Sistemas Operativos I

Módulo IV

ADMINISTRACIÓN DE LA MEMORIA

1

Temas del Módulo IV

- Introducción.
- Preparación de un programa para su ejecución.
  - Compilación.
  - Linkeado.
  - Cargado.
  - Ejecución.
- Esquemas simples de administración de memoria.
  - Particiones fijas.
  - Particiones variables. Compactación.
  - Overlay.
  - Swapping.
- Principios de memoria virtual.
  - Segmento único por proceso.
  - Múltiples segmentos por proceso.
  - Paginado.
  - Segmentación con paginado.
- Implementación de memoria virtual.
  - Tablas de segmentos.
  - Tablas de páginas.
- Esquemas de asignación de memoria.
  - First Fit, Best Fit.
  - Estático, Dinámico.

2

Introducción

Sistemas multitarea

Característica principal: múltiples procesos de usuarios y del sistema operativo comparten dinámicamente los recursos físicos y lógicos del sistema.

Recursos del sistema: procesador, memoria, archivos, dispositivos de entrada-salida, etc.

Módulos siguientes de Sistemas Operativos I

Se estudiarán temas relacionados con la administración de los recursos más importantes del sistema:

Módulo III: Planificación del procesador.

Módulo IV: Administración de la memoria.

Módulo V: Administración de archivos y dispositivos de E/S.

3

Introducción

Objetivos de los Sistemas Multiprogramados.

Para que los recursos del sistema sean compartidos dinámicamente por todos los procesos de manera eficiente, es necesario:

- En general, respecto del sistema:

  mejorar la respuesta global del sistema, incrementando la utilización de estos recursos.

- En particular, respecto de la memoria:

  tener un alto grado de multiprogramación. Esto significa: mantener múltiples procesos activos en la memoria principal del sistema.

  Esta tarea es realizada por el módulo del SO denominado **Administrador de la Memoria**.

4

Introducción

Tarea de Administración de la Memoria

- Lograr un alto grado de multiprogramación implica:
  asignar espacios de memoria a múltiples procesos administrando eficientemente la memoria
- La tarea de administración de la memoria es realizada por:
  - **Módulo de Administración de Memoria (SO)**
  - **MMU o Unidad de Manejo de Memoria (Hw)**

  Ambos elementos están fuertemente relacionados.

5

Preparación de un programa para su ejecución

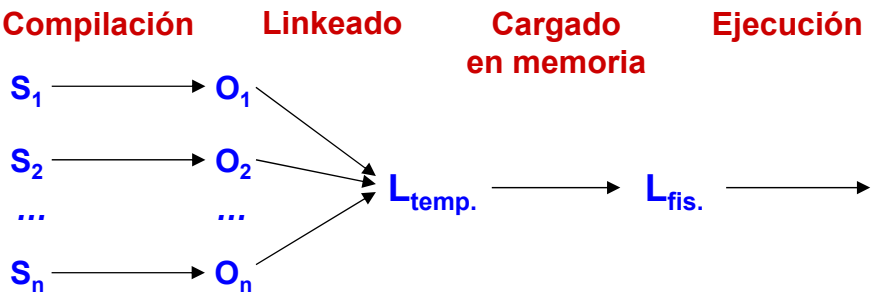
Para poder comprender los diferentes esquemas de manejo de la memoria, antes es conveniente recordar cómo se prepara un programa para su ejecución.

Téngase presente que:

- La mayoría de los programas de usuario y del Sist. Operativo están escritos en lenguajes de alto nivel (C, C++). Un programa así escrito se denomina **programa fuente (S_i)**.
- Para que el programa fuente pueda ser ejecutado, necesita pasar por una serie de transformaciones previas:

6

Preparación de un programa para su ejecución



S_i: programa fuente

O_i: módulo objeto

L_{temp}: módulo cargable (reubicable)

L_{fis}: módulo ejecutable

7

Preparación de un programa para su ejecución

Compilación.

Objetivo:

- Construir **Módulo objeto (O_i)** a partir del programa fuente **(S_i)**.

Tarea:

- Traducción de lenguaje de alto nivel a lenguaje de máquina.

Descripción de la tarea:

- Puede hacerse en más de una etapa. Implica el análisis sintáctico, semántico y léxico de las instrucciones de alto nivel. A partir de ese análisis se realizan las optimizaciones de hardware, por ejemplo, elegir qué registros del procesador utilizarán las variables.
- En una segunda etapa la compilación genera el código de máquina a partir de un archivo intermedio llamado "lenguaje ensamblador" (assembler). El resultado es un archivo binario.

8

Preparación de un programa para su ejecución

Linkeado

Objetivo:

Construir L_{temp} (módulo cargable y ejecutable en memoria)

Tarea:

- Unión de varios módulos objeto **(O_i)** en un único binario.

Descripción de la tarea:

- Normalmente, los programas complejos son escritos en varios módulos, los que deben unirse para formar un único programa.
- Función principal de esta etapa: resolver las referencias de localizaciones de operandos, instrucciones, estructuras de datos de los distintos módulos a referencias (direcciones) de memoria.
- Las referencias de memoria en L_{temp} son direcciones relativas, no absolutas (direcciones reales), debido a que el SO no sabe, en este momento, en qué lugar de la memoria será ubicado.

9

Preparación de un programa para su ejecución

Cargado en memoria

Objetivo:

- Construcción de L_{fis} (módulo cargado en memoria principal para ser ejecutado).

Tarea:

- Transformación de L_{temp} a L_{fis}

Descripción de la tarea:

- La ubicación en memoria se determina en el momento en que L_{temp} va a ser cargado desde disco para ser ejecutado.
- Programa cargador o loader: carga el programa en memoria principal y transforma las referencias relativas en absolutas.
- Referencias absolutas: direcciones de memoria principal (direcciones físicas)

10

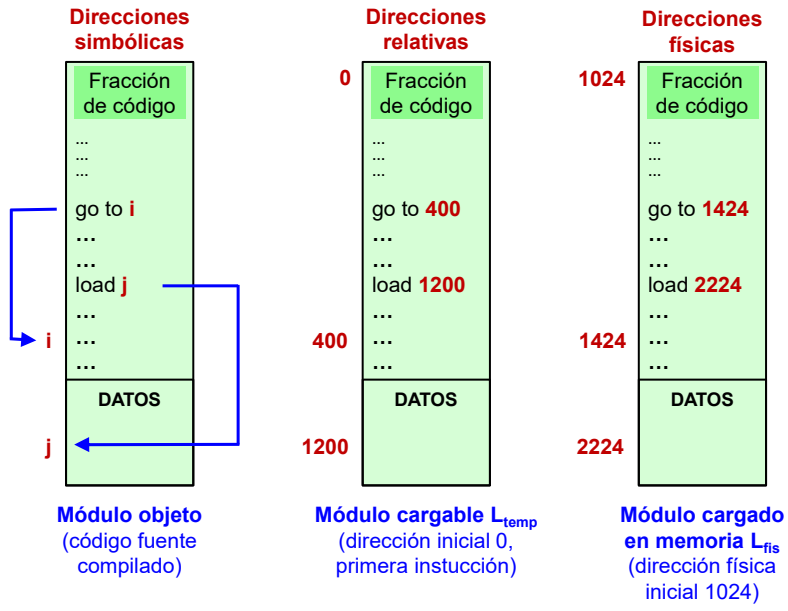
Preparación de un programa para su ejecución

Resumen de traducciones de direcciones

- Cuando se programa en alto nivel: se usan direcciones simbólicas. (el programa solo está escrito o editado).
- Cuando se construye L_{temp} se traducen las direcciones simbólicas en relativas. (varios módulos unidos en una tarea)
- Cuando se instala en memoria (L_{fis}): se traducen direcciones relativas en absolutas o físicas (direcciones reales de memoria)

11

Ejemplo de traducción de direcciones



12

Esquemas simples de administración de memoria

Planteo del problema de administración de la memoria.

La necesidad de administrar la memoria nace del objetivo de tener multiprogramación en el sistema.

(1) Esto implica que: varios procesos activos deben compartir la memoria al mismo tiempo para poder ejecutar.

Si bien el costo del hardware decrece continuamente, la memoria es un recurso siempre escaso en el computador debido, precisamente, a su costo.

Conclusión: no se puede satisfacer (1) de manera directa y simple puesto que el tamaño resultante de la suma de todos los programas en el sistema, normalmente, excede la capacidad de la memoria principal del computador.

- Se plantea así un problema a resolver: **debe administrarse la capacidad disponible de memoria principal.**

13

Esquemas simples de administración de memoria

Planteo del problema de administración de la memoria

(cont.)

- Para solucionar el problema expuesto, necesariamente debe realizarse la siguiente tarea:

  Los programas deben ser divididos de alguna forma en partes que serán cargadas desde el disco a memoria, sólo cuando esos programas vayan a ser ejecutados.

- A lo largo de la historia de la computación se han desarrollado y usado distintas técnicas para llevar a cabo la tarea mencionada.
- Se estudiarán esas técnicas, comenzando por las más simples.

14

Esquemas simples de administración de memoria

Particiones Fijas de la Memoria

- El esquema de partición de memoria más simple y antiguo.
- La memoria se divide en un número de espacios separados llamadas particiones que tienen tamaños diferentes para almacenar procesos de distintos tamaños.
- Los tamaños son fijados en el momento en que el sistema se inicializa (levanta) y no pueden ser modificados más; es decir, la memoria se divide antes de iniciarse la ejecución de los programas.
- Problema de partición fija. Hay dificultad en estimar el tamaño de las particiones: es complicado predecir el tamaño preciso de los programas que correrán en el sistema.
- Una estimación incorrecta de las particiones puede llevar a una mala utilización de la memoria.

15

Esquemas simples de administración de memoria

Particiones Fijas de la Memoria *(cont.)*

- Las particiones fijas imponen la necesidad de planificar los procesos que las van a ocupar; es decir, se debe asignar un proceso a cada partición.
- Existen dos esquemas: asignación estática y dinámica.

1 - Asignación Estática: se crea una cola de procesos por cada partición.

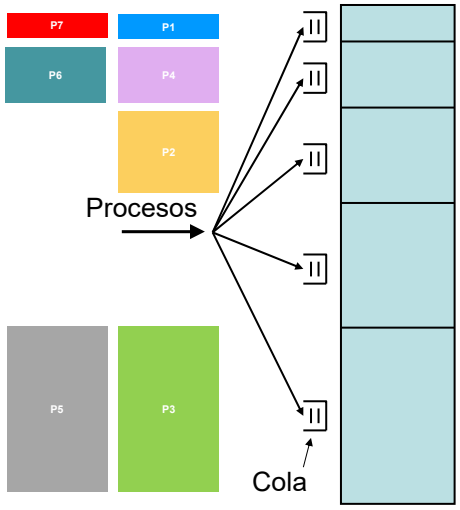
Los procesos son planificados para ocupar la partición que más se ajuste a su tamaño.

Ventaja: es la más simple de implementar.

Desventaja: algunas particiones pueden quedar vacías mientras en otras colas pueden haber procesos de tamaño menor que estas particiones. Esto lleva a una baja utilización de la memoria.

16

Esquemas simples de administración de memoria



Nunca se utilizó…

Asignación estática
Una cola de procesos
por cada partición

17

Esquemas simples de administración de memoria

Particiones Fijas de la Memoria *(cont.)*

2 - Asignación dinámica: una única cola de procesos para todas las particiones.

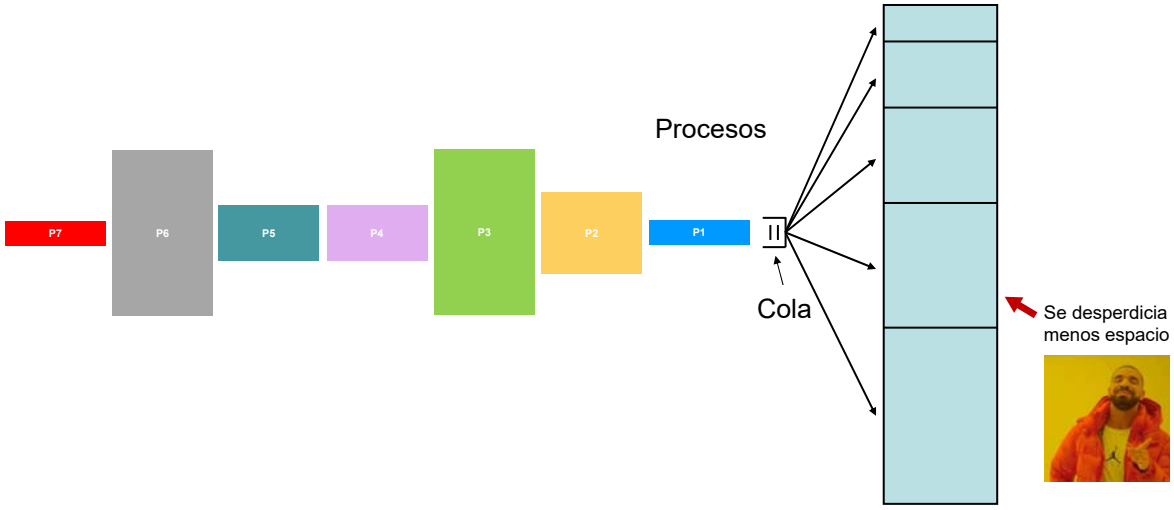
Los procesos que ingresan al sistema son puestos en una única cola. Desde esta cola cada proceso va ubicándose en la partición adecuada de acuerdo a su tamaño.

Ventaja: se obtiene una mejor utilización de la memoria, puesto que si existe un espacio desocupado en ésta que pueda albergar a un proceso listo a ser ejecutado, es usado aunque no se ajuste exactamente al tamaño del proceso.

Desventaja: complejidad en la implementación.

18

Esquemas simples de administración de memoria



Asignación dinámica
Una única cola de procesos
para todas las particiones

19

Esquemas simples de administración de memoria

Particiones Variables de la Memoria.

Este esquema se caracteriza porque la memoria se asigna según el tamaño de cada proceso y en el momento en que será cargado en memoria; es decir, es una asignación dinámica.

- Aquí se deben considerar 2 cuestiones:
  1° - los procesos no finalizan su ejecución en el mismo orden en que ingresaron al sistema, además,
  2° - los procesos tienen distinto tamaño,
  al final, la memoria es un conjunto de bloques ocupados y vacíos de diferentes tamaños.
- Esta situación lleva a un problema, llamado fragmentación externa de la memoria.

20

Esquemas simples de administración de memoria

Particiones Variables de la Memoria. *(cont.)*

Fragmentación externa de la memoria: es cuando existen varios bloques libres de memoria de tamaño tan reducido que no pueden ser ocupados por ningún proceso.

Tarea principal del módulo Administrador de la Memoria en este esquema:

- Mantener una tabla con los espacios libres que se van asignando a los procesos nuevos a medida que ingresan al sistema.
- Los espacios liberados por los procesos que finalizaron deben ser unidos para formar grandes bloques libres de memoria y evitar así la fragmentación de la memoria.

Solución: compactar la memoria.

21

Esquemas simples de administración de memoria

Particiones Variables de la Memoria. Compactación

- Considérese la siguiente situación: en un momento dado ingresa un proceso al sistema que no cabe en ninguno de los huecos vacíos de memoria pero, sin embargo, el total de varios huecos no contiguos es mayor que el tamaño del proceso.
- Pregunta: qué debe hacer el SO? (Ver Ejemplo en figura sigte.)

Hi = huecos (espacios vacíos de memoria)

A, B, C = Procesos instalados en memoria.

Por ejemplo, supóngase que **B** finaliza dejando un hueco de tamaño **H2** entre **A** y **C**.

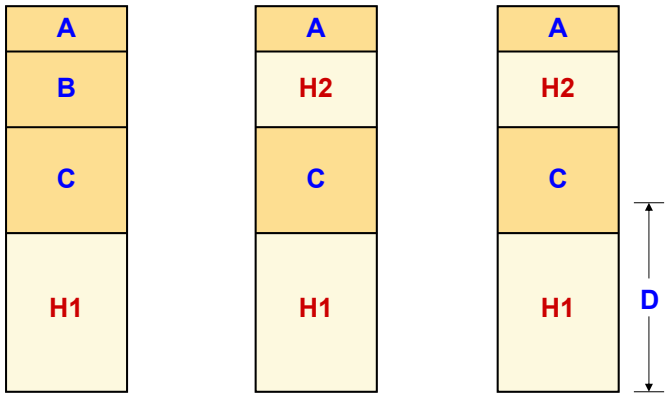
Luego el proceso **D** es elegido para ejecutar: no podrá ocupar la memoria porque su tamaño excede a **H1** y también a **H2**.

No obstante, si se juntan los huecos **H1** y **H2**, el espacio total alcanza y sobra para contener al proceso de tamaño **D**.

22

Esquemas simples de administración de memoria

Particiones Variables de Memoria. Compactación



D > H1 ; D > H2

D < H1 + H2

23

Esquemas simples de administración de memoria

Particiones Variables de la Memoria. Compactación *(cont.)*

Solución 1: D espera hasta que finalice **C**.

Solución no óptima desde el punto de vista de performance y utilización de memoria.

Solución 2: se reubica **A** y **C** para lograr un espacio libre mayor.

Compactación de la memoria

Es la acción de desplazar bloques de datos dispersos en la memoria y agruparlos para crear espacios libres de gran tamaño.

24

Esquemas simples de administración de memoria

Particiones Variables de la Memoria. Compactación *(cont.)*

Ejemplo.

(en la figura sigte. el número en cada bloque indica su tamaño).

Supóngase que ingresa un nuevo proceso cuyo tamaño es 10.

Evidentemente, ningún hueco puede alojar a este proceso - fig.(a) -

Solución de Compactación 1: desplazar todos los bloques de memoria a un extremo, p/ej: al principio -fig.(b)-.

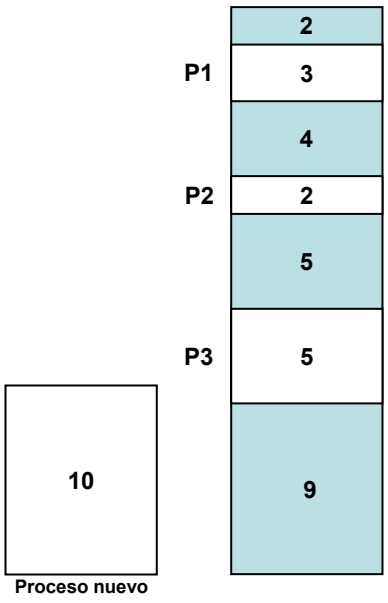
De esta manera se crea un hueco de tamaño 20 y así se puede cargar el proceso nuevo.

Ventaja: es simple.

Desventaja: movimiento excesivo de bloques ocupados. Reduce la performance global del sistema.

25

Esquemas simples de administración de memoria



Compactación de Memoria: Solución 1.

26

Esquemas simples de administración de memoria

Particiones Variables de la Memoria. Compactación *(cont.)*

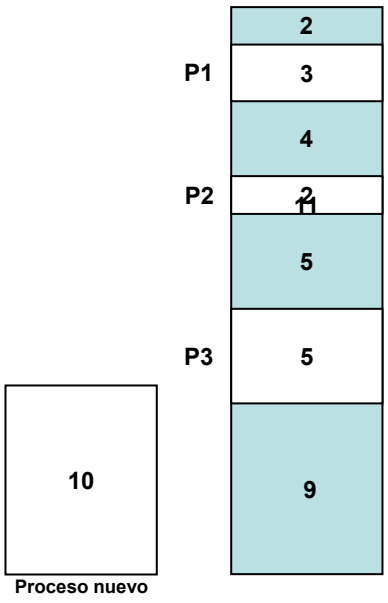
Solución de Compactación 2: desplazar sólo los bloques necesarios hasta crear un hueco igual o mayor que 10 -fig.(c)- Se movieron P1 y P2 para crear un hueco de tamaño 11 (fig. sigte.)

Ventaja: menor movimiento de bloque ocupados.

Desventaja: algoritmo no tan simple.

27

Esquemas simples de administración de memoria



Compactación de Memoria: Solución 2.

28

Esquemas simples de administración de memoria

Solución de Compactación 3: Buscar el mínimo movimiento de bloques -fig.(d)-. Se ha movido sólo el proceso P2 y con ello se ha logrado un hueco de tamaño 11.

Ventaja: Mínimo desplazamiento de bloques llenos.

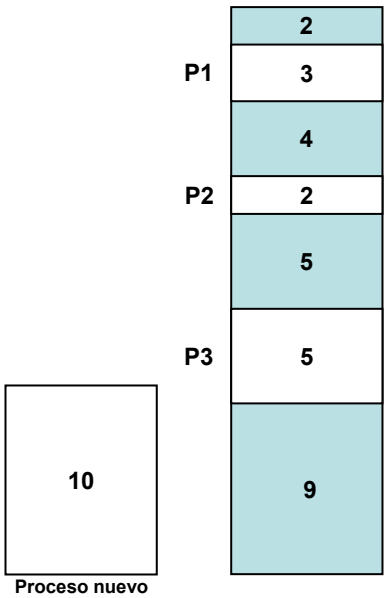
Desventaja: alta complejidad de los algoritmos encargados de determinar el movimiento mínimo de bloques.

Conclusión

Debido a la desventaja señalada se usan en mayor medida las soluciones propuestas en las figs. (b) y (c).

29

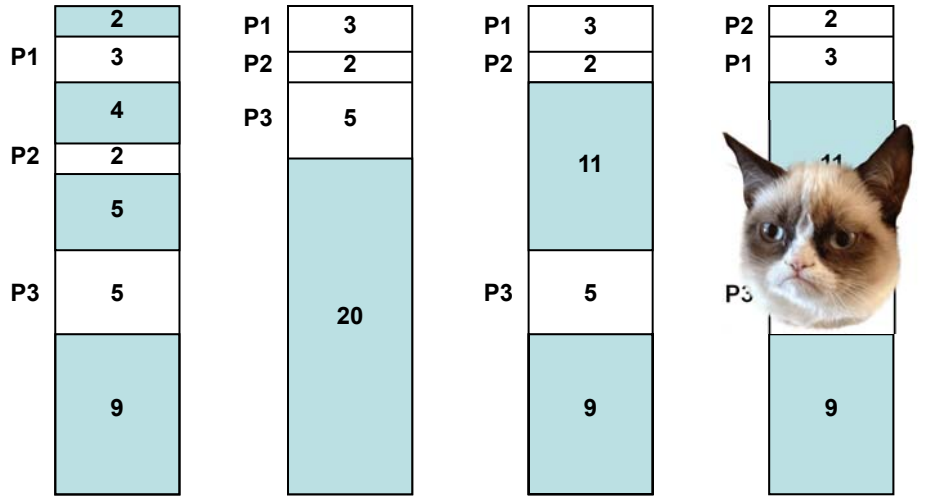
Esquemas simples de administración de memoria



Compactación de Memoria: Solución 3.

30

Esquemas simples de administración de memoria



31

Otras técnicas de administración de memoria

Técnica de Overlay

El tamaño físico - real - de la memoria principal impone varias limitaciones en sistemas mono y multiprogramas:

- **Sistemas Monoprograma.** El tamaño máximo de un proceso no debe ser mayor que el espacio disponible de memoria.
- **Sistemas Multiprograma.** El tamaño máximo posible de un proceso (en un sistema con partición fija de la memoria) es el de la mayor partición disponible.

Cuando se exceden esos límites, los programas deben ser divididos en módulos o segmentos que pasarán a ocupar la memoria en el momento en se los acceda, dejando en el disco los módulos no necesarios del programa en ejecución.

Esta técnica se denomina **overlay** o uso de overlays: significa reemplazo de módulos.

32

Otras técnicas de administración de memoria

Técnica de Overlay *(cont.)*

Esta tarea era realizada por el programador, quién debía especificar, en cada momento, cuáles de los segmentos del programa debían residir en memoria en forma simultánea para una ejecución satisfactoria.

Desventaja de la técnica de overlay: dependencia del programador en una tarea que debería ser realizada por el SO.

33

Otras técnicas de administración de memoria

Técnica de Swapping

- Es el pasaje de procesos de memoria primaria a secundaria y viceversa.
- Sea el escenario siguiente: arriba un proceso de alta prioridad al sistema requiriendo una cierta cantidad de memoria que no hay disponible.
- Ante esta situación: el SO desaloja de memoria uno o más procesos y los pasa a disco para dejar libre el espacio suficiente requerido por el proceso prioritario.
- Esta operación se denomina **swapp-out**.
- Cuando este proceso finalice su ejecución, los procesos serán pasados nuevamente de disco a memoria primaria (**swapp-in**).

34

Características de las soluciones vistas

Las soluciones anteriores no son óptimas porque:

- **Pueden llevar a una baja utilización de la memoria:**
  En asignación estática de memoria con particiones fijas: mucho desperdicio de espacio al no coincidir exactamente el tamaño de los segmentos de datos con los espacios disponibles.
- **Requieren la utilización de algoritmos de alta complejidad:**
  En asignación variable de memoria: para búsqueda de espacios libres en donde ubicar un segmento de datos y para compactación.
- **Necesitan la participación directa del programador:**
  Como en la técnica de overlay.

35

Principios de Memoria Virtual

Conceptos preliminares de memoria virtual

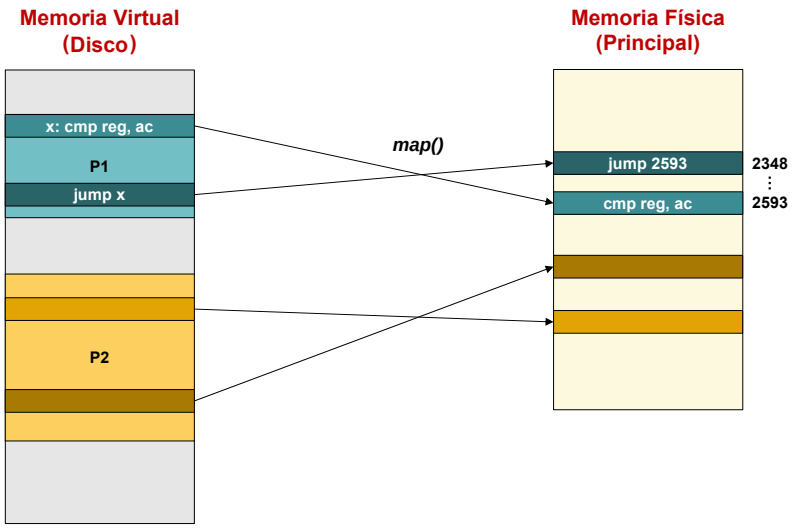
- Se trata de ocultar al usuario que la memoria principal tiene un tamaño físico limitado, de tal forma que él tenga la idea que la misma es continua e infinita.
- Esto se logra usando el disco como complemento de la memoria principal.

En concreto: al usuario de un sistema que usa memoria virtual:

1. Se le oculta que la memoria física es limitada en espacio y está compartida por varios procesos.
2. En cambio, se le presenta la visión de que a cada proceso se le provee un espacio privado e infinito de memoria.
   Estos espacios otorgados a los procesos son contiguos, es decir, están uno al lado del otro.
   ¿Cómo se materializa lo dicho?

36

Principios de Memoria Virtual



37

Principios de Memoria Virtual

Conceptos preliminares de memoria virtual *(cont.)*

En la figura anterior se muestra un ejemplo en el que se observa:

- Dos espacios de memoria virtual (MV1 y MV2) que pertenecen a un mismo proceso P
- Parte de MV1 y MV2 se han cargado en memoria física o principal
- La función map realiza el mapeo de direcciones

Mapeo

- Significado: traducción de direcciones virtuales a direcciones físicas.
- Tarea de traducción: es llevada a cabo por la función map( ).
- Implementación de la función map(): MMU (Memory Manager Unity).

38

Principios de Memoria Virtual

Conceptos preliminares de memoria virtual *(cont.)*

Direcciones virtuales

Conceptualmente, son similares a las direcciones relativas de L_{temp} vistas en el ejemplo anterior.

Direcciones físicas

Son similares a las direcciones absolutas o físicas de L_{fis}. Sus valores dependen del espacio físico en dónde está ubicado el proceso en memoria principal.

En la diapositiva que sigue se muestra el gráfico de módulo objeto, recargable y cargado en memoria.

39

Implementación de la memoria virtual

¿Cuándo debe realizarse el mapeo?

- Cuando los procesos listos para ser ejecutados o en ejecución contienen en sus códigos direcciones que referencian a alguna otra instrucción o dato, se trata de direcciones virtuales.
- Cuando una instrucción va a ser ejecutada y contiene una dirección virtual, antes debe realizarse el mapeo, es decir, debe encontrarse la dirección física correspondiente.
- Lo anterior resulta obvio, puesto que el procesador solo maneja direcciones físicas (direcciones de memoria principal)

Conclusión:

La función mapeo debe actuar cada vez que el procesador invoca una instrucción que contiene una dirección virtual.

40

Implementación de la memoria virtual

En la **implementación de la memoria virtual** se estudiarán diferentes

- **Modos de asignación de memoria virtual a los procesos** y, para cada modo de asignación, se verá la
- **Implementación del mapeo**, es decir, cómo se traducen direcciones virtuales a direcciones físicas.

41

Implementación de la memoria virtual

Modos de asignación del espacio de memoria virtual

- **Espacio de Segmento Único por Proceso.**
  Asignación de almacenamiento contiguo: cada proceso ocupa un bloque de memoria que es indivisible.
- **Paginado.**
  Asignación de páginas: cada proceso ocupa un espacio de memoria que se divide en bloques pequeños llamados páginas.
- **Espacio de Múltiples Segmentos por Proceso.**
  1. Asignación continua de segmentos: cada proceso ocupa varios segmentos indivisibles de memoria.
  2. Asignación de múltiples segmentos con paginado: cada proceso ocupa varios segmentos los que, a su vez, son divididos en páginas.

42

Espacio de segmento único por proceso

Significa que:

- Cada proceso ocupa un único segmento indivisible.
- Por lo tanto, cuando es cargado desde el disco a memoria principal, se trae todo el segmento que ocupa el proceso.

Asignación de almacenamiento contiguo

Debido a que cada proceso ocupa un segmento único e indivisible en memoria virtual, cuando un dado proceso es traído a memoria física ocupará una región contigua (bloque único) de memoria física.

43

Espacio de segmento único por proceso

Implementación del Mapeo

- Se usa un **Registro de Reubicación** (**RR**) contenido en MMU. **RR** contiene la dirección base del espacio en memoria física que ocupa el proceso que se está ejecutando en ese momento.

Secuencia de ejecución de un proceso que está en memoria virtual

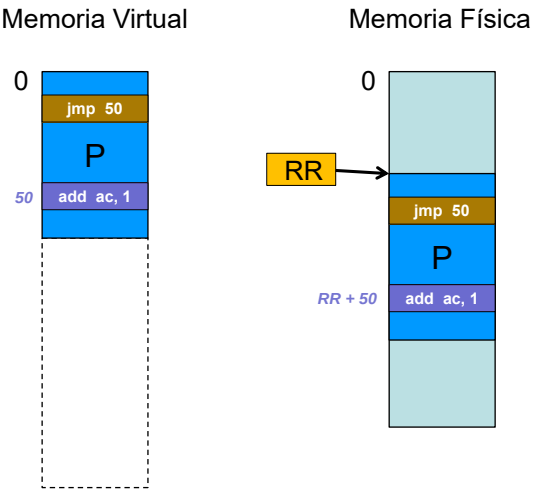
1. El SO carga el segmento que ocupa el proceso (código y datos) en un espacio libre de memoria principal.
2. A continuación carga en el **RR** la dirección física de comienzo del segmento. Si esta dirección es también el comienzo del código a ejecutar, deberá cargarse en el PC (program counter).
3. Como ya se dijo, se realiza el mapeo cada vez que se ejecuta una instrucción del programa que invoque una dirección virtual.

44

Espacio de segmento único por proceso

Implementación del Mapeo en un proceso cargado.

```
map (dir_virtual)
{
    dir_fisica = RR + dir_virtual;
    return dir_fisica;
}
```



45

Espacio de segmento único por proceso

Implementación del Mapeo en un proceso cargado.

Supóngase que se va a ejecutar un proceso y se ha realizado lo indicado en los puntos 1 y 2 anteriores.

3. El mapeo se implementa de la siguiente forma:

```
map(dir_virtual)
{
   dir_fisica = RR + dir_virtual
   return dir_fisica
}
```

(Aquí se ha supuesto que al inicio del segmento en memoria virtual se le ha otorgado la dirección virtual 0)

Ver esquema siguiente.

46

Espacio de segmento único por proceso

Implementación del Mapeo en varios procesos cargados.

La implementación del mapeo vista permite la existencia de varios procesos cargados en memoria física simultáneamente, donde cada proceso ocupa un único segmento.

En el caso de múltiples procesos activos en el sistema, el SO debe tener almacenado un **Vector de Estado** en el PCB de cada proceso **P** con la siguiente información:

- Un contador de instrucciones de memoria virtual ("program counter of virtual memory"): $\mathbf{PC_V}$.
- La dirección base ($\mathbf{DB_P}$) de la memoria física que ocupa **P**.

Vector de Estado: PC_V ; DB_P

47

Espacio de segmento único por proceso

Ejemplo

Se produce un cambio de proceso entre **P1** y **P2**. (P1 deja el procesador y P2 comienza a ejecutar en su lugar).

Pasos para realizar el mapeo de direcciones de P2

1. $\mathbf{PC_{P1} = PC_v}$; $\mathbf{DB_{P1} = RR}$ [se almacena en el PCB de P1 el vector de estado, es decir, el estado de ejecución de P1 dado por los registros PC_V(virtual) y RR]

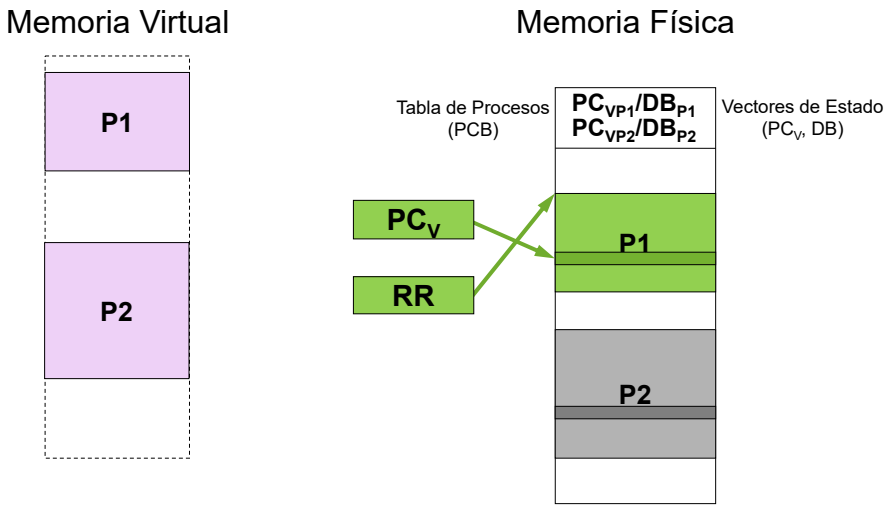
2. $\mathbf{PC_v = PC_{P2}}$; $\mathbf{RR = DB_{P2}}$ [desde el PCB de P2 se carga el vector de estado a los registros PC_V(virtual) y RR].

Una vez cargado $\mathbf{PC_v}$ desde $\mathbf{PC_{P2}}$ y **RR** desde $\mathbf{DB_{P2}}$, se obtiene la dirección física mediante el mapeo que implementó el programa expuesto anteriormente.
Si **RR** es la dirección física de inicio del código de P2, es cargada además en el $\mathbf{PC_f}$ (program counter de memoria física).

48

Espacio de segmento único por proceso

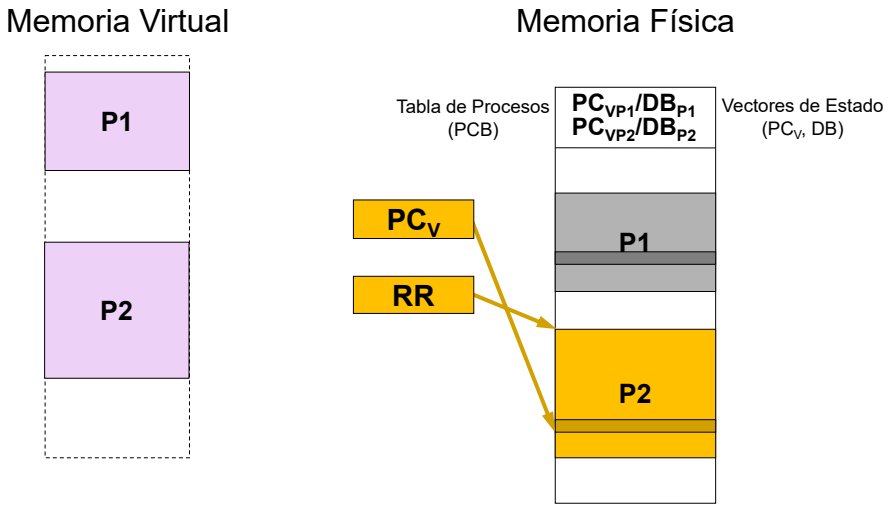
Implementación del Mapeo en varios procesos cargados.



49

Espacio de segmento único por proceso

Implementación del Mapeo en varios procesos cargados.



50

Espacio de segmento único por proceso

Ventaja y Limitación de Segmento Único por Proceso

- Ventaja: es muy fácil de implementar la ubicación y el mapeo. Para realizar el mapeo sólo se necesita especificar la dirección base (DB) del segmento que se va a ubicar en memoria física.
- Limitación: el tamaño del segmento en memoria virtual debe ser igual o menor que el espacio disponible en memoria física.

  Esta limitación, más la necesidad de compactar la memoria por la fragmentación externa, hace que sea más conveniente emplear otra técnica, que es la **técnica de paginado**.

  En este sentido, la técnica de paginado divide el espacio ocupado por los procesos en pequeños segmentos, todos de un mismo tamaño, eliminando así la fragmentación externa.

  La técnica de paginado es utilizada en los sistemas actuales, con distintas variantes.

51

Técnica de paginado

- El término **paginado** se usa para describir una particular implementación de memoria virtual y organización de memoria física.

Páginas en Memoria Física y Dirección Física

- Se divide la memoria física en un número dado de páginas que son bloques de memoria con un tamaño fijo.
  Las páginas se identifican como $\mathbf{f_0, f_1, f_2, \ldots f_{F-1}}$.
- Tamaño típico de una página: entre 1.024 y 4.096 palabras.
- Una **dirección de memoria física - df -** es un par ordenado **(f, w),** donde:

  f: es el número de página física.

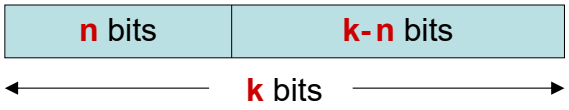
  w: es un número de palabra de la página **f**.

  (**w** se puede ver como un desplazamiento desde la dirección inicial de la página **f**).

52

Técnica de paginado

Suponiendo que una dirección **df** en memoria física tiene **k** bits, el esquema de paginado usa los primeros **n** bits para indicar el número de página **f** y los **k- n** restantes para indicar el número de palabra (desplazamiento **w**) dentro de la página.



Eso significa que está formada por f + w bits, pero para no confundirnos, cuando cuando nos refiramos a la cantidad de bits, vamos a encerrar **f** y **w** entre barras horizontales:

Entonces: **| df | = | f | + | w |**

53

Técnica de paginado

Podemos realizar los siguientes cálculos:

- Con |df| bits se pueden representar $2^{|df|}$ direcciones, por tanto, el tamaño de la memoria física es:

$$TMF = 2^{|df|}$$

- Con |f| bits se pueden representar $2^{|f|}$ páginas, es decir que la cantidad de páginas en que se puede dividir la memoria física, llamada F mayúscula, es:

$$F = 2^{|f|}$$

- Con |w| bits se pueden representar $2^{|w|}$ palabras de una página, por lo tanto, el tamaño de una página, medido en palabras o direcciones, es:

$$W = 2^{|w|}$$

54

Técnica de paginado

Páginas en Memoria Virtual y Dirección Virtual

De manera similar que para el espacio físico, un espacio virtual es dividido en un número de páginas contiguas denominadas $p_0, p_1, p_2, \ldots, p_{P-1}$,

donde: las páginas en memoria virtual son del mismo tamaño que las páginas en memoria física.

Una **dirección de memoria virtual - dv -** se considera como un par ordenado **(p, w)** donde **p** es el número de página virtual y **w** es la localización de la palabra referenciada dentro de la página **p** (desplazamiento dentro de la página).

55

Técnica de paginado

Páginas en Memoria Virtual y Dirección Virtual *(cont.)*

La correspondencia entre **dv** y **(p, w)** se obtiene de la misma manera que para la memoria física; es decir, dividiendo en dos la secuencia de bits de la dirección virtual:

- la 1ra. secuencia de **|p|** bits indica el número de página virtual.
- la 2da. secuencia de **|w|** bits indica el número de palabra dentro de una página p (desplazamiento).

Entonces, la cantidad de bits de una dirección virtual es:

$$|dv| = |p| + |w|$$

y la cantidad de páginas virtuales del proceso será:

$$P = 2^{|p|}$$

56

Técnica de paginado

- Cuando hablamos de palabras, no es otra cosa que una instrucción, y que ocupa una dirección. Por lo tanto estos tres términos van a ser equivalentes. El tamaño de la palabra dependerá de la arquitectura del hardware; hoy en día es de 64 bits.
- El número **w** es el mismo tanto en la página física como en la virtual, debido a que tienen el mismo tamaño. Esa palabra o instrucción es la que buscamos cuando hacemos el mapeo. Es decir, en alguna instrucción que ejecuta el proceso se hace referencia a otra instrucción o dato, entonces el mapeo consiste en encontrar en qué página física está.
- El valor **P** generalmente será mayor que **F**, ya que el proceso puede ser más grande que la memoria principal; veremos más adelante que esa es la principal ventaja del concepto de memoria virtual.

57

Implementación del mapeo en paginado

El objetivo de la función **map( )** es traducir una dirección virtual **(p, w)** en la correspondiente dirección física **(f, w).**

$$(p, w) \xrightarrow{map( )} (f, w)$$

Obsérvese que, en realidad, solo se debe encontrar **f** (a partir de **p**), puesto que **w** es el mismo en la dirección virtual y en la dirección física. (Ver gráfico. sigte.)

Una vez encontrada **f**, la dirección física **df** se obtiene como:

$$df = f \times 2^{k-n} + w$$

Obsérvese que: 2^{k-n} es el tamaño de una página (k-n = **|w|**)
y $f \times 2^{k-n}$ es la dirección inicial de la página **f**

58

Implementación del mapeo en paginado

|df| = 5 bits

|f| = 2 bits

|w| = 3 bits

T_{MF} = $2^{|df|}$ = 32 direcciones

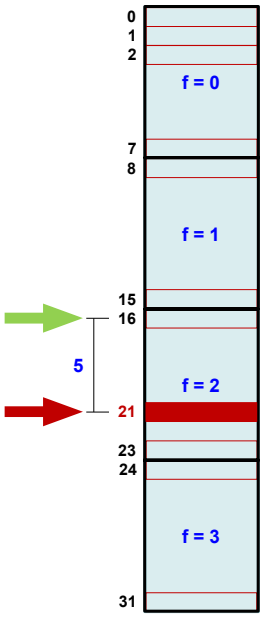
F = $2^{|f|}$ = 4 páginas

W = $2^{|w|}$ = 8 direcciones

df = (f, w) = (2, 5)

df = f x $2^{|w|}$ + w **=** 2 x 2^3 + 5 **=** 2 x 8 + 5

= 16 + 5 **= 21**



59

Implementación del mapeo en paginado

¿Cómo encontramos el valor de **f** ?

Con una **Tabla de Páginas**: arreglo que contiene las referencias entre páginas virtuales y físicas.

Esta tabla permite encontrar en qué página física se encuentra una página virtual.

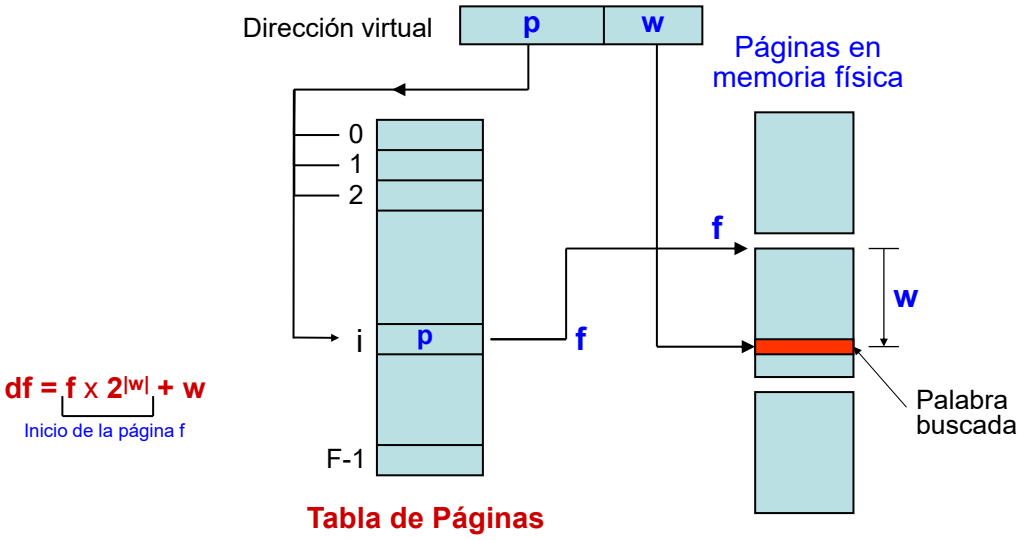
En pocas palabras, nos permite encontrar el valor de f a partir del valor de p.

La tabla tiene F entradas, igual a la cantidad de páginas físicas en memoria principal.

60

Implementación del mapeo en paginado

Implementación de la Tabla de Páginas



61

Paginado de memoria: ventajas y desventajas

Ventajas

1. La estrategia de emplazamiento o ubicación es simple.
   Esto se debe a que la unidad mínima de asignación de memoria es una página: si un proceso ocupa k páginas de memoria virtual; si hay disponibilidad en memoria física se las puede traer desde disco, no importando dónde van a ubicarse en memoria física
2. Permite la ejecución de programas de mayor tamaño que la memoria física, en forma totalmente transparente al usuario.
   Esto se implementa utilizando un esquema de asignación dinámica de almacenamiento: cuando se invoca una dirección virtual de una página que no está en memoria principal, esta página debe ser cargada desde disco.

   Este modo de operar se denomina "por demanda de páginas".
   Se necesita un mecanismo que señalice cuando una página no se encuentra en memoria principal: cuando el MMU detecta esto, se dice que hay fallo de página.

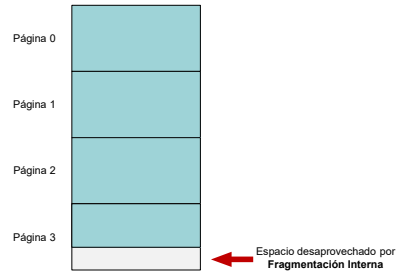
62

Paginado de memoria: ventajas y desventajas

Desventaja

Fragmentación interna: En general ningún programa ocupa con su código y sus datos el espacio disponible en la última página en forma completa. Este espacio no puede ser ocupado por ningún otro proceso ya que una página pertenece sólo a un único proceso.

Espacio promedio desperdiciado: 50% del tamaño de una página por cada proceso.



63

Espacio de múltiples segmentos por proceso

Asignación de Múltiples Segmentos a un Proceso

- Un proceso está, normalmente, compuesto por varios módulos (código, procedimientos, tablas de datos, librerías, …)
- Cuando se asigna un segmento a cada módulo, entonces cada proceso tiene asignados varios segmentos.
- Cada segmento es manejado por el SO como una unidad lógica independiente.

Ventajas:

- El SO puede acceder a estas unidades lógicas de forma más simple.
- La implementación del compartimiento y protección de acceso de estos módulos por los procesos en el sistema es natural.

64

Espacio de múltiples segmentos por proceso

La asignación de múltiples segmentos por proceso puede implementarse mediante alguna de las dos formas siguientes:

1. **Se asignan varios segmentos indivisibles a cada proceso.**

   El segmento es la mínima unidad de asignación de memoria.

   En la ejecución de un proceso, se van cargando en memoria principal los segmentos de tamaño variable a medida que se necesitan. Esta acción se denomina asignación dinámica de memoria.

2. **Se asignan segmentos a cada proceso y, a su vez, cada segmento es dividido en páginas.**

   La página es la mínima unidad de asignación de memoria

   Este esquema de asignación se denomina: paginado de segmentos.

65

Asignación de segmentos indivisibles

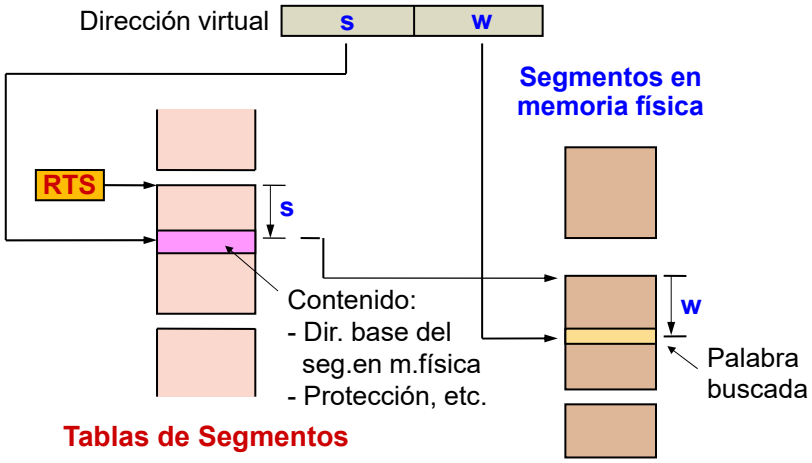
- Cada proceso consta de varios segmentos indivisibles.
- Aquí la dirección virtual que se invoca es de la forma **(s, w)** donde **s** es el número de segmento en memoria virtual y **w** la ubicación de la palabra en el segmento **s.**
- Para realizar el mapeo se utiliza una **Tabla de Segmentos** que estará almacenada en memoria principal.

  Debe existir una tabla por cada proceso activo en el sistema.
- Cada elemento o entrada de esta tabla contiene la dirección base de un segmento (en memoria física) que pertenece al proceso.
- El hardware del computador debe proveer un **Registro de Tabla de Segmentos (RTS)** que apunta al comienzo de la **Tabla de Segmentos** del proceso que se encuentra ejecutando en ese momento en la CPU.

66

Asignación de segmentos indivisibles

Tabla de Segmentos de longitud S en Memoria Principal



67

Asignación de segmentos indivisibles

Ventaja:

- Divide al programa en componentes que preservan su estructura lógica. De esta manera se simplifica el linkeado y el compartimiento y protección de procedimientos y datos.

Desventajas:

- Alta complejidad de las rutinas de ubicación y reubicación de segmentos debido a que estos tienen diferentes tamaños.
- Fragmentación externa. Se requiere compactar la memoria cada tanto.

68

Segmentación con paginado

- Significa que a cada proceso se asignan varios segmentos y, a su vez, cada segmento consta de varias páginas.
- De esta forma se combinan las ventajas de los dos esquemas vistos (segmentación simple y paginado simple).
- Para especificar una dirección virtual en este esquema, se debe agregar un nivel más de indirección que en paginado simple:

  Una **dirección virtual** consta, entonces, de una terna de la forma **(s, p, w)**, donde:

  s: número de segmento.
  p: número de página dentro del segmento **s**.
  w: desplazamiento dentro de la página **p**.

69

Segmentación con paginado

Implementación del Mapeo

La función de mapeo se puede implementar usando:

- una **Tabla de Segmentos** por proceso,
- una **Tabla de Páginas** por segmento, y
- un registro **RTS** que apunta a la Tabla de Segmentos del proceso que en ese momento está ejecutando en la CPU.

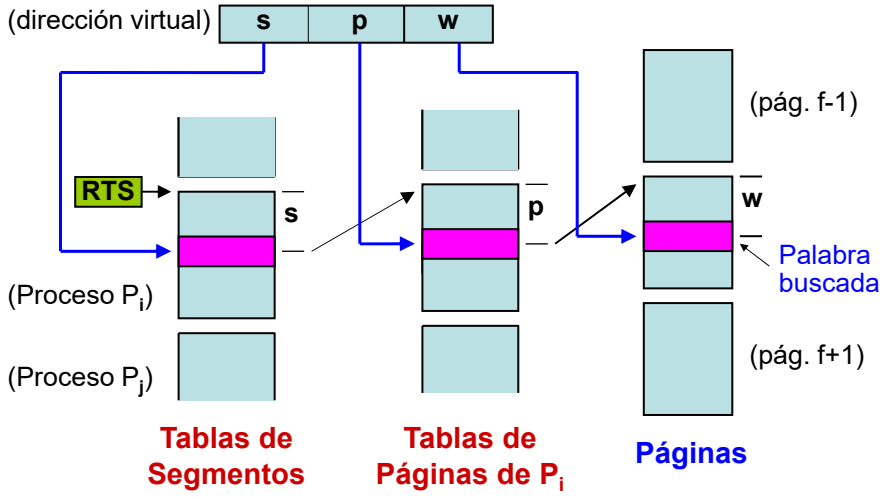
En este esquema de mapeo propuesto:

- Con el contenido de cada entrada de la Tabla de Segmentos se puede calcular la dirección de inicio de una Tabla de Páginas
- Con el contenido de cada entrada de la Tabla de Páginas se puede calcular la dirección inicio de una página en memoria física.

70

Segmentación con paginado

Implementación del Mapeo



71

Segmentación con paginado

Ventaja

Suma las ventajas que tienen por separado la segmentación y el paginado.

Desventajas

1. Se requiere un espacio considerable de memoria principal para albergar las tablas de segmentos y, especialmente, de páginas.
2. La administración de este esquema tan complejo de memoria virtual puede llevar a una disminución de la performance global del sistema.
3. Se necesitan hacer 3 referencias para acceder a una palabra en memoria física: una a la T. de Segmentos, otra a la T. de Páginas, y otra a la palabra misma.

72

Esquemas de asignación de memoria primaria

Consiste en:

Ante el pedido del SO de espacio en memoria primaria para albergar un bloque de datos que se traerá desde disco, se debe buscar un espacio igual o mayor al solicitado.

Pueden ocurrir 2 cosas:

- Que exista un espacio libre adecuado en memoria principal. En ese caso dicho espacio es asignado al bloque de datos.
- Que no exista un espacio libre adecuado en mem. principal. En ese caso, primero se debe liberar un espacio y luego asignarlo al bloque de datos (segmentos o páginas) que se traerán desde disco.

Se van a estudiar a continuación 2 esquemas de asignación:

- **Asignación de memoria en sistemas no paginados (segmentos)**
- **Asignación dinámica de memoria en sistemas paginados**

73

Asignación de memoria en sistemas no paginados

Estas operaciones pueden realizarse invocando los siguientes comandos o primitivas:

- ***Request (****Mem_Primaria, Tamaño_Pedido****)***
- ***Release (****Mem_Primaria, Tamaño_Liberado, Dir_Base****)***

Donde:

Tamaño_Xxx: número de unidades contiguas solicitadas (*Request*) y liberadas (*Release*).

Dir_Base: dirección donde comienza el bloque de celdas liberadas.

Resumen de la Operación: se solicita (*request*) un determinado espacio de memoria primaria; a partir de lo cual, se libera (*release*) una cantidad igual o mayor de memoria para asignar

74

Asignación de memoria en sistemas no paginados

Ejemplo

El módulo de administración de memoria de un SO necesita traer desde disco un bloque de datos de tamaño ***k*** de un proceso.

El comando correspondiente será:

Request *(Mem_Primaria, k)*

A partir de este pedido, pueden suceder 2 cosas:

a) **Existe** un hueco H de tamaño *h*, tal que *h* >= *k*.
En tal caso se asignan *k* unidades de H al proceso solicitante.

b) **No existe** un hueco de tamaño *h* tal que *h* >= *k* .
Aquí se pueden adoptar varios criterios:

- Bloquear el proceso solicitante hasta que se disponga de un hueco del tamaño necesario (liberación voluntaria), o
- 1) desalojar uno o más bloques de memoria, o 2) reubicar esos bloques en la misma memoria, ó 3) compactar la mem.

75

Asignación de memoria en sistemas no paginados

Algoritmos de Ubicación de Bloques de Tamaño Variable

Encuentran cuál de todos los bloques disponibles, que cumplen con la condición *h* >= *k*, se asigna al proceso que invoca el comando *Request ()*. Hay dos algoritmos: First-Fit y Best-Fit.

Antes de analizar los algoritmos mencionados, se debe asumir que la memoria está dividida en 2 conjuntos de bloques de tamaños variables:

a) Un conjunto de huecos: $H = \{H_i \mid i = 1, \ldots, n\}$, y

b) Un conjunto de bloques llenos: $F = \{F_i \mid i = 1, \ldots, m\}$.

- **Algoritmo "First-Fit":** ante la solicitud de un bloque de tamaño ***k***, recorre la memoria y asigna el primer hueco encontrado que cumple con la condición $h_i >= k$.
- **Algoritmo "Best-Fit":** entrega un hueco H_i tal que su tamaño es el mínimo posible que cumple con la condición $h_i >= k$.

76

Asignación de memoria en sistemas no paginados

Ventajas y Desventajas

Los métodos usados por los algoritmos vistos presentan las siguientes ventajas y desventajas.

	First-Fit	Best-Fit
Ventaja	Veloz	Eficiencia en la utilización de la memoria
Desventaja	Posible desperdicio de gran cantidad de memoria.	Puede resultar lento.

77

Modos de asignación de memoria principal

Asignación a Sistemas Paginados y No Paginados

¿Cuánto de memoria principal se asigna a un proceso cuando inicia su ejecución?

Existen 2 formas de asignar memoria a procesos en disco:

- **Asignación estática:** se asigna memoria física a todo el proceso al inicio de la ejecución del mismo.
- **Asignación dinámica:** se asigna la memoria física a una porción del proceso al inicio de la ejecución del mismo.
  En este caso se tiene que:
  - El resto del proceso permanece en memoria secundaria.
  - Durante la ejecución, distintas partes del proceso serán trasladadas entre memoria virtual y memoria física.

78

Modos de asignación de memoria principal

Asignación a Sistemas Paginados y No Paginados *(cont.)*

Ventajas y desventajas de las asignacˢ estática y dinámica

Asignación estática

- Es simple (ventaja)
- El espacio que utiliza el proceso debe ser menor que la memoria física disponible (desventaja).
- El grado de multiprogramación es bajo (desventaja).

Asignación dinámica

- Mejor uso del espacio de memoria primaria (ventaja)
- Mayor número de procesos activos en el sistema listos para hacer uso del procesador (mayor grado de multiprogramación) (ventaja).
- Mayor complejidad del algoritmo de asignación (desventaja).

79

Asignación dinámica de memoria en el tiempo

En sistemas Paginados y No Paginados

- El espacio que ocupan los procesos en el sistema, normalmente, es mucho mayor que el tamaño de la memoria física.
- Por ese motivo, cuando el sistema tiene muchos procesos activos es probable que la memoria principal esté casi completamente ocupada; por lo tanto, las solicitudes de espacios para páginas que están en disco no podrán ser satisfechas inmediatamente.
- En tal caso, un proceso que necesite estar en memoria principal, normalmente, entra en bloqueo hasta tanto el SO desocupe un espacio.

A continuación se verá: **Asignación Dinámica de Memoria en Sistemas Paginados**

80

Asignación dinámica en sistemas paginados

Esquema de Reemplazo de Páginas

- Es el método más complejo de implementar, pero también,
- Ofrece una mejor performance global al sistema.

Por esta segunda virtud, este método es utilizado por todos los SO´s modernos.

Situación típica en un sistema multiprogramado

- En cualquier momento, las páginas pertenecientes a distintos procesos están dispersas en distintos lugares de la memoria principal.
- Cuando ocurre un fallo de página - page fault - y no existe memoria física libre, el SO debe seleccionar una de las páginas residentes para su reemplazo.

81

Asignación dinámica en sistemas paginados

Esquema de Reemplazo de Páginas *(cont.)*

Se puede emplear alguna de las 2 estrategias de reemplazo:

1. **Estrategia local:** la página a ser reemplazada se selecciona del conjunto de páginas residentes del propio proceso que produce la falla.
2. **Estrategia global:** la página a reemplazarse puede pertenecer a cualquiera de los procesos en memoria principal.

82

Asignación dinámica en sistemas paginados

Estrategias

Cuando se debe traer desde disco a memoria física una página de un proceso, existen 2 tipos de estrategias:

- **Estrategia de emplazamiento o de ubicación**
  Determina **dónde** se debe cargar la página.
- **Estrategia de reemplazo**
  Determina **qué** página se debe eliminar de la memoria física cuando no hay espacio suficiente.

83

Algoritmos de reemplazo de páginas

Algoritmo de reemplazo más eficiente: es el que produce el menor número de fallos de página durante la ejecución de un programa. (Ver tablas a partir de pág. 273 del libro de Saade).

Se estudiarán los algoritmos considerando ejemplos, en los que:

1. Se supone que la memoria física sólo puede alojar 4 páginas que ya han sido cargadas cuando comienza la ejecución del programa.
2. **Z** representa la secuencia de las páginas referenciadas durante la ejecución del programa.
3. **a, b, …, k** son páginas virtuales referenciadas, donde **k** es la página de datos que se referencia.
4. Fallo: se marca con un asterisco **(*)** cuando ocurre un fallo de página.
5. **p** y **q** son la página cargada en memoria y la página reemplazada, respectivamente.

84

Algoritmos de reemplazo de páginas

Algoritmo de Reemplazo Óptimo

- Es el más eficiente, pues está basado en decidir el reemplazo de aquella página en memoria principal que no será accedida por mayor tiempo en el futuro.
- Es teórico. Lamentablemente, es de imposible implementación práctica, puesto que no se puede saber cuál será la página que no será accedida por mayor tiempo en el futuro.
- Sin embargo, sirve como referencia para evaluar algoritmos prácticos de reemplazo de páginas.

85

Algoritmos de reemplazo de páginas

Algoritmo de Reemplazo Óptimo

Tiempo	0	1	2	3	4	5	6	7	8	9	10
Z		c	a	d	b	e	b	a	b	c	d
Página 0	a										
Página 1	b										
Página 2	c										
Página 3	d										
Fallo											
p											
q											

86

Algoritmos de reemplazo de páginas

Algoritmo de Reemplazo Aleatorio

- Como su nombre lo indica, se remueve cualquier página de memoria física.
- Es el esquema más simple de reemplazo. No obstante, no es eficiente porque no tiene en cuenta el

  Principio de cercanía o localidad de la referencia, que dice:

  cuando un programa referencia un lugar de la memoria, existe una alta probabilidad que referencie a la misma posición de memoria o a localidades cercanas en el futuro inmediato.

  Este principio tiene validez intuitiva si se considera que la ejecución de un programa sigue los patrones siguientes:

87

Algoritmos de reemplazo de páginas

Justificación del Principio de Localidad de la Referencia

- Excepto por las instrucciones de salto (del tipo jump que son entre un 10 y 20% del total), la ejecución de los programas es de tipo secuencial (sigue la secuencia de la ubicación de las instrucciones en memoria principal).
- La mayoría de las construcciones iterativas consisten de un número relativamente pequeño de instrucciones consecutivas repetidas un gran número de veces.
- La computación de estructuras de datos como arreglos, o secuencias de registros, constituyen la gran parte del código de un programa. Una porción significativa de esta computación requiere un procesamiento secuencial. De esta forma, instrucciones consecutivas tienden a referenciar elementos contiguos de las estructuras de datos.

88

Algoritmos de reemplazo de páginas

Algoritmo FIFO - First In, First Out -

Un puntero apunta a la página más antiguamente referenciada; en caso de fallo, se reemplaza la página apuntada y avanza el puntero a la siguiente página en orden cronológico.

Ventaja. Simplicidad de implementación: solamente se necesita una lista de ***m*** elementos y un puntero que apunta a la página más antigua en memoria física para su reemplazo. El puntero se incrementa cada vez que ocurre un fallo de página.

Desventaja. Supone que las páginas que residen en memoria por mayor tiempo son las que tienen menor probabilidad de que se referencien en el futuro. Esto no responde exactamente a lo establecido por el principio de localidad de referencia.

Puede ocurrir que algunas páginas que fueron cargadas en el pasado pueden volver a ser referenciadas con frecuencia, con lo que el "principio" puede ser violado repetidamente.

89

Algoritmos de reemplazo de páginas

Algoritmo FIFO (First In, First Out)

Tiempo	0	1	2	3	4	5	6	7	8	9	10
Z		c	a	d	b	e	b	a	b	c	d
Página 0	a										
Página 1	b										
Página 2	c										
Página 3	d										
Fallo											
p (fifo)											
q (fifo)											

90

Algoritmos de reemplazo de páginas

Algoritmo LRU - Last Recently Used -

- Este algoritmo ha sido diseñado específicamente para cumplir con el principio de localidad de referencia.
- En este sentido, reemplaza la página que no ha sido referenciada por mayor tiempo en el pasado.
- La diferencia c/FIFO es que éste elimina la página que reside en memoria por mayor tiempo.
- **LRU** debe mantener una lista en la que las páginas se ordenan de acuerdo a la antigüedad de la referencia.

  Cuando ocurre un fallo, la página que se encuentra en la base de la lista (página referenciada en el pasado más lejano y todavía presente en memoria física) es reemplazada, luego la nueva página es ubicada al principio de la lista. (Ver ejemplo en libro)

91

Algoritmos de reemplazo de páginas

Algoritmo LRU (Last Recently Used)

Tiempo	0	1	2	3	4	5	6	7	8	9	10
Z		c	a	d	b	e	b	a	b	c	d
Página 0	a										
Página 1	b										
Página 2	c										
Página 3	d										
Fallo											
p (lru)											
q (lru)											
Stack											

92

Algoritmos de reemplazo de páginas

Algoritmo LRU *(cont.)*

Se observa que se debe mantener un orden cronológico de las referencias a las páginas, por lo que la lista debe ser reordenada cada vez que se realiza una referencia.

Desventaja: La implementación de una lista por software es impráctica, por el tiempo que lleva reordenar la lista en cada referencia. No obstante, existen varios métodos alternativos por hardware para implementar este algoritmo.

93

Algoritmos de reemplazo de páginas

Algoritmo de Reemplazo de Reloj

- Este algoritmo trata de sustituir al algoritmo LRU cuya implementación por software es compleja e ineficiente.
- En el algoritmo de reemplazo de reloj se mantiene una lista circular de todas las páginas residentes y un puntero similar al utilizado en el algoritmo FIFO.
- Se adiciona un bit *u*, denominado bit de *uso*, asociado a cada página en memoria física y cuyo valor indica su antigüedad.
- Cuando se produce un fallo:
  - Si el puntero se encuentra posicionado en una página cuyo bit *u* es 0, entonces se reemplaza dicha página y se avanza el puntero.
  - En caso contrario (*u* = 1), el bit *u* se pone en 0, se avanza el puntero a la próxima página en la lista y se repite el mismo paso.

94

Algoritmos de reemplazo de páginas

Algoritmo de Reemplazo de Reloj

Tiempo	0	1	2	3	4	5	6	7	8	9	10
Z		c	a	d	b	e	b	a	b	c	d
Página 0	a/1										
Página 1	b/1										
Página 2	c/1										
Página 3	d/1										
Fallo											
p											
q											

95

Algoritmos de reemplazo de páginas

Algoritmo de Reemplazo de Reloj (cont.)

Este algoritmo tiene un comportamiento aproximado al LRU, y su ventaja respecto de éste, es no se utiliza la lista donde se van ordenando las páginas, con lo que se optimiza el espacio utilizado y el tiempo que lleva el ordenamiento.

96

Estrategias de control de carga

Prepaginación

- **Un extremo:** en asignación estática, antes de ejecutar un programa son cargadas en memoria física todas las páginas del mismo.
  Consecuencia: pocos procesos cargados
- **Otro extremo:** Al inicio no se carga ninguna página. El programa comienza a ejecutar y las páginas se van cargando a medida que ocurren los fallos de página.
  Consecuencia: hasta que se estabiliza la ejecución del programa (varios fallos de página y varias páginas cargadas), hay mucha pérdida de tiempo en tareas de I/O de páginas.
- Conclusión: es necesario, entonces, buscar una solución intermedia entre los extremos; es decir, cargar en memoria física un número adecuado de páginas del programa antes de que comience a ejecutar. Esta acción se llama **prepaginación.**

97

Estrategias de control de carga

Prepaginación *(cont.)*

En la prepaginación se tienen las siguientes cuestiones a resolver:

- Proceso nuevo: es difícil tomar una decisión correcta de cuántas y qué páginas cargar inicialmente. (Ver siguiente diapositiva)
- Proceso viejo: una buena elección es cargar las páginas que tenía en memoria en el momento en que el proceso fue suspendido o bloqueado.

98

Estrategias de control de carga

Trashing

- Uno de los principales problemas a resolver en un esquema de memoria virtual es mantener un balance apropiado entre: **grado de multiprogramación** y **cantidad de prepaginado**

  Grado de multiprogramación: intervienen algoritmos de control de carga (long-term-schedulers) que determinan el número y tipos de procesos que debe ser admitidos para su ejecución.

  Cantidad de prepaginado:

1) Alto número de procesos activos→baja cantidad de prepaginado →muchos de fallos de página. Como consecuencia:
   El sistema tiende a destinar todo su tiempo sólo a mover páginas entre memoria virtual y memoria física, y viceversa.
   Este comportamiento se denomina **trashing** ← debe evitarse.
2) Alta cantidad de prepaginado→bajo grado de multiprogramación.

99

Evaluación de los sistemas de paginado

Método general para evaluar los sistemas de paginado:

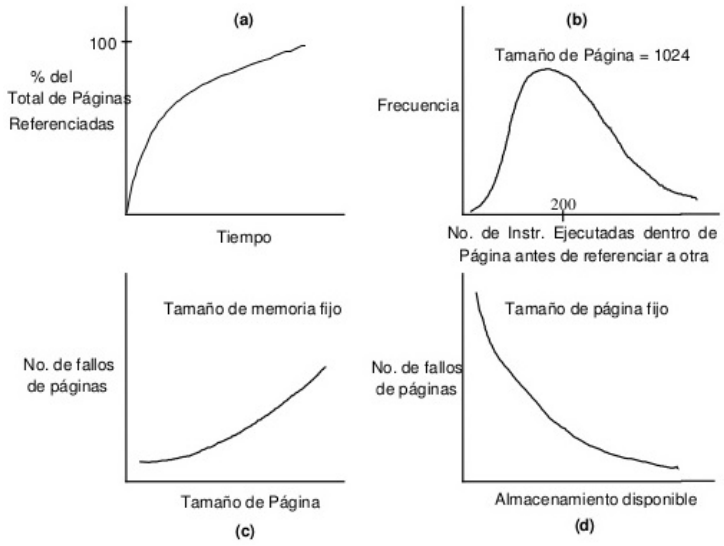
Se ejecuta una muestra de programas representativos para distintos mecanismos de paginado y variando ciertos parámetros tales como: el tamaño de la página o la cantidad de almacenamiento de páginas disponible en memoria física.

Conclusiones obtenidas:

1. La mayoría de los procesos requieren un alto porcentaje de sus páginas en un breve período, luego de su activación.
P/ejemplo: alrededor del 50% del total de las páginas de un proceso son referenciadas durante el primer quantum de tiempo de CPU, luego de la activación del proceso. [Parte (a) de la Figura 6-13] (Las figuras a que se hace referencia son del libro de Saade)

100

Evaluación de los sistemas de paginado



101

Evaluación de los sistemas de paginado

2. Un número relativamente pequeño de instrucciones son ejecutadas antes de que se referencie una instrucción en otra página. En la mayoría de las simulaciones, para páginas de 1024 palabras, fueron ejecutadas menos de 200 instrucciones antes de que otra página fuese referenciada. [Fig.(b)].
3. En el rango entre 64 y 1024 palabras por página y con espacio fijo de memoria, el número de fallos de páginas aumenta a medida que se incrementa el tamaño de la página. [Fig.(c)].
4. Cuando la cantidad de páginas en memoria (por proceso) decrece, para un tamaño de página fijo, existe un punto en que el número de fallos de página aumenta exponencialmente (trashing). [Fig.(d)].

102

Evaluación de los sistemas de paginado

Discusión sobre el Tamaño de Página más Adecuado

Tamaño Pequeño

Ventaja: menor desperdicio de memoria (fragmentación interna).

Desventaja: gran tamaño de las tablas de páginas y mucho movimiento de páginas.

Tamaño Grande

Ventajas:

- Tamaño reducido de las tablas de páginas.
- Mayor eficiencia en la transferencia de datos desde disco a memoria principal:

Eficiencia = cantidad de datos transferidos en la operación de E/S

Desventaja: mayor desperdicio de memoria (fragmentación interna)

103

Fin del Módulo IV

104