Administración de Memoria Principal

Explicación de práctica 5

Introducción a los Sistemas Operativos

Facultad de Informática
Universidad Nacional de La Plata

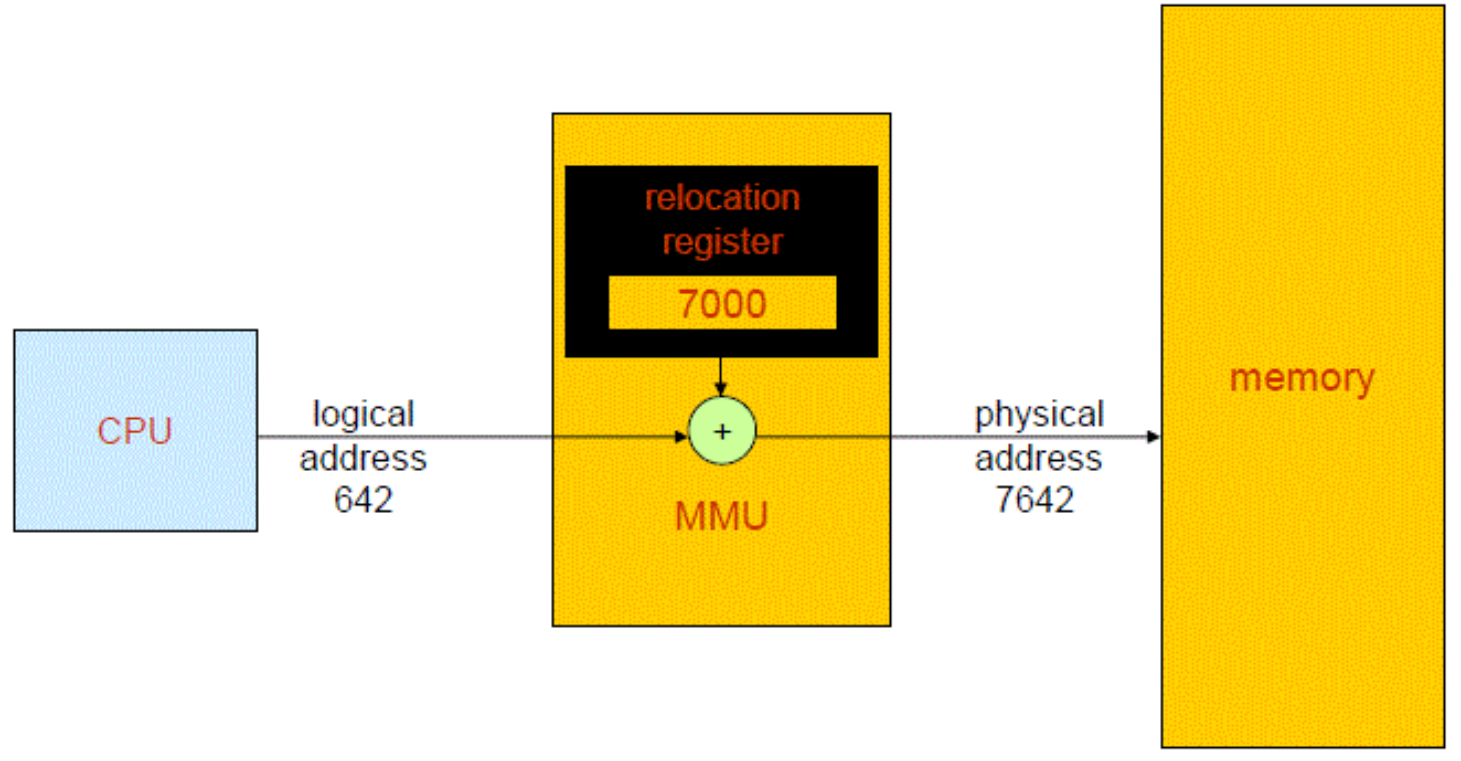
2024

- La organización y administración de la *memoria RAM* es uno de los factores más importantes en el diseño de un SO
- Los programas y datos deben residir en ella para:
  - Poder ejecutar
  - Referenciarlos directamente

- La parte del SO que administra esta memoria se llama "*administrador de la memoria*":
  - Lleva un registro de las partes de la memoria que se están utilizando y de aquellas que no
  - Asigna espacio en memoria a los procesos cuando estos la necesitan
  - Libera espacio de momoria asignada a procesos que han terminado
- Se espera que el SO haga uso eficiente de esta memoria con el fin de alojar el mayor número de procesos $\rightarrow$ repercute en la *multiprogramación*

- **Dirección Lógica**:
  - Es una dirección que enmascara o abstrae una dirección física
  - Referencia a una localidad en memoria
  - Se la debe traducir a una dirección física
- **Dirección Física**:
  - Es la dirección real. Es con la que se accede efectivamente a memoria
  - Representa la dirección absoluta en memoria principal
- La CPU trabaja con direcciones lógicas. Para acceder a la memoria se deben transformar en direcciones físicas
- El mapeo entre direcciones virtuales y físicas se realiza mediante *hardware* $\rightarrow$ **MMU** (Memory Management Unit)

Traduccion **MMU**



- **Particiones Fijas**:
  - La memoria se divide en particiones o regiones de tamaño fijo $\rightarrow$ tamaños iguales o diferentes
  - Alojan un único proceso
  - Cada proceso se coloca en alguna partición de acuerdo a algún criterio:
    - **First Fit**
    - **Best Fit**
    - **Worst Fit**
    - **Next Fit**
- **Particiones Dinámicas**:
  - Las particiones varían en tamaño y número
  - Alojan un proceso cada una
  - Cada partición se genera en forma dinámica del tamaño justo que necesita el proceso

¿Qué problemas se generan en cada caso?

- **Particiones Fijas**:
  - La memoria se divide en particiones o regiones de tamaño fijo $\rightarrow$ tamaños iguales o diferentes
  - Alojan un único proceso
  - Cada proceso se coloca en alguna partición de acuerdo a algún criterio:
    - **First Fit**
    - **Best Fit**
    - **Worst Fit**
    - **Next Fit**
- **Particiones Dinámicas**:
  - Las particiones varían en tamaño y número
  - Alojan un proceso cada una
  - Cada partición se genera en forma dinámica del tamaño justo que necesita el proceso

¿Qué problemas se generan en cada caso?

Fragmentación

- La fragmentación se produce cuando una localidad de memoria no puede ser utilizada por no encontrarse en forma contigua
- **Fragmentación Interna**:
  - Se produce en el esquema de particiones fijas, por ejemplo
  - Es interna a la localidad asignada
  - Es la porción de la localidad que queda sin utilizar
- **Fragmentación Externa**:
  - Se produce en el esquema de particiones dinámicas, por ejemplo
  - son huecos que van quedando en la memoria a medida que los procesos finalizan
  - Al no encontrarse en forma contigua puede darse el caso de que tengamos memoria libre para alocar un proceso, pero que no la podamos utilizar
  - Solución $\rightarrow$ *compactación* $\rightarrow$ muy costosa

Paginación

- La memoria se divide en porciones de igual tamaño llamadas *marco*
- El espacio de direcciones de los procesos se divide en porciones de igual tamaño denominadas **páginas**
- Tamaño página = tamaño *marco* = 512 bytes (generalmente)
- El SO mantiene una tabla de páginas para cada proceso, la cual contiene el *marco* donde se encuentra cada página
- La paginación bajo demanda es una técnica eficiente de manejar esta estrategia $\rightarrow$ *Thrashing*

Programa

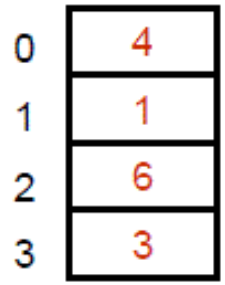
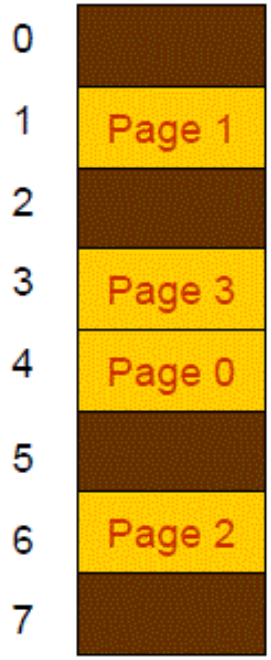


Tabla de Páginas



Memoria

- Un proceso en ejecución hace referencia a una dirección virtual $\rightarrow v = (p,d)$
- El SO busca la página p en la *tabla de páginas* del proceso y determina en que *marco* se encuentra
- La dirección de almacenamiento real se forma por la concatenación de la resolución de p (dirección inicio del marco que aloca la página) y d, donde p es el número de página y d es el desplazamiento

- Memoria administrada por sistema de paginacion
- Tamaño de página $\rightarrow$ 515 Bytes
- Cada dirección de memoria referencia a 1 Byte
- Los *marco* en memoria principal se encuentran desde la dirección física 0
- Tenemos un proceso con un tamaño de 2000 Bytes y con la siguiente tabla de páginas

Paginación - Ejemplo (cont.)

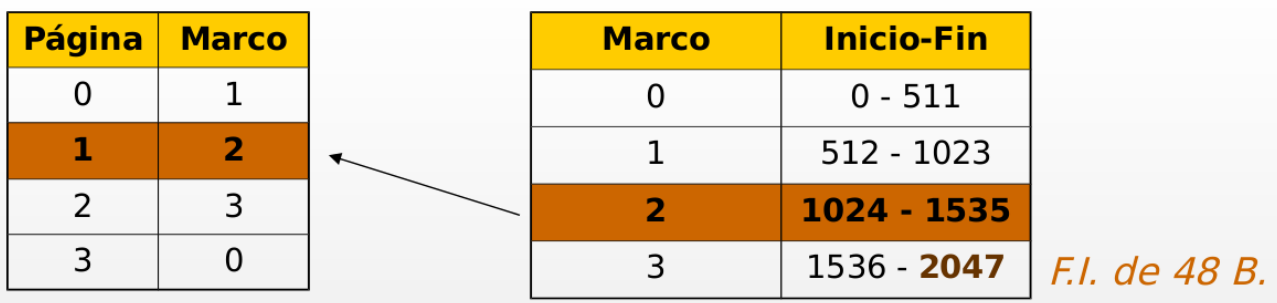
Página	Marco
0	1
1	**2**
2	3
3	0

Marco	Inicio-Fin
0	0 - 511
1	512 - 1023
2	**1024 - 1535**
3	1536 - **2047**

F.I. de 48 B.

- Si tenemos una dirección **virtual**, por ejemplo 580:
  - Para averiguar el número de página hacemos 580 div 512 = 1. Luego esta dirección corresponde a la página 1 que se encuentra en el *marco* 2
  - Para averiguar el desplazamiento hacemos 580 mod 512 = 68
  - La dirección física es 1024 + 68 = 1092

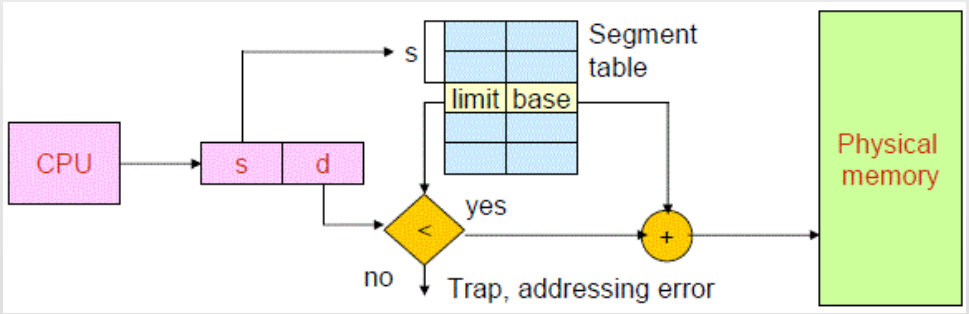
Paginación - Ejemplo (cont.)



- Si tenemos una dirección **física**, por ejemplo 1092:
  - Para averiguar el número de *marco* hacemos 1092 div 512 = 2. En el *marco* número 2 tenemos la página número 1
  - Para averiguar el desplazamiento hacemos 1092 mod 512 = 68
  - La dirección virtual es 512 + 68 = 580

Segmentación

- La segmentación básicamente la podemos ver como una mejora de la paginación (*no hay F.I., sino externa*)
- Ahora la tabla de segmentos, además de tener la dirección de inicio del mismo, tiene la longitud o límite
- Las direcciones lógicas constan de dos partes $\rightarrow$ un número de segmento s y un desplazamiento d dentro del segmento ($0 \langle d \langle$ *límite*)



Memoria Virtual *con Paginación*

- La técnica de paginación intenta alocar la mayor cantidad de páginas necesarias posibles
- Cada vez que hay que alocar una página en un *marco*, se produce un **fallo de página** (page fault) $\rightarrow$ *hard page fault*
- ¿Qué sucede si es necesario alocar una página y ya no hay espacio disponible?
- Se debe seleccionar una página víctima, para lo cual existen diversos algoritmos
- ¿Cuál es el mejor algoritmo?:
  - El que seleccione como página víctima aquella que no vaya a ser referenciada en un futuro próximo
- La mayoria de los algoritmos predicen el comportamiento futuro mirando el comportamiento pasado

Memoria Virtual *con Paginación*

- La técnica de paginación intenta alocar la mayor cantidad de páginas necesarias posibles
- Cada vez que hay que alocar una página en un *marco*, se produce un **fallo de página** (page fault) $\rightarrow$ *hard page fault*
- ¿Qué sucede si es necesario alocar una página y ya no hay espacio disponible?
- Se debe seleccionar una página víctima, para lo cual existen diversos algoritmos
- ¿Cuál es el mejor algoritmo?:
  - El que seleccione como página víctima aquella que no vaya a ser referenciada en un futuro próximo
- La mayoria de los algoritmos predicen el comportamiento futuro mirando el comportamiento pasado

Memoria Virtual *con Paginación*

- La técnica de paginación intenta alocar la mayor cantidad de páginas necesarias posibles
- Cada vez que hay que alocar una página en un *marco*, se produce un **fallo de página** (page fault) $\rightarrow$ *hard page fault*
- ¿Qué sucede si es necesario alocar una página y ya no hay espacio disponible?
- Se debe seleccionar una página víctima, para lo cual existen diversos algoritmos
- ¿Cuál es el mejor algoritmo?:
  - El que seleccione como página víctima aquella que no vaya a ser referenciada en un futuro próximo
- La mayoria de los algoritmos predicen el comportamiento futuro mirando el comportamiento pasado

- Selecciona la página cuyo próxima referencia se encuentra más lejana a la actual
- Imposible de implementar → no se conoce los futuros eventos

Marco/Página	**1**	**2**	**1**	**3**	**4**	**1**	**4**	**3**	**5**
F1	1	1	1	1	1	1	1	1	?
F2		2	2	2	4	4	4	4	?
F3				3	3	3	3	3	?
PF?	X	X		X	X				X

Continuación de la secuencia: 4 6 3 5 8 1

- El algoritmo *LRU* (Least Recently User) reemplaza la página que no fue referenciada por más tiempo
- Cada página debe tener información del instante de su última referencia $\rightarrow$ el overhead es mayor

Marco/Página	**1**	**2**	**1**	**3**	**4**	**1**	**4**	**3**	**5**
F1	1	1	1	1	1	1	1	1	5
F2		2	2	2	4	4	4	4	4
F3				3	3	3	3	3	3
PF?	X	X		X	X				X

- El algoritmo *FIFO* (Fist In First Out) trata a los *frames* en uso como una cola circular
- Simple de implementar
- La página más vieja en la memoria es reemplazada
- La página puede ser necesitada pronto

Marco/Página	**1**	**2**	**1**	**3**	**4**	**1**	**4**	**3**	**5**
F1	1	1	1	1	4	4	4	4	4
F2		2	2	2	2	1	1	1	1
F3				3	3	3	3	3	5
PF?	X	X		X	X	X			X

Algoritmo **FIFO con segunda chande**

- Se utiliza un *bit* adicional $\rightarrow$ bit de referencia
- Cuando la página se carga en memoria, el *bit R* se pone a 0
- Cuando la página es referenciada el *bit R* se pone en 1
- La víctima se busca en orden *FIFO*. Se selecciona la primer página cuyo *bit R* esta en 0
- Mientras se busca la víctima cada *bit R* que tiene el valor 1, se cambia a 0

- El algoritmo *FIFO* (Fist In First Out) trata a los *frames* en uso como una cola circular
- Simple de implementar
- La página más vieja en la memoria es reemplazada
- La página puede ser necesitada pronto

Marco/Página	**1**	**2**	**1**	**3**	**4**	**1**	**4**	**3**	**5**
F1	1	1	1*	1*	1	1*	1*	1*	1
F2		2	2	2	4	4	4*	4*	4
F3				3	3	3	3	3*	5
PF?	X	X		X	X				X

$* = bit\ R = 1$

- La asignación de *marco* se puede realizar de dos modos:
  - **Asignación Fija**: a cada proceso se le asigna una cantidad arbitraria de *marco*. A su vez para el reparto se puede usar:
    - **Reparto equitativo**: se asigna la misma cantidad de *marcos* a cada proceso $\rightarrow$ *m div p*
    - **Reparto proporcional**: se asignan *marco* en base a la necesidad que tiene cada proceso $\rightarrow$ V_p . *m* / V_t
  - **Asignación dinámica**: los procesos se van cargando en forma dinámica de acuerdo a la cantidad de marcos que necesiten

- Al momento de seleccionar una página víctima, podemos utilizar:
  - **Reemplazo global**: el fallo de página de un proceso puede reemplazar la página de cualquier proceso
  - **Reemplazo local**: el fallo de página de un proceso solo puede reemplazar sus propias páginas

Paginación - Descarga asincrónica de páginas

- El sistema operativo reserva uno o varios *marco* para la descarga asincrónica de páginas
- Cuando es necesario descargar una página modificada:
  - La página que provoco el fallo se coloca en un *frame* designado a la descarga asincrónica
  - El SO envía la orden de descargar asincrónicamente la página modificada mientras continua la ejecución de otro proceso
  - El *frame* de descarga asincrónica pasa a ser el que contenía a la página víctima que ya se descargó correctamente

Descarga asincrónica de páginas (ejemplo)

- Ejemplo de algoritmo *FIFO*

Marco/ Página	1	2	1ᴹ	3	4	3ᴹ	5		6	7	
F1	1	1	1ᴹ	1ᴹ	1ᴹ	1ᴹ	1ᴹ			7	7
F2		2	2	2	2	2	2	2	6	6	6
F3				3	3	3ᴹ	3ᴹ	3ᴹ	3ᴹ	3ᴹ	
F4					4	4	4	4	4	4	4
F5							5	5	5	5	5

- La técnica de paginación por demanda puede generar una degradación de rendimiento del sistema debido a que el reemplazo de páginas es costoso
- Tasa de *page faults* $0 \langle p \langle 1$:
  - Si $p = 0$, no hay *page faults*
  - Si $p = 1$, cada referencia es un *page fault*
- *Effective Access Time*: medida utilizada para medir este costo:

  - **Am** = tiempo de acceso a la memoria real
  - **Tf** = tiempo de atención de un fallo de página
  - **At** = tiempo de acceso a la tabla de páginas. Es igual al tiempo de acceso a la memoria (*Am*) si la entrada de la tabla de páginas no se encuentra en la *TLB* (cache donde residen las traducciones de direcciones realizadas)

$$\mathbf{TAE = At + (1 - p) * Am + p * (Tf + Am)}$$

Thrashing

- *Thrashing* (hiperpaginación):
  - decimos que un sistema está en *thrashing* cuando pasa más tiempo paginando que ejecutando procesos
- Si un proceso cuenta con todos los *frames* que necesita, no habría *thrashing*. Salvo excepciones como la *anomalía de* **Belady**
- Existen técnicas para evitarlo $\rightarrow$ estrategia de *Working Set*

Working Set - **Modelo de Localidad**

- Las referencias a datos y programas dentro de un proceso tienden a agruparse
- La localidad de un proceso en un momento dado se da por el conjunto de páginas que son referenciadas en ese momento
- En cortos períodos de tiempo, el proceso necesitará pocas "piezas" del proceso (una página de instrucciones y otra de datos)
- Entonces se define una ventana de trabajo o *working set* (Δ) que contiene las referencias de memoria más recientes
- **Working Set**: es el conjunto de páginas que tienen las Δ referencias a páginas más recientes

- Δ chico: no cubrirá la localidad. Toma muy pocas referencias
- Δ grande: puede tomar varias localidades. Toma referencias de la localidad y algunas más, posiblemente viejas
- Para determinar la medida del *working set* debemos tener en cuenta:
  - m = cantidad de *frames* disponibles
  - D = demanda total de *frames*
  - WSS_i = medida del *working set* del proceso$_i$
  - $\sum WSS_i = D$
  - Si $D \rangle m$ habrá **thrashing**

¿Preguntas?