Sistemas operativos
2ª edición

Capítulo 5
Gestión de memoria
(extracto de las transparencias del libro)

Contenido

- Introducción
- Aspectos generales de la gestión de memoria
- Modelo de memoria de un proceso
- Esquemas de gestión de la memoria del sistema
- Memoria virtual

Introducción

- SO multiplexa recursos entre procesos
  - Gestión de procesos: Reparto de procesador
  - Gestión de memoria: Reparto de memoria
- Gestión integral de la memoria: no sólo SO
  - Compilador, montador y hardware de gestión de memoria
- Gestor de memoria: elevada complejidad

Aspectos generales de la gestión de memoria

- Niveles de gestión de memoria
- Objetivos del sistema de gestión de memoria
- El problema general de la asignación de memoria

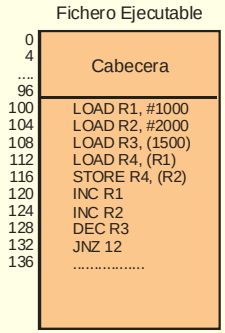
Niveles de gestión de memoria

- **Nivel de procesos**
  - Reparto de memoria entre procesos
- **Nivel de regiones**
  - Reparto de memoria del proceso entre regiones
- **Nivel de zonas** (si aplicable)
  - Reparto de espacio de región entre sus zonas
    - No gestionado por sistema operativo
    - Por ejemplo, región de *heap*

Objetivos del sistema de gestión de memoria

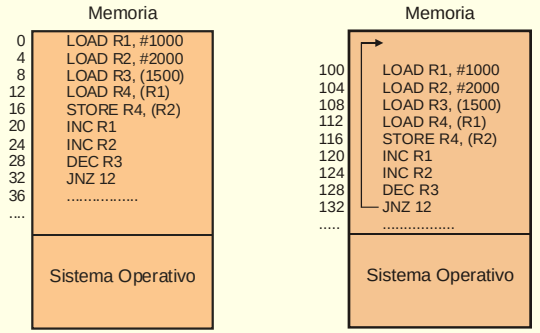
- Necesidades de los programas y del SO
  - Espacios lógicos independientes
  - Protección
  - Compartir memoria
  - Aprovechamiento del espacio de memoria
  - Soporte de regiones

Espacio lógico independiente



- Código en ejecutable incluye referencias entre 0 y N

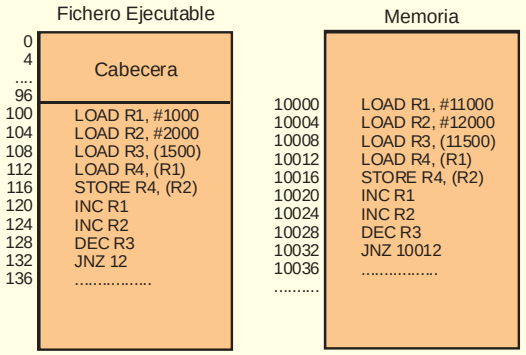
Ejecución en SO mono y multiprogramado



Reubicación y protección

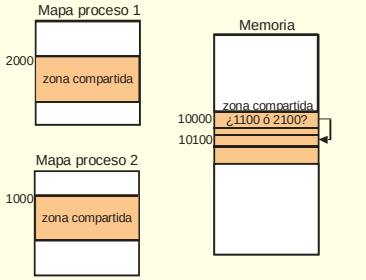
- **Reubicar**: Traducir direcciones lógicas a físicas
- **Reubicación en nivel de procesos**
  - Crea espacio lógico independiente para proceso
- Dos alternativas:
  - **Reubicación software**: previa a la ejecución del proceso
  - **Reubicación hardware**: en tiempo de ejecución
- ¿Es necesaria reubicación de direcciones en SO?
  - Uso de reubicación proporciona más flexibilidad
- SO no sólo requiere utilizar su mapa sino toda la memoria
  - Además, necesita "ver" espacio lógico de cada proceso
- **Protección**: Aislamiento del SO y de procesos entre sí
  - Necesita apoyo hardware

Reubicación software en la carga



Compartimiento de memoria para comunicación

- Dir. lógicas de procesos corresponden con misma dir. física



Problema de las autorreferencias

Compartimiento de memoria para optimizar su uso

- Compartir memoria permite mejor aprovechamiento
  - Compartir código de programas o de bibliotecas
- Datos de programa y de biblioteca no deben compartirse
  - Pero inicialmente idénticos
  - Diferir copia de cada dato hasta que se modifique (COW)
  - También aplicable a `fork`

Buen aprovechamiento de la memoria

- Todo byte debería almacenar información de utilidad pero...
  - Desperdicio inherente a la propia gestión (**fragmentación**)
  - Gasto de propia gestión de memoria (estructuras de datos)
- Mejor aprovechamiento → mayor grado de multiprogramación
  → mejor rendimiento
- Para mejorar rendimiento, uso de memoria virtual

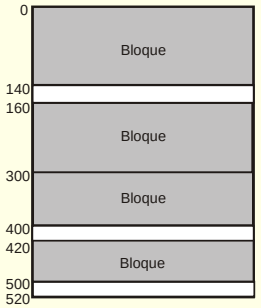
Soporte de regiones

- Mapa de proceso no homogéneo
  - Conjunto de regiones con distintas características
- Mapa de proceso dinámico
  - Regiones cambian de tamaño, se crean y destruyen
  - Huecos en el mapa del proceso

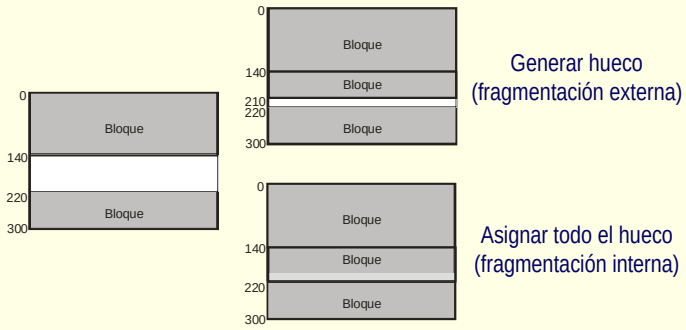
El problema general de la asignación de memoria

- Planteamiento del problema general
  - Espacio de almacenamiento de N bytes
  - Peticiones de espacio contiguo de diversos tamaños
  - Cuando ya no se necesita espacio, se libera
  - Según evoluciona: bloques y huecos
  - Objetivos: buen aprovechamiento (fragmentación) y eficiencia
- Cada nivel es un caso independiente del problema general
  - En cada nivel se puede usar esquema diferente
  - Un nivel reparte espacio proporcionado por nivel superior

Fragmentación



Fragmentación externa versus interna



Algoritmo de asignación de espacio

- El mejor ajuste (***best fit***).
  - Selección: comprobar todos u ordenados por tamaño
- El peor ajuste (***worst fit***)
  - Selección: comprobar todos u ordenados por tamaño
- El primero que ajuste (***first fit***)
  - Suele ser la mejor política en muchas situaciones
- El próximo que ajuste (***next fit***)
  - Variación del primero que ajuste

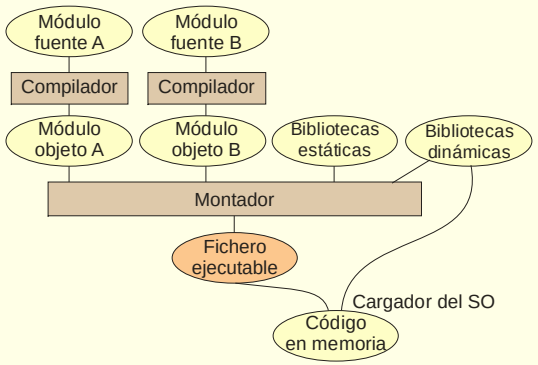
Gestión de la información de estado

- Información sobre bloques y huecos almacenada:
  - Internamente o externamente
- Múltiples soluciones (no tratadas en esta exposición)
  - Lista única
  - Múltiples listas con huecos de tamaño variable
  - Múltiples listas con particiones estáticas
  - Sistema *buddy* binario
  - Mapa de bits

Modelo de memoria de un proceso

- Ciclo de vida de un programa
- Mapa de memoria de un proceso

Ciclo de vida de un programa

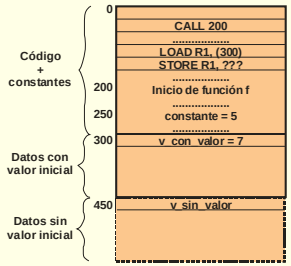


Compilación

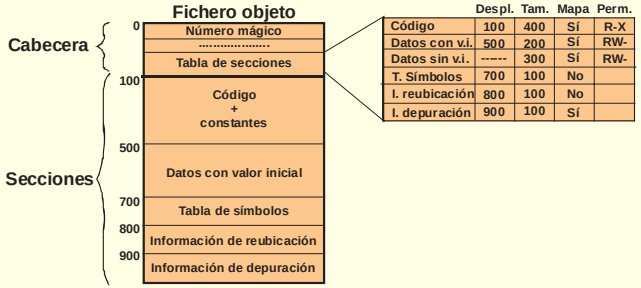
- Genera código y calcula tamaño datos estáticos
  - Tres secciones: código+constantes; datos con y sin v. inicial
- Asigna direcciones a datos estáticos definidos en el módulo
- Resuelve referencias a símbolos estáticos (código o datos)
  - Si están definidos en el módulo
  - Si externos, pendiente; Montador resolverá
- Otras secciones:
  - Info. de reubicación y depuración + Tabla de símbolos

Construcción de código objeto

```
const int constante = 5;
static int v_con_valor = 7;
int v_sin_valor;
extern int v_externa;
int main() {
  f(5); /* CALL /200 */
  v_externa = v_con_valor;
     /* LOAD R1, /300
        STORE R1, ??? */
  ...................
void f(int x) {
  ...................
```



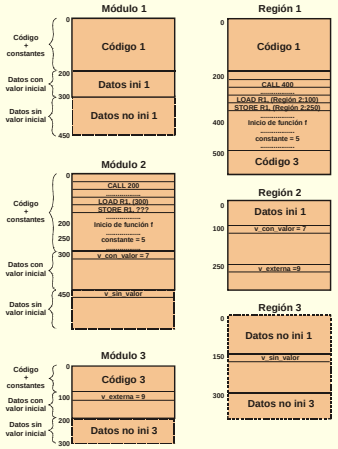
Formato de objeto



	Despl.	Tam.	Mapa	Perm.
Código	100	400	Sí	R-X
Datos con v.i.	500	200	Sí	RW-
Datos sin v.i.	-----	300	Sí	RW-
T. Símbolos	700	100	No	
I. reubicación	800	100	No	
I. depuración	900	100	Sí	

Montaje

- Termina resolución símbolos
- Ejecutable: unión de objetos
- Agrupa secciones similares
  - N secciones → 1 región
- Reubicación de módulos
  - dir. módulo → dir. región



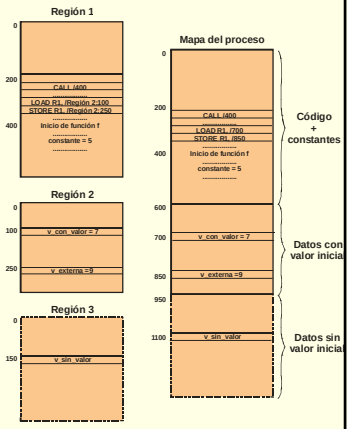
Reubicación de regiones

Montaje realiza reubicación regiones

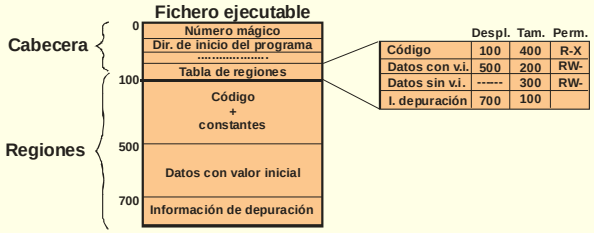
- dir. región → dir. proceso

Excepto si MMU usa segmentación:

- 1 región → 1 segmento



Formato del ejecutable



	Despl.	Tam.	Perm.
Código	100	400	R-X
Datos con v.i.	500	200	RW-
Datos sin v.i.	-----	300	RW-
I. depuración	700	100	

Carga y ejecución

- Falta reubicación en nivel de procesos:
  - dir. dentro de mapa de proceso → dir. física
    - Puede hacerlo el hardware
    - O el software: reubicación en la carga
- Si MMU usa segmentación:
  - Hardware reubica regiones y procesos

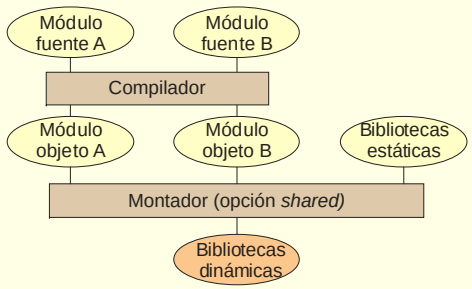
Bibliotecas

- Biblioteca: colección de módulos objeto relacionados
- Bibliotecas de SO, de cada lenguaje, creadas por usuario...
- Usuario especifica bibliotecas requeridas en montaje
- Si programa referencia símbolo en biblioteca
  - Montador extrae objeto(s) requerido(s) de biblioteca
- Una vez extraídos objetos requeridos, montaje convencional
- Desventajas del montaje estático de bibliotecas:
  - Ejecutables grandes
  - Código de biblioteca repetido en ejecutables y memoria
  - Actualización de biblioteca implica volver a montar

Bibliotecas dinámicas

- Solución: *Dynamically Linked Library* (biblioteca dinámica)
  - Carga y montaje en tiempo de ejecución
  - Transparente y con sobrecarga tolerable
- Generada también por el montador
  - Reubicación de módulos resuelta (organizada en regiones)
  - Pendiente resolución de símbolos y reubicación
- Montaje de programa que usa biblioteca
  - No se realiza resolución ni se incluye código de biblioteca
  - En ejecutable: anota su uso e incluye cargador/montador

Generación de una biblioteca dinámica



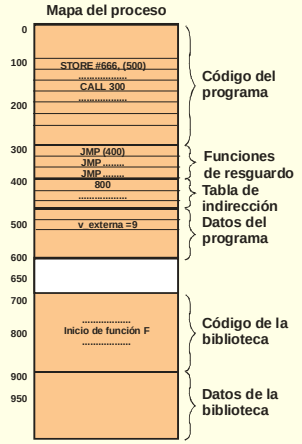
Implementación de bibliotecas dinámicas

- Dos aspectos a resolver:
  - Referencias a símbolos de biblioteca
    - Desde programa o desde otra biblioteca dinámica
  - Cuándo cargar/montar la biblioteca dinámica
- Primero ¿ya resuelto?: montaje en ejecución
  - **Pero modifica código de programa y biblio: ~~compartir~~**

Implementación de bibliotecas dinámicas

- Resolución símbolos y reubicación de regiones en t. ejecución
  - Permite técnicas sofisticadas como interposición
- Uso de código PIC (independiente de la posición) en biblioteca
- Direccionamiento indirecto a través de tabla
  - En ejec. resolución de símbolos externos: actualiza tabla
  - Compilador no sabe si se precisa indirección en refer. ext.
    - Referencia a función de biblioteca: resguardo
    - Referencia a variable de biblioteca: más complicado
      - Desde otra biblioteca: uso de acceso indirecto a través de tabla
      - Desde programa: variable se incluye en región datos de programa
- Resolución t. de ejecución puede ser ineficiente:
  - Preenlazado: se rellena a priori tabla de indirección

Programa que usa bib. dinámicamente montada



Tiempo de carga y montaje

- Solución inmediata:
  - Carga y montaje de todas bibliotecas antes de empezar
- Solución perezosa:
  - Esperar primer acceso a símbolo para cargar/montar
  - Si símbolo es función: valor inicial, llamada a cargador
  - Si símbolo es dato: ¿Cómo enterarse?
- Solución intermedia:
  - En inicio se cargan bibliotecas y se resuelven ref. a datos
  - Referencias a funciones igual que en solución perezosa

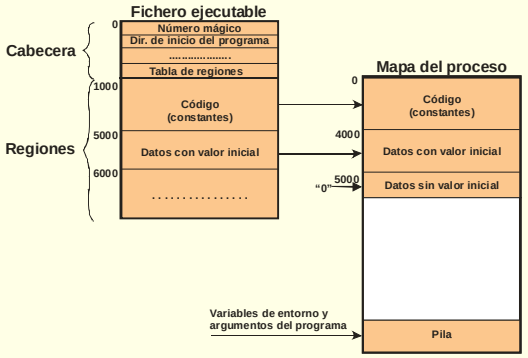
Ventajas de bibliotecas dinámicas

- Ventajas: las desventajas de las estáticas
  - Cuidado con actualización automática: Uso de versiones
- Desventajas:
  - Ejecutable no autocontenido (“El infierno de las DLL”)
  - Menos eficiente
- Montaje explícito de bib. dinámicas:
  - No se especifica biblioteca en mandato de montaje
  - Programa solicita carga de bib. mediante servicio del sistema
  - Acceso “no” transparente a símbolos de la biblioteca
  - Permite carga dinámica de código

Mapa de memoria de un proceso

- Evolución del mapa de memoria del proceso: Nivel de regiones
- Mapa de memoria o imagen del proceso: conjunto de regiones
  - Uso de tabla de regiones
- Región: zona contigua tratada como unidad
  - dirección de comienzo y tamaño inicial
  - soporte: En fichero o sin soporte (anónima)
  - protección: RWX
  - uso compartido o privado
  - tamaño fijo o variable
- Ejecución de programa: Crea mapa a partir de ejecutable
  - Regiones de mapa inicial → Regiones de ejecutable

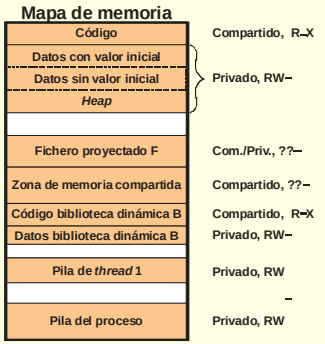
Crear mapa desde ejecutable



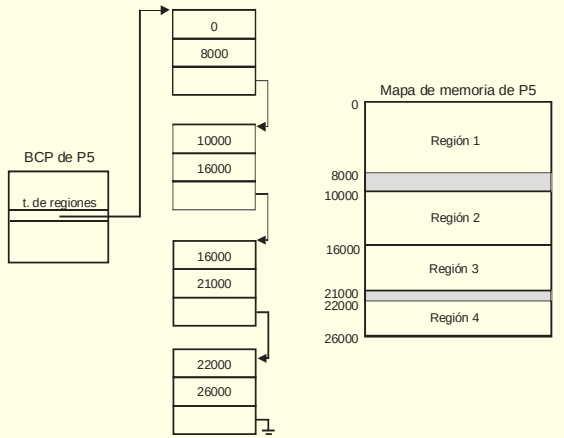
Regiones del mapa de memoria

Región	Soporte	Protección	Comp/Priv	Tam	Ubicación
Código	Fichero	RX	Compartida	Fijo	Prefijada
Dat. con v.i.	Fichero	RW	Privada	Fijo	Prefijada
Dat. sin v.i.	Sin soporte	RW	Privada	Fijo	Prefijada
Pilas	Sin soporte	RW	Privada	Var	Cualquiera
Heap	Sin soporte	RW	Privada	Var	Cualquiera
F. Proyect.	Fichero	por usuario	Comp./Priv.	Fijo	Cualquiera
M. Comp.	Sin soporte	por usuario	Compartida	Fijo	Cualquiera
Bib.dinám.	Regiones para código y datos (con y sin valor inicial)				

Mapa de memoria de un proceso hipotético



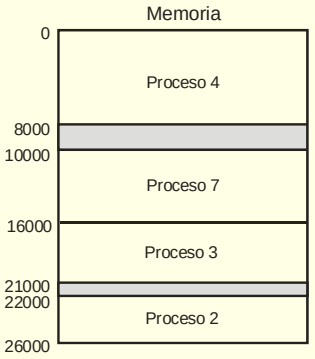
Implementación de la tabla de regiones



Esquemas de gestión de la memoria del sistema

- Nivel de gestión de procesos
  - ¿Cómo se reparte la memoria entre mapas de los procesos?
  - Muy ligado al hardware de gestión de memoria
- Esquemas de gestión analizados:
  - Asignación contigua
  - Segmentación
  - Paginación
  - Segmentación paginada

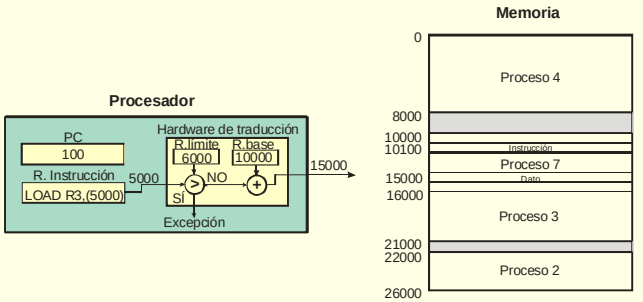
Asignación contigua



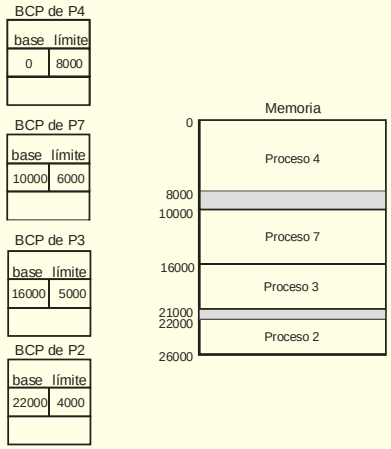
- Deficiencias de asignación contigua
  - No soporte regiones, no compartir, no base de m. virtual

Registros base y límite

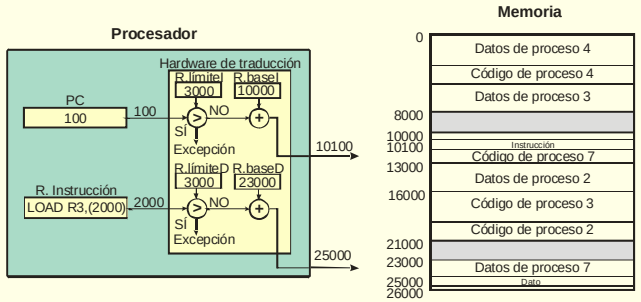
- Protección + reubicación de procesos por hardware



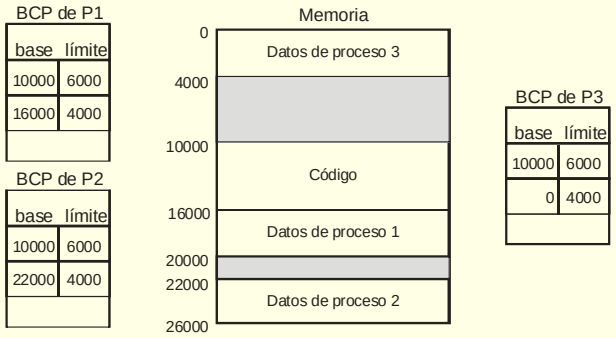
Creación de espacio lógico independiente



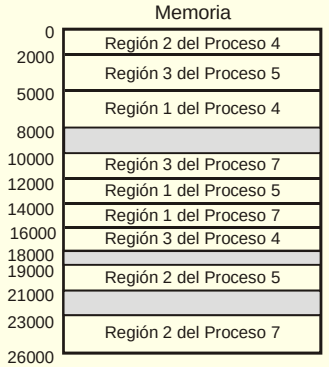
Espacio de instrucciones y de datos separados



Compartir con espacio de I/D separados



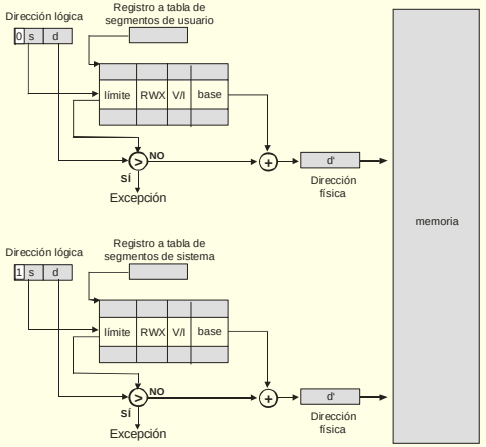
Segmentación



Segmentación: Aspectos hardware

- Esquema que intenta dar soporte directo a las regiones
- Generalización de registro base y límite: 1 pareja/segmento
- Dirección lógica: núm. de segmento + dirección en segmento
- MMU usa una tabla de segmentos (TS)
- Entrada de TS contiene (entre otros):
  - r. base y límite del segmento
  - protección: RWX + bit de validez
- Dir. lógicas de usuario y de sistema (p.e. empiezan por 0 ó 1)

Esquema de traducción con segmentación



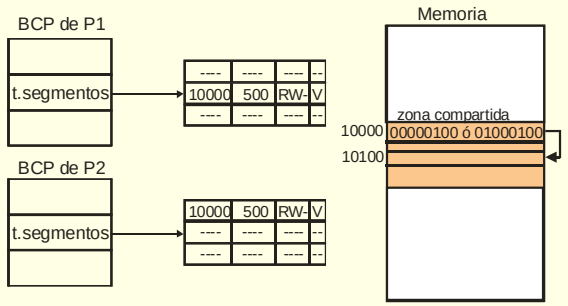
Creación de espacio lógico independiente

- SO mantiene TSU por proceso (t. de regiones)
  - En c. contexto notifica a MMU qué TSU debe usar
- SO mantiene una única TSS que no cambia
  - Procesos comparten mapa del sistema operativo
  - SO interpreta directamente direcciones de usuario de p. actual
- Segmentación: reubicación (doble) de regiones y de procesos

Soporte de regiones, compartir y m. virtual

- Soporte directo de regiones (punto fuerte)
- Compartir: directo (región = segmento)
  - 2 entradas iguales en 2 TSU
  - Pero sigue habiendo problemas con autorreferencias
- Soporte de memoria virtual
  - No adecuado para m. virtual por tamaño variable de segmentos
- Nivel de procesos y regiones fusionado
  - Espacio de regiones se busca en m. del sistema no en mapa

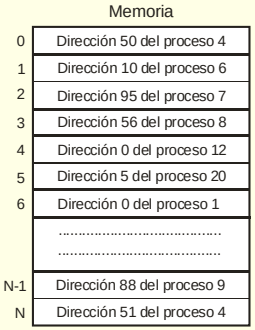
Problemas al compartir una región



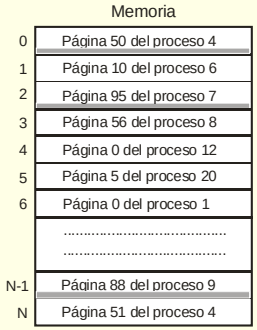
Paginación: Fundamento

- Asignación contigua o segmentación:
  - Mal aprovechamiento por frag. externa
- Óptimo es irrealizable
- Paginación: cambio de escala de byte a página
  - Cualquier página de proceso en cualquier marco de página
  - Peor aprovechamiento (f. interna) pero t. de traducción menor
  - Asignación no contigua: Reubicación no lineal
- Función de reubicación: tabla de páginas

Aprovechamiento óptimo



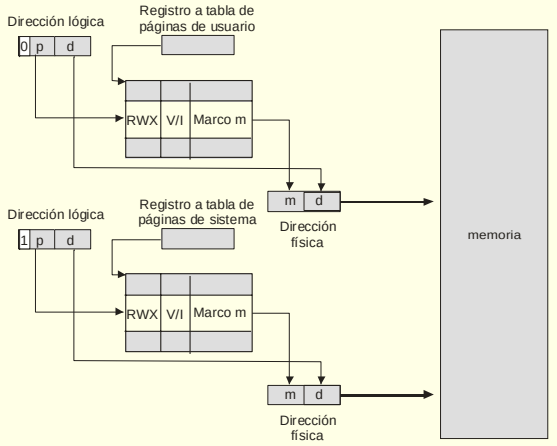
Aprovechamiento con paginación



Paginación. Aspectos hardware

- Mapa de memoria del proceso dividido en páginas
- Memoria principal dividida en marcos (tam. marco=tam. página)
- Tabla de páginas (TP): Asocia página y marco que la contiene
- Normalmente espacio lógico ≥ físico (bits de p ≥ bits de m)
- Tabla de páginas única (bit *S*) vs. 2 tablas de páginas separadas
  - Usamos 2 tablas en los ejemplos por sencillez

Esquema de traducción con TP usuario y sistema



Contenido de entrada de TP

- Número de marco asociado
- Información de protección: RWX
- Bit de página válida/inválida
- Bit de página accedida (*Ref*)
- Bit de página modificada (*Mod*)
- Bit de desactivación de caché (para direcciones de E/S)
- Entrada de sistema (*S*): Presente en MMU con TP única
- Indicador de superpágina

Tamaño de la tabla de páginas

- Condicionado por diversos factores contrapuestos:
  - Potencia de 2 y múltiplo de sector de disco
  - Compromiso (entre 1K y 16K)
  - Pequeño: Menor f. interna, mejor ajuste a conjunto de trabajo
  - Grande: Tablas más pequeñas, mejor rendimiento de disco
- Lo fija el procesador
  - Algunos permiten configurar distintos tamaños
  - Algunos implementan superpáginas
    - Por ejemplo: superpáginas de 4M y páginas de 4K

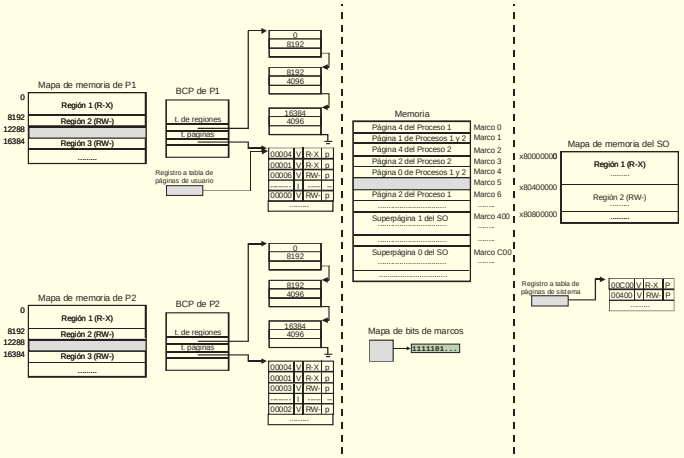
Creación de espacio lógico independiente

- SO mantiene una TP por cada proceso
  - En c. contexto notifica a MMU cuál debe usar
- Con TP separadas, SO mantiene una única TP para propio SO
- Proceso modo sistema acceso directo a su mapa y al de SO
  - Procesos comparten mapa del sistema operativo
  - SO interpreta directamente direcciones de usuario de p. actual
  - SO usa asociaciones temporales para acceso a resto memoria

Soporte de regiones, compartir y m. virtual

- No soporte directo de regiones pero fácil conseguirlo
  - Región ocupa número entero de páginas
  - SO mantiene tabla de regiones por cada proceso
  - Protección de región: uso de bits de protección de sus páginas
  - No reserva espacio para huecos: bit de validez desactivado
  - También soporte de regiones de SO (uso de superpáginas)
- Compartir región:
  - Entradas corresponden a mismo marco (contador de refs.)
- Permite esquemas de memoria virtual
  - Unidad de transferencia: página (tamaño fijo)
  - Uso de bit validez: página no residente con bit desactivado

Visión global de la paginación



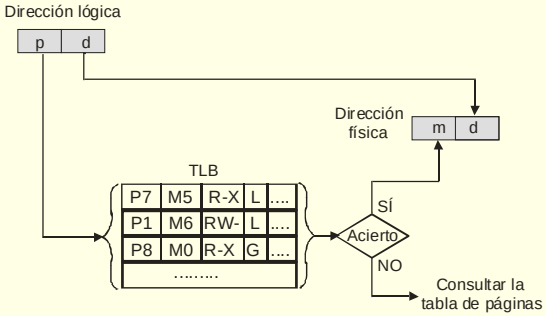
Implementación de TP

- TP se mantiene normalmente en memoria principal
- 2 problemas: eficiencia y gasto de almacenamiento
- Eficiencia:
  - Cada acceso lógico requiere 2 accesos a memoria principal
  - Solución: *caché* de traducciones → TLB
- Gasto de almacenamiento: Tablas muy grandes
  - Ejemplo: páginas 4K, dir. lógica 32 bits y 4 bytes/entrada
    - Tamaño TP: 2^{20} *4 = 4MB/proceso
  - Solución: tablas multinivel y tablas invertidas

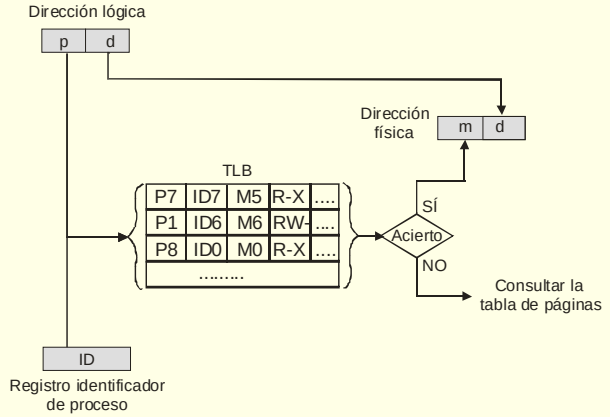
Translation Look-aside Buffer (TLB)

- Memoria asociativa con info. sobre últimas páginas accedidas
- Entradas en TLB no incluyen información sobre proceso
  - Invalida TLB en c. contexto (no entradas de sistema)
- Entradas en TLB incluyen información sobre proceso
  - Registro UCP mantiene ID de p. actual:
    - No invalidar excepto si se reutilizan
- Gestionada por MMU: Si fallo usa la TP en memoria
  - MMU activa *Mod* y *Ref* en TP
- "Casi" transparente al SO
  - Invalidar, si no PID, y coherencia en memoria virtual

TLB sin información de proceso



TLB con información de proceso



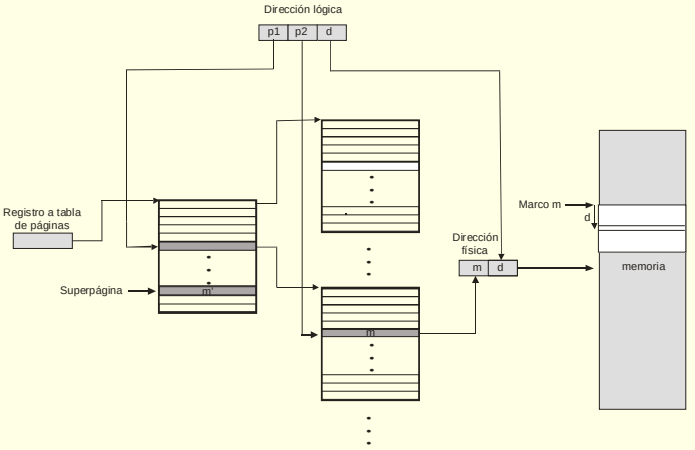
TLB gestionada por software

- Alternativa: traspasar al SO parte del trabajo de traducción
- MMU no usa tablas de páginas, sólo consulta TLB
- SO mantiene TPs que son independientes del HW
- Fallo en TLB → Activa SO
- SO se encarga de:
  - Buscar "a mano" en TP la traducción
  - Rellenar TLB con la traducción
  - Propagar bits *Ref* y *Mod* a TP
- Flexibilidad en diseño de SO pero menor eficiencia

Tablas de páginas multinivel

- Fundamento
  - Tablas de páginas muy grandes con muchas entradas nulas
  - Fragmentar tabla y acceder mediante tabla maestra
- Tablas de páginas organizadas en M niveles:
  - Entrada de TP de nivel K apunta a TP de nivel K+1
  - Entrada de último nivel apunta a marco de página
- Dirección lógica especifica la entrada a usar en cada nivel:
  - 1 campo por nivel + desplazamiento
- Si todas las entradas de una TP son inválidas
  - No se almacena esa TP e inválida entrada de TP superior

Esquema de traducción con 2 niveles



Ventajas de tablas de páginas multinivel

- Si proceso usa una parte pequeña de su espacio lógico
  - Ahorro en espacio para almacenar TPs
- Ejemplo: Proceso que usa 12MB superiores y 4MB inferiores
  - 2 niveles, pág. 4K, dir. lógica 32 bits (10 bits/nivel), 4B/entrada
    - Tamaño: 1 TP N_1 + 4 TP N_2= 5 * 4KB = 20KB (frente a 4MB)
- Ventajas adicionales:
  - Permite compartir TPs intermedias
  - Sólo se requiere que esté en memoria la TP de nivel superior
- Facilita implementación de superpáginas
  - Entrada de primer nivel de superpágina apunta a m. física

Ventajas de tablas de páginas multinivel

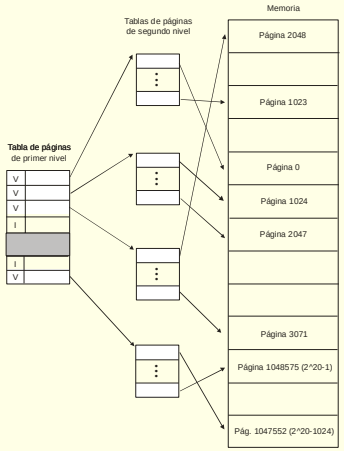
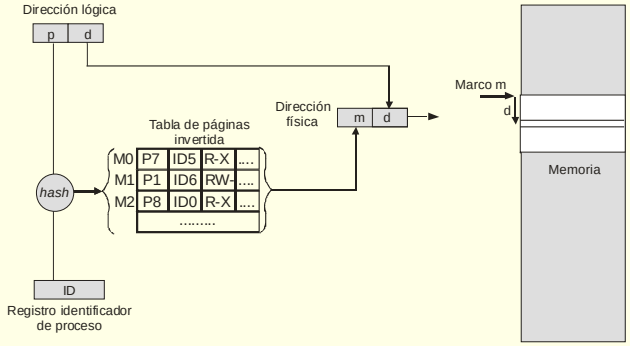


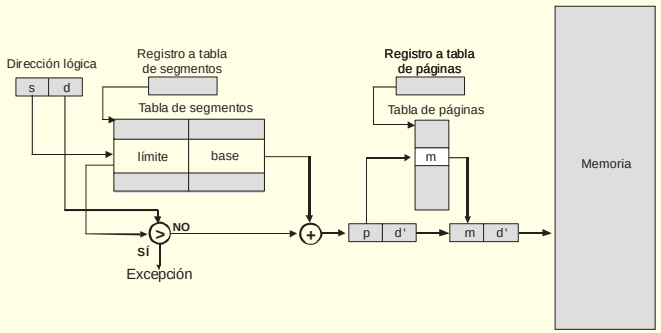
Tabla de páginas invertida

- Procesadores actuales espacio lógico enorme (dirs. de 64 bits)
  - TPs muy grandes incluso usando multinivel
- Uso de TPs invertidas
  - Entrada por cada marco indica página almacenada en él
  - Necesario guardar núm. de página e id. de proceso
- Procedimiento de traducción:
  - MMU usa TLB convencional
  - Si fallo en TLB → se busca traducción en TP invertida
- Tabla *hash* para evitar búsqueda secuencial
- Difícil compartir. Alternativa: guardar marco en entrada de TP
- TP pequeña pero SO debe guardar info. de páginas no residentes

Esquema de traducción con TP invertida



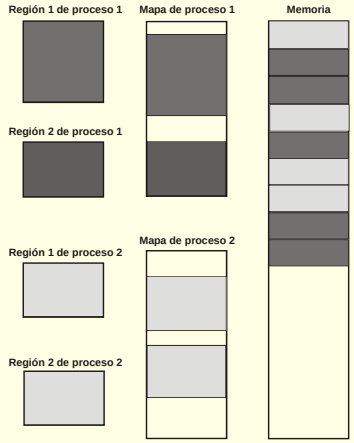
Segmentación paginada: esquema de traducción



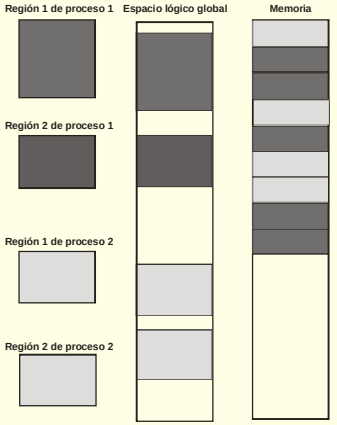
Creación de espacio lógico independiente

- Segmentación paginada local: 1TS/proceso y 1TP/proceso
  - Espacio lógico por proceso
  - Mayor similitud con paginación
- Segmentación paginada global: 1TS/proceso y TP única
  - Espacio lógico global
  - Mayor similitud con segmentación simple

Segmentación paginada local



Segmentación paginada global



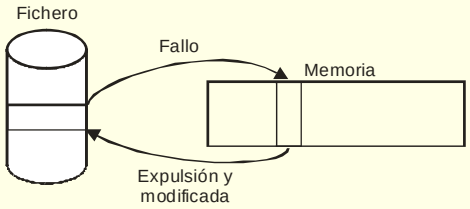
M. virtual: Antecedentes

- **Intercambio** (*swapping*): más procesos de los que caben en mem
  - Disco (*swap*): respaldo de memoria
  - *Swap out*: expulsa/suspende proceso si no hay sitio
    - Diversos criterios para expulsar: mejor si bloqueado
  - *Swap in*: reanudación de proceso expulsado (y listo)
  - Preasignación de *swap* o no
- ***Overlays***: Programas más grandes que memoria disponible
  - No transparente a programador: info. de uso de módulos
  - Montador genera ejecutable con código de carga y descarga

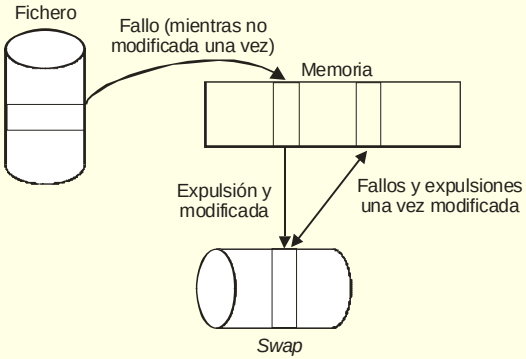
Fundamento de la memoria virtual

- M. virtual: SO gestiona niveles de m. principal y m. secundaria
  - Sube por demanda; Baja por expulsión
- Aplicable por proximidad de referencias
  - Procesos sólo usan parte de su mapa en intervalo de tiempo
  - Parte usada (*cjto de trabajo*) en m. principal (*cjto residente*)
- Beneficios:
  - Aumenta el grado de multiprogramación
  - Permite ejecución de programas que no quepan en mem. ppal
- No adecuada para sistemas de tiempo real
- Basada en paginación: Uso del bit de validez
  - Página no residente se marca como no válida
  - En acceso: Excepción de fallo de página

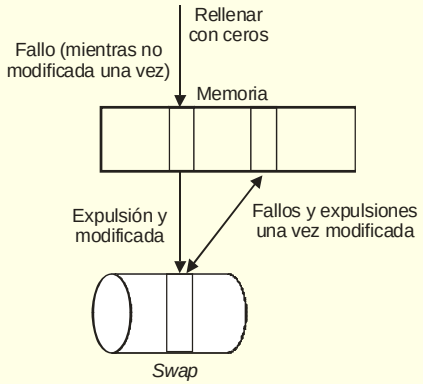
Ciclo de vida de página compartida y en fichero



Ciclo de vida de página privada y en fichero



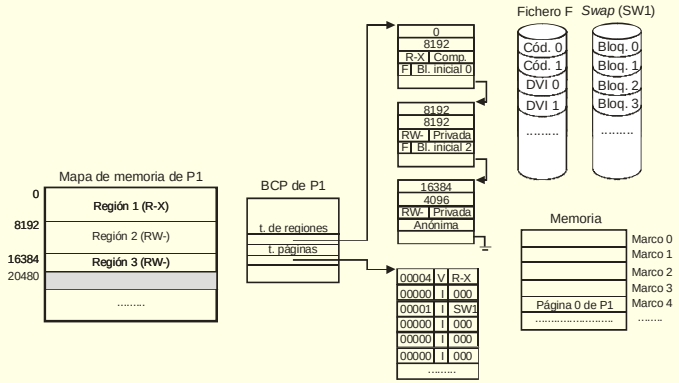
Ciclo de vida de página anónima



Políticas de administración

- Localización
- Extracción
- Ubicación:
  - Cualquiera, aunque puede usarse coloración de páginas
- Reemplazo
- Actualización
  - Escritura diferida
- Reparto de espacio

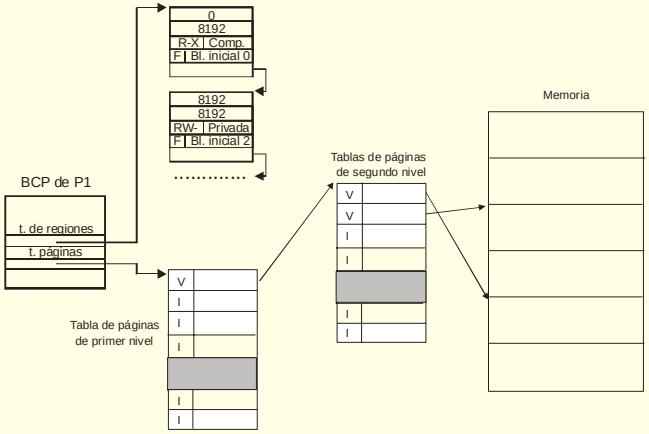
Política de localización: posibles ubicaciones



Política de extracción: Fallo de página

- Si dirección inválida → Aborta proceso o le manda señal
- Si no hay ningún marco libre (consulta T. marcos)
  - Reemplazo: pág P marco M → P inválida
    - Si *mod* → **escribir** fichero o *swap*
- Hay marco libre (se ha liberado o lo había previamente):
  - Inicia **lectura** de página en marco M
  - Conecta entrada de TP a M
- Fallo de página en modo sistema no siempre es error:
  - Acceso a página de usuario no residente
- Prepaginación: trae páginas por anticipado (no por demanda)

Creación de tablas de páginas por demanda



Política de reemplazo

- Tipo de reemplazo: local o global
- También en caché de sistemas de ficheros
- Objetivo: Minimizar la tasa de fallos de página.
  - Poca sobrecarga y MMU estándar
- Algoritmo óptimo (MIN): Irrealizable
  - Página residente que tardará más en accederse

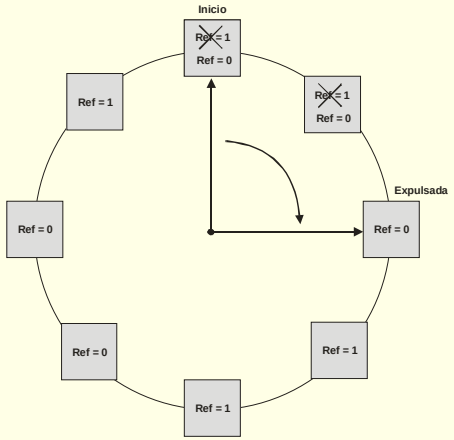
Algoritmo FIFO

- Página que lleva más tiempo residente
- Fácil implementación:
  - Páginas residentes en orden FIFO
  - No requiere el bit de página accedida (*Ref*)
- No es una buena estrategia: basado sólo en tiempo de residencia
- Anomalía de Belady
  - Ejemplos donde: ↑ nº marcos ⇒ ↑ nº fallos

Algoritmo LRU (*Least Recently Used*)

- Página residente menos recientemente usada
- Proximidad de referencias: pasado reciente → futuro próximo
- No anomalía de Belady: algoritmo de pila
- Sutileza: ¿en tiempo de proceso o de sistema?
- Difícil implementación estricta (hay aproximaciones):
  - Precisaría una MMU específica
- Sí se usa como tal en caché de sistemas de ficheros

Algoritmo del reloj (o 2ª oportunidad)



Buffering de páginas

- Reemplazo bajo demanda: Mejor por anticipado
- Reserva de marcos libres
- Fallo de página: siempre usa marco libre (no reemplazo)
- Si nº marcos libres < umbral
  - "demonio de paginación" aplica algoritmo de reemplazo
    - páginas no modificadas → lista de marcos libres
    - páginas modificadas → lista de marcos modificados
      - ► cuando se escriban a disco pasan a lista de libres
- Si se referencia una página mientras está en estas listas:
  - fallo de página la recupera directamente de la lista (no E/S)

Caché de páginas

- Encontrar eficientemente si página está residente:
  - Necesario en *buffering* y al compartir
  - [Fichero|Disp. *swap*, nº bloque] → {nº marco | !residente}
  - Págs. anónimas sin *swap* asignado no en caché de páginas
- Cargar página de fichero o *swap* en fallo
  - Insertar en caché de páginas
- Sistema de ficheros también incluye una caché de bloques
  - Tendencia: fusión de caché de páginas y de bloques

Retención de páginas en memoria

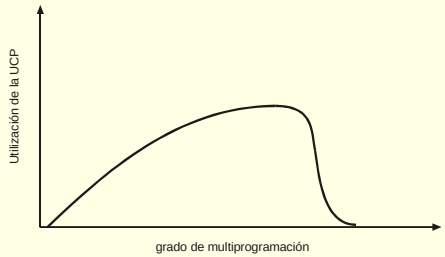
- Páginas marcadas como no reemplazables
- Se aplica a páginas del propio SO
  - SO con páginas fijas en memoria es más sencillo
- También se aplica mientras se hace DMA sobre una página
- Servicio para fijar en memoria una o más páginas de su mapa
  - Adecuado para procesos de tiempo real
  - Puede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

Política de reparto de espacio

- Estrategia de asignación fija (reemplazo local)
  - Nª marcos asignados a proceso (cjto residente) es constante
  - No se adapta a las distintas fases del programa
  - Comportamiento relativamente predecible
  - Arquitectura impone nº mínimo
- Estrategia de asignación dinámica
  - Nº marcos varía según evolución de proceso(s)
  - Asignación dinámica + reemplazo local
    - comportamiento relativamente predecible
  - Asignación dinámica + reemplazo global
    - comportamiento difícilmente predecible

Hiperpaginación (*Thrashing*)

- Tasa excesiva de fallos de página de proceso o de sistema
- Con asignación fija: Hiperpaginación en proceso
- Con asignación variable: Hiperpaginación en el sistema

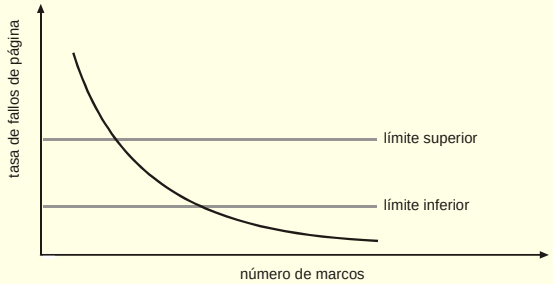


Políticas de control de carga

- Estrategia del conjunto de trabajo
  - Páginas usadas por proceso en 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)
  - Requiere MMU específica
- Estrategia basada en frecuencia de fallos de página (PFF)
- Control de carga y reemplazo global
  - No control de hiperpaginación
  - Algoritmo de control de carga empírico
    - Si nº marcos frecuentemente debajo de umbral

Estrategia basada en frecuencia de fallos

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



Dispositivo de paginación (*swap*)

- Disco, partición o fichero (más lento)
  - Incorporación dinámica y configurable de espacio de *swap*
- Estructura: cabecera y bloques; mapa de bits sólo en memoria
- Preasignación de espacio
  - Al crear región privada o sin soporte
  - Permite detectar síncronamente la falta de espacio de *swap*
- Sin preasignación de espacio
  - En primera expulsión se le asigna espacio en *swap*
  - Mejor aprovechamiento de espacio de almacenamiento
- Regiones privadas o sin soporte usan el *swap*
  - Compartidas con soporte usan directamente fichero
- Bloque de *swap* puede compartirse: contador de referencias

Compartimiento de páginas

- Escenarios:
  - Zona de memoria compartida
  - Compartir código de programa o biblioteca
  - Fichero proyectado en modo compartido
  - Regiones compartidas después de `fork`
- Compartir igual que con paginación convencional pero:
  - Caché de páginas para localizar pág compartida ya residente
  - Traducción inversa al expulsar página compartida
    - De marco de página a entradas de TP que lo referencian

Duplicado perezoso de páginas

- Escenarios de uso duplicado:
  - Datos con valor inicial de programas y bibliotecas
  - Regiones compartidas después de `fork`
  - Fichero proyectado en modo privado
- Optimización: Duplicado por demanda (*copy-on-write*, COW)
  - Se comparte una página mientras no se modifique
  - Si un proceso la modifica se crea una copia para él
- Implementación de COW
  - Se comparten páginas de regiones duplicadas pero:
    - se marcan de sólo lectura en TP (no en t. regiones)

Tratamiento del fallo de COW

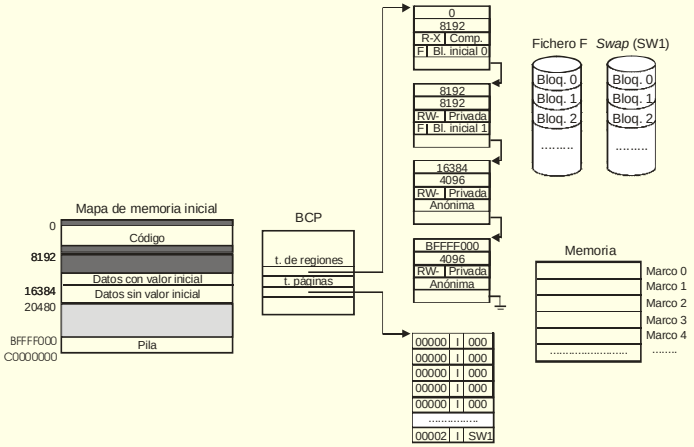
- Si dirección ∉ región WR → Aborta (o señal) proceso
- Si ref > 1
  - Reserva marco libre, copia contenido y conecta a TP
  - Devuelve permiso WR a entrada, ref--, no inserta caché págs
- Si ref == 1
  - Devuelve permiso WR a entrada, elimina de caché págs
- Si página con bloque de *swap* asignado, desvincular del mismo

- **Recordatorio importante**:
  - Sólo se asigna espacio (marcos) en fallo de página y de COW
    - **Nunca** al crear el mapa o una región del mismo

Operaciones en el nivel de regiones

- Crear una región: No asigna m. principal ni entradas TP
  - Busca zona libre en mapa de proceso usando t. regiones
    - Excepto si ubicación de la región está prefijada
  - Reserva y rellena entrada t. regiones
    - Si preasignación, reserva *swap*
- Eliminar una región:
  - Libera entrada t. regiones e invalida entradas TP
  - Marcos y bloques de *swap*: ref--
- Redimensionar una región (*heap* o pila)
  - Ajusta entrada t. regiones
    - Decrece: Invalida entradas TP + marcos y *swap*: ref--
    - Crece: Nada más, excepto reserva *swap* si preasignación

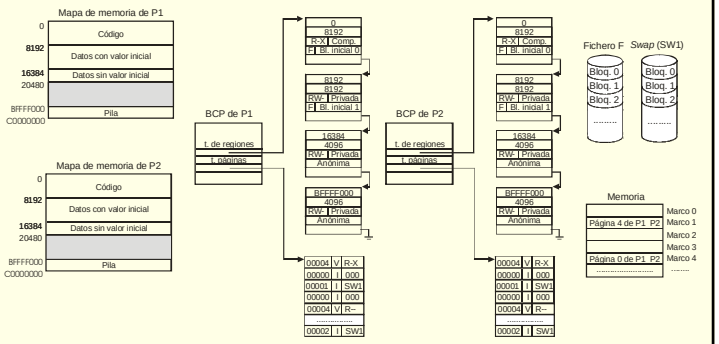
Creación del mapa inicial



Expansión de la región de pila

- Expansión automática: programa ↓SP y accede → fallo página
- Extensión de tratamiento de fallo de página:
  - Si dirección inválida (∉ región)
    - Si dirección < SP → Aborta proceso o le manda señal
    - Si no → Expansión de pila
- Sutileza:
  - Al menos 1 página inválida entre pila y región más cercana

Resultado del `fork`



Ficheros proyectados en memoria

- Generalización de memoria virtual
  - Entradas de TP referencian a un fichero de usuario
- Programa solicita proyección de fichero (o parte) en su mapa
  - Puede especificar protección y si privada o compartida
- Programa accede a posición de memoria asociada a fichero
  - Está accediendo al fichero
- Forma alternativa de acceso a ficheros frente a *read*/*write*
  - Menos llamadas al sistema
  - Se evitan copias intermedias en caché de sistema de ficheros
  - Se facilita programación: acceso a estructuras de datos

Proyección de un fichero