos a la memoria del usuario, según la capacidad del bus de datos. Para liberar al CPU de esta carga de trabajo, se inventó el acceso directo a memoria o DMA (por sus siglas en inglés).

El DMA consiste en que el procesador envía los requerimientos de entrada salida al módulo de DMA indicándole, además de los parámetros del comando específico, la dirección de memoria donde se encuentra el bloque de datos a trabajar, el CPU continua su trabajo, mientras que el adaptador inicia el procesamiento de la petición en forma paralela.

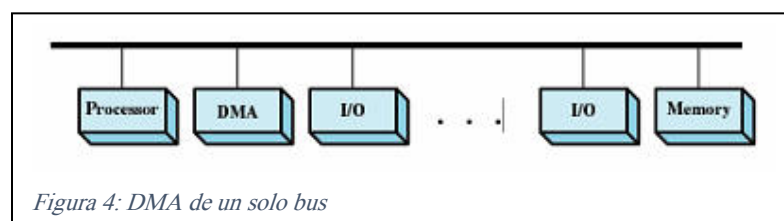
El módulo de DMA, se encarga de interactuar con el dispositivo en las siguientes operaciones de lectura/escritura y transferir los datos de/hacia la memoria principal, al adaptador. Cuando el adaptador termina su proceso, los datos ya están en la memoria del usuario y generará una interrupción para verificar el estado de la operación.



El acceso por DMA representó una mejora sustancial en el rendimiento del acceso a los dispositivos, por lo que si hay problemas de rendimiento de un disco duro, se debe verificar si el DMA está activado. Por ejemplo, Windows 98 en su configuración de fábrica no tenía habilitado el DMA, pero se podía habilitar con algunos cambios en la configuración. También puede suceder que un sistema moderno como Linux una detección incorrecta de un driver de disco puede funcionar con DMA deshabilitado.

5 Configuraciones de DMA

El diseño del DMA ha evolucionado en diferentes configuraciones de hardware a lo largo de la historia, que nos lleva a entender las ventajas y desventajas de cada una.



La primera configuración de DMA es la de bus simple, que se puede apreciar en la Figura 4, en la cual el procesador, el DMA, los dispositivos y la memoria comparten el mismo bus de datos.

Aunque funciona y es relativamente simple, tiene el problema que durante la transferencia de datos entre el dispositivo y el DMA, y este y la memoria, el procesador no puede acceder a la memoria, ya que el bus de datos se accede de forma exclusiva.

Para solucionar estos problemas, se utilizó un esquema de DMA con dispositivos integrados (ver Figura 5) en el que los diferentes módulos de DMA se conectan directamente con dispositivos, a través de canales específicos, de forma que la transferencia de información entre el DMA y el dispositivo ya no entra en conflicto con los accesos del procesador a memoria. El problema con este esquema es que, como podemos apreciar, hay varios módulos de DMA conectados con diferentes dispositivos y aunque se resuelve el problema de competencia de acceso al bus de datos en su mayor parte, resulta más caro (el DMA es un procesador especializado) por lo que no fue muy comercial.

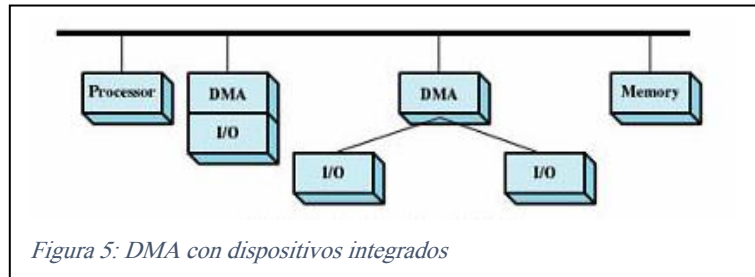


Figura 5: DMA con dispositivos integrados

Al final, estos problemas se resuelven con el esquema de doble bus que se muestra en la

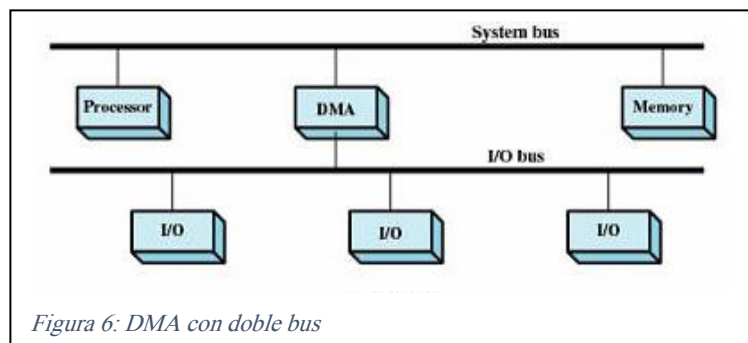


Figura 6: DMA con doble bus

Figura 6, en el cual además del bus del sistema, se agrega un bus adicional, el bus de entrada/salida, el cual conecta un único módulo de DMA a los dispositivos, mientras el procesador pues tiene completo acceso a la memoria a través del bus del sistema, lo que provoca menos conflicto que el esquema de un solo bus y no resulta tan caro como la arquitectura de múltiples

DMA con dispositivos integrados.



Sistemas Operativos 2

Unidad 2: Administración de dispositivos de E/S

Arquitectura de la entrada salida

René Ornelis
Primer semestre 2023

Contenido

1	Módulos del administrador de entrada/salida	4
1.1	Manejadores de interrupciones.....	4
1.2	Manejadores de dispositivos (driver)	5
1.3	Subsistema de gestión	5
1.4	Interfaz del SO	5
1.5	Aplicaciones del usuario	6
2	Control de acceso a dispositivos compartidos	6
3	Plug and Play (PnP)	7
4	Buffering	7
4.1	Orientaciones del buffering.....	8
4.2	Tipos de buffering	8

Índice de figuras

Figura 1: Arquitectura del administrador de dispositivos.....	4
Figura 2: SPOOL de una impresora.....	6
Figura 3: Código PnP de un dispositivo	7
Figura 4: Entrada/salida sin buffering.....	7
Figura 5: Entrada/salida con buffering simple.....	8
Figura 6: Buffering doble.....	8
Figura 7: Buffering múltiple	9

Arquitectura de la entrada/salida

1 Módulos del administrador de entrada/salida

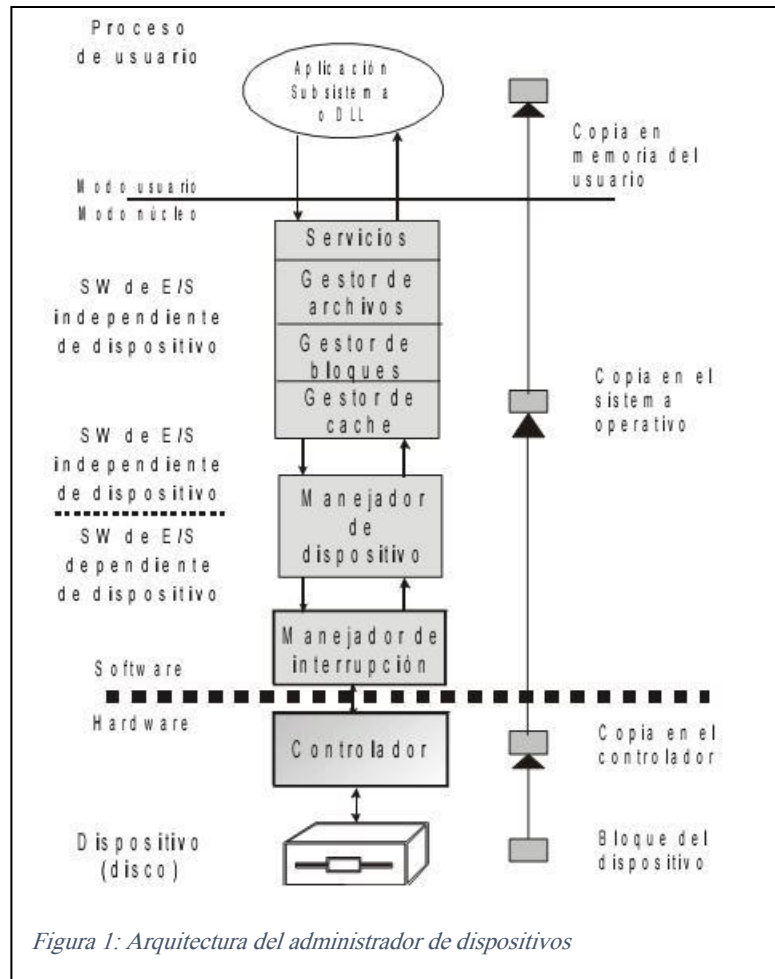
El sistema operativo o como cualquier pieza de software es una pieza de software compleja la cual se tiene que estructurar en módulos que hagan eficiente su administración. Adicionalmente recordemos que el sistema operativo debe de buscar el principio de independencia de los dispositivos por lo que esta arquitectura interna también debe orientarse a este objetivo. Como se ve en la Figura 1, en general los sistemas operativos estructura sus módulos de siguiente manera (de abajo hacia arriba):

- Manejador de interrupciones
- Manejadores de dispositivos
- Subsistema de gestión
- Interfaz del SO
- Aplicaciones del usuario

1.1 Manejadores de interrupciones

Son el nivel más bajo de la arquitectura. El módulo de administración de interrupciones es el que controla la asignación de interrupciones por dispositivo de forma que cuando recibe una interrupción, invoca la rutina de servicio definida por el driver registrado para ese dispositivo. Este módulo está muy vinculado al hardware ya que tiene que tener en cuenta las características de cada procesador y dispositivo, por lo que decimos que es la parte de la arquitectura *dependiente del dispositivo*.

Este módulo está obligado a correr en modo kernel (modo supervisor o modo privilegiado), ya que necesita tener acceso a todo el hardware, la memoria (con el MMU deshabilitado) y los puertos de entrada/salida.



1.2 Manejadores de dispositivos (driver)

Tienen el código específico para cada dispositivo. Su función es recibir información del dispositivo y trasladarla a las capas superiores. Asimismo, acepta solicitudes generales del sistema operativo y asegurarse que se cumpla la requisición. También convierte los datos del dispositivo al formato general del SO.

El driver marca la línea divisoria entre el software dependiente del dispositivo y el software independiente del dispositivo. Parte del driver y el manejador de interrupciones conforman la parte dependiente del dispositivo, ya que ambos módulos deben trabajar con las características de los dispositivos y el procesador y se debe manejar los comandos específicos para cada dispositivo. Otra parte del driver debe trabajar con comandos genéricos del sistema operativo y forma parte del software independiente del dispositivo.

1.3 Subsistema de gestión

Este subsistema está conformado por todos los módulos necesarios para la administración genérica de los dispositivos. Por ejemplo la administración de bloques, el sistema de archivos la administración de la caché, las colas de prioridades, el buffering y etcétera. que trabajan y administran sin interactuar directamente con los dispositivos ya que esto se hace a través del driver. Este subsistema es parte del software independiente del dispositivo

1.4 Interfaz del SO

Es la parte del sistema operativo por medio de la cual presenta a los procesos los servicios para que estos puedan hacer uso de los recursos del sistema (dispositivos). El API de todo sistema operativo debe buscar ser independiente del dispositivo, de forma que las complejidades de cada dispositivo quedan ocultas dentro de la arquitectura del sistema operativo.

Por ejemplo, en el caso de acceder un archivo:

1. El proceso, a través del API del sistema le indica que quiere leer un bloque de 10k del archivo c:\texto.txt. Estos parámetros son enviados como un código de servicio (leer de archivo), nombre del archivo y buffer donde el proceso requiere que quede la información leída.
2. A través del sistema de archivos convierte el nombre de archivo (c:\texto.txt) en una dirección de bloque dentro del disco en forma de cilindro-cabeza-sector.
3. Se revisa si la información correspondiente ya está en la memoria cache. De ser así, ya no se lee del dispositivo, sino que se toma de la memoria cache y se transfiere al proceso en la dirección de memoria solicitada por éste.
4. Si la información no está en caché, el sistema prepara un buffer para recibir la información y agrega el requerimiento a la cola de atención del driver.
5. El driver toma el requerimiento y lo convierte en instrucciones específicas para el disco y el DMA.
6. El disco trabaja en el requerimiento, en conjunto con el DMA, y cuando finaliza la petición, el DMA genera la interrupción de notificación de aviso.
7. El driver informa a las capas superiores de la finalización del requerimiento

8. El módulo de gestión de buffers traslada la información en el buffer del kernel (con la información leída) y lo traslada al buffer del proceso
9. Se le notifica al proceso que la petición está completa.

1.5 Aplicaciones del usuario

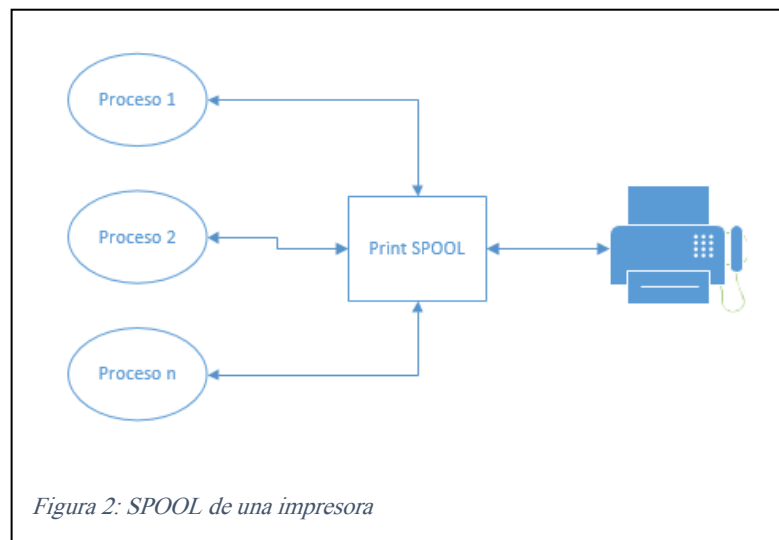
Las aplicaciones del usuario, tiene acceso al API del SO para trabajar con los dispositivos de E/S. Sin embargo, existen otras aplicaciones como los **demonios**, o **bibliotecas** que hacen procesos de **manejo por cola**, como colas de impresión, servidores de correo, etc.

2 Control de acceso a dispositivos compartidos

Como se ve en el estudio de los interbloqueos, existen recursos (dispositivos) son exclusivos e inapropiativos, es decir que el dispositivo sólo puede ser utilizado por un proceso a la vez (exclusivo) y el sistema no desasignar el recurso dado a un proceso (inapropiativo). Estos dispositivos representan para el programador y para el sistema operativo complejidad adicional, tanto para el control del interbloqueo como para la coordinación del acceso. Por eso, gran parte de la arquitectura del sistema de entrada salida está basada en abstraer y ocultar de los procesos el uso compartido de dispositivos. En general, cualquier dispositivo será visto por los procesos como un dispositivo compartido, ya que no tiene que coordinar con otros procesos es acceso al mismo, sino solo enviar un comando y esperar por la respuesta.

Si los procesos accedieran directamente a los dispositivos tendrían que coordinar con los demás procesos el acceso. Por ejemplo: acceder a una impresora implica que una vez concedido a un proceso y este inicie la impresión, los demás procesos deberán esperar hasta que el proceso favorecido termine de imprimir. Esto era evidente en sistemas operativos simples como el DOS donde cada proceso accedía directamente a la impresora y el usuario no podía realizar nada más ya que este sistema carecía de la habilidad de procesos concurrentes.

Este concepto es el SPOOL (del inglés Simultaneous Peripheral Operation Online) que consiste en que el API del sistema operativo presenta a los procesos un servicio compartido y este servicio es el único que accede al dispositivo exclusivo. Por ejemplo, en caso de una impresora (ver Figura 2) los procesos no acceden a la impresora, sino que invocan al servicio de SPOOL y este es el que se encarga de ordenar los requerimientos, priorizar y organizar la impresión. Al mismo tiempo reporta a los procesos que el requerimiento fue recibido, por lo que estos pueden continuar con otros trabajos.



Este concepto se puede generalizar a cualquier dispositivo y tiene la ventaja de que se pueden implementar más fácilmente las políticas de acceso a los dispositivos.

3 Plug and Play (PnP)

Es una especificación de hardware/software que permite que un dispositivo conectado pueda funcionar sin la intervención del usuario. Cuando se conecta un dispositivo el sistema procede así:

1. El sistema detecta el nuevo dispositivo y recupera su identificación. Cada dispositivo PnP tiene un identificador único (PnP Id) que es la combinación de un código de fabricante y código de producto (ver Figura 3). El código de fabricante es concedido por la organización Unified Extensible Firmware Interface Forum (uefi.org)
2. El sistema asigna (o reasigna) dinámicamente los recursos requeridos por el dispositivo como puertos, interrupciones, canales de DMA y memoria dedicada.
3. Con la identificación del dispositivo se busca en la base de datos de drivers del sistema y se instala y configura el driver adecuado.

De esta forma, como usuarios hemos observado que cuando se conecta un nuevo dispositivo, se ve brevemente actividad en la bandeja del sistema y luego obtenemos un mensaje de que el nuevo dispositivo está listo para ser utilizado.

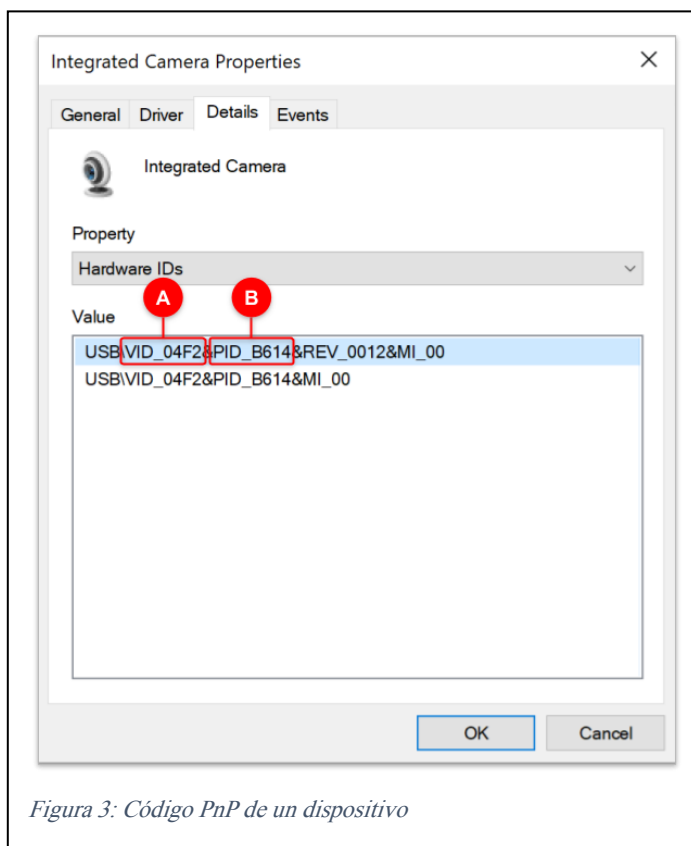


Figura 3: Código PnP de un dispositivo

4 Buffering

Tal como se indicó anteriormente, el subsistema de gestión de la entrada/salida, incluye el manejo de los buffers del sistema. Recordemos que cuando un proceso solicita una operación de entrada salida uno de los parámetros es la dirección de memoria donde está la información a

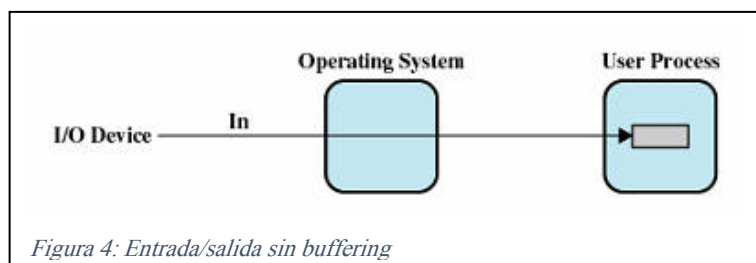


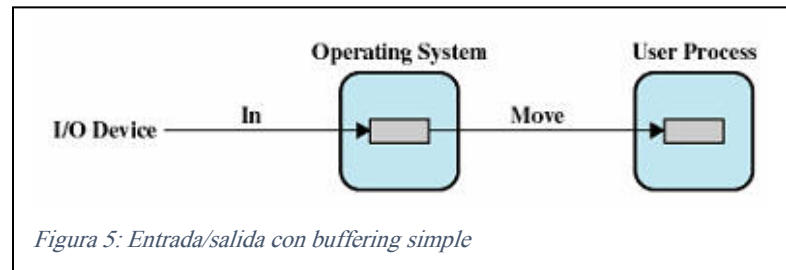
Figura 4: Entrada/salida sin buffering

escribir o donde se solicita que queden los datos leídos. Este buffer del proceso está en la **memoria del proceso** y es una dirección lógica.

Tal como se ve en la Figura 4, si la operación de entrada salida se realiza sin la utilización de buffering, implica que la página del proceso debe ser excluida del swapping de memoria virtual. Recordemos que las páginas de los

procesos están dentro de los marcos de página, que es el área designada para uso de memoria virtual. Aunque una página puede ser marcada como “siempre en memoria”, supone o un desperdicio de espacio, si el buffer no ocupa toda la página, o un aumento de la hiperpaginación al descartar de la política de reemplazo la página (o páginas) donde reside el buffer.

Por esto es necesario que en toda operación de entrada salida, el DMA/dispositivo transfieran la información a un buffer del sistema operativo, y al finalizar se traslada a la memoria del proceso, tal como se muestra en la Figura 5. Esto tiene la ventaja que, dado que los procesos deben esperar hasta que las operaciones de E/S terminen, el proceso puede ser suspendido y sus páginas descargadas a disco sin problema, hasta que finalice la operación de entrada/salida.



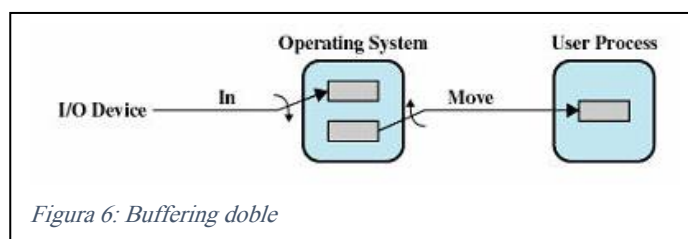
4.1 Orientaciones del buffering

El manejo del buffering depende en gran manera del tipo de dispositivo para el cual se utilice, ya que la complejidad de manejo varía entre los dispositivos de bloque y los dispositivos carácter.

El manejo de un buffer para un dispositivo de bloque tiene la ventaja que los bloques son de tamaño fijo y se maneja un bloque a la vez. Por su parte el buffer para un dispositivo de carácter debe manejar un flujo de datos constante, lo cual no se puede manejar como un solo bloque, por lo que lo usual es utilizar buffer como una cola circular de información. En este caso, la rutina de servicio de interrupción solo ingresa información al buffer, mientras que otra parte del driver se encarga leer de la cola y dar el debido proceso. Esto representa un problema de productor-consumidor, con la complejidad de que la rutina de servicio de interrupción no puede utilizar ningún tipo de coordinación interproceso (IPC) como semáforos, monitores y otros, ya que de ninguna forma puede suspenderse y la información en la cola debe mantenerse íntegra.

4.2 Tipos de buffering

Ya vimos la necesidad de mantener un buffering para el manejo de la entrada/salida y con ello, el primer tipo de buffering; **buffering simple**, el cual funciona y permite que los procesos requirientes puedan ser suspendidos mientras finaliza la operación de entrada/salida. Sin

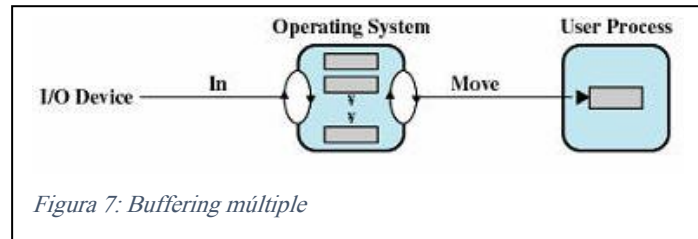


embargo, mientras el sistema transfiere la información del buffer del sistema a la página del proceso (implica posiblemente subir la página del proceso desde disco), no se puede realizar otra operación con el dispositivo. Entonces surge la idea de utilizar un **buffering doble**, el cual, tal

como se ilustra en la Figura 6, es conformado por un buffer de entrada y un buffer de salida. Así, un dispositivo puede iniciar otra operación de entrada/salida mientras la información de la operación anterior es transferida a la memoria del proceso solicitante, lo cual mejora el rendimiento de la entrada/salida.

Este mismo concepto, se puede expandir a **buffering múltiple**, el cual consiste en múltiples espacios de memoria que se utilizan como buffer para la operación con un dispositivo (ver Figura 7) los cuales se utilizan de forma cíclica.

Este esquema es muy útil en caso de dispositivos con alta demanda, como un disco duro, el cual que atiende a muchos procesos, por lo que puede estar transfiriendo información a muchos procesos mientras realiza una operación de entrada/salida.





Sistemas Operativos 2

Unidad 2: Administración de dispositivos de E/S

Dispositivos especiales

René Ornelis
Primer semestre 2023

Contenido

1	Introducción	4
2	El reloj.....	4
2.1	Señal de reloj del procesador	4
2.2	Reloj del sistema	4
2.3	Temporizadores.....	5
2.3.1	Mantenimiento de fecha y hora.....	5
2.3.2	Estadísticas.....	6
2.3.3	Gestión de temporizadores.....	6
2.3.4	Soporte para la planificación de procesos.....	8
3	La terminal	8
3.1	El teclado.....	9
3.2	El monitor.....	9
3.2.1	Modo texto	9
3.2.2	Modo gráfico.....	9
4	La red	11

Índice de figuras

Figura 1: Cola del temporizador	7
Figura 2: Cola para múltiples temporizadores	7
Figura 3: Múltiples colas de temporización	7
Figura 4: Arquitectura de la terminal	8
Figura 5: Configuraciones del monitor	10
Figura 6: Configuraciones de la tarjeta de video	11
Figura 7: API para sockets	12
Figura 8: Estados de un socket	12

Dispositivos especiales

'''

1 Introducción

Aunque existe una clasificación de dispositivos y una abstracción de los componentes generales de un dispositivo, existen dispositivos que no se les podría clasificar fácilmente dentro de estas categorías por sus características especiales. En este capítulo veremos los dispositivos que, por sus funcionalidades únicas, tienen un tratamiento especial y no pueden utilizarse como el resto de los dispositivos. La existencia de estos dispositivos hace difícil aplicar el principio de independencia del dispositivo, por lo que la mayoría de los fabricantes de sistemas operativos optan por aplicar un API especial a cada uno de estos.

Los dispositivos que estudiaremos son:

1. El reloj
2. La terminal
3. La red

2 El reloj

El concepto de reloj tiene varias acepciones en el ámbito de una computadora:

- Señal de reloj del procesador
- Temporizadores
- Reloj del sistema

2.1 Señal de reloj del procesador

Los procesadores cuentan con un circuito temporizador que genera señal periódica llamada el reloj o reloj del sistema, el cual es la base de todas las microoperaciones del procesador y determina la velocidad de este. Este circuito normalmente está fuera del ámbito del sistema operativo y lo único que se puede hacer es **acelerar al procesador** (*overclocking*), pero esto se realiza con herramientas específicas para cada procesador.

2.2 Reloj del sistema

También denominado reloj BIOS, es el reloj digital que mantiene la fecha y hora en el sistema. Este reloj es alimentado por batería, la cual se recarga cuando la computadora está encendida, por lo que en caso de que una computadora pasa mucho tiempo sin encenderse, al inicio puede solicitar establecer la fecha y hora.

Al encender la computadora, una de las primeras tareas del sistema operativo es leer la fecha y hora del BIOS y guardarla para posteriormente mantener el control (ver siguiente sección).

2.3 Temporizadores

Todo procesador tiene un conjunto de registros temporizadores, que tienen la capacidad de generar interrupciones basado en el valor de tiempo que se les asigne. Uno de estos temporizadores es la **interrupción de tiempo** o **interrupción del sistema**, el cual está conectado a una línea de interrupción de hardware de alta prioridad, y se puede programar en determinada frecuencia para que el sistema operativo pueda realizar determinadas tareas.

La frecuencia con que se programe la interrupción de tiempo depende del sistema operativo y del hardware. Por ejemplo: Linux programa la frecuencia en 100 y 1000 hz. El sistema operativo, basado en el conocimiento del procesador debe elegir esta frecuencia cuidando tener un balance adecuado entre la exactitud de sus cálculos y el rendimiento del sistema. Así, si se tiene una frecuencia alta de interrupción, como 1000 veces por segundo, los controles que haga el sistema en cada interrupción serán más precisos, pero tendrán un impacto directo en el rendimiento del sistema.

Las tareas usuales que realiza el sistema operativo en la interrupción del sistema son:

- Mantenimiento de la hora del sistema
- Muestreo del uso de procesador para estadísticas

2.3.1 Mantenimiento de fecha y hora

El sistema operativo no depende del reloj del BIOS para controlar la fecha y del sistema. Solamente en el arranque del sistema, se lee el reloj del BIOS, lo almacena en memoria y en la interrupción del sistema lo incrementa según el valor de la frecuencia de interrupción. De esta forma el sistema operativo mantiene el control de la fecha y hora. Al apagar el sistema, el valor de la fecha y hora se actualiza de vuelta al reloj del BIOS.

La mayoría de sistemas operativos mantienen el reloj del BIOS en el horario universal coordinado (UTC), lo que para mantener la hora del sistema tomando en cuenta la configuración de zona horaria y políticas de cada país respecto al horario de verano.

La forma de calcular el tiempo es normalmente un número entero con la cantidad de segundos, milisegundo u otra unidad, a partir de una fecha determinada. Por ejemplo: UNIX maneja la cantidad de segundos desde el 01-01-1970 00:00. Otras variantes de UNIX usan la cantidad de milisegundos desde la misma fecha. En el caso almacenar segundos, una variable de 32 bits alcanza para 136 años. Si utilizamos una variable de 64 bits, se pueden almacenar más de 586 mil millones de años. Por otro lado, Windows almacena la fecha en centenas de nanosegundos (1/10,000,000), lo que permite más 58,000 años en una variable de 64 bits. El valor de esta variable es la que sirve de base para calcular la fecha y hora del sistema en el calendario gregoriano o cualquier otro calendario que se tenga configurado, lo cual es una operación relativamente complicada.

Adicional al mantenimiento automático, el sistema operativo también debe permitir al administrador cambiar la hora del sistema, lo cual se realiza de dos formas:

- Interfaz para cambiar la fecha y hora, lo cual usualmente solo se da acceso al superusuario.
- Actualización a través de servicios de red (protocolo ntp)

En ambos casos, el administrador debe tener cuidado, ya que el cambio de fechas muy adelante o muy hacia atrás, puede provocar efectos indeseados debido principalmente a tareas calendarizadas, que se dispararían muchas al mismo tiempo si se adelanta la hora, o que ya no se realizarán si se atrasa. También puede haber implicaciones en vencimientos de licencias o de programas que están programados con codificación fija para funcionar en determinado período de tiempo.

Es por esto que **ntp** no permite la sincronización si la diferencia entre la hora del servidor y la hora de la computadora es más de 1000 segundos (configurable). En todo caso, si una computadora se desvía mucho de la hora (por ejemplo, si pasó mucho tiempo apagada), lo mejor es hacer cambios incrementales hasta tener la hora exacta.

2.3.2 Estadísticas.

Otras de las funciones en que la interrupción del sistema ayuda es en la obtención de estadísticas de uso del sistema. Por ejemplo: el porcentaje de uso del procesador, de forma simplificada, se obtiene así:

1. La rutina de interrupción usa dos variables: número de muestras y número de esas muestras en la que el procesador estaba en uso. En cada interrupción, el número de muestra se incrementa incondicionalmente y si hay un proceso que esté en uso del procesador (durante la interrupción) entonces se incrementa la segunda variable.
2. Otros procesos se despiertan periódicamente para revisar la información y realizar los cálculos correspondientes. En este caso, el porcentaje de uso del procesador se calcula así:

$$\text{Porcentaje de uso del procesador} = \frac{\text{Muestras con uso de procesador}}{\text{Número de muestras tomadas}}$$

Claro que la implementación es un poco más complicada, ya que este porcentaje se determina por proceso y, además del uso del procesador se controlan otros indicadores como el uso del disco duro, uso de la red, etc. Sin embargo, el punto es que la rutina de interrupción no realiza los cálculos de las estadísticas, sino que solo cuenta los eventos y son otros procesos del kernel los que se encargan de hacer los cálculos correspondientes, ya que la rutina de interrupción debe ser lo más breve posible.

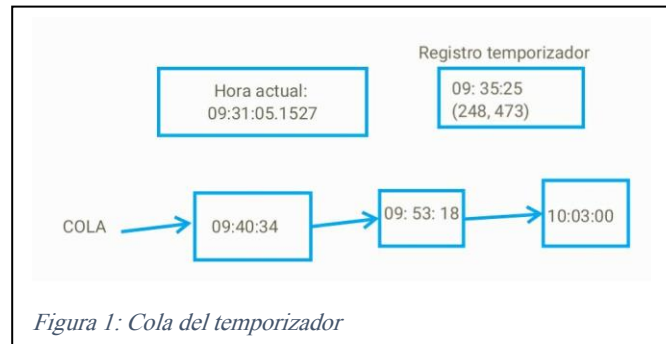
2.3.3 Gestión de temporizadores

Además de la interrupción del sistema por medio del registro temporizador de máxima prioridad, los procesadores suelen tener más registros temporizadores los cuales se utilizan con menor prioridad, pero nos sirven para las necesidades de temporización de los procesos como las agendas con avisos, los tiempos de espera de la red, cronómetros, alarmas, etc.

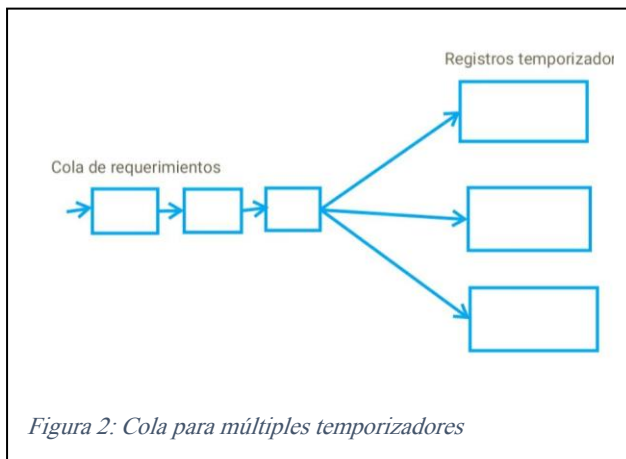
El sistema operativo logra atender las necesidades de temporización de todos los procesos a través del manejo de una cola de requerimientos (ver Figura 1), la cual está ordenada por la hora en la que el proceso necesita ser notificado. Los requerimientos de los procesos se muestran en hora, minutos, segundos y milisegundos, solo con fines didácticos, ya que usualmente el valor requerido se suele expresar en milisegundos.

Suponiendo que solo existe un registro temporizador, en el ejemplo se muestra que el requerimiento actual la hora el sistema es la 09:32:05.1527 y se atenderá una temporización para las 09:35:25, lo cual significa 248,473 milisegundos.

Cuando el temporizador llegue a cero, el sistema notificará al proceso que hizo el requerimiento y tomará el próximo requerimiento de la cola, que corresponde a las 09:40:34, lo que en su momento corresponderá a 309,000 milisegundo (el lector verifique el cálculo), y se repetirá el proceso.



Si el procesador cuenta con más de un registro temporizador, este mismo concepto se puede expandir a manejar una cola para varios temporizadores, tal como se muestra en la Figura 2, donde el primer temporizador que finalice su requerimiento tomará el siguiente elemento de la cola y lo atenderá.

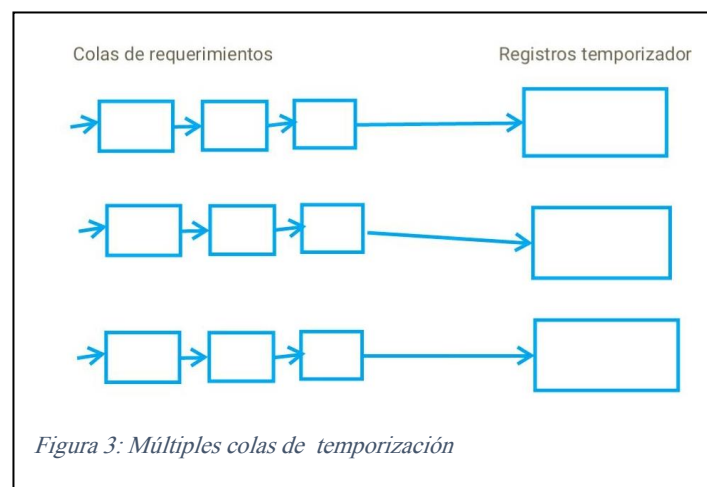


Esto tiene la ventaja que se aprovecha al máximo todos los temporizadores existentes ya que mientras haya requerimientos en cola, los registros temporizadores estarán trabajando.

Por otro lado, puede ser que el sistema operativo implemente políticas orientadas a especializar los registros temporizadores en ciertas tareas. Por ejemplo: un registro

temporizador para las tareas del kernel, otro para las necesidades de hardware (como espera de red, discos, etc.) y otro para atender los requerimientos de los procesos de usuario. En este caso, se deberá manejar una cola por temporizador según se muestra en la Figura 3, de tal forma que cada temporizador tiene su propia cola de requerimientos.

Para que los procesos puedan realizar las operaciones con temporizadores, el sistema operativo debe proporcionar una API especial el cual contiene las operaciones necesarias. En POSIX (la especificación de los UNIX y LINUX), la gestión de temporizadores está definida con las siguientes cinco primitivas:



Nombre	Descripción
Timer_create	Crea un temporizador
Timer_delete	Elimina un temporizador
Timer_getoverrun	Cuenta de desbordamiento
Timer_gettime	Tiempo restante
Timer_settime	Programa y arranca el temporizador

2.3.4 Soporte para la planificación de procesos

El planificador utiliza el reloj del sistema para determinar el tiempo adecuado para otorgar y quitar el procesador a cada proceso. Una parte del trabajo se realiza en la rutina de interrupción donde solo se cuentan los “ticks” que lleva activo el proceso que está en uso del procesador. Por otra parte, los otros procesos de planificación se encargarán de definir si al proceso activo le corresponde quitarle el procesador y dárselo a otro proceso, luego de tomar en cuenta el tiempo que lleva activo, la prioridad y otras consideraciones de la política de planificación.

Tradicionalmente la planificación de procesos y estadísticas se ha realizado a través de la interrupción periódica del reloj, o resolución del reloj (100hz-1000hz), pero en el kernel 2.6.21 de linux ya se implementa a través de temporizadores (*dynticks= dynamic ticks*), lo que supone mejoras en uso del CPU para ahorro de energía y optimización dentro de un hipervisor.

3 La terminal

Históricamente se le denomina terminal a la combinación de teclado y pantalla, pues las primeras terminales eran un solo dispositivo serial (terminal tonta) que recibía entradas y salida y se proyectaba en pantalla.

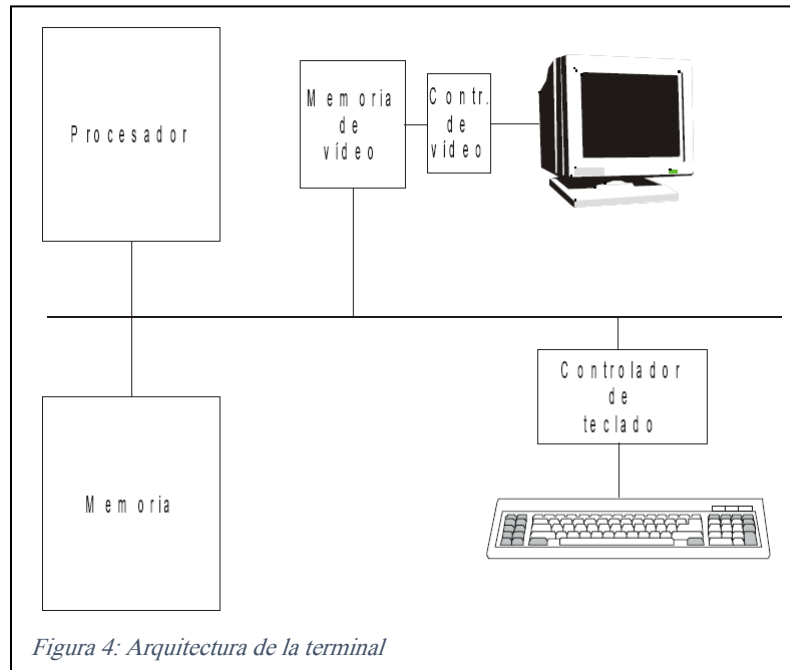


Figura 4: Arquitectura de la terminal

Recordemos que estos solían ser pantallas de modo texto donde no existía el posicionamiento del cursor sino solamente el despliegue secuencial de información. Actualmente los sistemas UNIX conservan el dispositivo `/dev/ttyX` los cuales se tratan como terminales que se pueden utilizar para la pantalla + teclado o para sesiones remotas.

Tal como se puede apreciar en la Figura 4, actualmente tanto el teclado como la pantalla se tratan separadamente. El teclado se ve como un dispositivo de interfaz, de carácter, proyectado en

puertos, mientras que la pantalla es un dispositivo de interfaz, de bloques, proyectado en memoria.

3.1 El teclado

- Entrada
 - Código de tecla (scan code) ==> Carácter ASCII
 - Se tiene en cuenta teclas modificadoras (Control, Alt, ...)
- Teclado genera interrupción al pulsar tecla
 - S.O. lee código de tecla de registro de controlador de teclado
 - Conversión a ASCII y manejo de teclas modif. por SW
 - Manejador proporciona “teclado anticipado” (type ahead)
 - Usuario teclea info. antes de que programa la solicite
 - Manejador debe usar zona de almacenamiento intermedio
 -
 -

3.2 El monitor

- Dependiendo de tipo de información manejada:
 - Terminales en modo alfanumérico
 - Terminales en modo gráfico

3.2.1 Modo texto

- Alfanumérico (texto):
 - La memoria de vídeo se trata como un arreglo de 80*25, contiene códigos ASCII, y atributos como colores y blink
 - Controlador de vídeo genera patrones de pixels

3.2.2 Modo gráfico

- Matriz de pixels con memoria de vídeo asociada
 - Memoria de vídeo directamente accesible al procesador
 - Cada posición de memoria (1 pixel), tiene el código del color de dicho pixel
 - La memoria de video total necesaria está dada por la resolución del monitor: filas x columnas x bytes de cada pixel
- Controlador de vídeo lee esta memoria y refresca pantalla
- Escritura en pantalla requiere escritura en memoria de vídeo
- El sistema debe transformar de código ASCII (o UTF) en el patrón de pixels correspondiente
 - canvas (Windows)
 - fonts



Settings

⌂ Advanced display settings

Choose display

Select a display to view or change the settings for it.

Display 1: Internal Display 

Display information



Internal Display

Display 1: Connected to Intel(R) HD Graphics 620

Desktop resolution	1366 × 768
Active signal resolution	1366 × 768
Refresh rate (Hz)	60.075 Hz
Bit depth	6-bit
Color format	RGB
Color space	Standard dynamic range (SDR)
Display adapter properties for Display 1	

Figura 5: Configuraciones del monitor

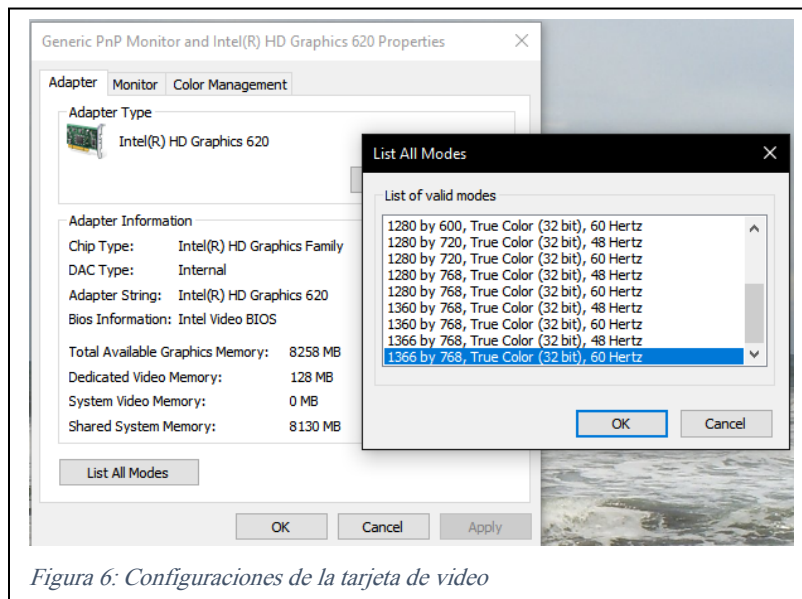


Figura 6: Configuraciones de la tarjeta de video

4 La red

Dispositivos de comunicación / de caracter

- Dada su creciente importancia, soporte de S.O. cada vez mejor
- API especial para la red (SOCKETS), debido a varios factores:
 - No se puede tratar como un archivo más
 - Direcciones de otras máquinas
 - Protocolo de comunicación
 - Timeout
 - Retry
- Sockets: se basa en las siguientes funciones

Primitives	Meaning
SOCKET	Create a New Communication Endpoint.
BIND	Attach a Local Address to a SOCKET.
LISTEN	Shows the Willingness to Accept Connections.
ACCEPT	Block the Caller until a Connection Attempts Arrives.
CONNECT	Actively Attempt to Establish a Connection.
SEND	Send Some Data over Connection.
RECEIVE	Receive Some Data from the Connection.
CLOSE	Release the Connection.

Figura 7: API para sockets

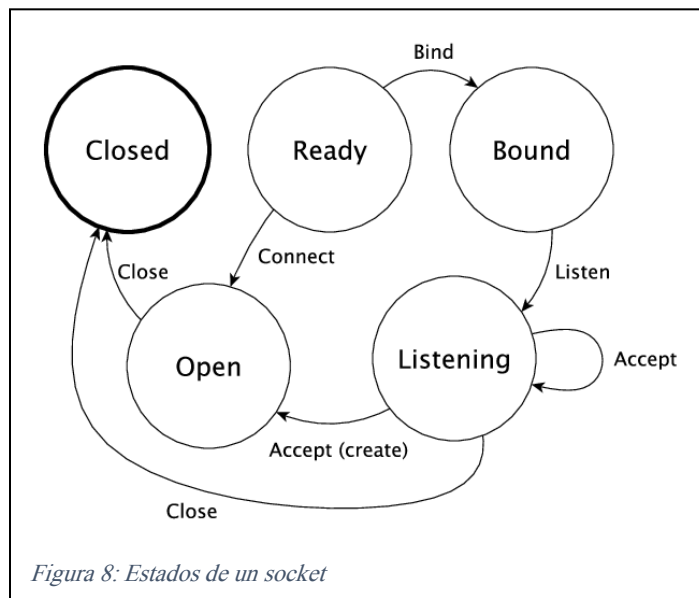


Figura 8: Estados de un socket

- Software de red organizado en tres niveles:
 - Nivel de interfaz a las aplicaciones
 - Típicamente, sockets (Winsock en Windows)
 - Puede considerarse como nivel de sesión OSI
 - Nivel de protocolos
 - Capa(s) que implementa(n) transporte y red OSI (o TCP/IP)
 - Incluye funciones de encaminamiento
 - Nivel de dispositivo de red
 - Manejadores de dispositivos de red (nivel de enlace OSI)