

CONCEPTOS IMPORTANTES

Un computador tiene cierto espacio de **Memoria Principal(RAM)** que utiliza para **mantener los programas en ejecución**. En los computadores modernos se pueden colocar varios programas en memoria al mismo tiempo.

En el caso más simple, la máxima cantidad de espacio de direcciones que tiene un proceso es menor que la memoria principal.

Hoy en día existe una técnica llamada **Memoria Virtual** en la que el SO mantiene una **parte del espacio de direcciones en Memoria Principal y otra parte en Disco**.

Memoria virtual

La memoria virtual proporciona la habilidad de ejecutar programas más extensos que la Memoria Principal (también llamada Memoria Física o RAM) de la computadora, trayendo pedazos entre la RAM y el Disco. También permite ligar dinámicamente bibliotecas en tiempo de ejecución (ya que las tiene en el Disco pero virtualmente ligadas dinámicamente).

Llamadas al sistema

FUNCIONES DEL SISTEMA OPERATIVO

- Proveer abstracciones a los programas de usuario
- Administrar recursos de la computadora

La **llamada al sistema** generalmente es el mecanismo usado por una aplicación para **solicitar un servicio al sistema operativo**. Normalmente usan una instrucción especial de la CPU.

Cuando una **Llamada al Sistema es invocada, la ejecución del programa es interrumpida** y sus datos son guardados para poder ejecutarse luego.

Para hacer más entendible el mecanismo de llamadas al sistema, vamos a echar un vistazo a la llamada **read**

```
cuenta = read(fd, buffer, nbytes);
```

La llamada devuelve el nº de bytes que se leen en *cuenta* (por lo general el valor de *cuenta* es el mismo que *nbytes* pero puede ser menor si se encuentra el fin del archivo al estar leyendo).

Si hay un error *cuenta* se establece a '-1' y se coloca el error en '*errno*'

Centrandonos más en el ejemplo...

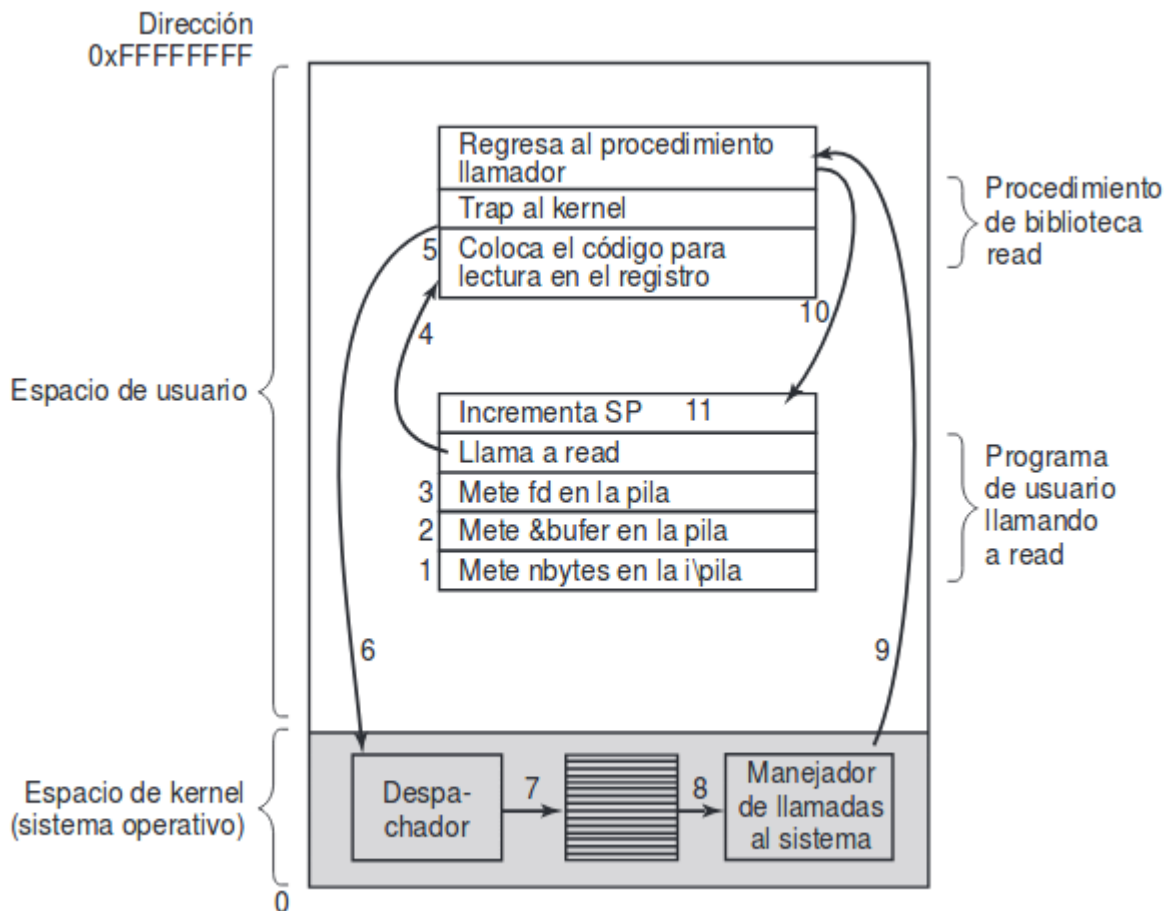


Figura 1-17. Los 11 pasos para realizar la llamada al sistema `read(fd, bufer, nbytes)`.

1. Se pasan los parámetros de la función como si fuera una pila (de ahí que estén en orden inversa). Además, en el caso de 'read', el segundo parámetro se pasa como referencia, al contrario que los otros, que se pasan como valor. Por esa razón hay que indicar 'contenido de bufer' con el &.
2. Llamada al procedimiento de biblioteca (seguimos en espacio de usuario, pero ya no en el espacio del programa llamador sino en la biblioteca llamada).
3. La biblioteca coloca por lo general el nº de la llamada al sistema en un lugar en el que el SO lo espera para buscarlo en la tabla de manejadores, por ejemplo lo guarda en un registro.
4. Se ejecuta un TRAP para cambiar a modo kernel. A diferencia de la anterior llamada, la instrucción TRAP no salta a una dirección relativa donde se encuentre un procedimiento (como la biblioteca) sino que salta a una dirección fija en el espacio kernel. Por lo que se salta a una dirección arbitraria donde hay un campo de 8 bits que proporciona un índice en una tabla de saltos.
5. El kernel examina el nº de llamada al sistema y le pasa al manejador correspondiente (elige el manejador mediante una tabla de apuntadores a manejadores indexados en una tabla según el nº de llamada al sistema que se ha pedido).
6. Se ejecuta el manejador de llamadas al sistema (que hace su trabajo, el que tenga que hacer según la llamada requerida).
7. Se regresa al procedimiento de biblioteca que está en espacio de usuario
8. Por último regresa en la forma usual a la que regresan las llamadas a procedimientos.

Por último el programa usuario limpia la pila.

Preludio

El trabajo del SO es abstraer una jerarquía de memoria en un modelo útil y administrarla; la parte que hace esto se llama **Administrador de Memoria**. En este capítulo nos concentraremos en el modelo del programador de la Memoria Principal (RAM) y cómo se puede administrar.

Sin Abstracción de Memoria

Cada programa veía simplemente la RAM: Cuando un programa veía la instrucción

```
MOV REGISTRO1, 1000
```

la computadora movía el contenido **100** a **REGISTRO1** dentro de la Memoria Física (RAM).

De esta forma **no había manera de tener 2 programas ejecutándose a la vez**. Ya que se borrarían mutuamente si intentaran acceder a la misma dirección mientras están en ejecución.

Los modelos (a) y (c) tienen la desventaja de que un error en el programa de usuario puede borrar el SO (al estar en Memoria de Escritura):

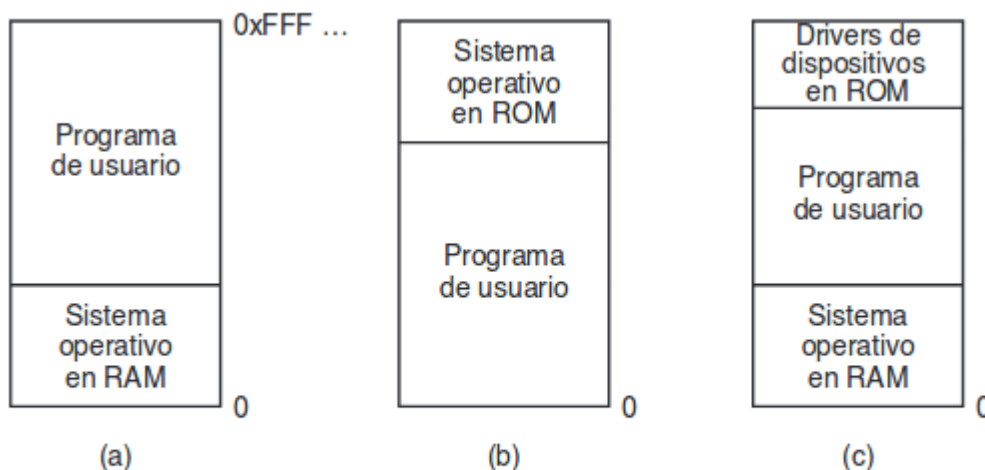


Figura 3-1. Tres formas simples de organizar la memoria con un sistema operativo y un proceso de usuario. También existen otras posibilidades.

Ejecución de múltiple programas sin abstracción

El programa hace un **swap** del programa que está en RAM a un archivo a **Disco**, más tarde cuando vuelve al programa lo vuelve a traer. A esto se le llama **Intercambio**

Con la adición de cierto hardware especial es posible ejecutar múltiples programas concurrentemente aún sin intercambio, que se realiza como marca la siguiente imagen. De tal forma que en la **Palabra de Estado del Programa** también contenía una llave de 4 bits de tal forma que no pudieran acceder a sus direcciones a no ser que tuvieran la llave (y un proceso no podía tener la llave del otro) .

Sin embargo esto trae un problema como también se puede comprobar...

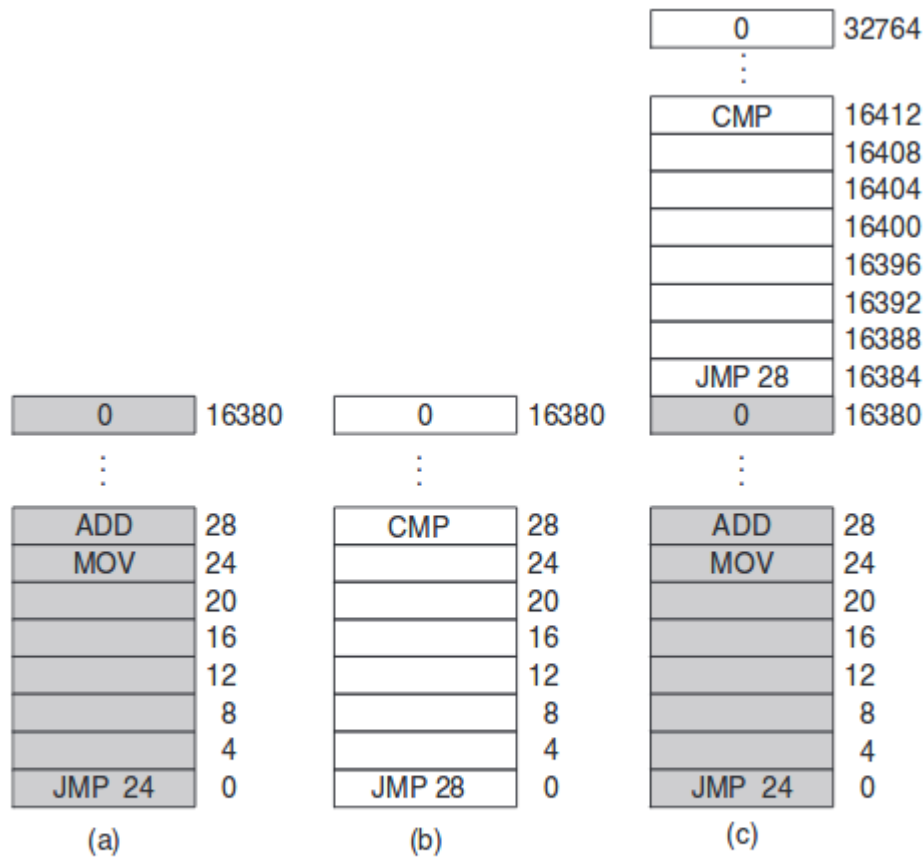


Figura 3-2. Ilustración del problema de reubicación. (a) Un programa de 16 KB. (b) Otro programa de 16 KB. (c) Los dos programas cargados consecutivamente en la memoria.

El primero de los programas tiene una llave de memoria diferente al segundo para poder distinguirlos. Al principio el programa salta a la dirección 24; el segundo salta a la 28. Cuando los dos programas se cargan consecutivamente en memoria empezando en dirección 0.

Con esto, los programas no se interfieren porque tienen distinta llave, pero las instrucciones (por ejemplo en los JMP) saltan a direcciones que no deberían saltar. Eso es porque los programas se están refiriendo a memoria física absoluta.

Una solución sería utilizar la **Reubicación Estática** que consiste en que cuando se trae un programa de Disco para RAM, sumarle el valor correspondiente a la posición en la en disco que se le asocia. Es decir, si un programa se cargaba en la dirección 16.384, se sumaba ese mismo valor.

Una Abstracción de memoria: Espacios de direcciones

Una solución para permitir que haya **varias aplicaciones en Memoria Principal o Física** al mismo tiempo es la usada anteriormente, con la llave que indica qué instrucción pertenece a cada programa.

La **otra solución** es crear la abstracción del **Espacio de Direcciones**.

Un **Espacio de Direcciones** es el conjunto de direcciones que puede utilizar un proceso para direccionar la memoria. Cada proceso tiene su propio espacio de direcciones. Lo complicado es proporcionar a cada programa su espacio de manera que la dirección 28 de un programa indique la ubicación física distinta de la 28 en otro programa.

Registros base y límite

A esta solución se le conoce como **Reubicación Dinámica**. Asocia el espacio de direcciones de cada proceso sobre una parte distinta de la Memoria Física. Esta solución contiene dos registros de hardware especiales **base** y **límite**. Cuando se utilizan estos registros, van aumentando, es decir, van acotando los límites que pueden tener los demás programas para evitar espacios usados.

Intercambio

Si la **Memoria Física de la computadora es lo bastante grande** para contener todos los procesos, los esquemas descritos hasta ahora funcionarán correctamente. Pero en la práctica la RAM es un recurso muy preciado por todos los programas.

La estrategia que se ha desarrollado con los años consiste en utilizar el **intercambio** que consiste en **llevar cada proceso completo a memoria, ejecutarlo durante un tiempo y regresarlo a disco**.

La otra estrategia es conocida como **MEMORIA VIRTUAL**

La operación de **Intercambio** se describe visualmente así:

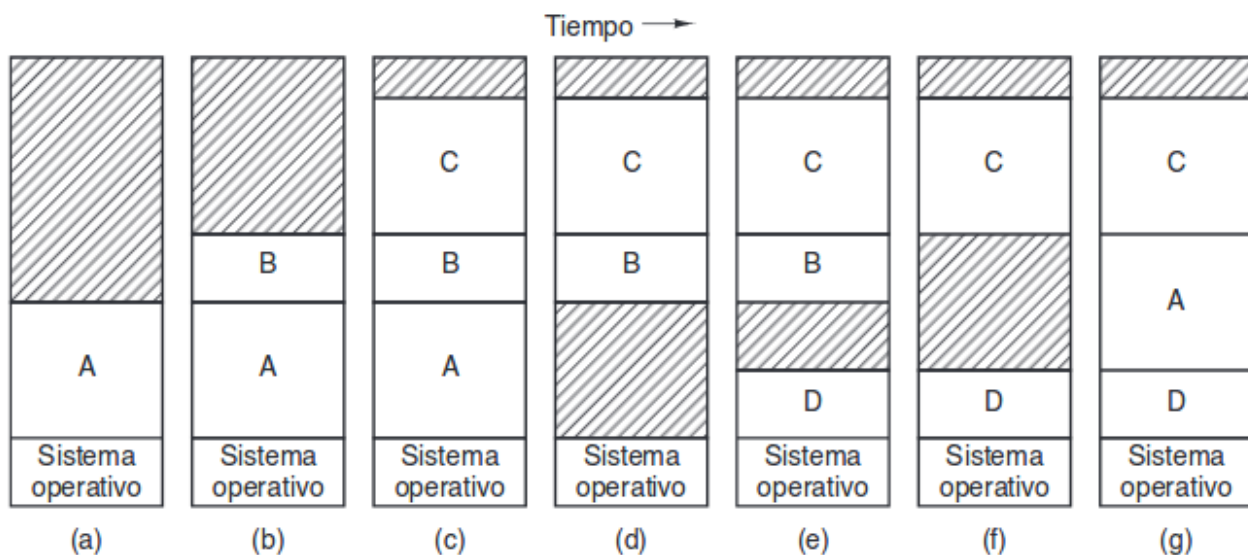


Figura 3-4. La asignación de la memoria cambia a medida que llegan procesos a la memoria y salen de ésta. Las regiones sombreadas son la memoria sin usar.

Administración de memoria libre

Administración con mapas de bits

La **Memoria se divide en unidades** de asignación (pueden ser del tamaño que convenga), en **cada unidad hay un bit** correspondiente en el mapa de bits (**0 libre y 1 ocupada**)

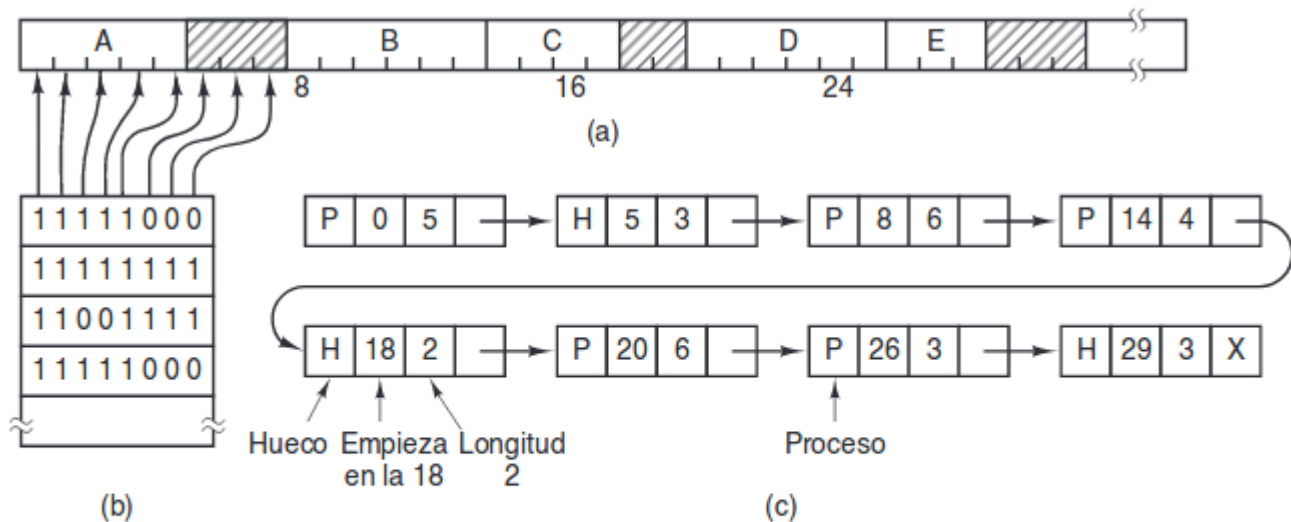


Figura 3-6. (a) Una parte de la memoria con cinco procesos y tres huecos. Las marcas de graduación muestran las unidades de asignación de memoria. Las regiones sombreadas (0 en el mapa de bits) están libres. (b) El mapa de bits correspondiente. (c) La misma información en forma de lista.

Administración con listas ligadas

Muy sencillo el concepto... se explica en la imagen anterior (c).

Cuando los procesos y huecos se mantienen en una lista ordenada por dirección, se pueden utilizar varios algoritmos para asignar memoria.

1. **Primer ajuste:** se explora la lista hasta encontrar un hueco lo suficientemente grande y se divide en 2: uno para el proceso y otro para su memoria sin utilizar.
2. **Siguiente ajuste:** igual que el **primer ajuste** pero se lleva un registro de los huecos. (rendimiento más pobre... mucha pérdida computacional).
3. **Mejor ajuste:** busca en toda la lista y toma el hueco más pequeño que sea adecuado. En vez de hacer como el primero y buscar también hueco para la memoria sin utilizar aún de es proceso, directamente busca el tamaño actual necesario. (es peor que los 2 anteriores).
4. **Peor ajuste:** toma siempre el hueco más grande disponible.
5. **Ajuste rápido:** en vez de mantener una misma lista para los procesos y huecos, únicamente se mantiene una para los huecos y sorprendentemente es más óptimo.

1. Introducción

El método llamado **Memoria Virtual** se basa en la idea de dividir programas en sobrepuestos (*Overlays*) de tal forma que **se mantienen en Disco y se intercambian hacia dentro de la Memoria**.

Aplicando el concepto a Memoria Virtual

Cada programa tiene su espacio de direcciones, que se divide en trozos llamados **Páginas**.

Cada página es un rango continuo de direcciones. Las **Páginas se asocian a la Memoria Física**. Con la **Memoria Virtual** todo el espacio de direcciones (incluido datos y texto) se puede asociar a la Memoria Física (RAM).

2. Paginación

Cuando un programa ejecuta una instrucción como **MOV REG, 100** copia el contenido de la dirección de memoria **100** a un registro generando así direcciones virtuales.

Estas direcciones virtuales generadas por el programa se conocen como **Direcciones Virtuales** y forman el **Espacio de Direcciones Virtuales**. (Si no hubiera Memoria Virtual, se colocaría directamente la Memoria Física en el bus de memoria y se lee/escribe directamente en esa dirección).

Cuando se utiliza una Memoria Virtual, las direcciones virtuales no van directamente al bus de memoria: van al **MMU** (*Memory Management Unit*) que **asocia Direcciones Virtuales a Direcciones Físicas**.

```
Direcciones Virtuales --> hasta 64KB
```

```
Memoria Física --> 32KB
```

```
El espacio de direcciones VIRTUALES se divide en PÁGINAS --> 4KB
```

```
El espacio de direcciones FÍSICAS se divide en MARCOS DE PÁGINA --> 4KB
```

```
*PAGINA y MARCO DE PÁGINA tienen mismo TAMAÑO
```

```
Por lo tanto obtenemos 16 PÁGINAS y 8 MARCOS DE PÁGINA
```

Imagen: Paginacion 1

Cuando el programa ejecuta **MOV REG, 0** ... trata de acceder a la dirección virtual 0. Esta dirección virtual se envía a la MMU. La MMU ve que está en la página 0 (de 0-4095), que esta ya asociada al marco de página 2 (de 8192-12287). Por lo que el bus recibe la Dirección Física **8192**.

Como se puede comprobar en la imagen, sólo 8 de las páginas virtuales se asocian a la Memoria Física. En hardware, existe un **bit de presente/ausente** que lleva el registro de las páginas presentes en Memoria Virtual.

Fallo de página

Cuando el programa ejecuta **MOV REG, 32780** está accediendo a un byte dentro de la Página virtual 8.

La MMU detecta que la página no está asociada y hace que la CPU haga un **TRAP (fallo de página)**. El SO selecciona un marco de página que se utilice poco y escribe su contenido en el Disco para no perderlo, después obtiene el Marco de Página que acaba de desalojar y lo asocia a la Página que queríamos (en este caso la que contiene la dirección 32780) obtiene la página y reinicia la instrucción del trap.

Ahora veamos un ejemplo de cómo funciona la MMU...

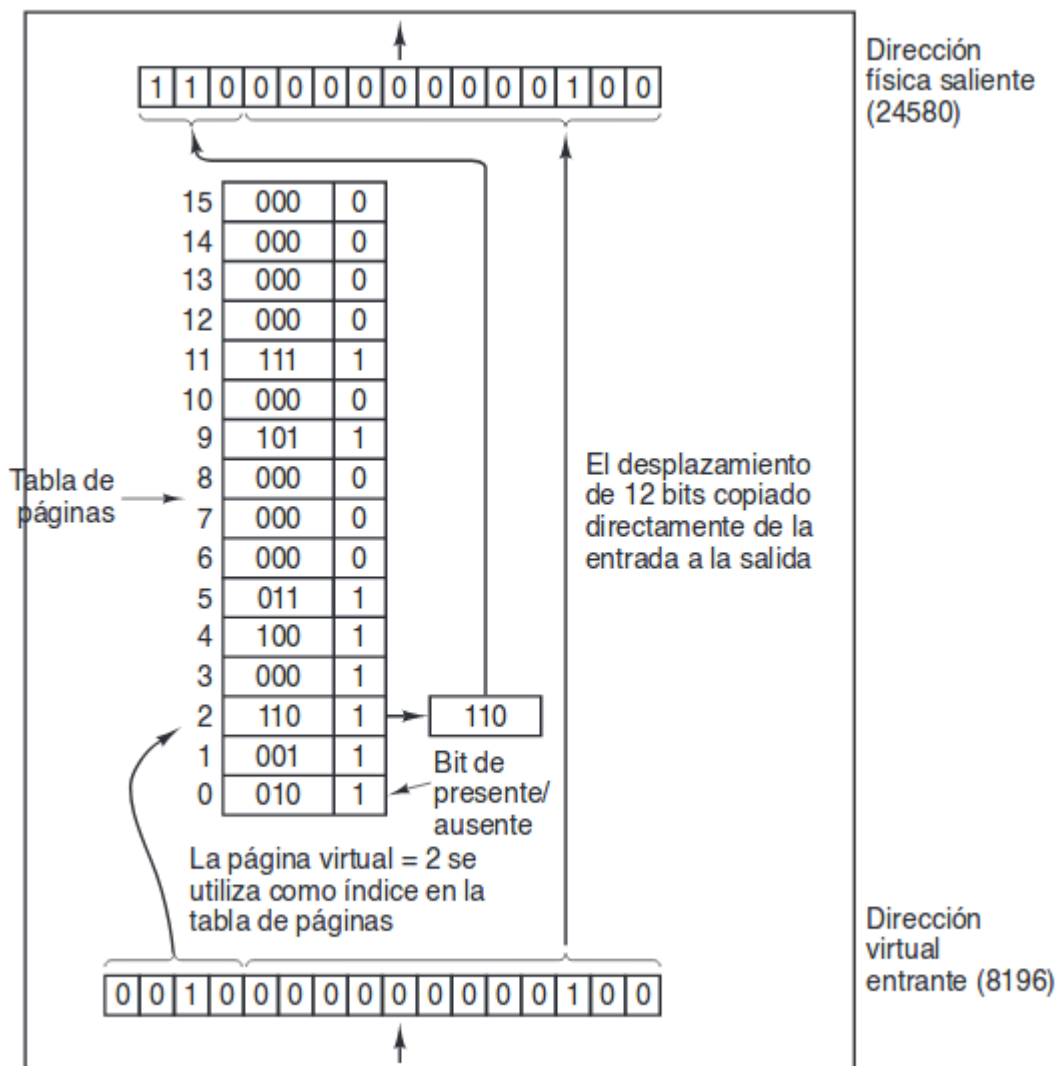


Figura 3-10. La operación interna de la MMU con 16 páginas de 4 KB.

Tenemos una dirección virtual: 8196 (00100000000000100)

La MMU va a asociar esta dirección con la Memoria Física.

1. 16 bits de entrada > 4 bits para el nº de página (16 páginas)
> 12 bits para desplaz. (4096 bytes dentro de una pág)
2. Los 4 bits identifican las páginas (ordenadas) y cada página tiene dentro un índice al marco de página físico (con su bit presente/ausente)
3. Se obtiene el marco de página y se le suma el desplaz.

En total obtenemos de una dirección virtual de 16 bits a una física de 15.

Finalmente el conjunto de 15 bits se coloca en el bus de salida como dirección de memoria física.

Tablas de páginas

El objetivo de las **tablas de página** es asociar las direcciones virtuales a las direcciones físicas. En sentido matemático, la tabla es una función donde el número de página virtual es un argumento y el número de

marco físico es un resultado.

Estructura de tablas

La distribución exacta depende en gran parte de la máquina, pero en rasgos generales siempre es igual.

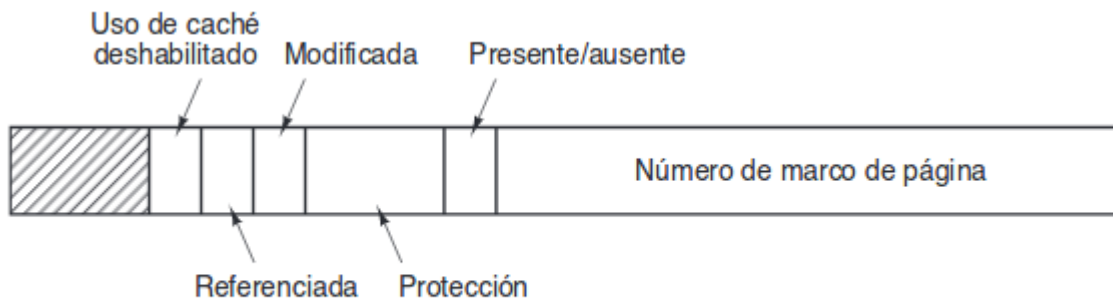


Figura 3-11. Una típica entrada en la tabla de páginas.

El tamaño puede variar pero 32 bits es muy común.

1. Número de marco de Página
2. Bit presente/ausente
3. Protección
4. Modificada (a veces es le refiere como 'bit sucio')
5. Referenciada
6. Uso de caché deshabilitado

*Los bits de Modificada son útiles a la hora de llevar un registro de páginas; si ha sido modificada debe escribirse de vuelta en el disco.

*Los bits de Referenciada son útiles a la hora de saber qué página se debe desalojar cuando ocurre un fallo de página. Las páginas que no se estén utilizando son buenas candidatas.

Optimización de paginación

La solución que se ha ideado es equipar a las computadoras con un pequeño dispositivo de hardware, el cual se usa para acceder a Direcciones Físicas sin pasar por la tabla de páginas (es decir, sin pasar 2 veces por RAM).

TLB

El dispositivo del que acabamos de hablar se llama **TLB** o **Memoria Asociativa**, éste se encuentra en la MMU y **tiene el mismo nº de datos que proporciona la tabla de páginas**. La diferencia es que aquí también se almacena el **Nº de Página** ya que en la tabla de RAM no se necesita ya que tiene asociado directamente el Nº de Página Virtual por orden (como se ha visto en las anteriores imágenes).

Funcionamiento

Cuando se presenta una dirección virtual a la MMU, el hardware comprueba si el número de página está presente en **TLB comparándola con todas sus entradas simultáneamente** (por eso está en hardware)

1. Si está presente y **no** viola los bits de protección --> MARCO
2. Si está presente y **sí** viola los bits de protección --> FALLO (por protección)
3. Si **no** está presente. La MMU lo detecta y la busca en la tabla de páginas, remplazándola en una de las entradas de la TLB.

El beneficio de la TLB es que ahorra área de CACHÉS DE CPU

OPTIMIZACIÓN: El Sistema Operativo intenta adivinar qué páginas tienen más probabilidad de ser utilizadas y las precarga en el TLB

Fallos de TLB

- **Fallo suave:** la página no está en TLB pero sí en Memoria Virtual.
- **Fallo duro:** la página ni siquiera está en Memoria Virtual.

Tablas de páginas Multinivel

Para estudiar este caso es necesario tener a mano esta imagen.

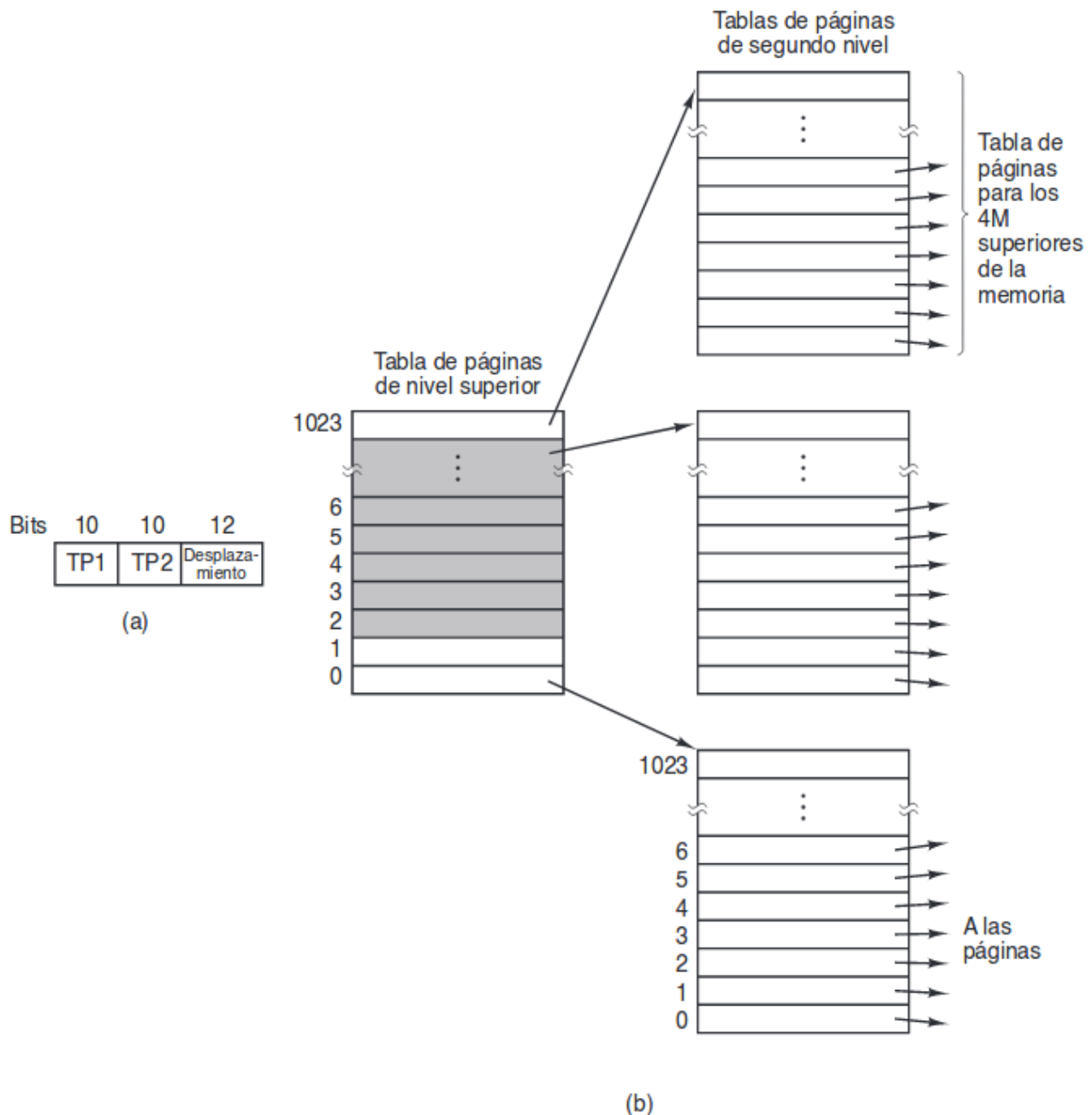


Figura 3-13. (a) Una dirección de 32 bits con dos campos de tablas de páginas. (b) Tablas de páginas de dos niveles.

Tenemos una dirección virtual de 32 bits:

> TP1 = 10 + TP2 = 10 + Despl. = 12

Desplazamiento = 12 bits --> $2^{12} = 4096$ Bytes = páginas de 4KB
(ya que el despl. indica los bytes escogidos dentro de un marco)

TP1 = 10 bits --> $2^{10} = 1024$ Bytes = 1024 entradas
y cada entrada es una página y cada una contiene sus 4KB

A la Tabla de Páginas de Nivel Superior se accede según el TP1

TP2 = 10 bits --> $2^{10} = 1024$ Bytes = 1024 entradas

y cada entrada es una página y cada una contiene sus 4KB

En total hay 2^{20} entradas en total.

Es decir, tenemos 1.048.576 entradas y cada una de ellas tiene un espacio de 4096 bytes. Tenemos en total 4.294.967.296 bytes

Ahora después de elegir una Tabla de Nivel Superior, vamos a elegir la entrada de esa tabla para elegir una página que nos conduzca a las últimas tablas.

Para eso sirve el TP2, para una vez tienes la Tabla Superior, llegar a las últimas.

*Estudiar MUY DETENIDAMENTE fijándose en la figura 7

*Para ver un ejemplo más concreto leer página 200 de Tanenbaum

Utilizar esta solución es muy útil, sin embargo, en las computadoras actuales de 64 bits, en vez de tener $10+10+12=32$ tendremos $x+y+x=64$. Imaginemos lo que podría ocupar una tabla de páginas con 64 bits de direccionamiento si una de 32 con espacios de 4KB utilizan más de 4.000 millones de bytes (unos 4GB).

Tablas de páginas Invertidas

En este diseño existe **una entrada por cada Marco de Página**, en vez de tener una entrada por cada **Página**. En este caso la CPU pide una **página y un proceso** que se busca una tupla que coincida en la Tabla de Páginas y devuelve un puntero a la Memoria Física.

El problema es que es mucho más costoso aunque se puede **solucionar utilizando la TLB** de la que hemos hablado antes. La TLB puede contener todas las páginas de uso frecuente y la traducción puede ocurrir con igual rapidez que las Tablas Regulares.

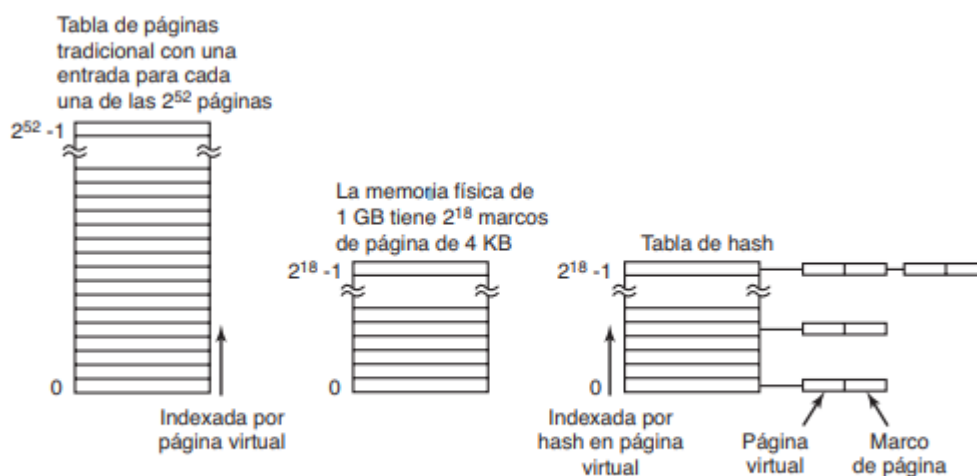


Figura 3-14. Comparación de una tabla de páginas tradicional con una tabla de páginas invertida.

Algoritmos de remplazo de Página

No es óptimo escoger una página al azar para remplazar, por eso se trata de elegir una página de uso poco frecuente.

El algoritmo utópico se basa en **etiquetar cada página con el número de instrucciones que se ejecutarán después de que se haga referencia a esa página**. El problema es que no hay forma de saber cuál será la próxima página referenciada.

1. No Usadas Recientemente (NRU)

La mayor parte de las computadoras tienen dos bits de estados:

- **Referencia** que se establece cada vez que se hace referencia a la página.
- **Modifica** que se establece cuando se modifica la página.

Al principio, cuando se carga por primera vez la página, sólo se carga en SÓLO LECTURA. Sin embargo, cuando se modifica y por ende se cambia el bit **Modifica** también se pone en modo LECTURA/ESCRITURA.

En el transcurso del programa el bit **R se borra en cada interrupción de reloj**. Cuando ocurre un fallo de página, el Sistema Operativo inspecciona todas las páginas y las divide en 4 categorías:

0. no R, no M. (solo se ha leído y posiblemente haya sido hace bastante tiempo)
1. no R, sí M. (se ha modificado pero no se accede a ella desde hace más de una interrupción)
2. sí R, no M. (se ha traído a memoria únicamente para leer, pero hace una interrupción que accedo a ella, lo cual es poco tiempo)
3. sí R, sí M. (la he leído y modificado hace muy poco tiempo, posiblemente no sea buena opción para eliminar de memoria)

El algoritmo NRU elimina una página al azar de la mínima numeración que no esté vacía

2. Remplazo FIFO

El Sistema Operativo mantiene una lista de todas las páginas actualizadas en memoria, las páginas que llegan se ponen en la parte reciente y se van empujando hacia el otro lado (FIFO = cola).

Cuando ocurre un fallo de página, se **elimina la página que esté al final** (la que se ha usado hace más tiempo).

Sin embargo, aunque puede parecer perfecta, puede dar la casualidad de que una página que se usa extremadamente a menudo no se usó durante X instrucciones y la estamos eliminando por usar este algoritmo.

3. Remplazo FIFO con Segunda Oportunidad

Es una **modificación del FIFO** que evita precisamente el problema comentado en el último párrafo.

Lo que hace es analizar siempre el bit **Referenciado** del último elemento, el que se va a eliminar:

- Si el bit es 0 --> se elimina inmediatamente (hace más de una rotación que no se usa).
 - Si el bit es 1 --> se pone el bit a 0 y la página se pone al final de la lista para una segunda rotación.
- *De ahí viene lo de **segunda oportunidad**.

4. Remplazo de Reloj (Versión Óptima de FIFO con Segunda Oportunidad)

Más óptimo que mantener una lista, es mantener una con la forma de reloj para no tener que mover el puntero de detrás a delante.

Por lo demás es exactamente igual a **segunda oportunidad** es decir, si el bit **R** es 1, se pone a 0 y la manecilla pasa al siguiente.

5. Menos Usadas Recientemente (LRU)

Este **algoritmo es excelente**, ya que se fija en la proporción de veces que se utiliza una página. El **problema es que es muy caro** de implementar y mantener.

Una forma es implementarlo mediante **Hardware**:

Se implementa un contador llamado C en hardware, que se incrementa después de cada instrucción.

Cada entrada de la Tabla de Páginas tiene un campo suficientemente grande donde se guarda ese contador cada vez que se haga referencia a esa página.

Cuando ocurre un fallo de página el SO examina todos los contadores y el menor lo elimina.

Otra forma con **Matrices**:

Tenemos una matriz $n \times n$ donde n es el nº de marcos de página.

Inicialmente la matriz esta en 0.

Cuando se referencia una página, se establecen todos los bits de su fila en 1 y de su columna en 0.

En cualquier momento, la fila cuyo valor binario sea menor se elimina.

6. No Usada Frecuentemente (NFU)

Realmente es como una **simulación del LRU implementado en Hardware** (es decir, el primero, el del contador) **pero en Software**.

Al igual que en el LRU en Hardware, también tenemos un **Contador asociado a una Página** (inicado a 0). Y en cada interrupción **agrega el bit R**.

Como se puede ver a simple vista, es una aproximación un poco burda ya que únicamente se guarda si se ha hecho referencia hace poco o mucho pero con muy poca certeza del tiempo que lleva sin ser utilizada.

7. NFU MODIFICADO: Envejecimiento

En este caso se hace igual que el anterior, una aproximación de LRU Hardware pero en Software.

La **diferencia con la anterior** es que en este **se guarda más de un bit del contador**, se guardan varios. De tal forma que se van **añadiendo los bits R empujándolos hacia la derecha**. Así en un momento de tiempo, podemos examinar el número completo y (siguiendo el sistema de numeración [centenas, decenas, unidades...]) podemos ver que el que tenga el contador con el número menor es el que hay que eliminar.


8. Remplazo de conjunto de trabajo

Es la **forma más pura de paginación**: los procesos inician sin ninguna de sus páginas en la memoria. **A medida que se vayan solicitando páginas, van surgiendo fallos de página**. * Si se cargan las páginas del último WS antes de reanudar el proceso se llama **Prepaginación**.

Lo bueno es que se evita cargar páginas no necesarias al principio, de tal modo que al cabo de un cierto tiempo, después de muchos fallos de página, seguramente va a haber muy pocos fallos ya que cada programa tiende a referirse a una fracción pequeña de todas sus páginas. (Este conjunto de páginas se llama **conjunto de trabajo**).

Esta propiedad de que los procesos normalmente hacen referencia a una fracción pequeña de sus páginas se llama **Localidad Espacial**.

*Nota: se dice que un programa que produce fallos de página cada pocas instrucciones está **sobrepaginando**.

Aquí se ve una gráfica de este método para entenderlo:  Conjunto de Trabajo

Más acerca del Espacio de Trabajo (WS)

El WS varía lentamente con el tiempo... pero se puede predecir el WS en función del propio WS en el momento en el que se detuvo el proceso

Para entender cómo funciona realmente este algoritmo hay que presentar varios conceptos que trabaja con él.

Bit R y M	-> se trabajan con ellos como se hacía hasta ahora (1 bit para cada 1).
Tiempo Virtual Actual	-> cantidad de tiempo que la CPU lleva con un proceso.
Tiempo de Último Uso	-> campo que existe dentro de cada página. -> cada vez que se hace referencia a esa página se copia el Tiempo Virtual Actual dentro de este campo.
Edad	-> $\text{Tiempo Virtual Actual} - \text{Tiempo de Último Uso}$

```
Si (R == 1)                                // Ha sido referenciada dentro del
último pulso.
{
    Tiempo de Último Uso = Tiempo Virtual Actual
}
```

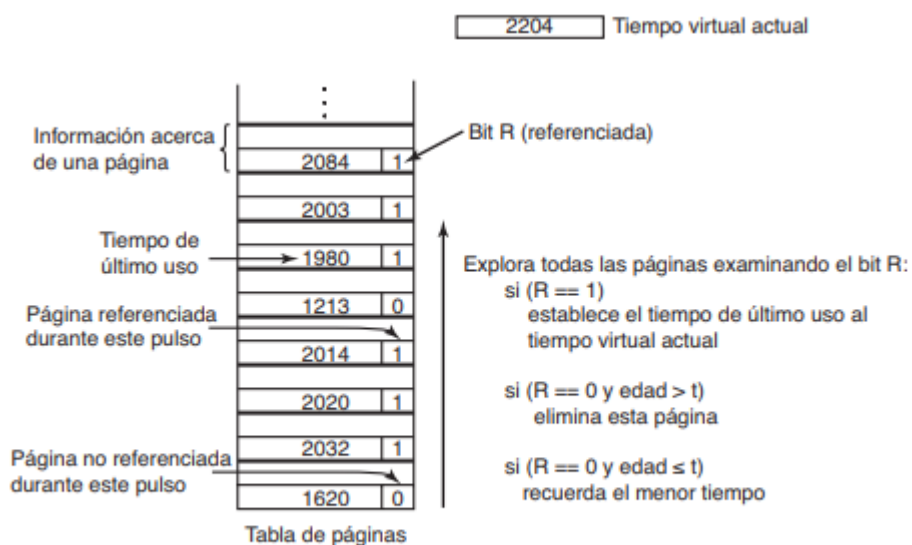


```

Si (R == 0) && (Edad > Tiempo del WS)    // Es decir, no pertenece al
Workspace
{
    La página se ELIMINA
}

Si (R == 0) && (Edad < Tiempo del WS)    // Sigue perteneciendo al WS, es
decir, seguramente se use próximamente
{
    No se elimina, pero si no hay otra página más vieja, se elimina esta.
}

```



7. Remplazo WSClock

Este algoritmo fusiona el **Algoritmo del Reloj** con el **Conjunto de trabajo**.

Al principio existe una lista circular de marcos de página. Cuando se carga una página se agrega a la lista. En cada entrada se incluye el **Tiempo de Último Uso**. Por lo que, al igual que en el algoritmo de reloj, se analizan los bits de **R** y **M**. Pero además se mira su edad.

```

Si (R == 1)
{
    Se establece R = 0
}

Si (R == 0) && (Edad > Tiempo del WS)    // No se encuentra en el conjunto
de trabajo
{
    Se elimina y se coloca otra página
}

Si (R == 0) && (Edad < Tiempo del WS)    // Aún está en el conjunto de
trabajo

```

```
{
    El algoritmo continua con la siguiente página
}
```

Algoritmo	Comentario
NRU (Not Recently Used)	Poco efectivo
FIFO	Podría descartar páginas importantes
FIFO Segunda Oportunidad	Gran mejora del FIFO
Reloj	Implementacion más realista del FIFO/Seg. Oportunidad
LRU (Less Recently Used)	Excelente, pero muy costosa de implementar
NFU (Not Frequently Used)	Aproximación burda del LRU
NFU con Envejecimiento	Eficiente que se aproxima a LRU
WS	Eficiente pero costoso de implementar
WSClock	Buen balance entre eficacia y sencillez

Cuestiones de Diseño para Paginación

Asignación Local vs Global

- **Algoritmo Local:** asignan una **fracción fija** de memoria a cada proceso.
- **Algoritmo Global:** asignan **dinámicamente** la memoria a los procesos.

Lo más **óptimo** es usar el **algoritmo global** ya que cambia dinámicamente y no se desperdicia memoria.

Algunos algoritmos de remplazo pueden necesitar uno u otro, por ejemplo WSClock sólo tiene sentido con estrategia local. (dado que juega con el **WS -> local**)

Algoritmo PFF (Page Fault Frequency)

Indica cuando se debe aumentar o disminuir la asignación de marcos de página a un proceso.

Tamaño de Página

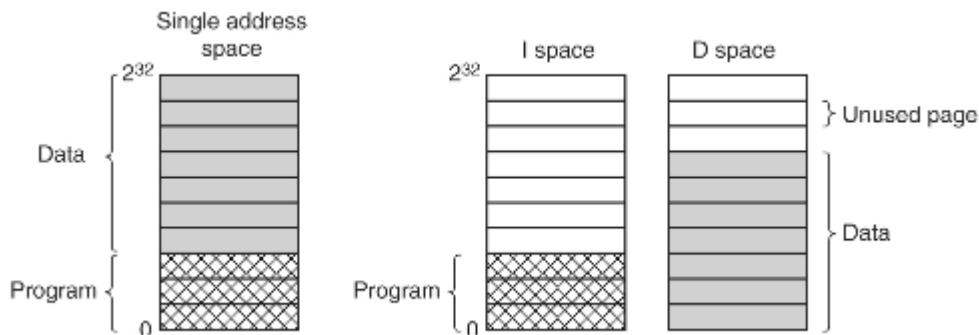
Puede ser elegido por el SO y no hay un tamaño óptimo por lo que se rigen por distintos factores:

- **Página pequeña:** Menor fragmentación interna.
- **Página grande:** La tabla de páginas es más pequeña. Pero el **tiempo de transferencia de una página grande no mejora** tanto como para que merezca la pena con respecto a una pequeña.

Espacios de instrucciones y datos

En espacios pequeños no caben instrucciones y datos; se usan espacios separados.

Lo que provoca el duplicado de espacio de direcciones como se puede ver en la imagen.



Páginas compartidas

Si varios usuarios ejecutan el mismo programa es recomendable compartir páginas en vez de varias copias en memoria.

Las páginas de sólo **lectura** (código) son muy sencillas, sólo necesitan un puntero y evitar que se desalojen páginas compartidas al finalizar un proceso (lo cual requiere estructuras especiales).

Cuando en un momento **un proceso intenta escribir** se produce un **TRAP** y se **crea una copia donde se escriben los datos** que quiera el proceso ofensor. Ahora ambas se establecen como **LECTURA/ESCRITURA**.

A este proceso se le llama **Copiar en Escritura**.

Bibliotecas compartidas

Son bibliotecas extensas que se utilizan por muchos procesos. Se pueden establecer dos enlaces:

- **Enlace Estático:** las funciones se establecen en el espacio de direcciones de cada proceso. Sólo se cargan las páginas que se utilizan. --> **utiliza mucha memoria con páginas repetidas**.
- **Enlace Dinámico (DLL):** Tiene una rutina que enlaza la función al proceso llamador en tiempo de ejecución.

Archivos asociados a memoria

Un proceso puede asociar un archivo a una porción de su espacio de direcciones virtuales; se cargan las páginas según su demanda.

Varios procesos pueden asociar el mismo archivo. Es lo que se llama **Memoria Compartida**, que puede ser un canal de comunicación entre procesos (Ej: un .txt en el que van escribiendo dos procesos).

Política de Limpieza

Cuando se pide un marco de página libre y no hay, es necesario pedir una página y escribir la anterior en el disco.

Para eso existen los **Demonios de Paginación**, está la mayor parte del tiempo inactivo pero se despierta de forma periódica para liberar páginas mediante el algoritmo seleccionado.

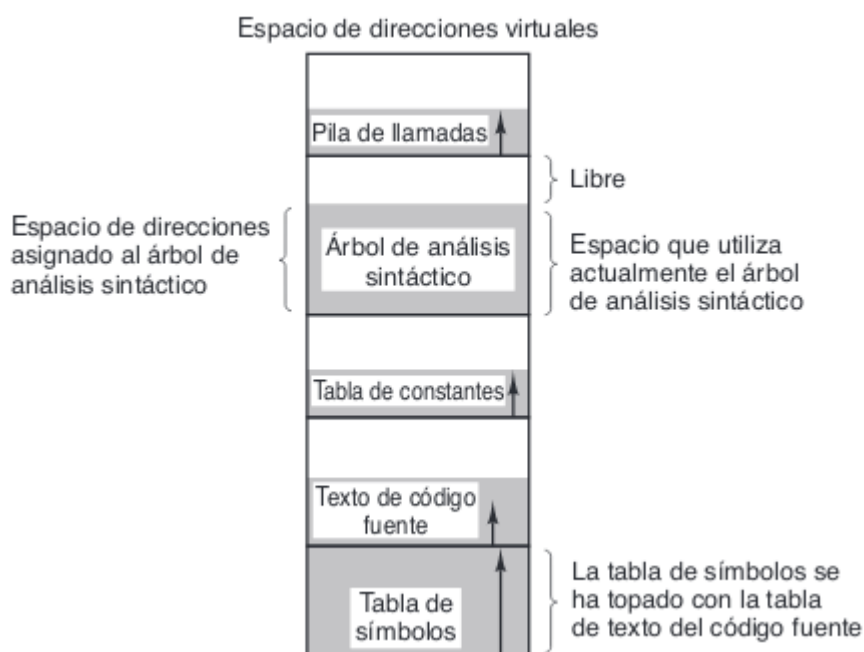
*NOTA: También se asegura que no necesita escribir en disco cuando se requiere un marco de página.

Segmentación

La Memoria Virtual analizada hasta ahora era unidimensional. Sin embargo puede haber casos en los que sería mejor tener dos o más direcciones virtuales.

Por ejemplo, un Compilador necesita tener su tabla de constantes, su pila, el texto del código...

Algunas de esas tablas crecen durante la compilación, otras crecen y disminuyen (pila). En una memoria unidimensional, se deberían usar trozos contiguos como en la figura.



Si suponemos que puede existir algún programa muy extenso que puede llenar alguna de las secciones se puede considerar varias formas... desde interrumpir la compilación, reasignar el espacio quitándoselo a secciones poco llenas u alguna otra.

Sin embargo, la **solución** es realmente **liberar al programador de administrar sus tablas**.

Esta solución consiste en proporcionar muchos **Espacios de Direcciones independientes** llamados **Segmentos** (cuyo valor va de 0 a x). De esta forma los segmentos pueden expandirse y contraerse sin afectar unos a otros.

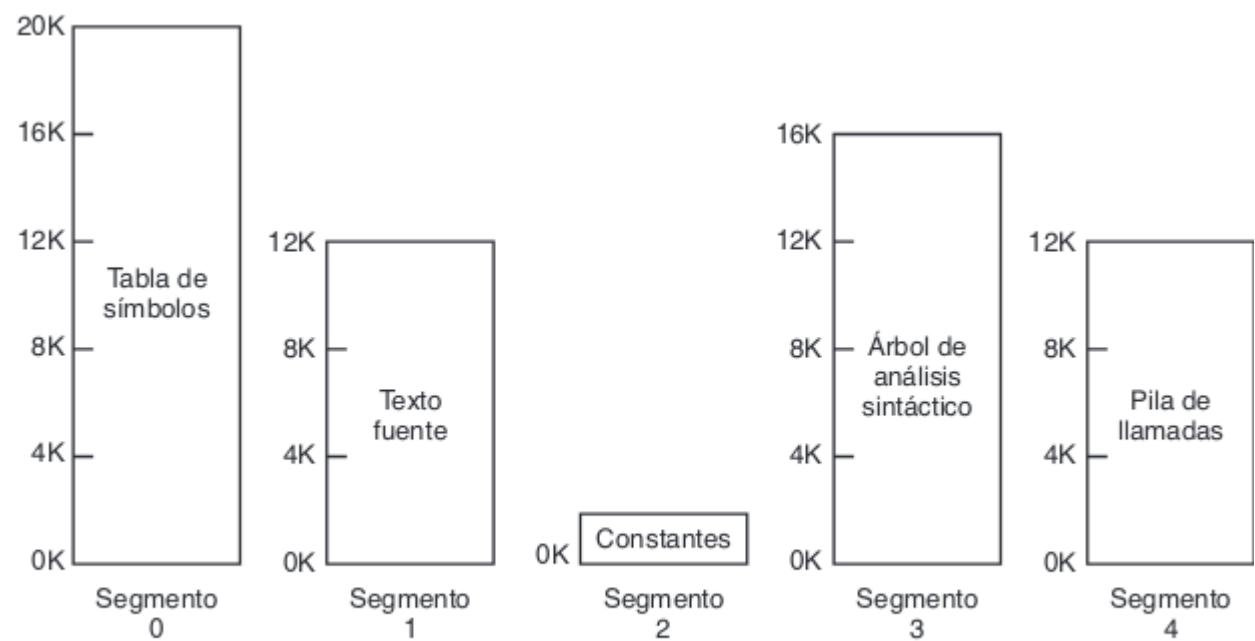


Figura 3-32. Una memoria segmentada permite que cada tabla crezca o se reduzca de manera independiente a las otras tablas.

Hay que considerar que otra ventaja es la **facilidad de compartir bibliotecas, procedimientos....** Ya que las bibliotecas se pueden colocar en un segmento y varios procesos pueden compartirla, eliminando la necesidad de tenerla en el espacio de direcciones.

Implementación de segmentación pura.

Consideración	Páginas	Segmentación
Necesita el programador saber qué técnica se usa?	No	Sí
Cuantos espacios de direcciones lineales hay?	1	Muchos
Puede exceder el espacio de direcciones virtual el tamaño total de RAM?	Sí	Sí
Pueden tener distinta protección los datos y procedimientos?	No	Sí
Pueden las tablas fluctuar el tamaño con facilidad?	No	Sí
Se facilita la compartición de procedimientos entre usuarios?	No	Sí

- **Paginación:** Se inventó para obtener un **gran espacio de direcciones lineal** sin tener que comprar más memoria física.
- **Segmentación:** Se inventó para permitir a los programas y datos dividirse en **espacios de direcciones lógicamente independientes**, ayudando a la compartición y protección.

Segmentación con Paginación

Si los segmentos son extensos es recomendable paginarlos y combinar los dos métodos. Se necesita una **Tabla de Segmentos** por proceso y para cada segmento su **Tabla de Páginas**.