



Sistemas Operativos

Gestión de memoria
Espacio de direcciones. Memoria virtual.



Agenda

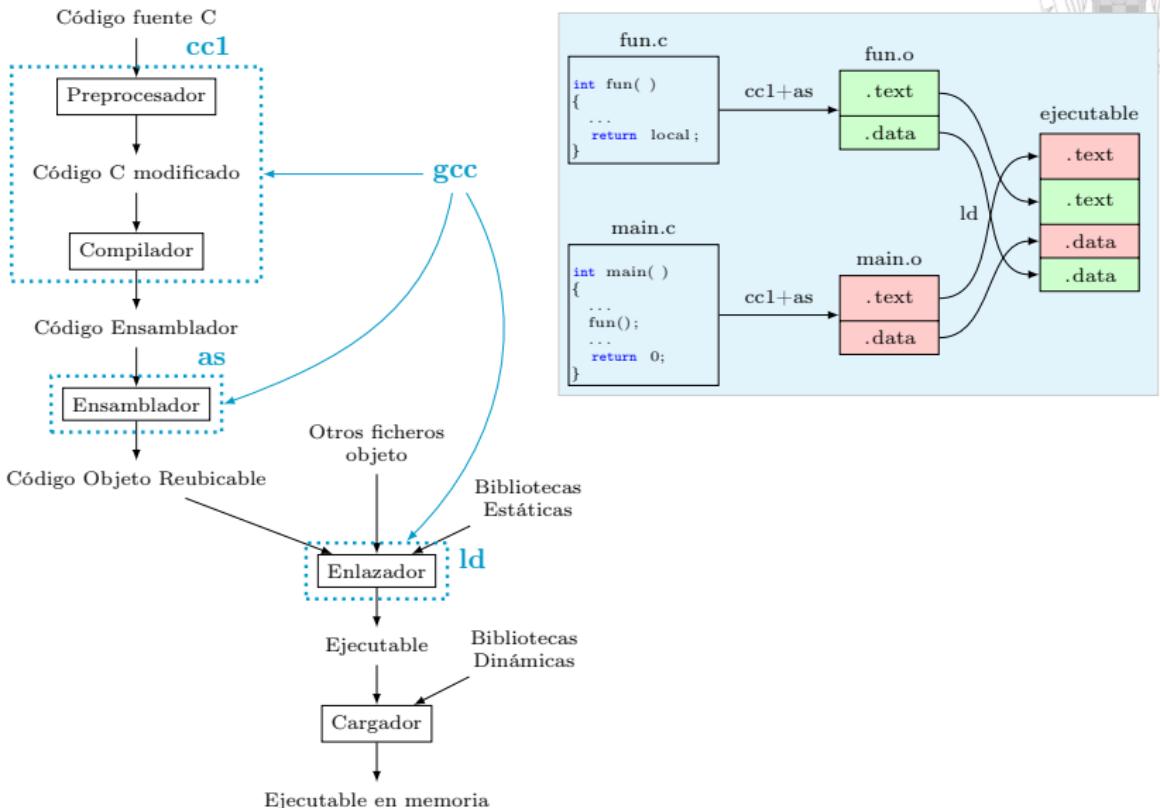
- 1 Modelo de memoria de un proceso**
- 2 Servicios POSIX para el manejo de regiones**
- 3 Requisitos para la gestión de memoria**
- 4 Asignación contigua**
- 5 Memoria virtual con Paginación Bajo Demanda**



Agenda

- 1 Modelo de memoria de un proceso**
- 2 Servicios POSIX para el manejo de regiones**
- 3 Requisitos para la gestión de memoria**
- 4 Asignación contigua**
- 5 Memoria virtual con Paginación Bajo Demanda**

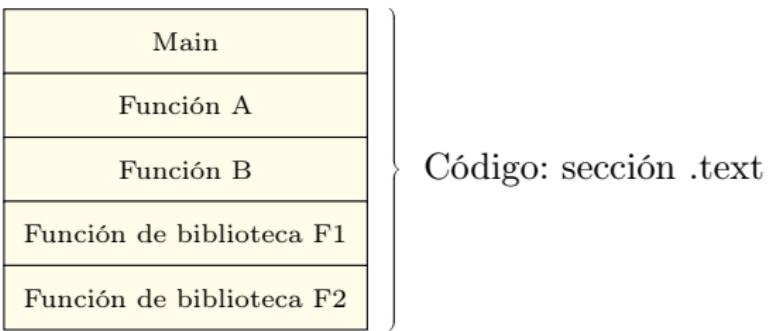
Construcción de un ejecutable





Bibliotecas estáticas

- Colección de módulos objeto relacionados que exportan un conjunto de símbolos globales (funciones, variables, ...)
- Estáticas: el *linker* enlaza los módulos objeto del programa y de las bibliotecas creando un ejecutable autocontenido



Desventajas de las bibliotecas estáticas

- Ejecutables grandes
- Código de biblioteca repetido en muchos ejecutables y memoria
- Actualización de biblioteca implica volver a generar el ejecutable



Bibliotecas dinámicas

Enlazado y carga en tiempo de ejecución:

- El ejecutable contiene:
 - 1 Nombre de la biblioteca
 - 2 Rutina de carga y montaje en tiempo de ejecución
- La rutina de carga se invoca en la primera referencia a un símbolo de la biblioteca.

Ventajas

- Menor tamaño ejecutables
 - Código de rutinas de biblioteca sólo en archivo de biblioteca
- Procesos pueden compartir código de biblioteca dinámica
- Actualización automática de bibliotecas: Uso de versiones

Desventajas

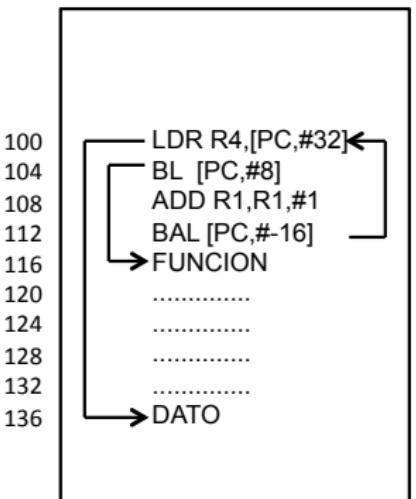
- Ejecutable no es autocontenido
- Mayor tiempo de ejecución debido a carga y montaje
 - Tolerable, compensado por las ventajas



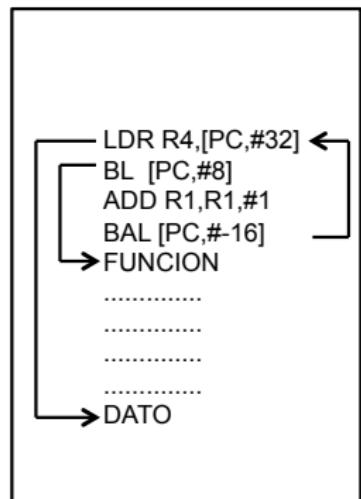
Compartición de bibliotecas dinámicas

- Cualquier biblioteca dinámica contiene referencias a símbolos internos
- El código debe ser independiente de la posición de carga (PIC)
 - Simplifica carga de bibliotecas en el mapa de memoria
 - Permite la compartición de bibliotecas en memoria

Memoria



Memoria



Formato del ejecutable



- En UNIX *Executable and Linkable Format* (ELF)

Estructura simplificada

1 Cabecera

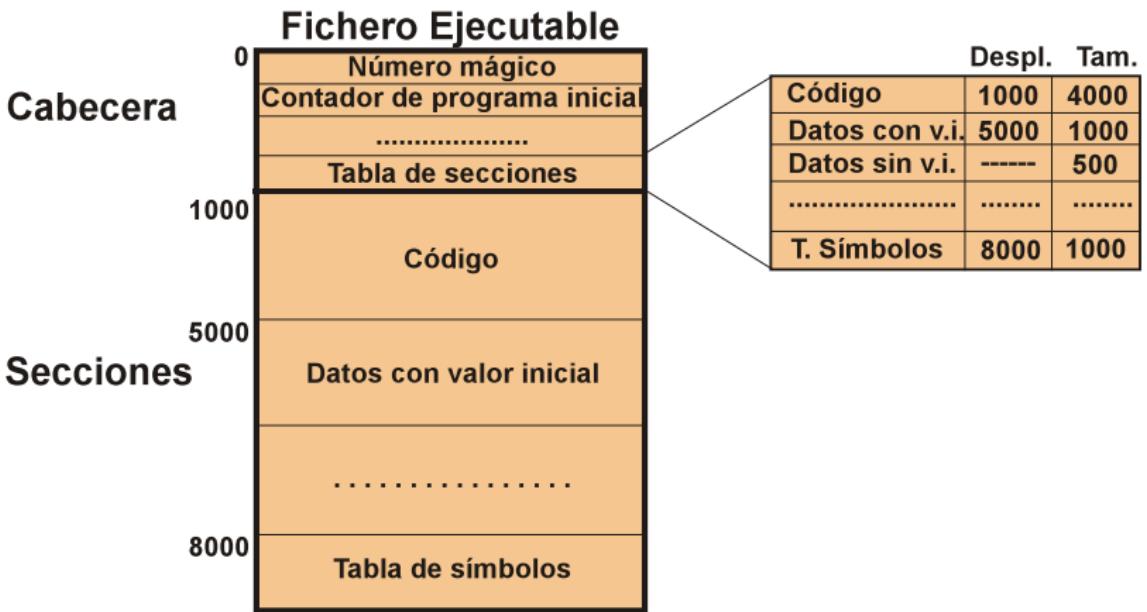
- Número mágico que identifica a ejecutable (0x7f+'ELF')
- Punto de entrada del programa
- Tabla de secciones

2 Secciones

- Código (text)
- Datos inicializados
- Datos no inicializados (sólo se especifica el tamaño)
- Tabla de símbolos de depuración
- Tabla de bibliotecas dinámicas



Formato del ejecutable





Variables globales vs. locales

Variables globales (con sección)

- Estáticas
- Se crean al iniciarse programa
- Existen durante ejecución del mismo
- Dirección fija en memoria y en ejecutable

Variables locales y parámetros (sin sección)

- Dinámicas
- Se crean al invocar función
- Se destruyen al retornar
- La dirección se calcula en tiempo de ejecución
- Recursividad: varias instancias de una variable



Variables globales vs. locales

Ejemplo

```
int x=8;          /* Variable global con valor inicial */
int y;            /* Variable global sin valor inicial */
int j=5;          /* Inicialización y reserva de espacio
                     para var. global */

int f(int t) {    /* Parámetro, es una variable local */
    int z;          /* Variable local */
    ...
}

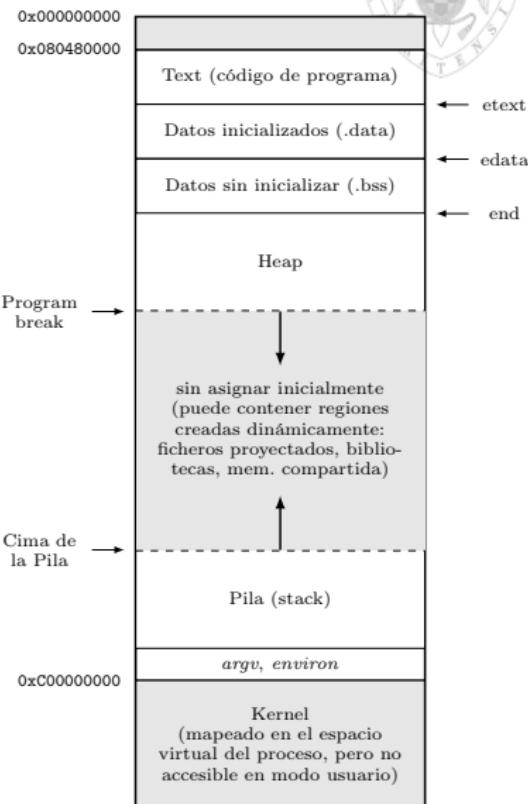
int main() {
    ...
    f(4);           /* Inicialización dinámica del parámetro */
    ...
}
```

Mapa de memoria de un proceso (MMP)



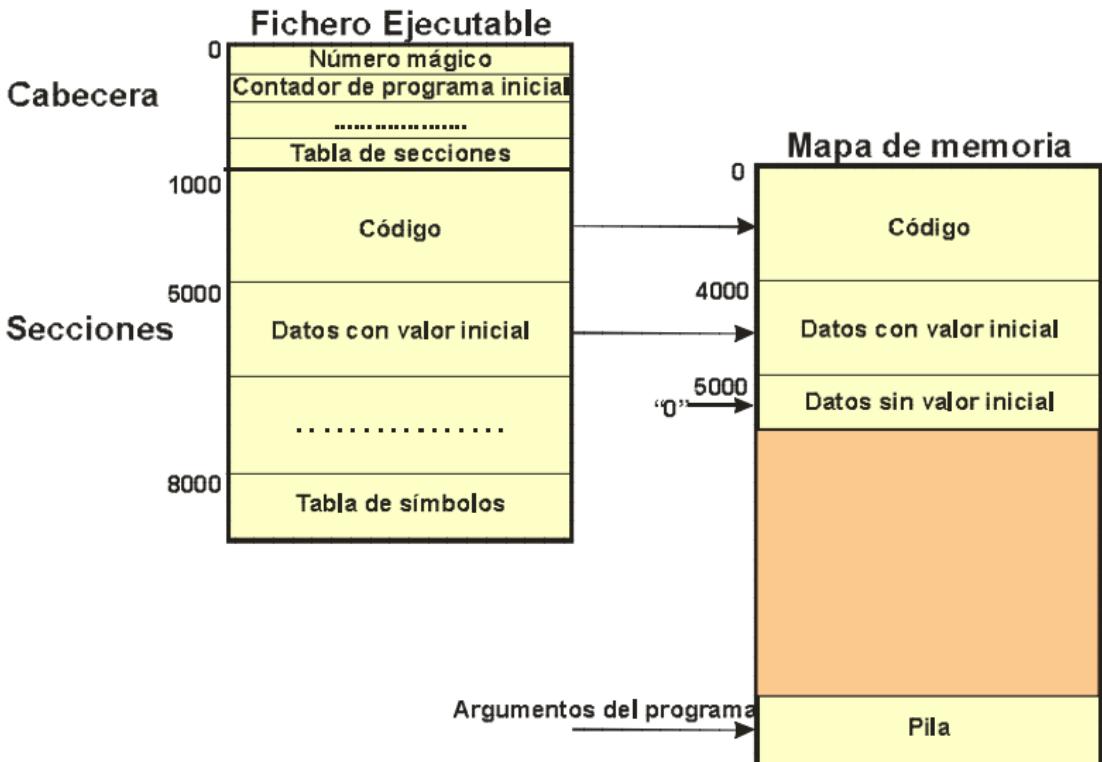
Conjunto de regiones de memoria accesibles

- Cada región es un rango contiguo de direcciones virtuales con unos atributos
 - Dirección de comienzo
 - Tamaño inicial
 - Tamaño fijo o variable
 - Modo de crecimiento: ↑↓
 - Soporte: fichero del que se toma el contenido inicial, si lo tuviese
 - Protección: RWX
 - Carácter compartido o privado
 - Privado: sólo un proceso puede acceder a la región. En fork se marca como *copy on write*.
 - Compartido: región de memoria accesible a otros procesos, en fork no se duplica.





Creación del MMP desde el ejecutable (exec*)





Resto de regiones del MMP

Creadas durante la ejecución del proceso:

■ Región de Heap

- Soporte de memoria dinámica (`malloc()` en C)
- Privada, RW, T. Variable, Sin Soporte (rellenar 0's)
- Crece hacia direcciones más altas

■ Regiones de bibliotecas dinámicas

- Se crean regiones asociadas al código y datos de la biblioteca

■ Pilas de threads

- Cada pila de *thread* corresponde con una región
- Mismas características que pila del proceso

■ Memoria compartida

- Región asociada a la zona de memoria compartida
- Compartida, T. Variable, Sin Soporte (rellenar 0's)
- Protección especificada en proyección

■ Fichero proyectado en memoria

- Región asociada al archivo proyectado
- T. Variable, Soporte en fichero
- Protección y carácter compartido o privado especificado en proyección



Características de las regiones

Mapa de memoria

Código
Datos con valor inicial
Datos sin valor inicial
Heap
Fichero proyectado F
Zona de memoria compartida
Código biblioteca dinámica B
Datos biblioteca dinámica B
Pila de thread 1
Pila del proceso

Región	Soporte	Protección	Comp/Priv	Tamaño
Código	Fichero	RX	Compartida	Fijo
Dat. con v.i.	Fichero	RW	Privada	Fijo
Dat. sin v.i.	Sin soporte	RW	Privada	Fijo
Pilas	Sin soporte	RW	Privada	Variable
Heap	Sin soporte	RW	Privada	Variable
F. Proyect.	Fichero	por usuario	Comp./Priv.	Variable
M. Comp.	Sin soporte	por usuario	Compartida	Variable



Agenda

- 1** Modelo de memoria de un proceso
- 2** Servicios POSIX para el manejo de regiones
- 3** Requisitos para la gestión de memoria
- 4** Asignación contigua
- 5** Memoria virtual con Paginación Bajo Demanda



Creación de regiones de memoria

La llamada al sistema `mmap()` permite crear una nueva región de memoria (*memory mapping*) en el espacio de direcciones virtuales del proceso. Estas regiones pueden ser de dos tipos:

- **Mapeo de Fichero.** Se inicializa la región de memoria copiando parte del contenido de un fichero. Suele llamarse región de fichero proyectado.
- **Mapeo anónimo (MAP_ANONYMOUS).** Se inicializa la región a 0. Se dice que no tiene respaldo.

A su vez, el mapeo puede tener un carácter privado/compartido:

- **Mapeo privado (MAP_PRIVATE):** los cambios en la región no actualizan el fichero de respaldo (si lo hay) y no son visibles en otros procesos.
- **Mapeo compartido (MAP_SHARED):** los cambios en la región actualizan el fichero de respaldo (si lo hay) y son visibles en otros procesos que comparten el mismo mapeo.



Llamada al sistema mmap

```
void* mmap(void *addr, size_t len, int prot, int  
flags, int fd, off_t offset);
```

Crea una región nueva en el mapa de memoria del proceso, dónde:

- **addr**, dirección sugerida para el mapeo
- **len**, tamaño de la región. Si no es múltiplo del tamaño de página se toma el siguiente múltiplo.
- **prot**, máscara de los bits de protección de la región (permisos de acceso): PROT_READ, PROT_WRITE, PROT_EXEC, PROT_NONE
- **flags**, máscara de bits que determina el tipo de región con la combinación (OR) de los flags MAP_PRIVATE y MAP_SHARED con MAP_ANONYMOUS
- **fd**: el descriptor de fichero del fichero a proyectar si es un mapeo de fichero, o -1 si es un mapeo anónimo.
 - Debe abrirse siempre con permiso de lectura, y si prot tiene PROT_WRITE y flags tiene MAP_SHARED deben incluirse también permisos de escritura.
- **offset**: el offset en el fichero a partir de donde se realiza la proyección. Debe ser 0 si el flag MAP_ANONYMOUS está activo.



mmap tipos de regiones

En función de la combinación de los flags MAP_SHARED y MAP_PRIVATE con MAP_ANONYMOUS tenemos cuatro tipos de regiones:

- MAP_PRIVATE: mapeo de fichero privado. Se usa el contenido del fichero para inicializar la región. No sirve como mecanismo IPC. Es una región *copy-on-write*.
- MAP_PRIVATE | MAP_ANONYMOUS: mapeo anónimo, sirve para crear una nueva región de memoria inicializada a 0. No sirve como mecanismo IPC. Es una región *copy-on-write*.
- MAP_SHARED: mapeo de fichero compartido. Se usa el contenido del fichero para inicializar la memoria. Todos los procesos que tengan el mismo mapeo usan las mismas páginas físicas. El fichero se actualiza con los cambios. Sirve como mecanismo IPC, con respaldo en el sistema de ficheros.
- MAP_SHARED | MAP_ANONYMOUS: mapeo anónimo compartido. La región se inicializa a 0. Los procesos hijo heredan este mapeo y usan las mismas páginas físicas. Sirve como mecanismo IPC entre procesos con relación de parentesco.



POSIX: munmap y msync

```
int munmap(void *addr, size_t len);
```

- Elimina las regiones mapeadas entre addr y addr + len
- En una región compartida con respaldo en fichero deberíamos hacer primero una llamada a msync para asegurarnos que se actualiza el fichero antes de eliminar la región
- Todos los locks (mlock y mlockall) se eliminan al desmapear la región
- En exec todas las regiones se eliminan automáticamente

```
int msync(void *addr, size_t len, int flags);
```

- Los ficheros de respaldo de las regiones MAP_SHARED los actualiza automáticamente el sistema, pero no se especifica cuándo se hará
- msync permite forzar la actualización:
 - MS_SYNC: bloqueando el hilo hasta que se complete la operación
 - MS_ASYNC: actualizando la cache de bloques, de forma que los reads al fichero obtendrán el nuevo valor, pero dejando la escritura a disco para el futuro
 - MS_INVALIDATE: se invalidan otros mapeos del mismo fichero para que se cargen de nuevo en el siguiente acceso



Agenda

- 1** Modelo de memoria de un proceso
- 2** Servicios POSIX para el manejo de regiones
- 3** Requisitos para la gestión de memoria
- 4** Asignación contigua
- 5** Memoria virtual con Paginación Bajo Demanda



Objetivos del sistema de gestión de memoria

El SO multiplexa recursos entre procesos

- Cada proceso cree que tiene una máquina para él solo
- Gestión de procesos: Reparto de procesador
- Gestión de memoria: Reparto de memoria

Objetivos:

- Ofrecer a cada proceso un espacio lógico propio
- Dar soporte a las regiones del proceso
- Proporcionar protección entre procesos
- Permitir que procesos comparten memoria
- Maximizar el grado de multiprogramación
- Proporcionar a los procesos mapas de memoria muy grandes

Espacios lógicos independientes



Reubicación:

- Traducir direcciones lógicas a físicas
 - Dir. lógicas: direcciones de memoria generadas por el programa
 - Dir. físicas: direcciones de mem. principal asignadas al proceso
- La traducción es diferente para cada proceso:
 - $\text{Traducción}(PID, \text{dir_lógica}) \rightarrow \text{dir_física}$
- Crea espacio lógico independiente para proceso
 - SO debe poder acceder a espacios lógicos de los procesos
- Dos alternativas de reubicación:
 - Software
 - Hardware

Reubicación software



- Traducción de direcciones durante carga del programa
- Programa en memoria distinto del ejecutable
 - Requiere código independiente de posición (PIC)
- Desventajas:
 - No asegura protección
 - No permite mover programa en tiempo de ejecución

Memoria

10000	LDR R1,#11000
10004	LDR R2,#12000
10008	LDR R3,#11500
10012	LDR R4,[R3]
10016	LDR R5,[R1],#1
10020	STR R5,[R2],#1
10024	SUBS R4,R4,#1
10028	BNE 10016
10032

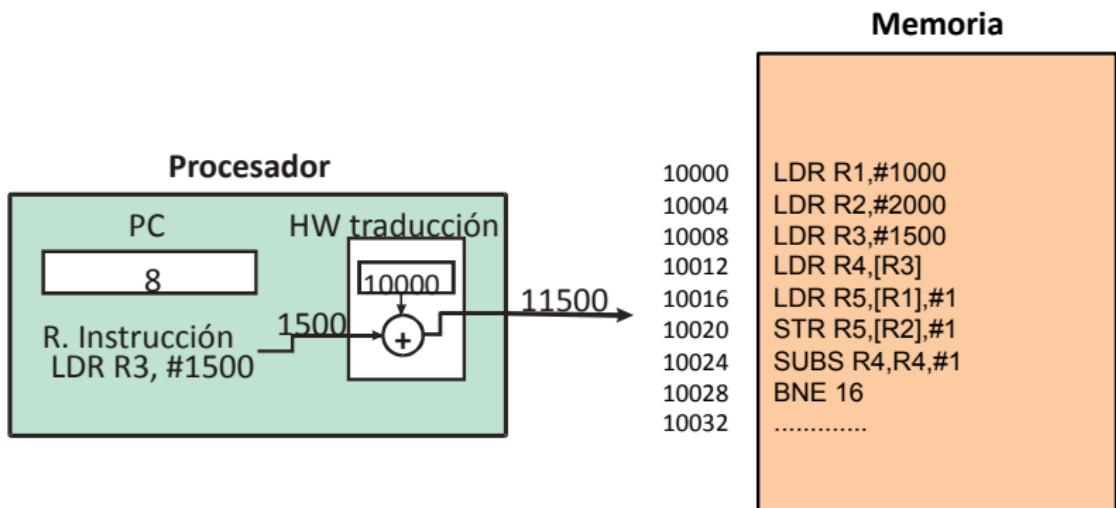


Reubicación hardware

Memory Management Unit (MMU): hw encargado de hacer la traducción de dirección lógica (virtual) a dirección física

- El SO almacena por cada proceso los valores con los que configurar la MMU para el proceso
- Estos valores de configuración determinan la función de traducción que se usará (será por tanto distinta para todos los procesos)

El programa se puede cargar en memoria sin modificar





Soporte de regiones

- Mapa de proceso no homogéneo
 - Conjunto de regiones con distintas características
 - Ejemplo: Región de código no modifiable
- Mapa de proceso dinámico
 - Regiones cambian de tamaño (p.ej. pila)
 - Se crean y destruyen regiones
 - Existen zonas sin asignar (huecos)
- Gestor de memoria debe dar soporte a estas características:
 - 1 Detectar accesos no permitidos a una región
 - 2 Detectar accesos a huecos
 - 3 Evitar reservar espacio para huecos
- SO debe guardar una tabla de regiones para cada proceso



Protección

- Monoprogramación: Proteger al SO
- Multiprogramación: Además procesos entre sí

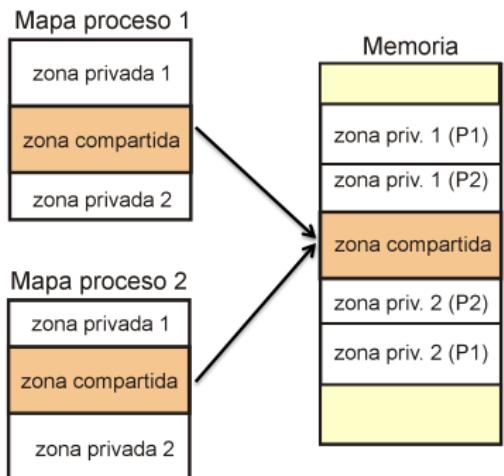
Requisitos mínimos:

- 1 La traducción debe crear espacios de direcciones disjuntos
- 2 Es necesario validar todas las direcciones que genera el programa
 - La detección se realiza por HW → generación de excepción
 - El tratamiento lo hace el SO



Compartición de memoria

- Direcciones lógicas de 2 o más procesos se corresponden con misma dirección física
- Bajo control del SO
- Beneficios:
 - Mecanismo de comunicación entre procesos muy rápido
 - Procesos ejecutando mismo programa comparten su código
- Requiere asignación no contigua



Utilización de la memoria



Reparto de memoria maximizando grado de multiprogramación:

- Se *desperdicia* memoria debido a:
 - Restos inutilizables (fragmentación)
 - Estructuras de datos requeridas por el propio gestor de memoria
- Mejor Compromiso → Paginación
 - Menor fragmentación implica tablas más grandes
 - Tablas más grandes implica más sobrecarga
 - Tablas multinivel

Aprovechamiento de memoria óptimo pero irrealizable
Memoria

0	Dirección 50 del proceso 4
1	Dirección 10 del proceso 6
2	Dirección 95 del proceso 7
3	Dirección 56 del proceso 8
4	Dirección 0 del proceso 12
5	Dirección 5 del proceso 20
6	Dirección 0 del proceso 1
.....	
N-1	Dirección 88 del proceso 9
N	Dirección 51 del proceso 4



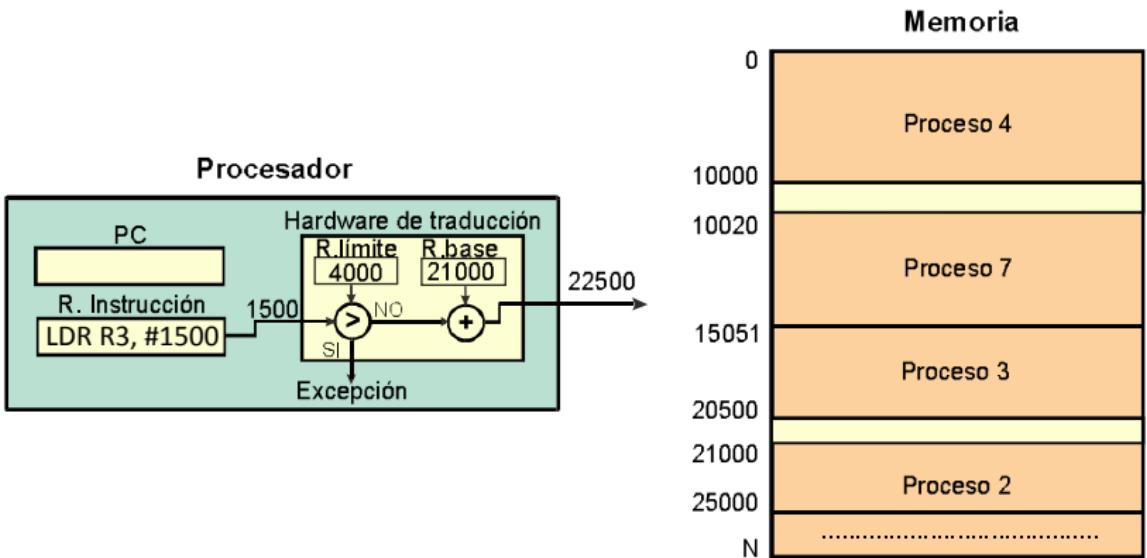
Agenda

- 1** Modelo de memoria de un proceso
- 2** Servicios POSIX para el manejo de regiones
- 3** Requisitos para la gestión de memoria
- 4** Asignación contigua
- 5** Memoria virtual con Paginación Bajo Demanda



Asignación contigua (1/2)

- Mapa de memoria de cada proceso en zona contigua de memoria principal
- Hardware requerido: Regs. valla (R. base y R. límite)
 - Sólo accesibles en modo privilegiado.





Asignación contigua (2/2)

SO mantiene información sobre:

- 1** El valor de regs. valla de cada proceso en su BCP
 - En cambio de contexto SO carga en regs. valor adecuado
- 2** Estado de ocupación de la memoria
 - Estructuras de datos que identifiquen huecos y zonas asignadas
 - Regiones de cada proceso

Este esquema presenta fragmentación externa

- Se generan pequeños fragmentos libres entre zonas asignadas
- Posible solución: compactación → proceso costoso



Operaciones sobre regiones con a. contigua

- Al crear un proceso se le asigna una zona de memoria de tamaño fijo
 - Suficiente para albergar regiones iniciales
 - Con huecos para permitir cambios de tamaño y añadir nuevas regiones (p.ej. bibliotecas dinámicas o pilas de threads)
 - Difícil asignación adecuada
 - Si es grande se desperdicia espacio, si es pequeña se puede agotar
 - Algoritmos: primer ajuste, mejor ajuste, peor ajuste y siguiente ajuste
- Operaciones:
 - Crear/liberar/cambiar tamaño usan la tabla de regiones para gestionar la zona asignada al proceso
 - Duplicar región requiere crear región nueva y copiar contenido
 - Limitaciones del hardware impiden compartir memoria



Intercambio/swapping (1/2)

¿Qué hacer si no caben todos los programas en mem. principal?

Swap out

- Cuando no caben en memoria todos los procesos activos, se expulsa un proceso de memoria copiando su imagen a *swap*
- Diversos criterios de selección del proceso a expulsar
 - P.ej. Dependiendo de la prioridad del proceso
 - Preferiblemente un proceso bloqueado
 - No expulsar un proceso si tiene una operación de DMA activa

Swap in

- Cuando haya espacio en memoria principal, se lee el proceso a memoria copiando la imagen desde *swap*
- También cuando un proceso lleva un cierto tiempo expulsado
 - En este caso antes de *swap in*, hay *swap out* de otro

Intercambio/swapping (2/2)



Zona de Intercambio o *Swap*: partición (fichero) de disco que almacena imágenes de procesos

- Con preasignación: se asigna espacio al crear el proceso
- Sin preasignación: se asigna espacio al expulsarlo

Con asignación contigua los procesos activos han de residir completamente en MP:

- Grado de multiprogramación depende del tamaño de proc. y MP
- Solución general → Uso de esquemas de memoria virtual paginada



Agenda

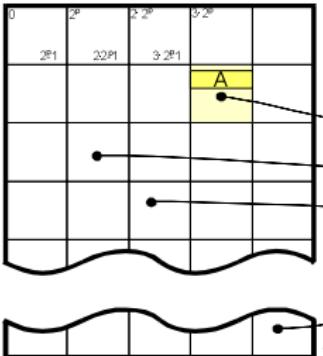
- 1** Modelo de memoria de un proceso
- 2** Servicios POSIX para el manejo de regiones
- 3** Requisitos para la gestión de memoria
- 4** Asignación contigua
- 5** Memoria virtual con Paginación Bajo Demanda
 - Fundamentos de la MV Paginada
 - Políticas de gestión de la MV paginada



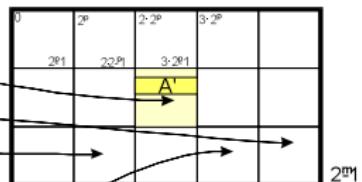
Fundamentos de la MV Paginada (1/2)

- Los procesos (y el SO) generan direcciones virtuales al ejecutar
 - Las dir. virtuales pertenecen al *mapa de memoria virtual*
- El mapa de memoria virtual de un proceso se divide en **páginas**
 - Página: unidad mínima de asignación de MV (Ej. 4KB)
- Parte del mapa de memoria virtual de un proceso está en MP (memoria principal) y parte en disco (*swap* o memoria secundaria)
- El SO se encarga de que estén en memoria principal las páginas *necesarias* del mapa de memoria virtual de cada proceso

MAPA VIRTUAL
(RESIDENTE EN DISCO)



MEMORIA PRINCIPAL



Proyección de página
virtual a memoria física

$n > m$



Fundamentos de la MV Paginada (2/2)

Transferencia de páginas entre swap y MP:

- De M. secundaria a principal: bajo demanda
 - El primer acceso genera un fallo de página
 - El SO copia la página del disco a memoria
- De M. principal a secundaria: por expulsión
 - El SO tiene que hacer sitio

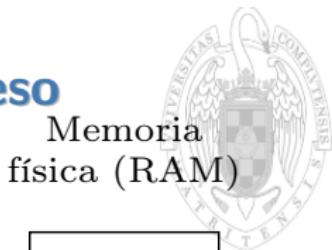
Es eficaz porque los procesos exhiben localidad de referencias:

- Localidad espacial (p.ej. páginas de la región de código.)
- Localidad temporal (p.ej. referencias repetidas a una variable)
- En la práctica, los procesos sólo usan parte de su mapa de memoria en un intervalo de tiempo
 - El SO intenta que las páginas que un proceso esté utilizado (*conjunto de trabajo*) residan en MP (*conjunto residente*)

Beneficios:

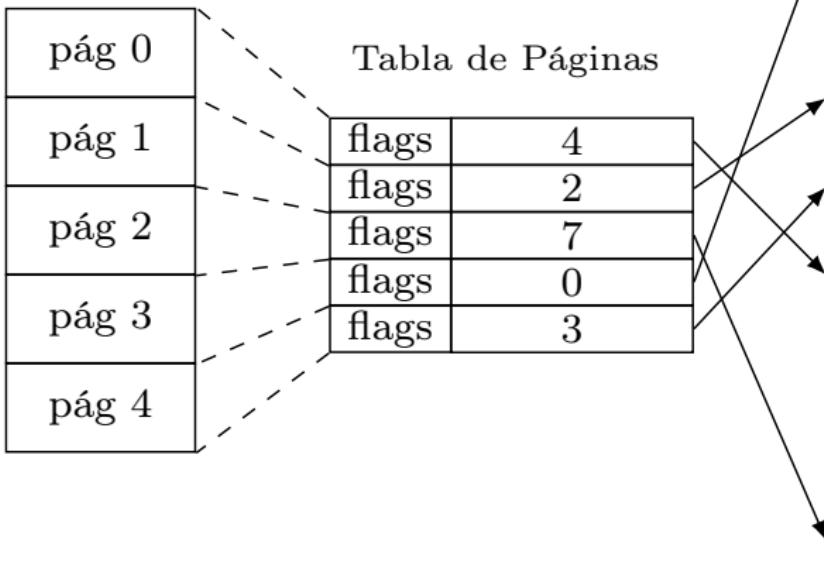
- Aumenta el grado de multiprogramación
- Permite la ejecución de programas que no quepan en MP

Tabla de páginas (TP) de un proceso

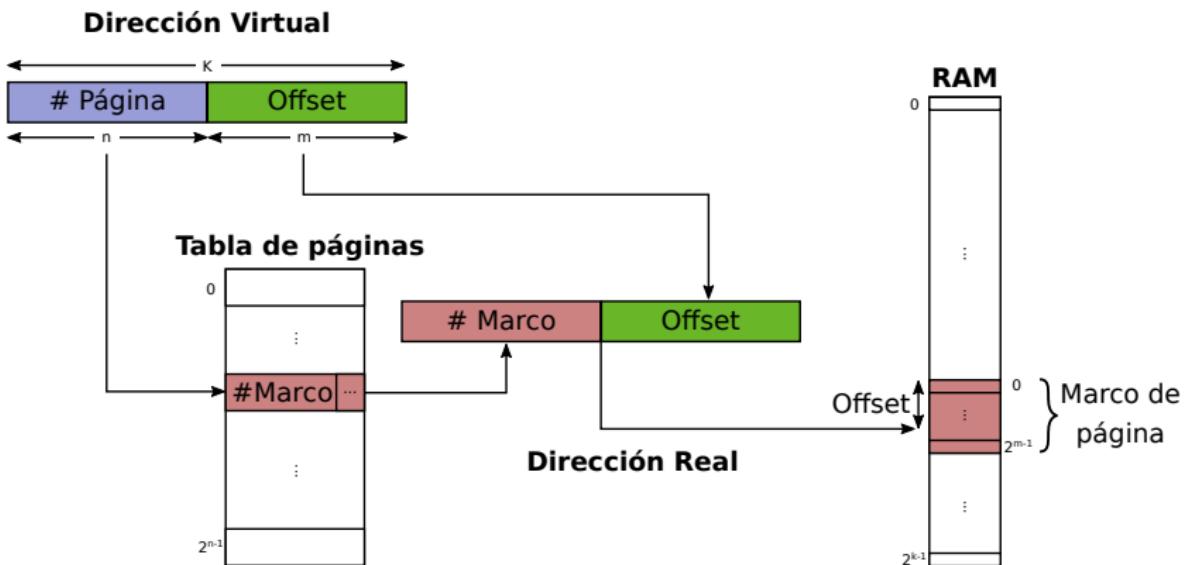


Memoria
física (RAM)

Espacio Virtual
de direcciones
del proceso



Traducción con tabla de páginas



Contenido de entrada de TP



- Número de marco asociado
- Flags
 - Bit de página válida/inválida
 - Indica si la página pertenece a una región del proceso y por tanto es válido referenciarla
 - Si es 0 → *Segmentation Fault*
 - Bit de Presencia
 - Si la página está cargada en RAM o no
 - Si página no presente → Excepción de fallo de página
 - Información de protección: RWX
 - Si operación no permitida → Excepción
 - Información de nivel de privilegio: Usuario/Kernel
 - Bit de página accedida (*Ref*)
 - MMU lo activa cuando se accede a esta página desde que la página se trajo a MP (lo puede borrar el algoritmo de reemplazo)
 - Bit de página modificada (*Mod*)
 - MMU lo activa cuando se escribe en esta página
 - Bit de desactivación de cache



Tamaño de página

Condicionado por diversos factores contrapuestos:

- Potencia de 2 y múltiplo del tamaño del bloque de E/S
- Mejor pequeño por:
 - Menor fragmentación interna
 - Se ajusta mejor al conjunto de trabajo
- Mejor grande por:
 - Tablas más pequeñas
 - Mejor rendimiento en las transferencias entre MP y disco
- Compromiso: entre 2KB y 16KB
 - En SSOO tipo UNIX actuales consultar con: `getconf PAGESIZE`



Fragmentación interna en paginación

- Tendremos *fragmentación interna* si Mem. asignada >Mem. requerida
 - Puede desperdiciarse parte de un marco asignado

T. Páginas Pr. 1

Página 0	Marco 2
Página 1	Marco N

Página M	Marco 3

T. Páginas Pr. 2

Página 0	Marco 4
Página 1	Marco 0

Página P	Marco 1

Memoria

Pág. 1 Pr. 2	Marco 0
Pág. P Pr. 2	Marco 1
Pág. 0 Pr. 1	Marco 2
Pág. M Pr. 1	Marco 3
Pág. 0 Pr. 2	Marco 4

Pág. 1 Pr. 1	Marco N



Páginas de usuario y sistema

El SO proyecta el espacio de direcciones del kernel en el mapa de memoria de cada proceso:

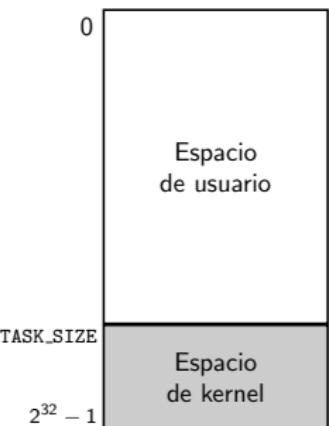
- El mapa de memoria del proceso queda dividido en dos regiones:
 - **Espacio de usuario:** formado por *páginas de usuario* que almacenan código o datos de un proceso
 - **Espacio de kernel:** formado por *páginas de sistema* que almacenan código o datos del SO
- Simplifica el direccionamiento de variables del kernel en las llamadas al sistema

Cuando el proceso ejecuta en modo usuario solo puede referenciar páginas del espacio de usuario

- La MMU genera una excepción de protección si se produce un acceso al espacio de kernel

Cuando el proceso invoca una llamada al sistema pasa a ejecutar código del SO en modo kernel

- Puede entonces acceder a ambos espacios





Valoración de la paginación

- Espacios independientes para procesos:
 - Mediante TP
- Protección:
 - Mediante TP
- Compartir memoria:
 - Entradas de las TPs de dos o más procesos corresponden con mismo marco (Varias dir. virtuales → 1 única dir. física)
- Soporte de regiones:
 - Bits de protección
 - Bit de validez: no se reserva espacio para huecos
- Maximizar utilización de la memoria y soporte de mapas grandes
 - permite mapas de memoria virtual >> cantidad de memoria física
- *Problema:* Mucho mayor gasto en tablas del SO que con asignación contigua
 - Es el precio de mucha mayor funcionalidad



Problemática de las TPs

Eficiencia:

- Cada acceso lógico requiere dos accesos a memoria principal (uno detrás del otro):
 - A la tabla de páginas + al propio dato o instrucción
- Solución: Cache de traducciones (TLB)

Gasto de almacenamiento:

- Tablas muy grandes
 - Ejemplo: páginas 4KB, dir. virtuales de 32 bits y 4 bytes por entrada
 - Tamaño TP: $2^{20} * 4 = 4\text{MB}/\text{proceso}$
- Solución:
 - Tablas multinivel
 - Tablas invertidas



Translation Look-aside Buffer (TLB)

- Memoria asociativa con info. sobre últimas páginas accedidas
 - cache de entradas de TP correspondientes a estos accesos

2 alternativas de diseño:

- 1 Entradas en TLB no incluyen información sobre proceso
 - Exige invalidar TLB en cambios de contexto
- 2 Entradas en TLB incluyen información sobre proceso
 - Registro de CPU debe mantener un ident. de proceso actual

Tratamiento del fallo de página



Tratamiento de excepción

- El HW almacena la dirección de fallo en un registro
- Si dirección inválida → Aborta proceso o le manda señal
- Consulta T. marcos, si no hay ningún marco libre
 - Selección de víctima: pág P marco M
 - Marca P como no presente
 - Si P modificada (bit Mod activo)
 - Inicia escritura P en mem. secundaria
 - Hay marco libre (se ha liberado o lo había previamente):
 - Inicia lectura de página en marco M
 - Marca entrada de página presente referenciando a M
 - Pone M como ocupado en T. marcos (si no lo estaba)
- Fallo de página puede implicar 2 accesos a disco



Políticas de administración de MV

Política de reemplazo:

- ¿Qué página reemplazar si fallo y no hay marco libre?
- Reemplazo local
 - Sólo puede usarse para reemplazo un marco asignado al proceso que causa fallo
- Reemplazo global
 - Puede usarse para reemplazo cualquier marco

Política de asignación de espacio a los procesos:

- ¿Cómo se reparten los marcos entre los procesos?
 - Asignación fija o dinámica



Algoritmos de reemplazo

- Objetivo: Minimizar la tasa de fallos de página
- Cada algoritmo descrito tiene versión local y global:
 - local: criterio se aplica a las páginas residentes del proceso
 - global: criterio se aplica a todas las páginas residentes
- Algoritmos presentados
 - Óptimo
 - FIFO
 - Reloj (o segunda oportunidad)
 - LRU
- Uso de técnicas de *buffering* de páginas



Algoritmo óptimo

- Criterio: se conoce la página residente que tardará más ser accedida
→ Irrealizable
- Interés para estudios analíticos comparativos
- Ejemplo:
 - Memoria física de 4 marcos de página
 - Referencias *a b g a d e a b a d e g d e*

Ref	a	b	g	a	d	e	a	b	a	d	e	g	d	e
0	a	a	a	a	a	a	a	a	a	a	a	g	g	g
1		b	b	b	b	b	b	b	b	b	b	b	b	b
2			g	g	e	e	e	e	e	e	e	e	e	e
3				d	d	d	d	d	d	d	d	d	d	d

6 fallos



Algoritmo FIFO

- Criterio: página que lleva más tiempo residente
- Implementación sencilla:
 - páginas residentes en orden FIFO → se expulsa la primera
 - no requiere el bit de página accedida (Ref)
- No es una buena estrategia:
 - Página que lleva mucho tiempo residente puede seguir accediéndose frecuentemente
 - Su criterio no se basa en el uso de la página
- Ejemplo:
 - Memoria física de 4 marcos de página
 - Referencias $a\ b\ g\ a\ d\ e\ a\ b\ a\ d\ e\ g\ d\ e$

Ref	a	b	g	a	d	e	a	b	a	d	e	g	d	e
0	a_1	a	a	a	a	e_6	e	e	e	e	e	e	d_{13}	d
1		b_2	b	b	b	b	a_7	a	a	a	a	a	a	e_{14}
2			g_3	g	g	g	g	b_8	b	b	b	b	b	b
3				d_5	d	d	d	d	d	d	d	g_{12}	g	g

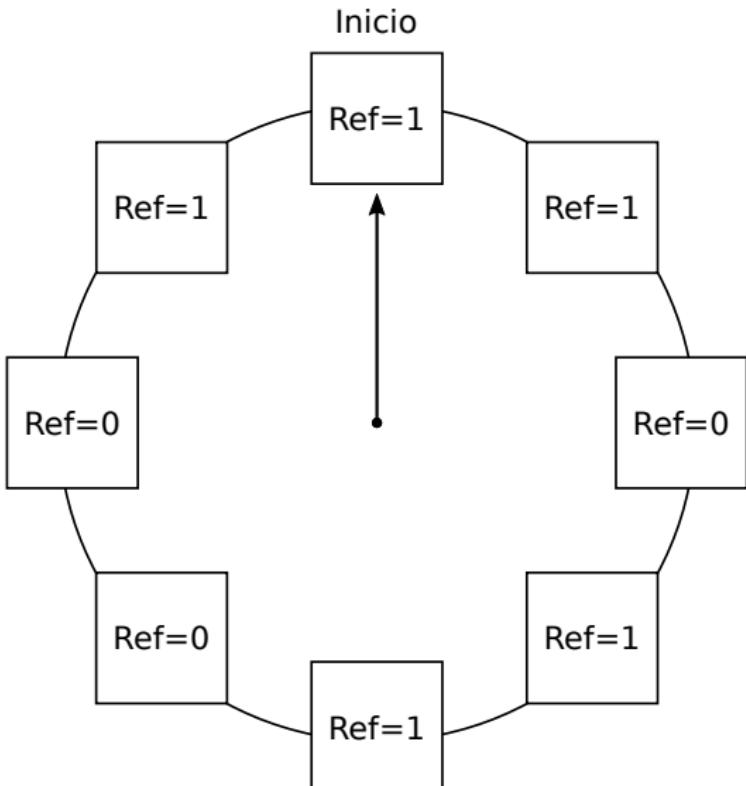
10 fallos

Algoritmo del reloj (o 2^a oportunidad)

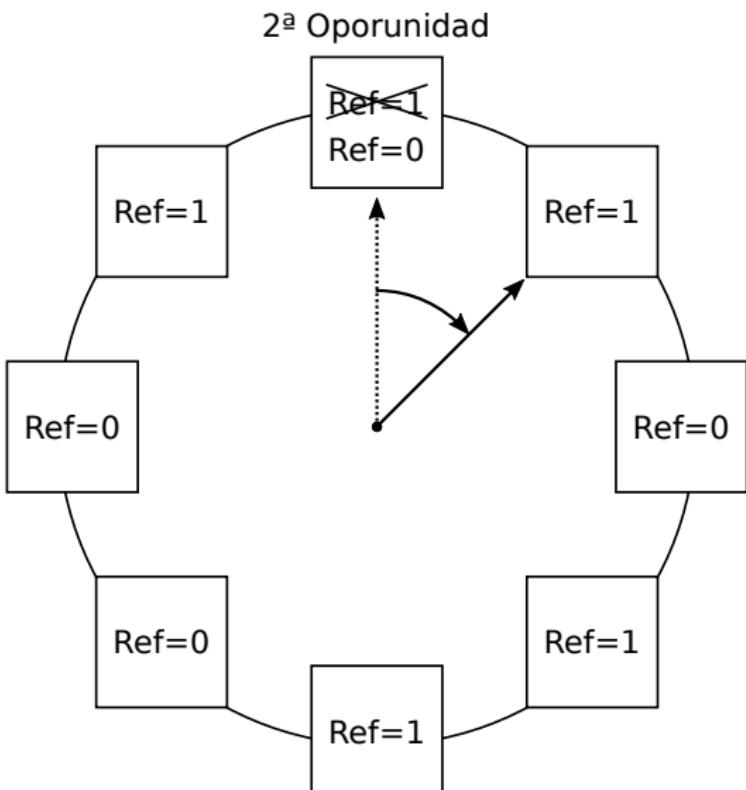


- FIFO + uso de bit de referencia Ref (de página accedida)
- Criterio:
 - Si página elegida por FIFO no tiene activo Ref
 - Es la página expulsada
 - Si lo tiene activo (2^a oportunidad)
 - Se desactiva Ref
 - Se pone página al final de FIFO
 - Se aplica criterio a la siguiente página
- Se puede implementar orden FIFO como lista circular con una referencia a la primera página de la lista:
 - Se visualiza como un reloj donde la referencia a la primera página es la aguja del reloj

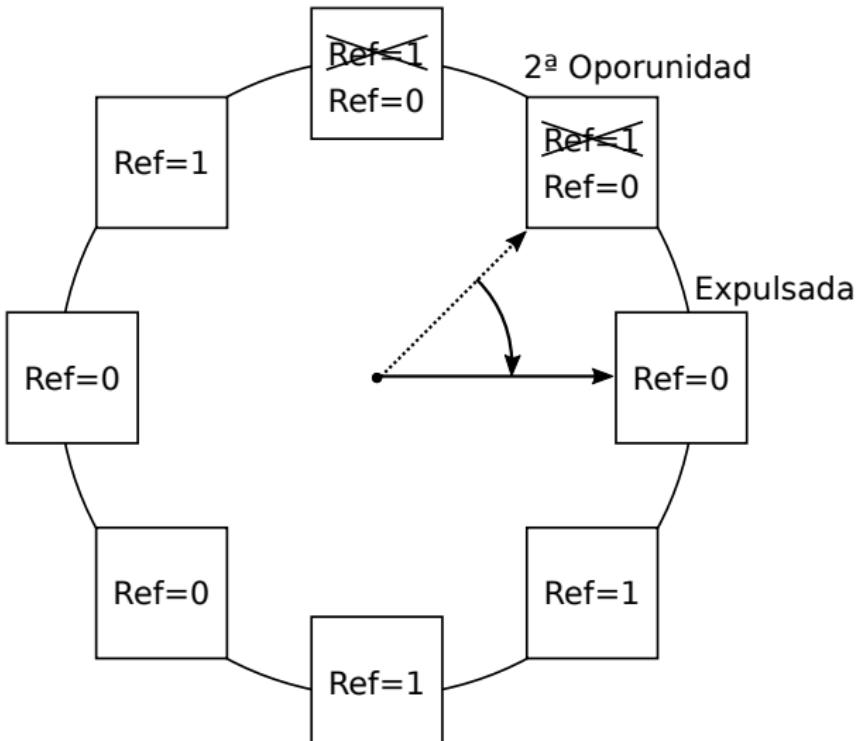
Algoritmo del reloj (o 2^a oportunidad)



Algoritmo del reloj (o 2^a oportunidad)



Algoritmo del reloj (o 2^a oportunidad)





Algoritmo del reloj (o 2^a oportunidad)

■ Ejemplo:

- Memoria física de 4 marcos de página
- Referencias $a\ b\ g\ a\ d\ e\ a\ b\ a\ d\ e\ g\ d\ e$

Ref	a	b	g	a	d	e	a	b	a	d	e	g	d	e
0	$\downarrow a_0$	$\downarrow a_0$	$\downarrow a_0$	$\downarrow a_1$	$\downarrow a_1$	a_0	a_0	a_1	a_1	a_1	a_1	a_0	a_0	a_0
1		b_0	b_0	b_0	b_0	$\downarrow b_0$	e_0	e_0	e_0	e_0	e_0	e_1	e_0	e_0
2			g_0	g_0	g_0	g_0	$\downarrow g_0$	$\downarrow g_0$	b_0	b_0	b_0	g_0	g_0	g_0
3				d_0	d_0	d_0	d_0	$\downarrow d_0$	$\downarrow d_0$	$\downarrow d_0$	$\downarrow d_1$	$\downarrow d_0$	$\downarrow d_1$	$\downarrow d_1$

7 fallos



Algoritmo LRU

- Criterio: página residente que fué accedida hace más tiempo (*menos recientemente*)
- Por proximidad de referencias:
 - Pasado reciente condiciona futuro próximo
- Sutileza:
 - En su versión global: accedida hace más tiempo en la escala lógica de cada proceso
- Difícil implementación estricta (hay aproximaciones):
 - Precisaría una MMU específica
- Posible implementación con HW específico:
 - La MMU mantiene un contador que se incrementa cada cierto tiempo
 - En entrada de TP hay una marca de tiempo (*timestamp*)
 - En cada acceso a memoria MMU copia contador del sistema a entrada referenciada
 - Reemplazo: Página con *timestamp* más bajo



Algoritmo LRU

■ Ejemplo:

- Memoria física de 4 marcos de página
- Referencias $a\ b\ g\ a\ d\ e\ a\ b\ a\ d\ e\ g\ d\ e$

Ref	a	b	g	a	d	e	a	b	a	d	e	g	d	e
0	a_1	a	a	a_4	a	a	a_7	a	a_9	a	a	a	a	a
1		b_2	b	b	b	e_6	e	e	e	e	e_{11}	e	e	e_{14}
2			g_3	g	g	g	b_8	b	b	b	g_{12}	g	g	
3				d_5	d	d	d	d	d_{10}	d	d	d_{13}	d	

7 fallos



Buffering de páginas

Mantener una reserva de marcos libres

- Fallo de página: siempre usa marco libre, no hay reemplazo

Si el número de marcos libres baja de un umbral entra el *Demonio de paginación* que aplica repetidamente el algoritmo de reemplazo:

- páginas no modificadas pasan a lista de marcos libres
- páginas modificadas pasan a lista de marcos modificados
 - cuando se escriban a disco pasan a lista de libres
 - pueden escribirse en tandas (mejor rendimiento)

Si se referencia una página mientras está en estas listas el fallo de página la recupera directamente de la lista, evitando el acceso a disco

- puede arreglar el comportamiento de algoritmos “malos”



Retención de páginas en memoria

- Páginas marcadas como no reemplazables
- Se aplica a páginas del propio SO
 - SO con páginas fijas en memoria es más sencillo
- También se aplica mientras se hace DMA sobre una página
- Algunos sistemas ofrecen a aplicaciones un servicio para fijar en memoria una o más páginas de su mapa
 - Adecuado para procesos de tiempo real
 - Puede afectar al rendimiento del sistema
 - En POSIX servicio `mlock()`



Estrategia de asignación fija

- Número de marcos asignados al proceso (conjunto residente) es constante
- Puede depender de características del proceso:
 - tamaño, prioridad,...
- No se adapta a las distintas fases del programa
- Comportamiento relativamente predecible
- Sólo tiene sentido usar reemplazo local
- Arquitectura impone nº mínimo: al menos una página de código, una de datos, y una de pila

Estrategia de asignación dinámica



- Número de marcos varía dependiendo de comportamiento del proceso (y posiblemente de los otros procesos)
- Asignación dinámica + reemplazo local
 - Proceso va aumentando o disminuyendo su conjunto residente dependiendo de su comportamiento
 - Comportamiento relativamente predecible
- Asignación dinámica + reemplazo global
 - Procesos se quitan las páginas entre ellos
 - Comportamiento difícilmente predecible

Hiperpaginación (Thrashing)



- Tasa excesiva de fallos de página de un proceso o en el sistema
- Con asignación fija: Hiperpaginación en P_i
 - Si conjunto residente de $P_i <$ conjunto de trabajo P_i
- Con asignación variable: Hiperpaginación en el sistema
 - Si nº marcos disponibles $< \Sigma$ conjuntos de trabajo de todos
 - Grado de utilización de CPU cae drásticamente
 - Procesos están casi siempre en colas de dispositivo de paginación
 - Solución (similar al *swapping*): Control de carga
 - Disminuir el grado de multiprogramación
 - Suspender 1 o más procesos liberando sus páginas residentes
- Problema: ¿Cómo detectar esta situación?

Estrategia del conjunto de trabajo

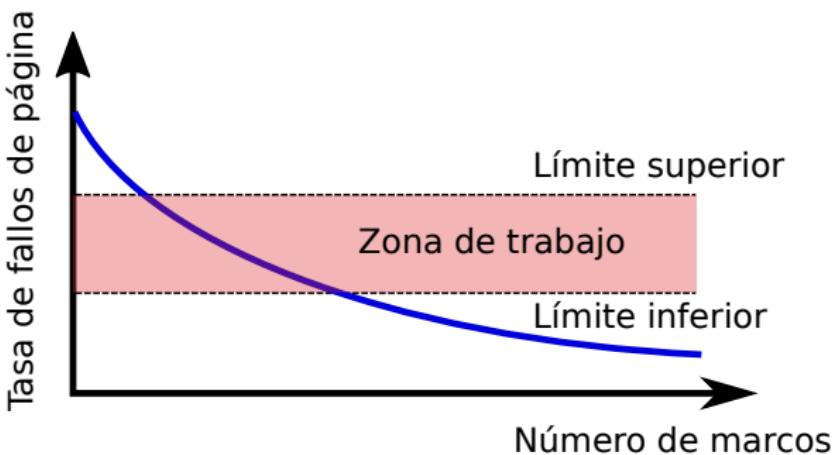


- Intentar conocer el conjunto de trabajo de cada proceso
 - Páginas usadas por el proceso en las últimas N referencias
- Si conjunto de trabajo decrece se liberan marcos
- Si conjunto de trabajo crece se asignan nuevos marcos
 - Si no hay disponibles: suspender proceso(s)
 - Se reactivan cuando hay marcos suficientes para c. de trabajo
- Asignación dinámica con reemplazo local
- Difícil implementación estricta
 - Precisaría una MMU específica
- Se pueden implementar aproximaciones:
 - Estrategia basada en frecuencia de fallos de página (PFF):
 - Controlar tasa de fallos de página de cada proceso



Estrategia basada en frecuencia de fallos

- Si tasa < límite inferior se liberan marcos aplicando un algoritmo de reemplazo
- Si tasa > límite superior se asignan nuevos marcos
 - Si no marcos libres se suspende algún proceso



Control de carga y reemplazo global



- Algoritmos de reemplazo global no controlan hiperpaginación
 - ¡Incluso el óptimo!
 - Necesitan cooperar con un algoritmo de control de carga
- Ejemplo: UNIX 4.3 BSD
 - Reemplazo global con algoritmo del reloj
 - Variante con dos “manecillas”
 - Uso de *buffering* de páginas
 - “demonio de paginación” controla nº de marcos libres
 - Si número de marcos libres < umbral
 - “demonio de paginación” aplica reemplazo
 - Si se repite con frecuencia la falta de marcos libres:
 - Proceso “swapper” suspende procesos