Memoria Virtual

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

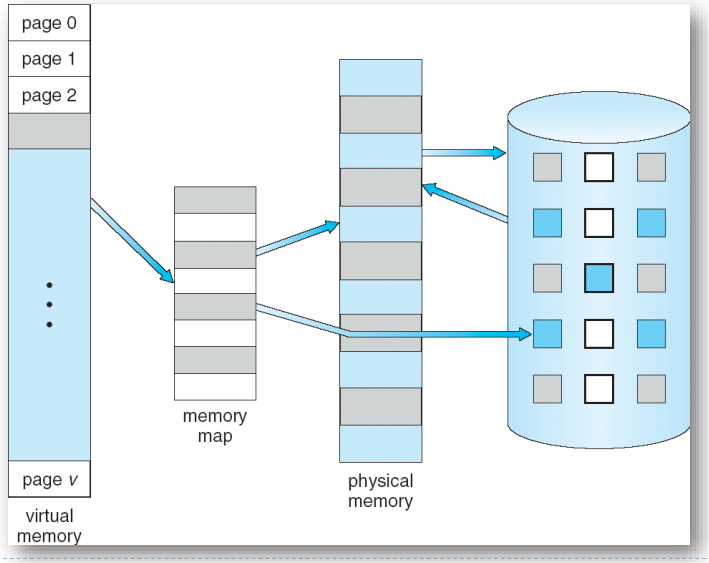
Objetivos

- Describir los beneficios del sistema de memoria virtual
- Explicar los conceptos de paginado por demanda, algoritmos de reemplazo de páginas y alocación de cuadros de páginas
- Discutir el principio del modelo de working-set

Base

- **Memoria Virtual**– separación de la memoria lógica del usuario de la memoria física.
  - Solo parte del programa necesita estar en memoria para su ejecución.
  - El espacio de direcciones lógicas puede ser más grande que el espacio de direcciones físicas.
  - Permite a varios procesos compartir el espacio de direcciones.
  - La creación de procesos sea más eficiente
- La memoria virtual puede ser implementada vía:
  - Paginado por demanda
  - Segmentación por demanda

Memoria Virtual más grande que la Memoria Física



Paginado por Demanda

- Traer una página a la memoria solo cuando es necesario.
  - Son necesarias menos E/S
  - Es necesario menos memoria
  - Respuesta más rápida
  - Más usuarios
- Cuando una página se necesita ⇒ se la referencia
  - referencia inválida ⇒ aborto
  - no está en memoria ⇒ se la trae a memoria
- Intercambiador "perezoso" – nunca intercambia en memoria hasta que la página se necesite.
  - El intercambiador (swapper) que trata con páginas es un *paginador* (pager)

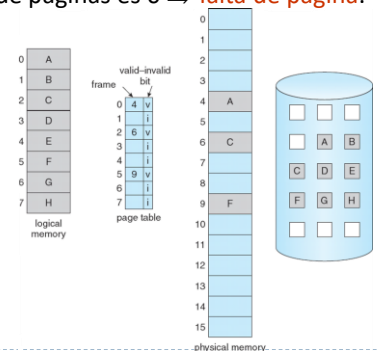
Paginación: Bit Válido-Inválido

- Se asocia a cada entrada a la tabla de páginas un bit válido–inválido (1 ⇒ en memoria, 0 ⇒ no en memoria)
- Inicialmente el bit válido–inválido es puesto a 0 en todas las entradas.

marco #	bit válido-inválido
	1
	1
	1
	1
	0
⋮	
	0
	0

tabla de páginas

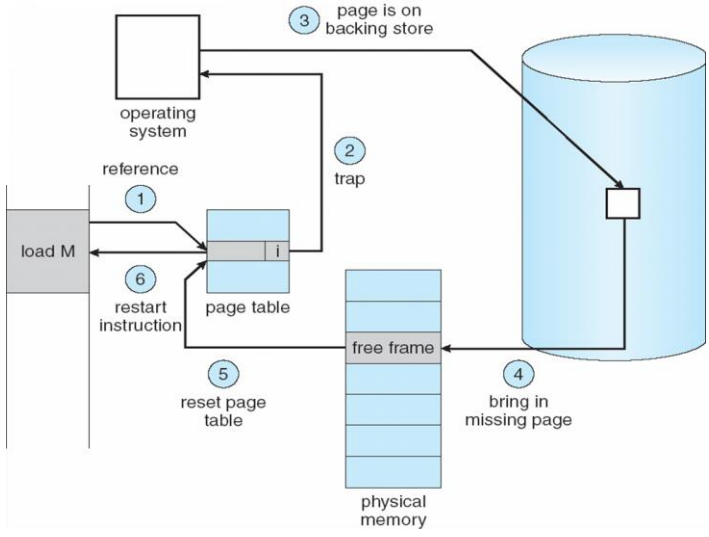
Durante la traducción de la dirección, si el bit válido–inválido en la entrada de la tabla de páginas es 0 ⇒ falta de página.



Falta de Página

- Si hay una referencia a una página, la primer referencia hace un trap al SO ⇒ **falta de página**
- El SO mira la tabla para decidir:
  - Referencia Inválida ⇒ aborto.
  - No está en memoria.
- Toma un marco libre.
- Lleva la página al marco.
- Reestablece las tablas, bit de validación = 1.
- Reinicio de la instrucción:
  - Movimiento de bloque
  - Locación con auto incremento/decremento

Pasos en el Manejo de una Falta de Página



Performance del Paginado por Demanda

- Ritmo de falta de páginas $0 \le p \le 1.0$
  - si p = 0 no hay falta de páginas
  - si p = 1, cada referencia es una falta de página
- Tiempo Efectivo de Acceso (TEA)

TEA = (1 – p) x acceso a memoria + p x (sobrecarga de falta de página + salida de la página + entrada de la página + sobrecarga de reinicio)

Ejemplo

- Tiempo de acceso a memoria = 200 nanosegundos
- Tiempo promedio de servicio de una falta de página = 8 milisegundos.

TEA = (1 – p) x 200 + p x (8 milisegundos)

= (1 – p) x 200 + p x 8000000

= 200 + p x 7999800

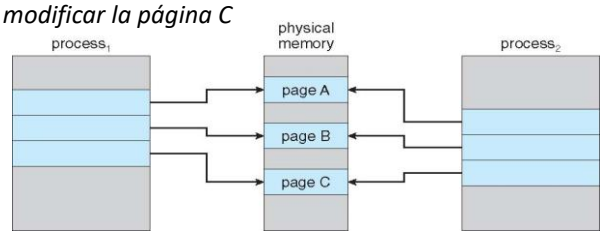
- Si uno de 1000 causa una falta de página, entonces TEA = 8.2 Microseconds.

Esto significa una reducción de 40!!

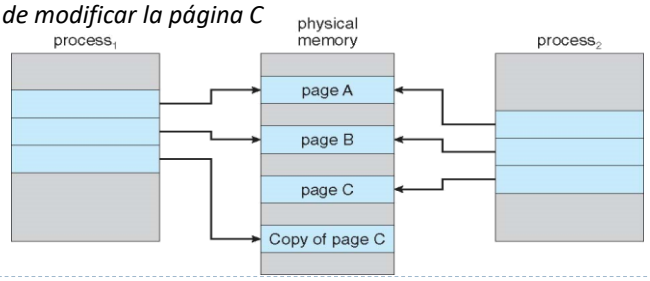
Copia en Escritura

Permite que un proceso padre comparta con el proceso hijo (fork())

Antes de modificar la página C



Después de modificar la página C



¿Qué ocurre cuando no hay marcos libres?

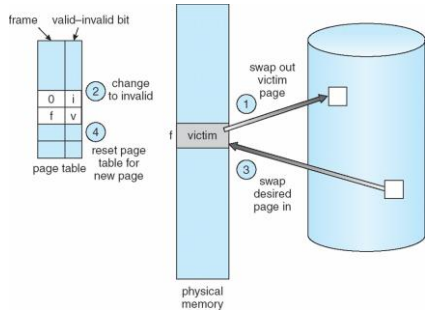
- Reemplazo de páginas – se busca alguna página en memoria que no está en uso y se la intercambia.
  - algoritmo
  - performance – se requiere un algoritmo que resulte en un mínimo número de falta de páginas.
- Algunas páginas pueden ser quitadas o volcadas en memoria varias veces.

Reemplazo de Páginas

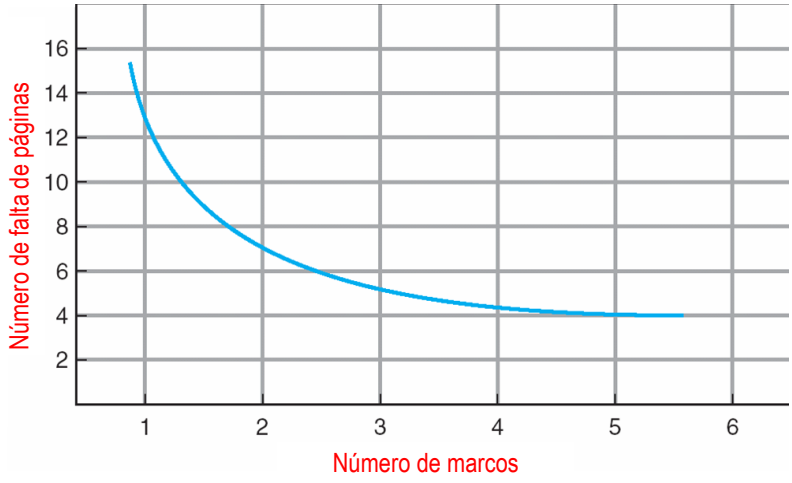
- Se previene sobrealocación de memoria por modificación de la rutina de servicio de falta de páginas para incluir el reemplazo de páginas.
- Uso del *bit de* ***modificado*** (*"sucio")* para reducir la sobrecarga de la transferencia de páginas – solo las páginas modificadas son escritas en disco.
- El reemplazo de páginas completa la separación entre memoria lógica y memoria física – puede ser provista una gran memoria lógica en una pequeña memoria física.

Reemplazo de Páginas

1. Encontrar la locación de la página deseada en el disco
2. Encontrar un marco libre:
   - Si hay un marco libre, usarlo
   - Si no hay marco libre usar un algoritmo de reemplazo de página para seleccionar el marco **víctima.**
3. Traer la página deseada al marco libre, modificar la tabla de páginas y la tabla de marcos.
4. Reiniciar el proceso



Grafo de Falta de Páginas Versus Número de Marcos



Algoritmos de Reemplazo de Páginas

- Procuran un ritmo de falta de páginas bajo.
- Se evalua el algoritmo ensayándolo sobre una secuencia particular de referencias a memoria (secuencia de referencia) y computando el número falta de páginas en la secuencia.
- En el ejemplo, la secuencia es

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Algoritmo Primero en entrar—Primero en salir (FIFO)

- Secuencia de referencia: **1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**
- 3 marcos (hay 3 páginas en memoria al mismo tiempo por proceso)

1	1	4	4	4	5	5	5	5	5	5	9 faltas de páginas
2	2	2	1	1	1	1	1	3	3	3	
3	3	3	3	2	2	2	2	2	4	4	

- 4 marcos

1	1	1	1	5	5	5	5	4	4	10 faltas de páginas
2	2	2	2	2	1	1	1	1	5	
3	3	3	3	3	3	2	2	2	2	
4	4	4	4	4	4	4	3	3	3	

- Reemplazo FIFO – Anomalía de Belady
  - más cuadros $\Rightarrow$ menos faltas de páginas

Algoritmo Óptimo

- Reemplace la página que no será usada por un período largo de tiempo.
- Ejemplo con 4 marcos

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	1	1	1	1	1	1	4	4
2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3
4	4	4	5	5	5	5	5	5

6 faltas de páginas

- ¿Cómo se conoce esto?
- Usado para medir como se comporta un algoritmo.

Algoritmo *Menos Recientemente Usado*(LRU)

- Secuencia de referencia: **1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**

1	1	1	1	1	1	1	1	5
2	2	2	2	2	2	2	2	2
3	3	3	5	5	5	5	4	3
4	4	4	4	4	4	3	3	4

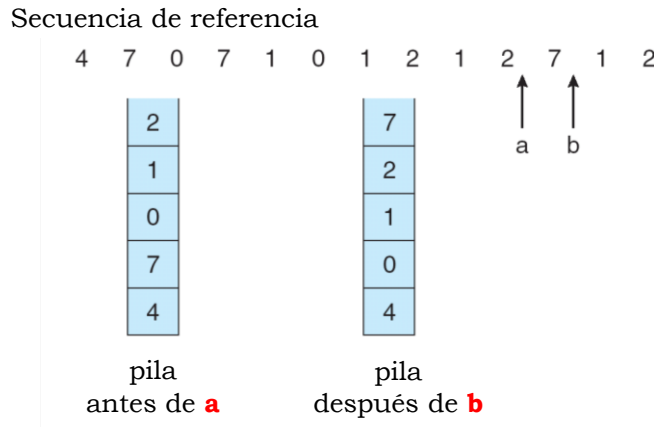
8 faltas de páginas

- Implementación del contador
  - Cada entrada a la tabla de páginas tiene un contador; cada vez que la página es referenciada se copia el reloj en el contador.
  - Cuando la página necesita ser cambiada, mira los contadores para determinar cuales hay que cambiar.

Algoritmo LRU

- Implementación por pila – mantiene una pila de números de páginas en forma de una lista doblemente enlazada:
  - Página referenciada:
    - se mueve al tope
    - Requiere cambios de punteros
  - No se necesita buscar para realizar el reemplazo

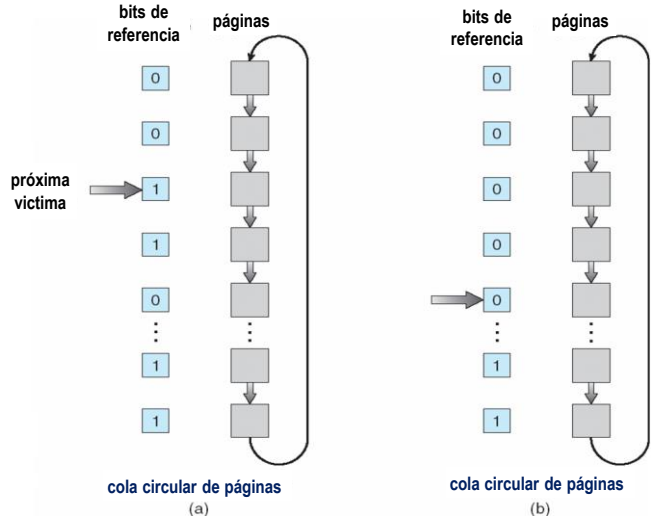
Uso de Pila para registrar las Referencias a Páginas Más Recientes



Algoritmos de Aproximación a LRU

- Bit de referencia. Con cada página se asocia un bit, inicialmente= 0
  - Cuando la página es referenciada el bit es puesto a 1.
  - Reemplace aquella en la cual es 0 (si existe). No se conoce el orden.
- Segunda oportunidad
  - Necesita el bit de referencia.
  - Reemplazo circular (Clock).
  - Si la página a ser reemplazada (en orden circular) tiene el bit de referencia = 1 entonces:
    - Se pone el bit de referencia en 0.
    - Se deja la página en memoria.
    - Se reemplaza la siguiente página (en el orden circular), sujeta a las mismas reglas.

Algoritmo de Reemplazo de Páginas Segunda Oportunidad (Reloj)



Algoritmos de Cuenta

- Se mantiene un contador del número de referencias que han sido hechas a la misma página.
- **Algoritmo LFU**: reemplaza la página con la menor cuenta.
- **Algoritmo MFU**: basado en el argumento que la página con la cuenta más chica fue recién puesta y todavía tiene que ser usada.

Alocación de Marcos

- Cada proceso necesita un número mínimo de páginas.
- Ejemplo: IBM 370 – 6 páginas para manejar la instrucción SS MOVE:
  - la instrucción es de 6 bytes, puede expandirse a 2 páginas.
  - 2 páginas para manejar **desde**.
  - 2 páginas para manejar **hacia**.
- Dos esquemas de alocación.
  - alocación fija
  - alocación con prioridad

Alocación Fija

- Alocación igualitaria – p.e., si hay 100 marcos y 5 procesos, a cada uno se les da 20 páginas.
- Alocación Proporcional – Aloca de acuerdo al tamaño del proceso.

s_i = tamaño del proceso p_i

$$S = \sum s_i$$

m = número total marcos

a_i = alocación para $p_i = \frac{s_i}{S} x\, m$

$m = 64$

$s_i = 10$

$s_2 = 127$

$$a_1 = \frac{10}{137} \times 64 \approx 5$$

$$a_2 = \frac{127}{137} \times 64 \approx 59$$

Alocación con Prioridad

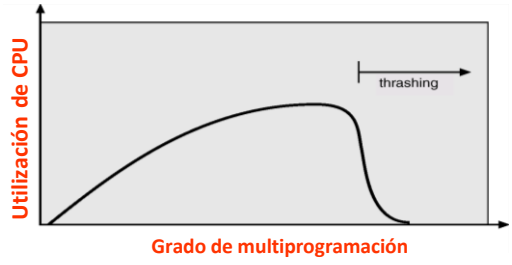
- Se usa un esquema de alocación proporcional usando prioridades antes que tamaño.
- Si el proceso P_i genera una falta de página
  - Se selecciona para reemplazar uno de sus marcos.
  - Se selecciona para reemplazar un marco de un proceso con menor número de prioridad.

Reemplazo Global vs. Local

- Reemplazo global – el proceso selecciona un marco de reemplazo de todos los marcos; un proceso puede tomar los marcos de otro.
- Reemplazo local – cada proceso selecciona de su propio conjunto el marco a reemplazar.

Thrashing

- Si un proceso no tiene suficientes páginas, el ritmo de falta de páginas es muy alto. Esto lleva a:
  - baja utilización de CPU.
  - el SO piensa que es necesario incrementar el grado de multiprogramación.
  - otro proceso se agrega al sistema.
- Thrashing ≡ un proceso está ocupado haciendo solamente intercambio de páginas.



Thrashing

- ¿Por qué trabaja el paginado?
  Modelo de Localidad
  - El proceso migra desde una localidad a otra.
  - Las localidades se pueden solapar.
- ¿Por qué ocurre el thrashing ?
  Σ tamaño de la localidad > tamaño total de memoria

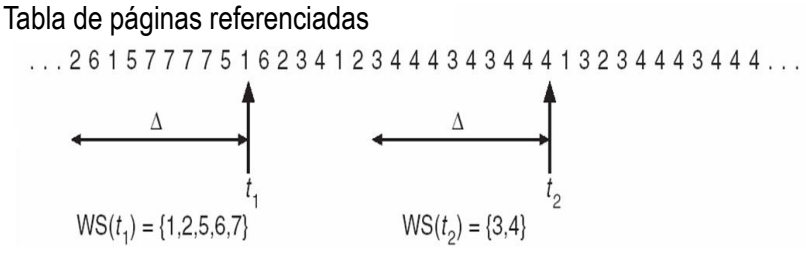
Modelo de Conjunto de Trabajo (Working-Set)

El modelo está basado en la localidad.

- $\Delta \equiv$ ventana working-set $\equiv$ un número fijo de referencias de páginas. Ejemplo: 10,000 instrucciones
- WSS_i (working set del proceso P_i) =
  número total de páginas referenciadas en el más reciente Δ (varía en el tiempo)
  - si Δ es demasiado chico no acompaña la localidad.
  - si Δ es demasiado grande acompaña varias localidades.
  - si $\Delta = \infty \Rightarrow$ acompañará al programa entero.
- $D = \Sigma\ WSS_i \equiv$ demanda total de marcos
- si $D > m \Rightarrow$ Thrashing
- Política: si $D >$ m, entonces suspende uno de los procesos.

m=# de marcos de memoria

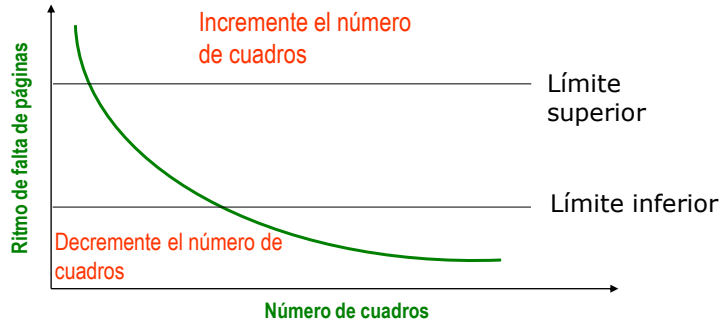
Modelo de Working-set



Control del Working Set

- Aproximar con un intervalo de tiempo + bit de referencia
- Ejemplo: Δ = 10,000
  - Las interrupciones del Timer se producen cada 5000 unidades de tiempo.
  - Se mantienen en memoria 2 bits por cada página.
  - Siempre que el timer interrumpe copia e inicializa los valores de todos los bits de referencia a 0.
  - Si uno de los bits en memoria = 1 $\Rightarrow$ página en el working set.
- ¿Por qué no es completamente preciso?
- Mejora = 10 bits e interrupción cada 1000 unidades de tiempo.

Esquema de Frecuencia de Falta de Página



- La idea es establecer un ritmo "aceptable" de falta de páginas.
  - Si el ritmo actual es demasiado bajo, los procesos pierden marcos.
  - Si el ritmo actual es demasiado alto, el proceso gana marcos.

Otras Consideraciones - Prepaginado

- Prepaginado
  - Para reducir el gran número de falta de páginas que ocurren en el inicio del proceso
  - Se necesitará prepaginar todas o algunas páginas del proceso antes de ser referenciadas
  - Pero si las páginas prepaginadas no son usadas se incurrió en gasto de E/S y memoria
  - Suponga que ***s*** páginas son prepaginadas y ***α*** de esas páginas son usadas
    - Es el costo de salvar ***s* α*** faltas > ó < qué el costo de prepaginar ***s* (1- α)*** paginas no necesarias?
    - ***α*** cercano a cero ⇒ prepaginado pierde

Otras Consideraciones – Tamaño de Página

- Selección del tamaño de página
  - fragmentación
  - tamaño de tabla
  - sobrecarga de E/S
  - localidad

Otras Consideraciones – Estructura del Programa

- Estructura de programa

  Arreglo de enteros A[1024, 1024]

  Cada fila está almacenada en una página

  Un cuadro

  - Programa 1

```
for j := 1 to 1024 do
    for i := 1 to 1024 do
        A[i,j] := 0;
```

  1024 x 1024 faltas de páginas

  - Programa 2

```
for i := 1 to 1024 do
    for j := 1 to 1024 do
        A[i,j] := 0;
```

  1024 faltas de páginas

- Fijación para E/S y direccionamiento

Otras Consideraciones – Fijación de E/S

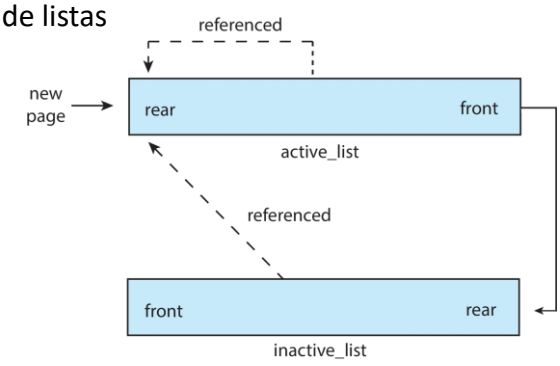
- **Fijación de E/S** – Algunas veces las páginas deben ser fijadas en la memoria
- Considere E/S – Las páginas que son usadas para copiar un archivo desde un dispositivo deben ser fijadas para no ser seleccionadas por el algoritmo de reemplazo de páginas

Ejemplos de Sistemas Operativos

- Linux
- Windows
- Solaris

Linux

- Usa demanda de páginas, aloca las páginas de una lista de marcos libres.
- Usa política de reemplazo global, similar a la aproximación de LRU utilizando el algoritmo del Reloj.
- Mantiene dos tipos de listas
  - Lista de Activa
  - Lista de Inactive



Windows

- ✓ Usa demanda de páginas con **clustering**. Es decir trae las páginas que rodean a la página demandada.
- ✓ A los procesos se les asigna un **working set mínimo** y un **working set máximo.**
- ✓ El *working set* mínimo es el mínimo número de páginas que requiere tener el proceso en memoria.
- ✓ A un proceso se le puede asignar tantas páginas como indica su *working set* máximo.
- ✓ Cuando la cantidad de memoria libre en el sistema cae bajo determinado umbral, se activa en forma automática hasta restaurar la cantidad de memoria libre.
- ✓ Este ajuste automático remueve páginas de los procesos que están excedidos de su *working set* mínimo

UNIX-SVR4

Formato Administración de Memoria

Entrada en la tabla de páginas

Page frame number	Age	Copy on write	Mod-ify	Refe-rence	Valid	Pro-tect

Descriptor de bloque de disco

Swap device number	Device block number	Type of storage

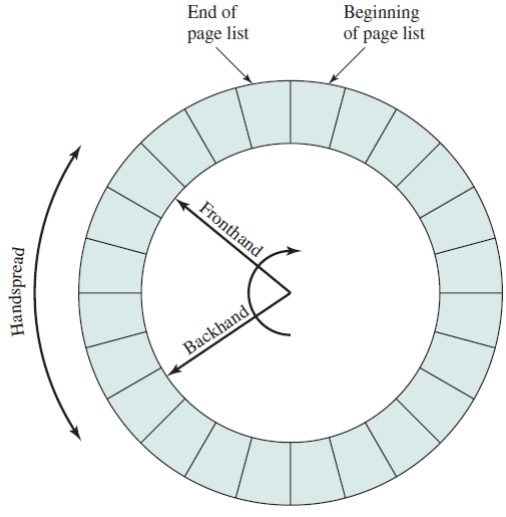
Entrada en la tabla de dato de página frame

Page state	Reference count	Logical device	Block number	Pfdata pointer

Entrada en la tabla de swap-use

Reference count	Page/storage unit number

UNIX- SVR4: Algoritmo Reemplazo de Página basado en el Reloj



UNIX-Solaris

- ✓ Mantiene una lista de páginas libres para asignar a procesos en falta
- ✓ ***Lotsfree*** – parámetro umbral (cantidad de memoria libre) para comenzar a paginar
- ✓ ***Desfree*** – parámetro umbral para incrementar el paginado
- ✓ ***Minfree*** – parámetro umbral para ser intercambiadas las páginas
- ✓ El paginado es realizado por un proceso ***pageout***
- ✓ Pageout barre las páginas usando un algoritmo de reloj modificado
- ✓ ***Scanrate*** es la frecuencia con que las paginas son barridas. Estos rangos varían entre ***slowscan*** y ***fastscan***
- ✓ La frecuencia de llamado a *pageout* depende de la cantidad de memoria libre disponible.

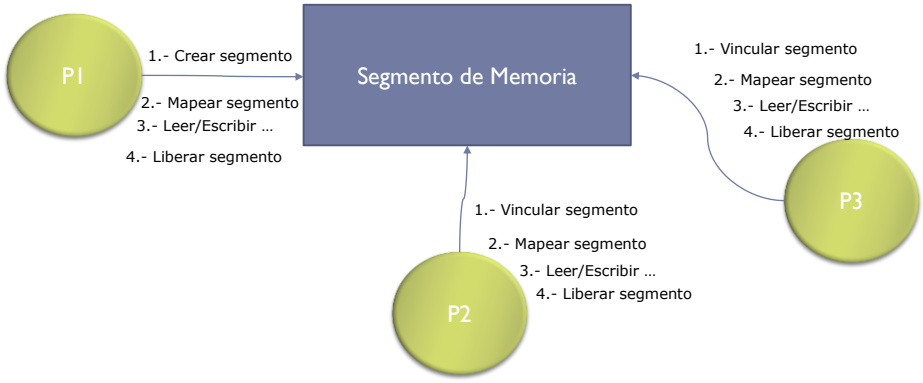
Bibliografía:

- Silberschatz, A., Gagne G., y Galvin, P.B.; "*Operating System Concepts*", 7ma Edición. 2009, 9na. Edición 2012.
- Stallings, W. *"Operating Systems: Internals and Design Principles"*, Prentice Hall, 6ta Edición, 2009; 7maEdición, 2011; 8va. Edición, 2014..

Comunicación entre Proceso

- Segmento de Memoria
- Colas de Mensajes

Segmento de Memoria



Segmento de Memoria

1. Creación de un nuevo segmento de memoria compartida o acceder a uno existente.
   - Llamada al sistema: **shmget (operación 1)**
2. Mapeado del segmento de memoria compartida al espacio de direcciones del proceso.
   - Llamada al sistema: **shmat (operación 2)**
3. Lectura o escritura. En esta zona se lee y se escribe como en cualquier dirección de memoria del proceso. Por ser una zona de memoria compartida, para realizar el acceso a esta zona hay que utilizar semáforos para obtener acceso exclusivo. Llamadas al sistema correspondientes a semáforos ya vistas en prácticas anteriores.
4. Desenlace del segmento de memoria compartida por parte del proceso.
   - Llamada al sistema: **shmdt**

Segmento de Memoria

1.- Crear u obtener un nuevo segmento de memoria

int shmget (key_t key, size_t size, int shmflg)

Por ejemplo:

```
struct info{
  char mensaje[255];        /* dato del segmento */
};
#define KEY   ((key_t) (1243))        /* número de llave */
#define SEGSIZE  sizeof (struct info)    /* longitud del segmento */
....
    id = shmget(KEY, SEGSIZE, IPC_CREAT | 0666);
id = shmget(KEY, SEGSIZE, 0);
```

Definiciones

Crea un segmento

Vincula a un segmento

Segmento de Memoria

2.- Mapear un segmento de memoria

char *shmat(int shmid, void *shmaddr, int shmflg);

Por ejemplo:

```
...
ctrl = (struct info*) shmat(id,0,0);
```

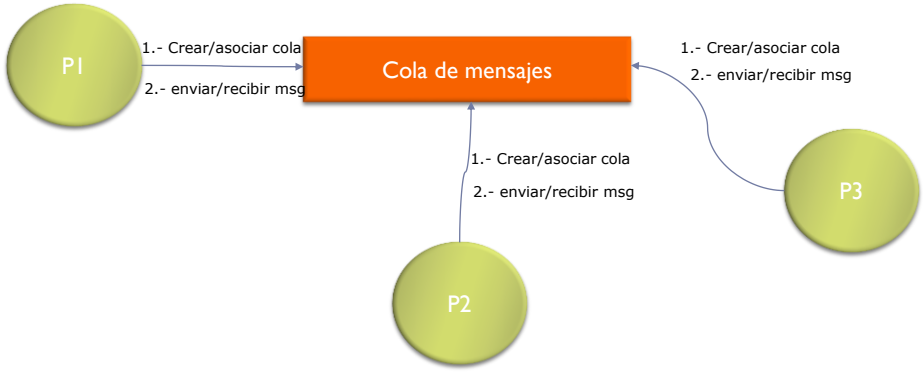
4.- Desvincular un segmento de memoria

int shmdt(void *shmaddr)

Por ejemplo:

```
...
resul = shmdt(ctrl);
```


Colas de Mensajes



Colas de Mensajes

1. Creación/asociación a una cola de mensajes.
   - Llamada al sistema: **msgget (operación 1)**
2. Lectura o escritura
   - Llamada al sistema: **msgsnd, msgrcv (operación 2)**

struct mensaje{ long tipo; int dato_1; int dato_2; };

Colas de Mensajes

1.- Crear o asociar cola de mensajes

int msgget(key_t key, int msgflg)

Por ejemplo:

queueID = msgget(key, IPC_CREAT|0666);

Crea y asocia

queueID = msgget(key, 0666);

asocia

Estructura del mensaje

```
struct mensaje{
        long tipo;
        int dato_1;
        …
};
```

En todas las estructuras

Colas de Mensajes

2.- Envío y recepción de mensajes.

int msgsnd(int msqid, struct msgbuf *msgp, int msgsz, int msgflg);

int msgrcv(int msqid, struct msgbuf *msgp, int msgsz, long msgtyp, int msgflg);

- Si msgtyp = 0 - se accederá al primer mensaje que se encuentre en la cola independientemente de su tipo.
- Si msgtyp > 0 - se accederá al primer mensaje del tipo ***msgtyp*** que se encuentre almacenado en la cola.
- Si msgtyp < 0 - se accederá al primer mensaje cuyo tipo sea menor o igual al valor absoluto de ***msgtyp*** y a la vez sea el menor de todos.

Colas de Mensajes

Ejemplo:

```
...
        longitud = sizeof(struct mensaje) - sizeof(long);
        if (msgsnd(idmsg, &mimensaje, longitud, 0) == -1) {
                fprintf(stderr, ``Error en la escritura del mensaje\n'');
                exit(1);
        }
...

        if (msgrcv(idmsg, &mimensaje, longitud, 1, 0) == -1) {
                fprintf(stderr, ``Error en la escritura del mensaje\n'');
                exit(1);
        }
```