Xestión de Memoria

Sumario

- 1 Concepto de Espacio de Direcciones de un Proceso
  - 1.1 Direccionamiento Físico o Real y Direccionamiento Virtual
- 2 Modalidades de Gestión de Memoria de los Procesos
  - 2.1 Particiones Fijas
  - 2.2 Particiones variables e Intercambio
- 3 Sistemas de Memoria Virtual. Paginación
  - 3.1 Modelo de de Memoria Virtual
  - 3.2 Rendimiento de sistemas de Memoria Virtual con paginación: Tamaño de página
  - 3.3 Implementación de la Memoria Virtual: Tablas de Páginas: Modelo simple, multinivel, TLB
  - 3.4 Estructura de la Tabla de Páginas
  - 3.5 Fallos de página
  - 3.6 Ejemplo: Gestión de memoria virtual en un sistema de 32 bits con 2GiB de RAM
  - 3.7 Mecanismos de optimización
  - 3.8 Resumen de los aspectos fundamentales de la gestión de memoria basada en Memoria Virtual
- 4 Algoritmos de Reemplazo de Páginas
  - 4.1 Objetivos de un buen Algoritmo
  - 4.2 Algoritmos
- 5 Otros Aspectos
  - 5.1 Políticas de reemplazo global y local
  - 5.2 Conjunto de Trabajo
  - 5.3 Segmentación
- 6 Referencias
  - 6.1 Semántica del comando free
  - 6.2 Linux Memory Management

Concepto de Espacio de Direcciones de un Proceso

Todo proceso se ejecuta en la memoria principal, es decir, sus datos e instrucciones están copiados en la memoria principal durante su ejecución. Cuando la CPU ejecuta una instrucción de un proceso debe poder identificar el dato o instrucción a procesar a través de una dirección de memoria que la identifica de modo único. Es sabido que las memorias se identifican a nivel de byte, esto es, cada byte de la memoria principal tiene una dirección única asociada. Ejemplo:

¿Cuál es el tamaño máximo de la memoria principal en arquitecturas de CPU de 32 bits?

Decir que una arquitectura de CPU es de 32 bits es lo mismo que decir que las direcciones de memoria son de 32 bits. Solo habrá que calcular por tanto cuántos bytes podemos direccionar, es decir, cuántos bytes podemos identificar con un número binario de 32 bits para la dirección:

2e32 bytes = 2e10x2e10x2e10x2e2 bytes = 1024x1024x1024x4 bytes = 4x1024x1024 Kibytes = 4x1024 Mibytes = 4 Gibytes

Por este motivo, 4 Gigabytes ya no es una cantidad de memoria principal RAM excesiva, se han creado los nuevos sistemas de 64 bits

Direccionamiento Físico o Real y Direccionamiento Virtual

Los Sistemas Operativos antiguos, de la época de los 60 y 70, utilizaban sistemas de gestión de memoria que manejaban directamente las direcciones de memoria físicas. Recordad que la memoria principal se etiqueta y direcciona mediante números que hacen referencia a los bytes de datos en la memoria principal. En este contexto, la creación de programas para computadoras requería conocer los detalles físicos del hardware de memoria de esa plataforma y adaptar éstos para que pudiesen direccionar un espacio de memoria físico en la RAM.

Esta estrategia es demasiado dependiente del hardware y por tanto susceptible de causar problemas durante la ejecución de los procesos. Por ejemplo, **¿qué ocurriría si un proceso que está en la memoria principal en, digamos las direcciones 1000 a 1100, tiene que ser interrumpido e intercambiado a disco?** (a veces es necesario hacer esto para poder alojar en memoria nuevos procesos que se van a ejecutar). La próxima vez que se vuelva a cargar en memoria para proseguir su ejecución debería de ubicarse en el mismo "trozo" de memoria, de lo contrario, las direcciones del

programa, que estarían en el rango 1000 a 1100, podrían hacer referencia a zonas de memoria fuera de su ámbito.

Llamaremos **relocalización** a la propiedad del Sistema Operativo de poder cargar procesos en direcciones de memoria diferentes en sucesivos intercambios del proceso de memoria a disco. Llamaremos **protección** a la propiedad de que un proceso no puede hacer referencia a direcciones de memoria fuera de su espacio de direcciones legítimo, esto es, aquella parte de la memoria principal en la que se ubica y residen sus direcciones válidas.

La administración de memoria tiene que garantizar:

1. **Un proceso tiene que poder reubicarse de forma transparente**, es decir, aunque cambie de ubicación física en la memoria tiene que poder funcionar igual
2. **Un proceso no puede acceder a zonas de memoria en las que residen otros procesos**, es decir, estará confinado en la parte de memoria principal asociada a su ejecución.

Según lo anterior, los espacios de direcciones que trabajan con direcciones de memoria físicas no son adecuados para esos dos objetivos; sobretodo llevan mal la recolocación. Por ese motivo no se utilizan en los Sistemas Operativos modernos **espacios de direccionamiento físicos**, sino que se utilizan los denominados **espacios de direcciones virtuales**.

En un **espacio de direcciones virtuales** el proceso "ve" un esquema de direcciones lógico, **que empieza en 0 y termina en un número máximo**, determinado por el máximo número de direcciones en la arquitectura de la CPU. Por ejemplo, para arquitectura de CPU de **32 bits un proceso podría tener una memoria virtual asociada de 4GB de tamaño**. Aunque realmente no hará uso de todas ellas, éste sería el máximo teórico, mucho más elevado, por supuesto, en sistemas de 64 bits.

Ahora bien, **cuando el proceso se ejecuta lo hará en la memoria principal física** y por tanto **necesita hacer corresponder sus direcciones de memoria virtuales (independientes de la memoria física) con las direcciones de memoria físicas** que la CPU debe direccionar para poder ejecutar el proceso. De esta labor de asociación se encarga el Sistema Operativo.

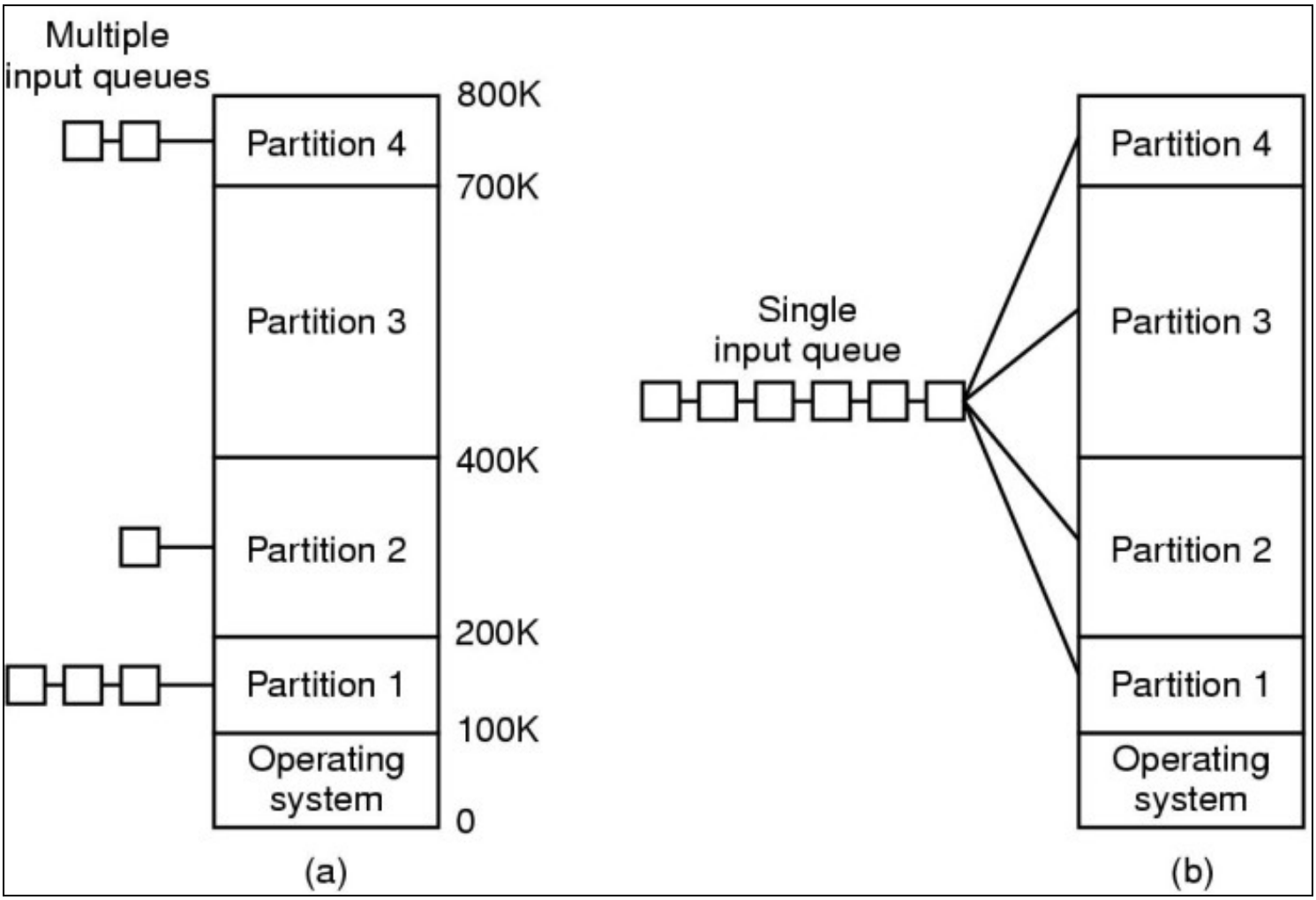
Modalidades de Gestión de Memoria de los Procesos

Existen varias maneras de gestionar la ubicación de los procesos en memoria para su ejecución

NOTA: En este apartado supondremos, en todos los casos, que el proceso tiene que estar completamente cargado en la RAM para poder ejecutarse. Más adelante, cuando se hable de Memoria Virtual eliminaremos esa restricción

Particiones Fijas

En esta modalidad los procesos se cargan en particiones (huecos) de la memoria principal de tamaño predefinido. Tiene a su favor que es sencillo de implementar, pero adolece de una serie de desventajas: > El máximo número de procesos en memoria, es decir, el número de procesos o tareas concurrentes, está limitado por el número de particiones en memoria > Si tenemos un hueco de, digamos, 100KB, un proceso que ocupe 2KB desperdiciará 98KB de ese hueco durante su ejecución, que no podrán ser utilizados por otro proceso ya que solo puede haber un proceso por hueco a la vez.


Multiple
input queues
Partition 4
Partition 3
Partition 2
Partition 1
Operating
system
800K
700K
400K
200K
100K
0
(a)
Single
input queue
Partition 4
Partition 3
Partition 2
Partition 1
Operating
system
(b)

- a) Tenemos una cola de procesos en espera de entrar en la partición de memoria
- b) Hay una única cola que se reparten los huecos fijos

Particiones variables e Intercambio

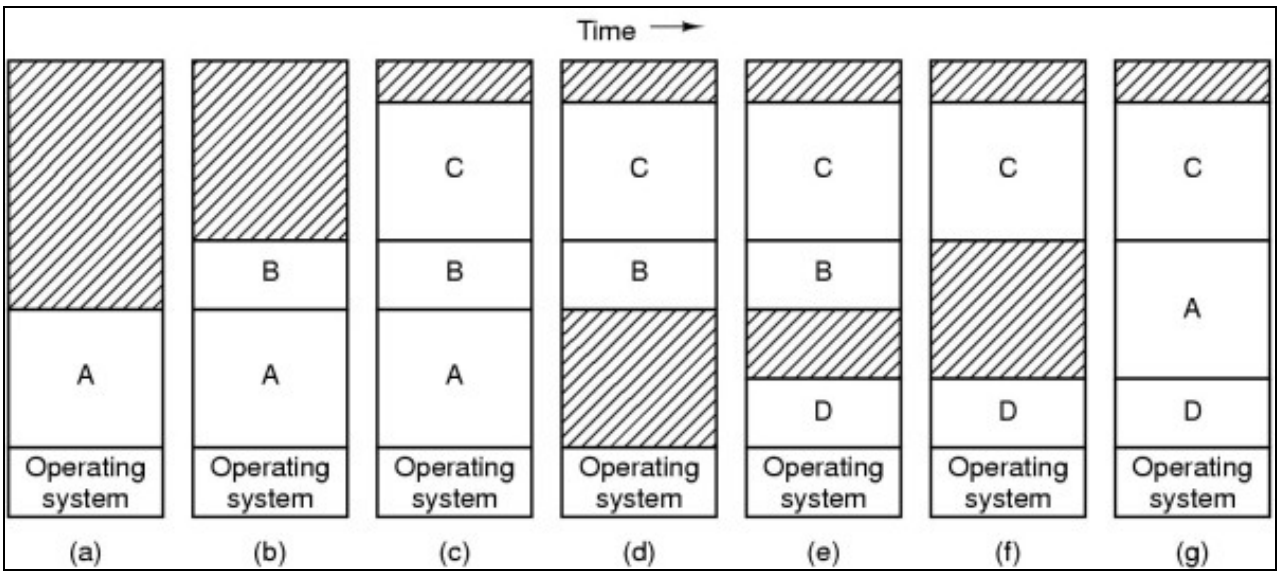
Un sistema de administración de memoria un poco más flexible consiste en poder ubicar los espacios de direcciones virtuales de los procesos en posiciones de memoria que pueden ser **relocalizadas**. Por ejemplo, un proceso podría ocupar un espacio de RAM determinado y ejecutarse ahí. Más tarde, si es necesario el hueco de memoria ocupado por ese proceso para otro, el nuevo proceso podría utilizarlo, el primero de ellos sería volcado a disco, **intercambiado**, y el hueco **liberado (partición)**, ser utilizado para el nuevo proceso. Cuando el proceso que ahora está en disco tenga que ser transferido a RAM, para volver a ejecutarse, podría ser ubicado en otro hueco diferente del anterior. No existen en este modelo, por tanto, a diferencia del anterior un esquema de huecos predeterminado, simplemente un espacio de memoria que puede ser dividido de forma variable.

El **intercambio**, es pues, el mecanismo mediante el cual un proceso es intercambiado a disco para que otro proceso ocupe el espacio que ocupaba el anterior. Esta circunstancia puede darse cuando no haya suficiente memoria disponible para albergar un proceso determinado que se va a ejecutar. Por tanto, será necesario quitar algún proceso de la memoria para crear un hueco adecuado al proceso que va a pasar a ejecución.

En resumen, en este modelo no hay huecos o particiones fijos sino que cada vez que el proceso se relocaliza puede ocupar zonas diferentes de la memoria principal. Este esquema tiene también sus desventajas:

1. Los huecos que van quedando en la memoria tiene un tamaño que puede ser adecuado o no para los procesos que lo ocupan. ¿Qué ocurre si el proceso aumenta de tamaño y no cabe en el hueco asignado?
2. Se puede producir fragmentación cuando quedan en memoria muchos huecos pequeños en los que no se puede ubicar ningún proceso

Estos y otros problemas sugieren la utilización de un modelo más flexible para la gestión de memoria que solvente los problemas planteados



a) Proceso A en memoria b) Entra proceso B c) Entra proceso C d) Intercambiamos A a disco e) Entra D en el hueco que dejó A f) Intercambiamos B a disco g) Entra A en el hueco dejado por B

Sistemas de Memoria Virtual. Paginación

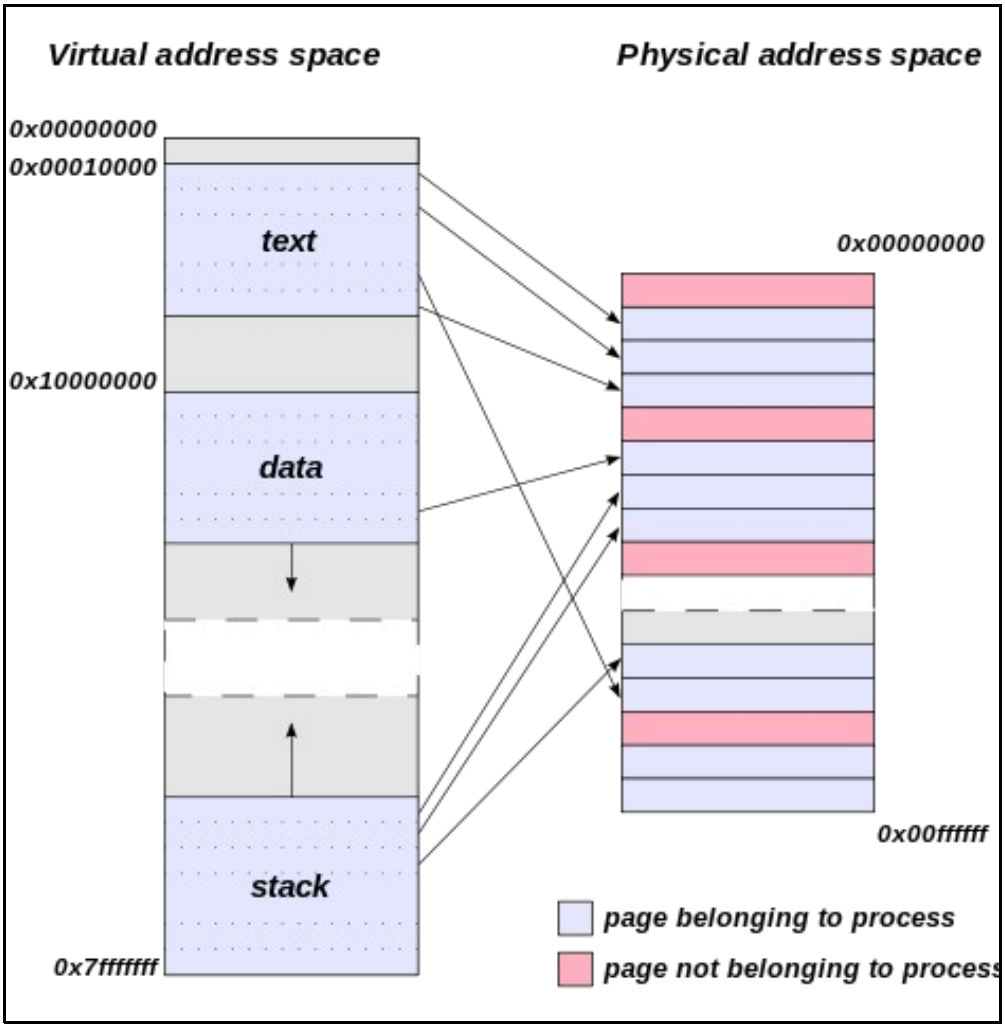
Modelo de de Memoria Virtual

La solución a los problemas de gestión de memoria en los sistemas planteados en el punto anterior viene de la mano de un nuevo concepto de gestión de la memoria ideado por Fotheringham (1961). En el modelo de Memoria Virtual dividimos la memoria en fragmentos, denominados **páginas**. Esta división nos proporciona la flexibilidad necesaria para poder realizar varias cosas importantes:

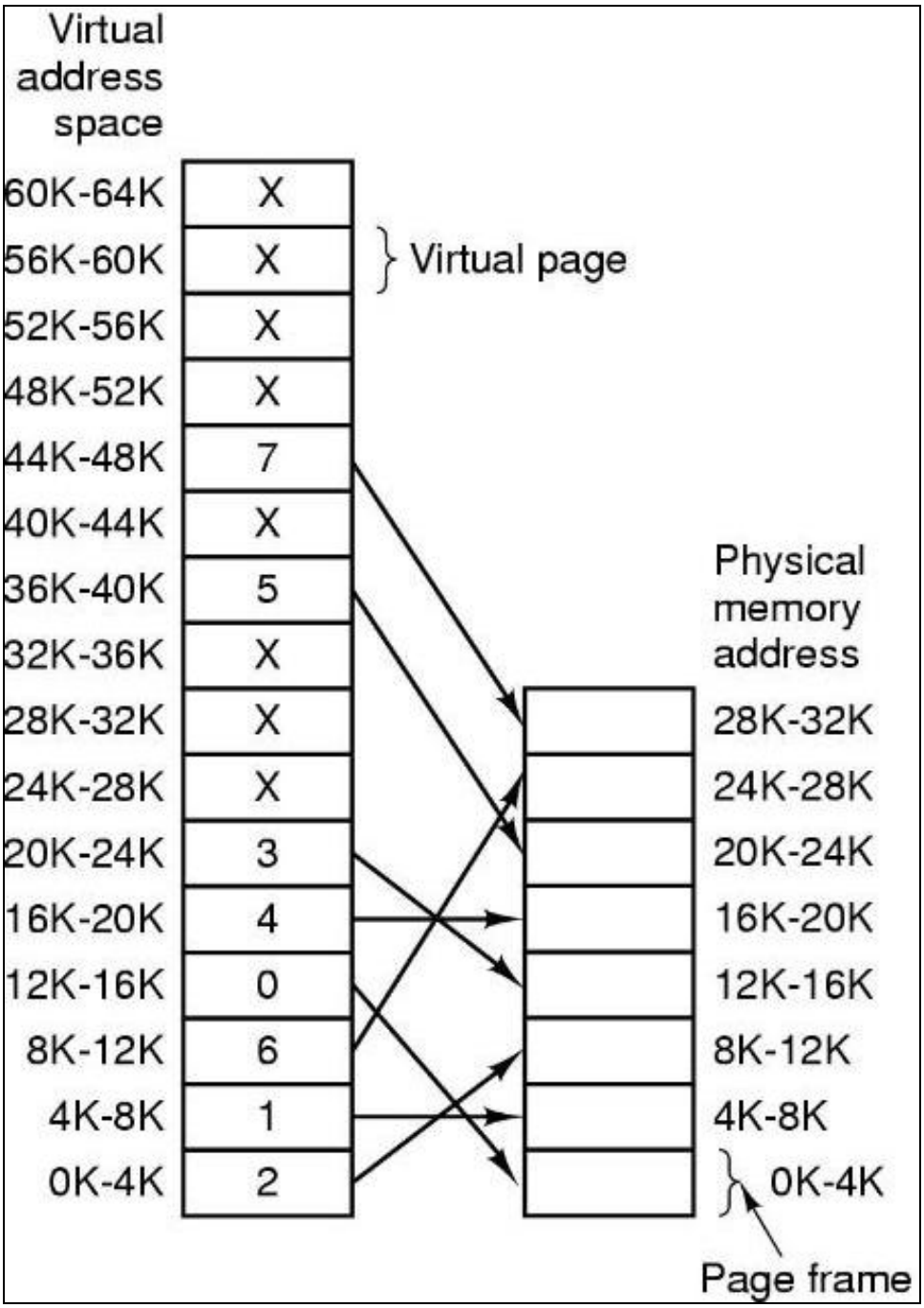
- Asignar la memoria a los procesos no como un todo, sino mediante pequeños fragmentos, o páginas, que permiten que un proceso se puede ejecutar aún cuando no resida por completo en la memoria principal
- En un momento de ejecución determinado de un proceso parte estará en memoria y parte en el disco
- Como consecuencia de lo anterior los procesos pueden ser virtualmente más grandes que la memoria principal, es decir, podrían tener una imagen de memoria cuyo tamaño fuese superior a la cantidad de memoria principal disponible

- Los procesos ya no se intercambian a disco completamente sino que sus fragmentos o páginas se intercambian por separado

Los Sistemas Operativos establecen, para poder gestionar la Memoria Virtual, una correspondencia entre **páginas virtuales** (páginas del espacio de direcciones virtuales de los procesos), es decir el direccionamiento virtual que el programador crea para ese proceso y que es independiente del espacio de direccionamiento físico de la memoria principal, según lo dicho en el apartado anterior, y las páginas físicas o **marcos de página**, fragmentos de la memoria principal del sistema que se utilizarán para albergar las páginas virtuales del proceso en el momento de su ejecución. Es fundamental el hecho de que **el tamaño de página virtual y página física o marco de página deben coincidir**. En la siguiente figura vemos una correspondencia de este tipo:



En el lado izquierdo del gráfico vemos el espacio de direcciones virtuales de un proceso, con su texto de programa (código ejecutable) sus datos y la pila (stack) de llamadas del proceso. A la derecha vemos un esquema que representa la memoria principal (direcciones físicas) y las correspondencias en forma de flechas indican cómo una página virtual es albergada en tiempo de ejecución en una determinada página física o marco de página. En consecuencia, aquellas partes de un proceso, páginas virtuales, necesarias para que el mismo ejecute en un momento dado deberán residir físicamente en la memoria principal, es decir, deberá de existir una asociación entre la página virtual y un marco de página de la memoria principal.



En la izquierda de la figura anterior vemos una representación del espacio de direcciones virtual de un proceso. En este caso comprendido entre 0 y 64KB. A la derecha vemos un esquema de representación de la memoria física de la máquina, en este caso un total de 32KB de memoria. Según estos parámetros es posible que el espacio de direcciones del proceso sea mayor que la memoria física, concretamente puede ser hasta del doble. También podemos observar que el tamaño de página es de 4KB, tanto en las direcciones virtuales como en las direcciones físicas. Esto siempre es así, es decir, coincide el tamaño de página virtual y de marco de página. Tenemos, según el esquema, **16 páginas virtuales** y **8 marcos de página**, por tanto para direccionar una página virtual necesitaríamos 4 bits (2e4=16) y para direccionar un marco de página usaríamos 3 bits (2e3=8). Dentro de una página, sea virtual o física (son del mismo tamaño), tendríamos que utilizar 12 bits para direccionar toda las palabras de memoria de un byte dentro de los 4KB, pues 2e12=4096 y 4096 bytes son 4KB. En conclusión, un byte dentro del espacio de direcciones virtual se direccionaría con un total de 4+12=16 bits (4 para la página y 12 para el desplazamiento dentro de ésta). Para direccionar un byte dentro de la memoria física necesitaríamos 3+12=15 bits. Entonces ¿cómo se traduce una dirección de memoria virtual, utilizadas por el proceso, a una dirección de memoria física, utiliza por la CPU?. La respuesta es sencilla si nos damos cuenta de que lo único que tenemos que hacer es corresponder páginas virtuales con marcos de página, pues el resto de la dirección, la correspondiente al desplazamiento dentro de una página virtual o marco de página coincide, al ser ambas del mismo tamaño. Por tanto lo que necesitamos es mantener una asociación entre páginas virtuales y marcos de página. En la figura vemos como, por ejemplo la página virtual 15, correspondiente a las direcciones 44k-48k, está en memoria principal en la página 7, correspondiente a las direcciones 28k-32k. También podemos ver como la página virtual 0, 0k-4k, está en el marco de página 2, 8k-12k, en la memoria principal. Siguiendo las flechas de la asociación la imagen vemos a que marcos de página corresponden las páginas virtuales. Las páginas virtuales marcadas con una X no están asociados a ningún marco de página. Esto puede ser, o bien, porque no son utilizadas por el proceso, o bien, porque no se han transferido todavía a la memoria principal. Según esto, obtenemos como conclusión que **en un esquema de gestión de memoria basado en Memoria Virtual, no es necesario que el proceso esté por completo en memoria principal para ejecutarse**. En los Sistemas modernos los espacios de direcciones virtuales pueden ser tan grandes como lo permita la arquitectura de direccionamiento del sistema (por ejemplo en sistemas de 32 bits, 4GB y en sistemas de 64 bits 2e32x4GB, algo del orden de los Exabytes), sin embargo la memoria principal está limitada a valores entre unos cuantos GB en la actualidad. La clave, según se ha dicho, está por tanto en la traducción de números de página virtual en números de marcos de página, esta traducción es una labor que realiza el Sistema Operativo utilizando una de las estructuras más importantes y fundamentales de los Sistemas Operativos, la **Tabla de Páginas**.

Rendimiento de sistemas de Memoria Virtual con paginación: Tamaño de página

Uno de los factores fundamentales que afectan al rendimiento a la hora de gestionar la memoria es el parámetro relacionado con el tamaño de página. En los Sistemas operativos actuales se manejan páginas que abarcan tamaños desde 1KB hasta 16KB. La pregunta es obvia, ¿que es preferible, un tamaño de página pequeño o un tamaño de página grande?.

Un **tamaño de página pequeño** implica que el espacio de direcciones de un proceso abarcará más páginas que el equivalente en un sistema con tamaños de página mayor. Por ejemplo un proceso que utilice un máximo de 400KB de memoria, necesitará 200 páginas en un sistema con un tamaño de página de 2KB, pero solamente 50 en un sistema con tamaño de página de 8K. Por tanto con un tamaño de página grande las tablas de página ocupan menos espacio en la memoria y la localización por parte del Sistema operativo de éstas es por lo general más rápido.

Ahora bien, cuando el espacio de direcciones de un proceso no ocupa una cantidad exacta de páginas, siempre se desperdicia un espacio de memoria correspondiente a la fracción del espacio de direcciones que no completa exactamente una página (fragmentación interna).

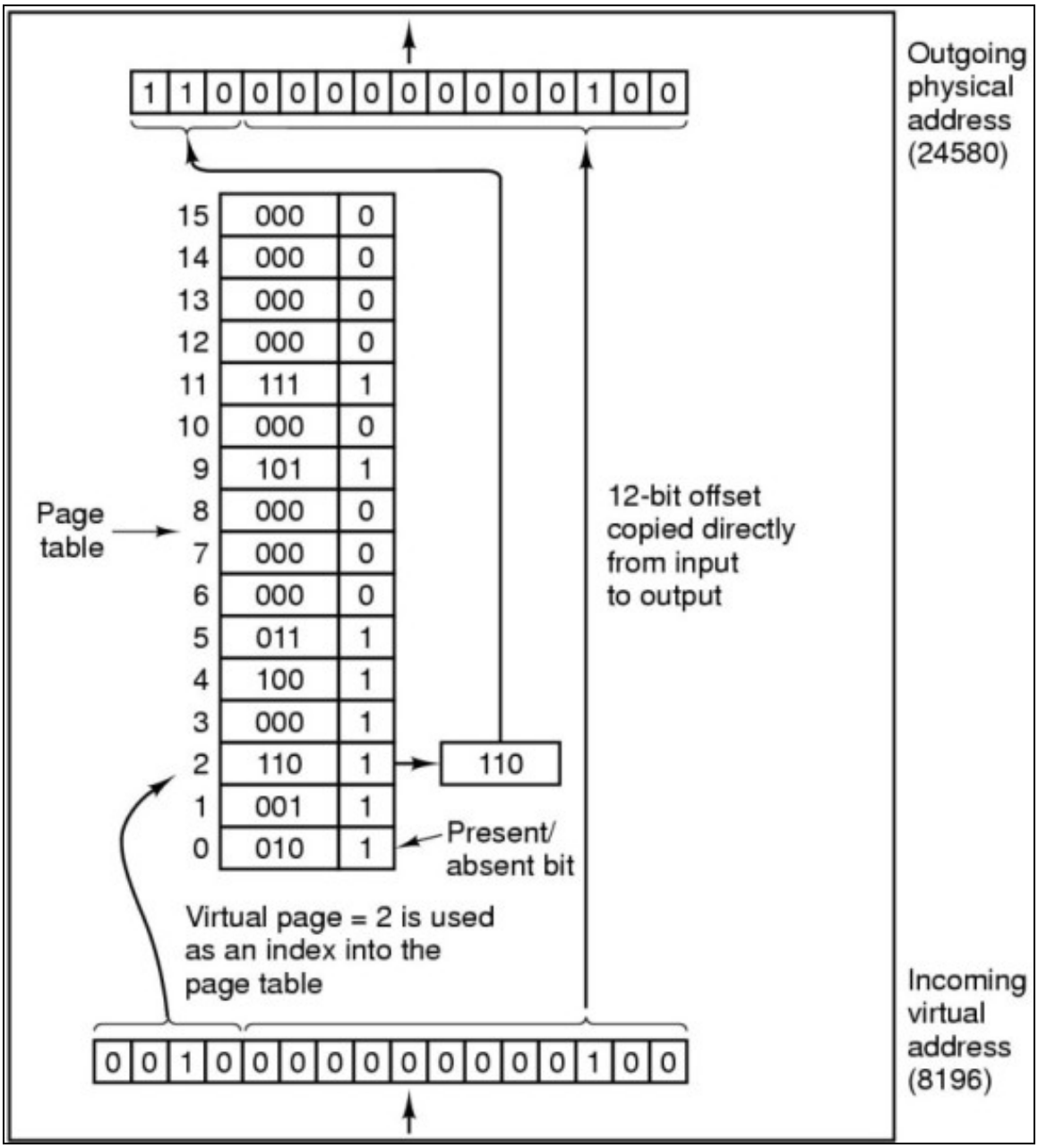
Veámoslo con un ejemplo: supongamos un sistema con un **tamaño de página de 8KB** y un proceso que necesita **17KB** de memoria para ejecutarse. Este proceso necesitará **3 páginas virtuales y 3 marcos de página** para poder ejecutarse, pues el espacio correspondiente a 2 marcos de página, 16KB, no es suficiente. Al necesitar 3 estamos ocupando 24KB de espacio en memoria para un proceso que necesita 17KB; por tanto estamos desperdiciando 24-17=7KB de memoria.

Este problema se atenúa en gran medida cuando usamos tamaños de página más pequeños, ese mismo proceso en un sistema que manejara tamaños de página de **2KB** ocuparía **9 marcos de página en memoria**, y solo desperdiciaría 18-17=1KB de memoria principal.

En resumen los **tamaños de página pequeños** disminuyen la fragmentación interna de la memoria pero necesitan tablas de páginas más grandes. Por el contario un **tamaño de página grande** reduce la memoria necesaria para albergar la tabla de páginas pero aumenta el espacio de memoria desperdiciado a causa de la fragmentación interna.

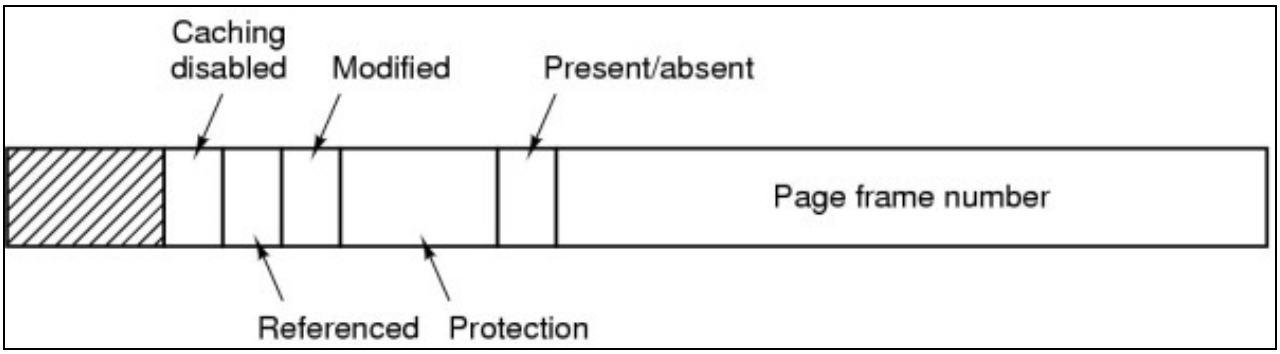
El tamaño de página es un factor importante a la hora de parametrizar un Sistema operativo, por tanto deberán tenerse en cuenta los pros y contras de ámbas estrategias para elegir el tamaño de página idóneo para cada sistema.

Implementación de la Memoria Virtual: Tablas de Páginas: Modelo simple, multinivel, TLB


Outgoing physical address (24580)
1 1 0 0 0 0 0 0 0 0 0 0 1 0 0
15 000 0
14 000 0
13 000 0
12 000 0
11 111 1
10 000 0
9 101 1
8 000 0
7 000 0
6 000 0
5 011 1
4 100 1
3 000 1
2 110 1
1 001 1
0 010 1
Page table
110
Present/ absent bit
12-bit offset copied directly from input to output
Virtual page = 2 is used as an index into the page table
0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0
Incoming virtual address (8196)

En el gráfico anterior se ilustra el mecanismo de traducción de direcciones virtuales a físicas. En la parte inferior tenemos una palabra de 16 bits que representa una dirección virtual de un proceso. Los 4 bits de la izquierda representan el número de página y los 12 bits a la derecha el desplazamiento dentro de la página. En la parte superior se ve la dirección física correspondiente. En este caso es una palabra de 15 bits, los 3 de la izquierda identifican el marco de página y los 12 de la derecha el desplazamiento dentro de éste. Notad que los 12 bits de la derecha de la dirección indican en ambos casos el desplazamiento dentro de la página virtual o marco de página correspondiente. Por tanto lo único que hay que hacer es establecer la relación entre los 4 bits de la izquierda de la dirección virtual, que identifican la página virtual, con los 3 bits de la izquierda que identifican al marco de página en la dirección física. Esta simplificación es posible al ser los tamaños de página virtual y marco de página idénticos. Para hacer la traducción necesaria utilizaremos una estructura del Sistema Operativo, la **Tabla de Páginas**. La Tabla de Páginas almacena la correspondencia **página virtual->marco de página** necesarios. El Sistema Operativo se encargará se gestionar y establecer estas correspondencias. Notad que hay más espacio de direcciones virtual que físico, por tanto las asociaciones serán dinámicas y a lo largo del tiempo de ejecución del proceso se irán modificando las ubicaciones físicas de las direcciones virtuales correspondientes del proceso. Por tanto, las direcciones virtuales **es lo que no varía**, de hecho esto es bueno, ya que hará que el proceso se pueda ejecutar independientemente del hardware, sin embargo **la asociación a los marcos de página (direcciones físicas) es dinámico** y cambiante, gestionado totalmente por el Sistema Operativo. En el gráfico vemos como se utilizan los 4 bits de la página virtual de la dirección como índice en la Tabla de Páginas. A partir del índice se busca la correspondencia, en este caso los 3 bits que identifica el número de marco de página en la memoria principal. **El resto, los 12 bits de la derecha se transcriben tal cual**. El identificador "presente-ausente" a la derecha de la Tabla de Páginas muestra si la página virtual tiene correspondencia, 1, o no, X, con un marco de página en memoria. Cuando un proceso hace referencia a una instrucción dentro de una página virtual que no tiene correspondencia con un marco de página en memoria, es decir, cuando esa página virtual no está en memoria principal, ocurre un **Fallo de Página** (interrupción) y será necesario que el **Sistema Operativo tome el control para transferir la página virtual del proceso de disco a memoria principal**.

Estructura de la Tabla de Páginas



La Tabla de Páginas tiene una estructura del tipo indicado en la figura anterior. El número de página virtual se usa como índice para localizar la entrada de la Tabla de Páginas correspondiente. Las entradas de la Tabla de Páginas tienen la siguiente estructura De derecha a izquierda:

- **Page frame number**: Número de marco de página asociado con la página virtual
- **Presente/Absent**: Indica si la página tiene correspondencia o no en memoria(presente/ausente)
- **Protection**: Indica si la página es de lectura (R) o lectura/escritura (RW)
- **Modified**: Indica si la página ha sido modificada en memoria, en cuyo caso deberá sincronizar su contenido con la copia en disco de la página virtual
- **Referenced**: Indica si se ha hecho referencia al contenido de la página en memoria principal. Esa información se utiliza por algunos algoritmos de reemplazo de páginas
- **Caching disabled**: Hay ciertas páginas que no pueden copiarse a la memoria caché. Por ejemplo una página de memoria virtual que corresponda con un búffer de dispositivo no puede almacenarse en caché, pues en ésta la información no se actualizaría (los datos leídos del dispositivo se actualizarían en la RAM, no en caché) y no se percibirían los cambios.

Fallos de página

Cuando un proceso al ejecutarse hace referencia a una dirección de memoria virtual, dentro de una página virtual, que no está cargada en un marco de página en memoria principal, decimos que ha ocurrido un **Fallo de Página**. La intención del proceso es ejecutar direcciones dentro de esa página, por tanto deberá cargarse en memoria (en un marco de página) y actualizar la Tabla de Páginas para reflejar la correspondencia entre la página virtual y el marco de página elegido para albergarla. Si ocurre que no hay marcos de página libres para albergar la página virtual, el Sistema Operativo deberá tomar la decisión de desalojar otra página virtual de la memoria principal para hacer "sitio" a la nueva página. Por este motivo es importante diseñar **Algoritmos de Reemplazo de Páginas**, que serán las políticas que el Sistema Operativo seguirá para desalojar un marco de página ocupado por otra página virtual, con el objetivo de albergar la página virtual que es preciso ubicar en memoria para que el proceso pueda continuar.

Ejemplo: Gestión de memoria virtual en un sistema de 32 bits con 2GiB de RAM

Las páginas virtuales de un proceso y los marcos de página que ocupan en memoria son elementos del mismo tamaño, simplemente se establece correspondencia entre ellas a la hora de ejecutar las instrucciones de la CPU.

Supón, como dice el enunciado, un SO que trabaja en 32 bits, con páginas de 4 KiB y tiene 2 GiB de RAM.

Los procesos pueden direccionar 2e32 bytes de memoria, es decir **4GiB**. Todas esas direcciones son virtuales y están divididas en un conjunto de páginas virtuales. Más concretamente, como **12 bits corresponden al desplazamiento** dentro de la página, ya que 2e12=4KiB, nos quedan **20 bits de los 32 para identificar páginas virtuales**.

Esas 2e20 páginas virtuales pueden ser asignadas al espacio de direcciones virtuales del proceso, aunque es probable que no sea necesario utilizar todo ese rango. Solamente aquellas páginas virtuales "ocupadas" son susceptibles de ser llevadas a memoria principal para ejecución. En este caso pasarían a ocupar un marco de página en memoria principal, también de 4KiB por definición, ya que el tamaño de las páginas virtuales y marcos de página coincide siempre.

Lo que hay que hacer a continuación es traducir direcciones virtuales a direcciones físicas para que la CPU pueda ejecutar instrucciones en el espacio de direccionamiento que conoce, es decir, el espacio de direcciones físicas de la memoria principal.

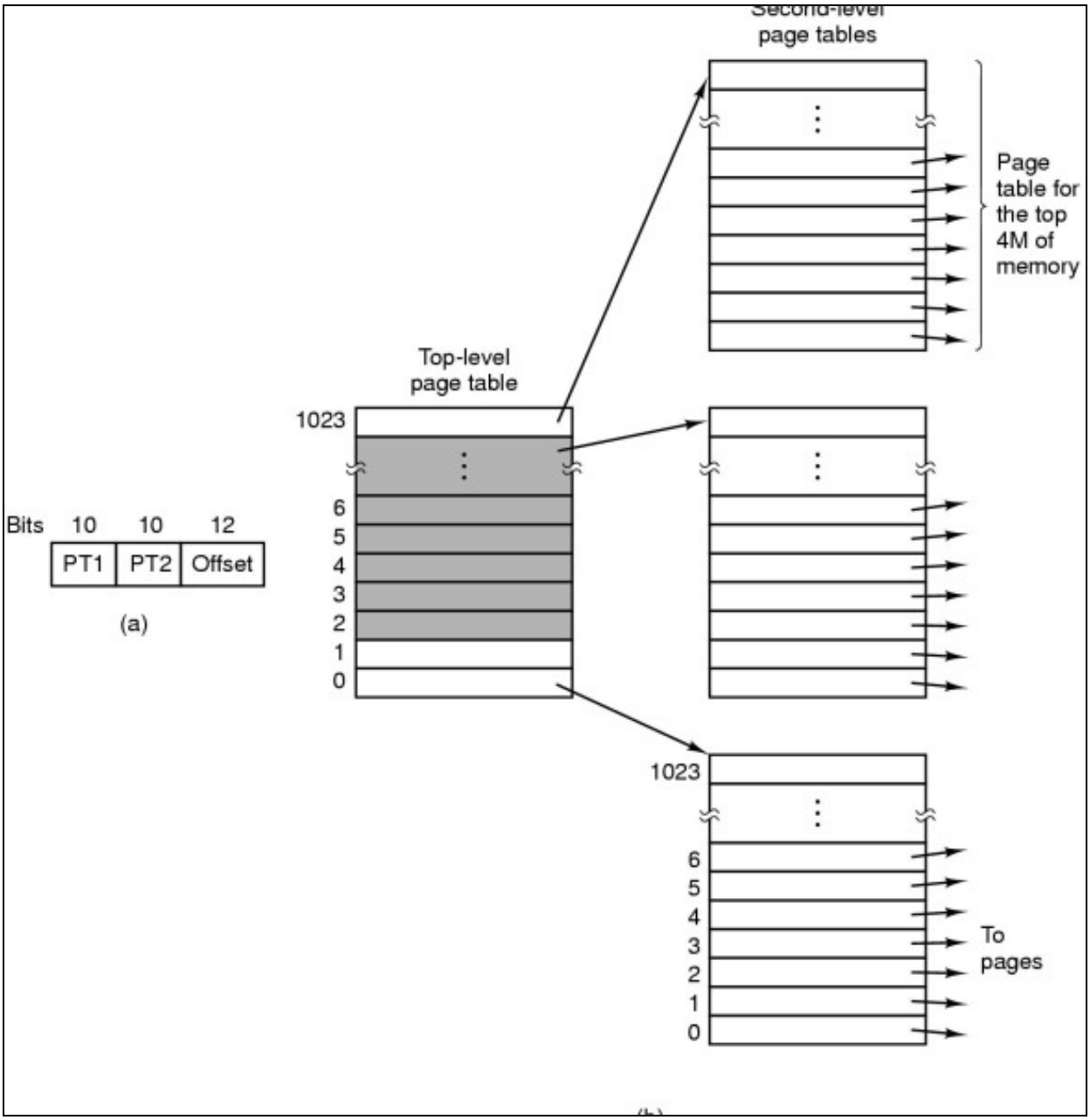
Al ser ambas del mismo tamaño, páginas virtuales y marcos, los 12 bits de la dirección virtual correspondientes al desplazamiento (y que indican la posición de un byte dentro del rango de 4KiB que abarca la página), se copian tal cual, en la **función de traducción de dirección virtual->física** de la dirección virtual a la dirección física, y suele corresponder con los bits menos significativos (los de la derecha) de la dirección. Los 20 bits a la izquierda identifican a las páginas virtuales del proceso, por tanto es con esas con las que vamos a establecer la correspondencia **página virtual->marco de página** a través de la tabla de páginas del proceso.

A continuación, **haciendo uso de la tabla de páginas del proceso**, obtenemos el número del marco de página que corresponde a esa página virtual, de esta manera obtenemos una combinación de 31 bits **[número_marco (19 bits), 12 bits desplazamiento]** correspondiente a la dirección física de la memoria (recuerda que ésta era de 2GiB), por tanto en teoría sería una dirección física de 31 bits, aunque es probable que se rellene un límite entero de bytes, en ese caso 32 bits.

Mecanismos de optimización

TLB: Las TLB, Búfer de Traducción Adelantada, son un mecanismo que mejora el rendimiento a la hora de resolver las referencias a memoria. Pensemos que, en cada referencia de un proceso de una dirección de memoria, deberemos determinar si la página está o no en memoria principal. Para ello deberá consultarse la Tabla de Páginas. Como la Tabla de Páginas está en la memoria principal, esto implicaría un acceso "extra" a memoria principal para consultar la Tabla, por tanto, cada referencia a memoria llevaría asociadas dos lecturas de la misma. Para agilizar y reducir el número de lecturas en memoria principal se utilizan estructuras de Memoria Caché (asociativa), denominadas TLB que almacenan las últimas entradas referenciadas de la Tabla de Páginas. De este modo, utilizando el mismo principio que utiliza la CPU para reducir las lecturas a memoria principal utilizando la Caché, conseguimos mejorar el rendimiento del sistema de gestión de memoria.

Tablas de Páginas multinivel: Este tipo de Tablas de Páginas reducen la cantidad de memoria principal consumida por la propia Tabla de Páginas. Recordad que las Tablas de Páginas están indexadas por el número de página virtual, por tanto en teoría debería tener tantas entradas como páginas virtuales podría tener un proceso. En sistemas de 32 bits este número es elevado, pero en sistemas de 64 es descomunal, por tanto no es posible mantener toda la Tabla de Páginas en memoria principal. Para solventar este problema se utilizan Tablas de Página multinivel, en los cuales el número de página virtual se dispersa en varios niveles que permite direccionar de modo independiente grandes trozos de memoria sin necesidad de tener una entrada en la Tabla de Páginas para cada página virtual.


Second-level
page tables
Page
table for
the top
4M of
memory
Top-level
page table
1023
6
5
4
3
2
1
0
Bits 10 10 12
PT1
PT2
Offset
(a)
1023
6
5
4
3
2
1
0
To
pages

En esta figura puede verse como la parte de la dirección que hace referencia al número de página, se divide en 2 trozos de 10 bits. El primero cubre un total de 2e10=1024 entradas en una Tabla de primer nivel. Cada entrada de ese nivel apunta a Tablas de segundo nivel que almacenan 1024 entradas cada una, se utiliza el segundo grupo de 10 bits de la dirección para ello, que apuntan a marcos de página. Los últimos 12 bits de la dirección, indican el desplazamiento dentro de la página. Con este esquema, incluso con esquemas de hasta 3 niveles en la práctica, se evita tener que disponer de toda la Tabla de Páginas en la RAM, ahorrando el correspondiente espacio de memoria. Cuando una dirección hace referencia, en el segundo nivel de la dirección (la tabla de primer nivel es residente, es decir, siempre está en RAM) a una Tabla de segundo nivel que no está en memoria, ésta se trae del disco y una vez en RAM puede continuar la resolución de la referencia para localizar el marco de página correspondiente. Si éste no está en memoria ocurrirá un Fallo de Página y habrá que buscar un marco de página libre para albergar la página virtual.

Resumen de los aspectos fundamentales de la gestión de memoria basada en Memoria Virtual

1. **Los procesos hacen referencia a sus direcciones virtuales**, es decir, las direcciones utilizadas por el proceso son virtuales y por tanto deberán ser traducidas por la CPU al ser ejecutadas
2. **Cada proceso tiene un espacio de direcciones virtuales dedicado**, el cual se divide en **páginas** que se hacen corresponder con **marcos de página** en la memoria física
3. **Para la traducción de dirección virtual a física se utiliza la Tabla de Páginas de cada proceso**. Esta es una labor desarrollada en hardware por la CPU
4. **El proceso de traducción debe ser rápido, directo y eficiente**, por ese motivo se realiza con soporte del hardware y de mecanismos de optimización en caché, como la **TLB**, y en consumo de memoria, como las **tablas de páginas multinivel o las tablas de páginas invertidas**

Algoritmos de Reemplazo de Páginas

Objetivos de un buen Algoritmo

La memoria principal es un elemento fundamental para el Sistema. Toda la información procesada por la máquina, programas y datos de programas deberá estar almacenada en la memoria principal para su tratamiento. Por tanto, es importante hacer un uso eficiente de ese recurso pues de ello dependerá en gran medida la ejecución estable y ágil del Sistema Operativo. A la hora de determinar qué páginas deben abandonar la memoria principal, para poder ubicar nuevas páginas necesarias para los procesos, tendremos en cuenta lo siguiente:

- Debe de, en la medida de lo posible, **elegirse páginas que no será necesario volver a traer a memoria en un período breve de tiempo**, pues de lo contrario se generarán muchos fallos de página
- Usaremos **políticas equitativas** en función de las características de los procesos

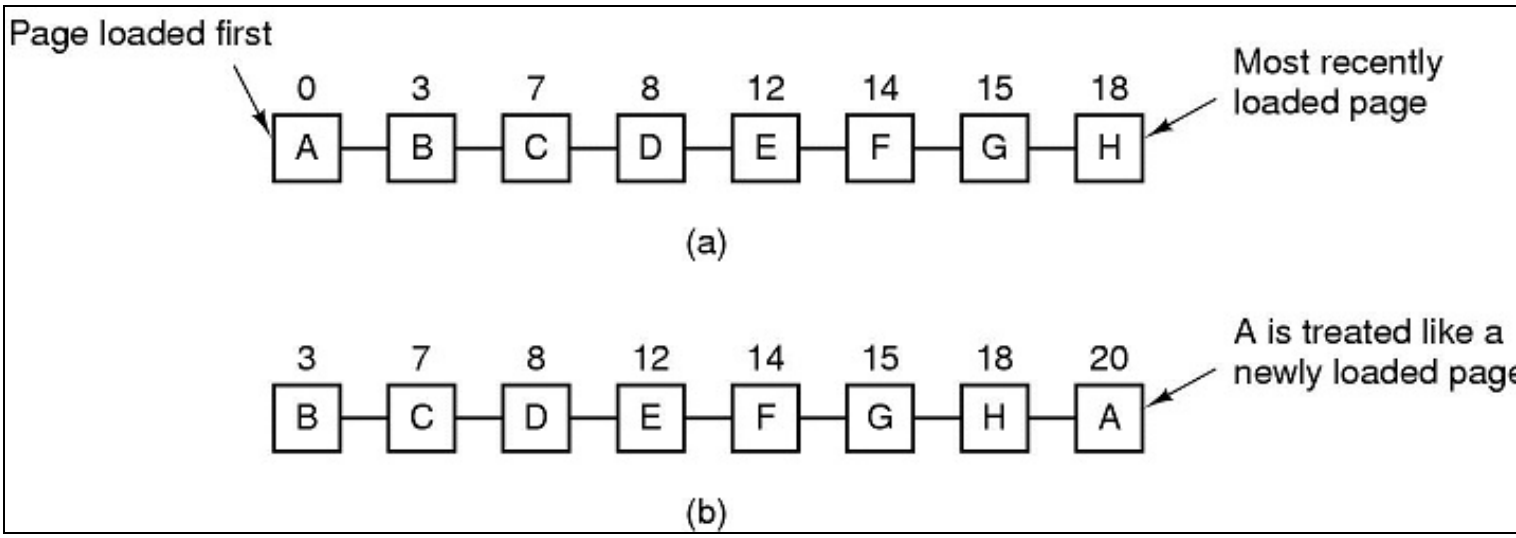
Algoritmos

- **Algoritmo óptimo**

Este algoritmo existe solamente en el campo teórico, en la práctica es irrealizable. La política de reemplazo sería la siguiente: *"Reemplazar aquel marco de página cuya información vaya a tardar más tiempo en ser accedida"* Aunque la afirmación es clara, no lo es tanto el implementar un algoritmo que permita obtener esa respuesta. Por tanto este algoritmo solo se utiliza para simulaciones y comparación de rendimiento con respecto a otros algoritmos que puedan ser implementados en la práctica.

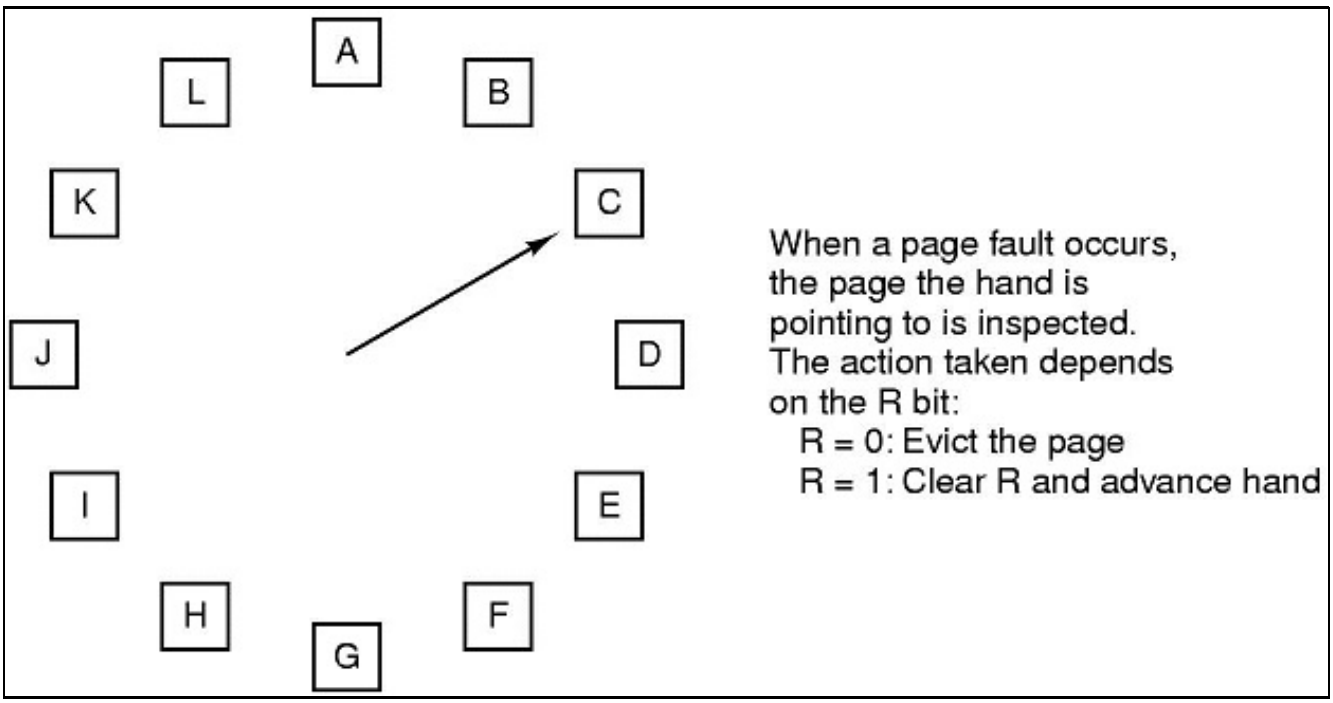
- **FIFO**

Del mismo modo que con los algoritmos de planificación de procesos, una estrategia simple y directa es elegir para reemplazo aquel marco de página que lleve más tiempo en la memoria, según una cola de tipo FIFO (Primero en llegar, Primero en ser atendido). Aquellas páginas que lleven más tiempo en memoria estarán a la cabeza de la cola y por tanto serán las primeras en ser reemplazadas. Esta estrategia no es óptima, pues que una página lleve mucho tiempo en memoria no es indicativo fiable de que sea una página que no se esté utilizando habitualmente. Si se reemplaza una página de memoria que pronto va a ser necesario traer de nuevo a ésta, se estarían generando lecturas de disco adicionales y el sistema se volvería ineficiente. Sería mucho mejor política utilizar otro criterio, como por ejemplo reemplazar de memoria aquellas páginas que lleven tiempo sin ser referenciadas por la CPU, es decir, que no está utilizando ningún proceso.



- **Segunda Oportunidad y Reloj**

Los algoritmos de Segunda Oportunidad y Reloj son realmente el mismo y se basan en utilizar algún tipo de información "extra" para determinar si una página va a ser reemplazada en la memoria o no, en fución de cierta información en la Tabla de Páginas. Revisando la información almacenada en la Tabla de Páginas, recordad que hay dos bits "M", modificado y "R", referenciada. En el caso de estos algoritmos utilizaremos una variación de FIFO que antes de reemplazar la página correspondiente compruebe si el bit R está encendido (R=1), lo que indica que la página fue accedida (referenciada desde un proceso) recientemente. En ese caso no se reemplaza esa página, sino que se pasa a la siguiente, colocando su bit R a 0 antes de pasar a la siguiente. El algoritmo del Reloj es ni más ni menos que Segunda Oportunidad que funciona de modo circular en la cola FIFO, adelantando la manecilla del Reloj que apunta a cada página candida a ser reemplazada y sobre la que se verificará la condición R=1 de reemplazo.



- **NRU**

NRU No Recently Used (no recientemente usada) **utiliza los bits M y R de las entradas de la Tabla de Páginas** El primero, **M**, indica si una página ha sido modificada en la memoria, en cuyo caso, antes de ser reemplazada deberá escribirse su contenido a disco para que los cambios queden almacenados. El otro bit, **R**, se utiliza para indicar que se ha hecho referencia a la página recientemente (en un intervalo de tiempo, por ejemplo, los últimos 5ms). Con esta información **el algoritmo consiste en crear categorías o grupos de páginas en función del valor de los bits M y R**. Los grupos son: **GRUPO 1:** R=0, M=0 (No referenciada, no modificada) **GRUPO 2:** R=0, M=1 (No referenciada, modificada) **GRUPO 3:** R=1, M=0 (Referenciada, no modificada) **GRUPO 4:** R=1, M=1 (Referenciada, modificada) Se recorrerá la cola FIFO de páginas en memoria buscando la primera ocurrencia de una del GRUPO 1, sino aparece ninguna en esas condiciones se dará otra vuelta a la cola en busca de una del GRUPO 2, y así sucesivamente. En caso de no encontrar ninguna en los GRUPOS 1, 2 y 3 habrá que tomar una de las GRUPO 4, la primera que aparezca. **El objetivo de este algoritmo es intentar no reemplazar aquellas páginas que se han utilizado recientemente** porque un proceso ha hecho referencia a direcciones de memoria dentro de la página, dando preferencia al reemplazo de aquellas que no han sido modificadas en memoria, pues desalojar una de esas implicaría tener que escribir al disco los cambios en memoria no respaldados, generando una operación adicional de escritura en disco.

- **LRU**

LRU, Least Recently Used (La menos usada recientemente) es un algoritmo más sofisticado que los anteriores. En el algoritmo de Segunda Oportunidad no podemos determinar si una página ha sido referenciada más veces que otra cuando su bit R esté a 1, solamente nos dice que se ha hecho referencia a ellas. LRU utiliza más información para determinar que página es la que menos se ha usado recientemente (LRU). Existen mecanismos por hardware y software para implementarlo, en cualquier caso el objetivo es siempre reemplazar páginas cuya probabilidad de ser utilizada de nuevo en un tiempo sea lo más bajo posible. Debido a la dificultad de implementación de **LRU**, sobretodo por hardware, existen aproximaciones bastante buenas en rendimiento a él, que imitan sus características. Una de estos algoritmos es **NFU** (Not Frequently Used, No Frecuentemente Usada) que implementa, mediante software, una política de aproximación, consistente en elegir la candidata a sustitución entre las páginas que no han sido utilizadas con frecuencia en un intervalo de tiempo reciente.

Otros Aspectos

Políticas de reemplazo global y local

Una decisión importante a la hora de realizar un reemplazo de página es considerar el ámbito en el que vea a realizarse la sustitución. **Una política de reemplazo global buscaría un marco de página candidato a reemplazo en toda la memoria**. **Una política de reemplazo local solamente consideraría para reemplazo los marcos de página asociados a un determinado proceso**.

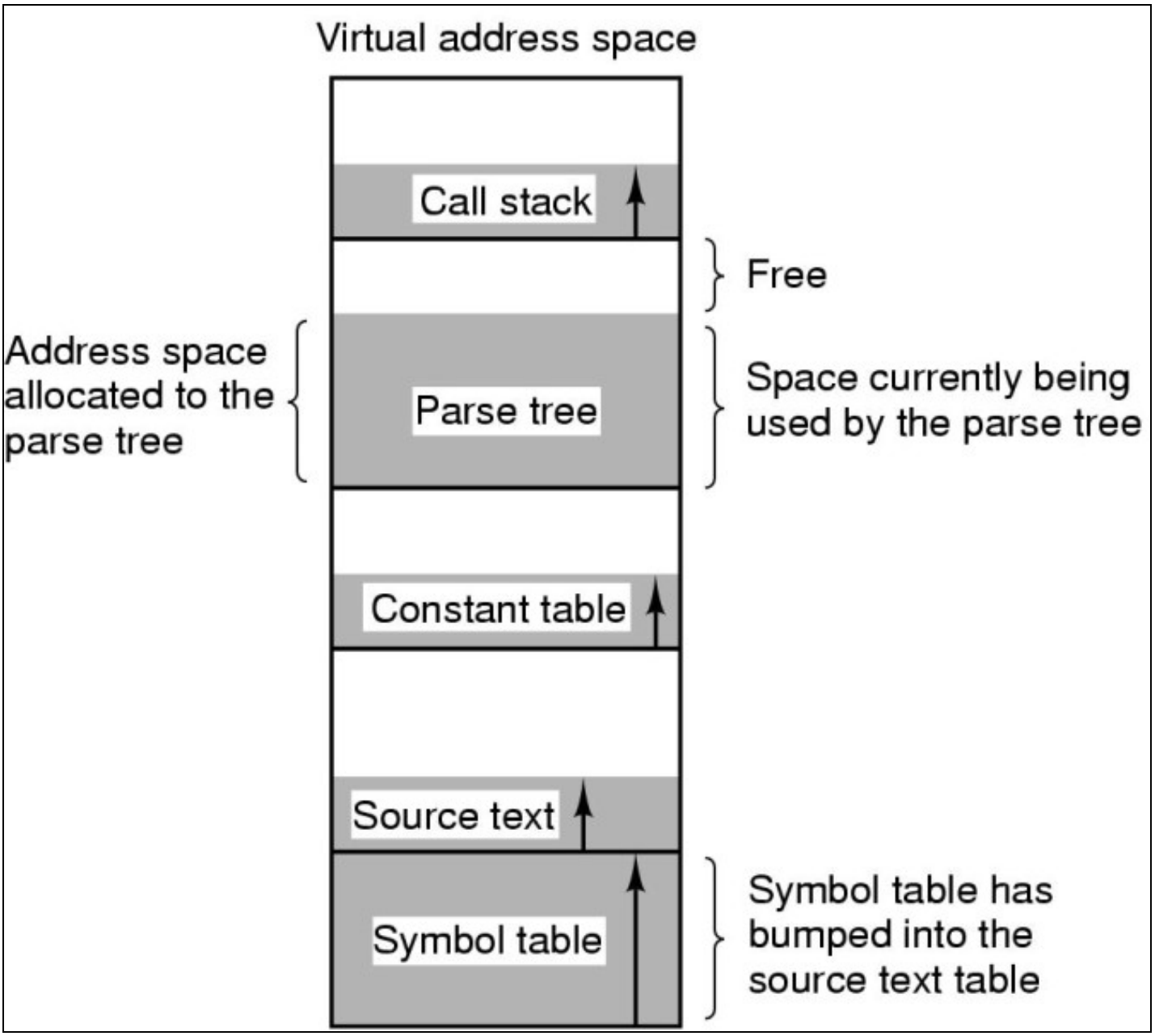
Conjunto de Trabajo

Una noción importante, fundamental para algunos algoritmos y políticas de gestión de memoria, es el Conjunto de Trabajo. **El Conjunto de Trabajo de un proceso son aquellas páginas que éste necesita para poder funcionar en un rango temporal determinado**. Evidentemente el Conjunto de Trabajo de un proceso va cambiando a lo largo del tiempo, pues las páginas a las que hace referencia van variando ya que sus direcciones de ejecución pasan de unas páginas a otras. Sin embargo en intervalos de tiempo más cortos, existe una **localización de cercanía espacial** de las referencias a memoria, esto quiere decir que por la propia construcción de los programas (ejecución de instrucciones en secuencia con saltos esporádicos), sus procesos derivados tienen a hacer referencia a direcciones de memoria concentradas en un conjunto de páginas determinado en pequeños intervalos de tiempo, **Principio de cercanía espacial**.

La noción de Conjunto de Trabajo permite establecer políticas de gestión de memoria consistentes en garantizar, en la medida de lo posible, que el Conjunto de Trabajo correspondiente a un proceso en un determinado instante corresponda a páginas que están en la memoria principal. Esto mejora el rendimiento ya que durante un tiempo no ocurrirán muchos fallos de página para el proceso.

Segmentación

La Segmentación es una **técnica consistente en dividir el espacio de direcciones virtual de los procesos en varios segmentos con direccionamiento virtual independiente**, esto es de 0 al máximo de la arquitectura. Con la segmentación se puede conseguir que los procesos puedan ser más grandes, aunque el objetivo principal no es éste.


Virtual address space
Call stack
Free
Address space allocated to the parse tree
Parse tree
Space currently being used by the parse tree
Constant table
Source text
Symbol table
Symbol table has bumped into the source text table

En este gráfico vemos un espacio de direcciones virtuales de un proceso genérico. Hay áreas del espacio de direcciones del proceso que crecen a lo largo del tiempo, como la pila y el área de memoria dinámica (se ven unas flechas apuntando hacia la zona de crecimiento). Puede darse el caso de que esas áreas crezcan demasiado y "colisionen" con otras áreas del proceso utilizadas para otros fines.

La segmentación permite definir "segmentos" diferentes, correspondientes a espacios de direcciones virtuales independientes, de modo que esas áreas puedan crecer en mayor medida y de modo mucho más controlado.

Otras aplicaciones de la segmentación son su utilización para albergar librerías de carga dinámica (**DLL**, Dynamic Loadable Libraries), que son librerías de código, archivos ejecutables, que pueden ser compartidos entre los procesos para poder así ahorrar espacio de memoria RAM. Por ejemplo, las librerías de control de gráficos en Windows o Linux son compartidas por todas aquellas aplicaciones gráficas. Por tanto, sería un desperdicio el repetir en el espacio de direcciones virtual de cada proceso la carga de esas librerías. De hacer esto así, cada copia de esa librería se paginaría de modo independiente para cada proceso y el consumo de memoria principal aumentaría en proporción al número de procesos que la utilizaran: si una librería ocupara 1 MB en memoria RAM y los procesos que la utilizan la cargan de modo estático, es decir, directamente en su espacio de direcciones virtual, suponiendo que tenemos 20 procesos que la utilicen, tendríamos 20x1MB = 20 MB de consumo de RAM, cuando la librería es la misma para todos y ésta ocupa solamente 1 MB en memoria!. Por tanto estaríamos desperdiciando 19 MB. Con las DLL se utilizan segmentos para ubicar estas librerías a los que pueden acceder los procesos a través de llamadas a funciones (no directamente incorporándolas en su espacio de direcciones) que resuelven la dirección de memoria física de los marcos de página en los que se ubican las páginas del segmento de la librería correspondiente. Este mecanismo es mucho más fácil de alcanzar y más flexible de gestionar con el uso de la segmentación.

En resumen, mediante la segmentación disponemos de un mecanismo que permite independizar las distintas áreas de crecimiento del direccionamiento virtual de un proceso, haciendo más flexible la gestión de ese crecimiento. Por otro lado también permiten ubicar librerías compartidas en memoria dentro de un segmento al que podrán hacer indirectamente referencia los procesos clientes de la librería, facilitando el acceso a la misma y ahorrando recursos de memoria.

A la vista de las ventajas ofrecidas por la segmentación, **los Sistemas Operativos actuales utilizan un modelo mixto de memoria virtual segmentada y paginada**, **de modo que las direcciones virtuales de un proceso hacen referencia a una dirección en una página dentro del espacio de direcciones de un determinado segmento del proceso**

Referencias

Semántica del comando free

https://tuxfiles.wordpress.com/2012/01/07/entendiendo-el-comando-free/

Linux Memory Management

http://www.linuxhowtos.org/System/Linux%20Memory%20Management.htm

http://www.slideshare.net/raghusiddarth/memory-management-in-linux-11551521

https://es.slideshare.net/anil_pugalia/linux-memory-management

Volver

JavierFP 17:31 03 dec 2018 (CET)