

# Sistemas Operativos

## Memoria

Ingeniería de Sistemas

Universidad Nacional

# Índice

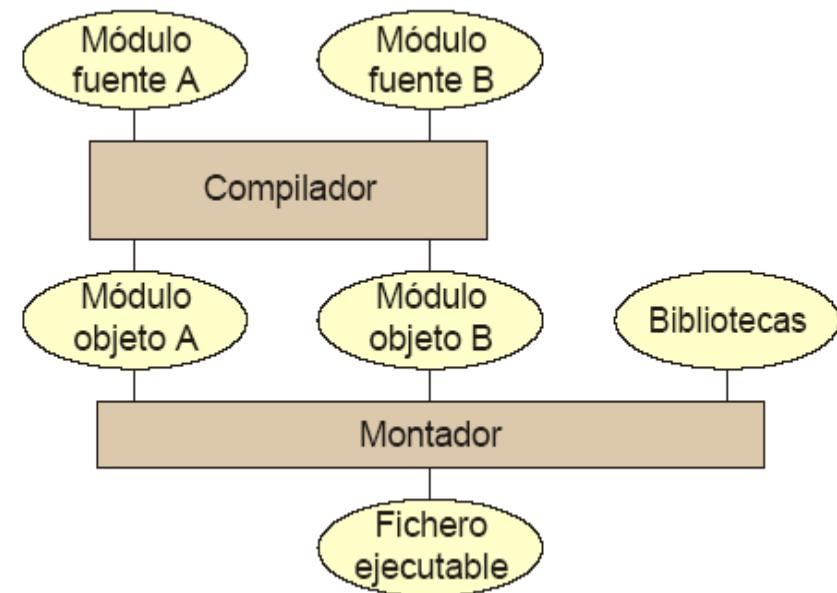
- Generalidades
- Objetivos del sistema de gestión de memoria
- Requerimientos para manejo de memoria
- Particionamiento de memoria
- Paginación y segmentación
- Memoria virtual
- Archivos proyectados en memoria

# Índice

- Generalidades
- Objetivos del sistema de gestión de memoria
- Requerimientos para manejo de memoria
- Particionamiento de memoria
- Paginación y segmentación
- Memoria virtual
- Archivos proyectados en memoria

# Generalidades

- **Fases en Generación de Ejecutable**
  - Compilación:
    - Resuelve referencias dentro cada módulo fuente
    - Genera módulo objeto
  - Montaje (o enlace):
    - Resuelve referencias entre módulos objeto
    - Resuelve referencias a símbolos de bibliotecas
  - Genera ejecutable incluyendo bibliotecas



# Generalidades

- **Bibliotecas de Objetos**
  - Biblioteca: colección de módulos objeto relacionados
  - Bibliotecas del sistema o creadas por el usuario
  - Bibliotecas Estáticas:
    - Montaje: enlaza los módulos objeto del programa y de las bibliotecas
    - Ejecutable autocontenido
  - Desventajas del montaje estático:
    - Ejecutables grandes
    - Código de función de biblioteca repetido en muchos ejecutables
    - Múltiples copias en memoria del código de función de biblioteca
    - Actualización de biblioteca implica volver a montar

# Generalidades

- **Bibliotecas Dinámicas**
  - Carga y montaje de biblioteca en tiempo de ejecución
  - Ejecutable contiene:
    - Nombre de la biblioteca
    - Rutina de carga y montaje en tiempo de ejecución
  - En 1<sup>a</sup> referencia a símbolo de biblioteca en tiempo de ejecución:
    - Rutina carga y monta biblioteca correspondiente
    - Ajusta instrucción que realiza referencia para que próximas referencias accedan a símbolo de biblioteca
    - Problema: Se modificaría el código del programa
      - Solución típica: Referencia indirecta mediante una tabla

# Generalidades

- **Ventajas de Bibliotecas Dinámicas**
  - Ventajas:
    - Menor tamaño ejecutables
    - Código de rutinas de biblioteca sólo en archivo de biblioteca
    - Procesos pueden compartir código de biblioteca
    - Actualización automática de bibliotecas: Uso de versiones
  - Desventajas:
    - Mayor tiempo de ejecución debido a carga y montaje
      - Tolerable: Compensa el resto de las ventajas
    - Ejecutable no autocontenido
  - Uso de bibliotecas dinámicas es transparente
    - Mandatos de compilación y montaje igual que con estáticas

# Generalidades

- **Uso explícito de Bibliotecas Dinámicas**
  - Forma de uso habitual:
    - Se especifica en tiempo de montaje qué biblioteca usar pero se pospone su carga y montaje a tiempo de ejecución
  - Uso explícito:
    - Requerido por aplicaciones que determinan en tiempo de ejecución qué bibliotecas deben usar
    - No se especifica biblioteca en mandato de montaje
    - Programa solicita carga de bib. mediante servicio del sistema
      - *dlopen* en UNIX y *LoadLibrary* en Win32
    - Acceso “no” transparente a símbolos de la biblioteca
    - Programa necesita usar servicios del sistema para ello
      - *dlsym* en UNIX y *GetProcAddress* en Win32

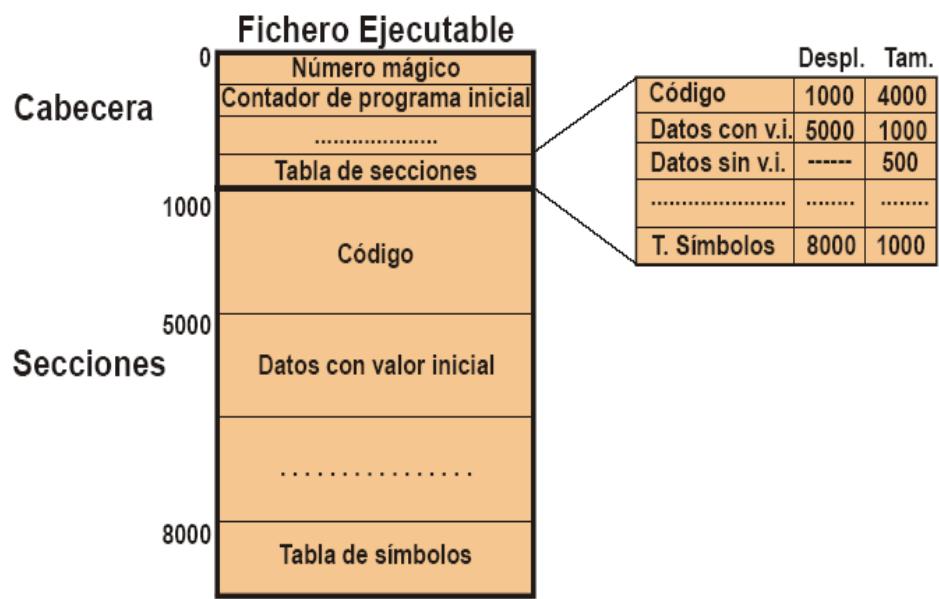
# Generalidades

- **Compartir Bibliotecas Dinámicas**
  - Biblioteca dinámica contiene referencias internas
    - Problema de zona compartida con autoreferencias
  - Tres posibles soluciones:
    - 1. A cada bib. dinámica se le asigna rango de direcciones fijo
      - Inconveniente: Poco flexible
    - 2. En montaje en t. de ejecución se reajustan autoreferencias
      - Inconveniente: Impide compartir código de biblioteca
    - 3. Crear biblioteca con código independiente de posición(PIC)
      - Se genera código con direccionamiento relativo a registro
      - Inconveniente (tolerable): dir. relativo menos eficiente

# Generalidades

- **Formato del Ejecutable**

- Distintos fabricantes usan diferentes formatos
  - Ejemplo: En Linux *Executable and Linkable Format* (ELF)
- Estructura: Cabecera y conjunto de secciones
- Cabecera:
  - Número mágico que identifica a ejecutable
  - Punto de entrada del programa
  - Tabla de secciones



# Generalidades

- **Secciones del Ejecutable**
  - Variedad de tipos de secciones. Ejemplo:
    - Tabla de símbolos para depuración
    - Lista de bibliotecas dinámicas usadas
  - Más relevantes en mapa de memoria del proceso:
    - Código, Datos con valor inicial y Datos sin valor inicial
  - Código (texto)
    - Contiene código del programa
  - Datos con valor inicial
    - Variables globales inicializadas
  - Datos sin valor inicial
    - Variables globales no inicializadas
    - Aparece en tabla de secciones pero no se almacena en ejecutable
  - ¿Por qué no hay sección vinculada a variables locales?

# Generalidades

- **Variable Globales y Locales**
  - Variables globales
    - Estáticas
    - Se crean al iniciarse programa
    - Existen durante ejecución del mismo
    - Dirección fija en memoria y en ejecutable
  - Variables locales y parámetros
    - Dinámicas
    - Se crean al invocar función
    - Se destruyen al retornar
    - La dirección se calcula en tiempo de ejecución
    - Recursividad: varias instancias de una variable

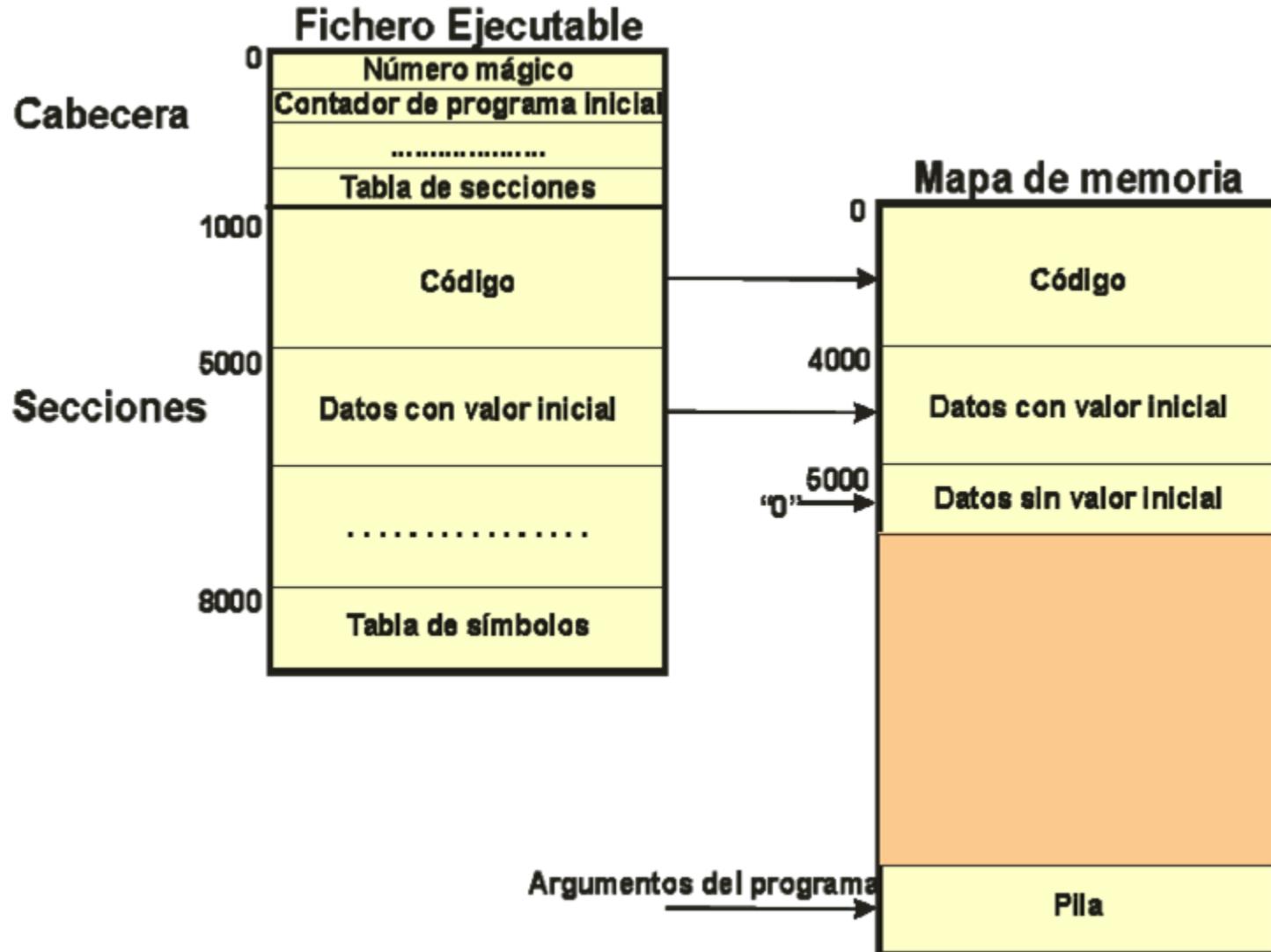
# Generalidades

- **Mapa de Memoria de un Proceso**
  - Mapa de memoria o imagen del proceso: conjunto de regiones
  - Región:
    - Tiene asociada una información (un “objeto de memoria”)
    - Zona contigua tratada como unidad al proteger o compartir
  - Cada región se caracteriza por:
    - dirección de comienzo y tamaño inicial
    - soporte: donde se almacena su contenido inicial
    - protección: RWX
    - uso compartido o privado
    - tamaño fijo o variable

# Generalidades

- **Creación de un mapa de memoria a partir del ejecutable**
  - Ejecución de un programa: Crea mapa a partir de ejecutable
    - Regiones de mapa inicial -> Secciones de ejecutable
  - Código
    - Compartida, RX, T. Fijo, Soporte en Ejecutable
  - Datos con valor inicial
    - Privada, RW, T. Fijo, Soporte en Ejecutable
  - Datos sin valor inicial
    - Privada, RW, T. Fijo, Sin Soporte (rellenar 0)
  - Pila
    - Privada, RW, T. Variable, Sin Soporte (rellenar 0)
    - Crece hacia direcciones más bajas
    - Pila inicial: argumentos del programa

# Generalidades



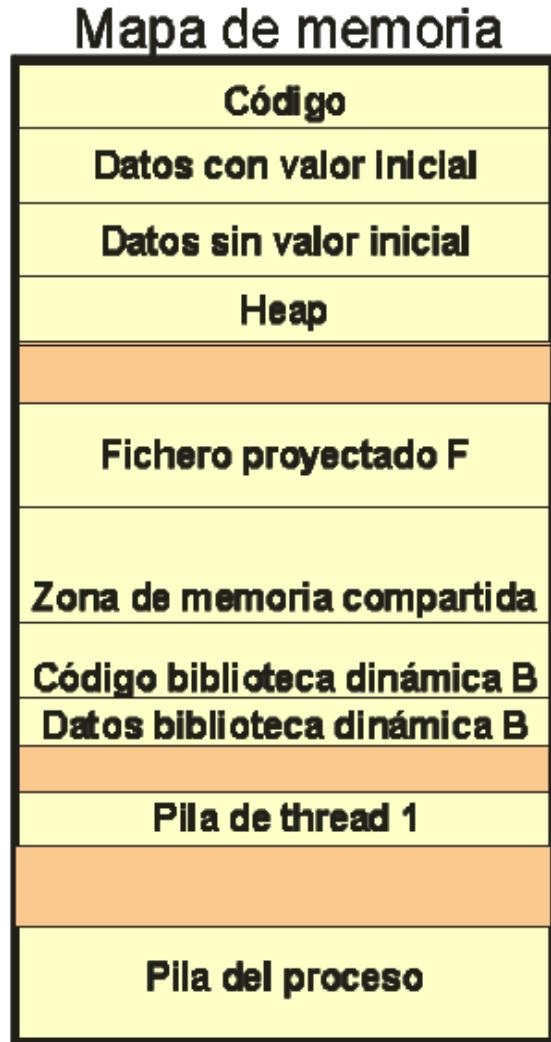
# Generalidades

- **Otras Regiones del Mapa de Memoria**
  - Durante ejecución de proceso se crean nuevas regiones
    - Mapa de memoria tiene un carácter dinámico
  - Región de Heap
    - Soporte de memoria dinámica (malloc en C)
    - Privada, RW, T. Variable, Sin Soporte (rellenar 0)
    - Crece hacia direcciones más altas
  - Archivo proyectado
    - Región asociada al archivo proyectado
    - T. Variable, Soporte en Archivo
    - Protección y carácter compartido o privado especificado en proyección

# Generalidades

- **Otras Regiones del Mapa de Memoria**
  - Memoria compartida
    - Región asociada a la zona de memoria compartida
    - Compartida, T. Variable, Sin Soporte (rellenar 0)
    - Protección especificada en proyección
  - Pilas de threads
    - Cada pila de thread corresponde con una región
    - Mismas características que pila del proceso
  - Carga de biblioteca dinámica
    - Se crean regiones asociadas al código y datos de la biblioteca

# Generalidades



# Generalidades

- **Operaciones sobre Regiones**

- Para estudiar evolución del mapa de memoria se pueden distinguir las siguientes operaciones:
- Crear región
  - Implícitamente al crear mapa inicial o por solicitud del programa en t. de ejecución (p.ej. proyectar un archivo)
- Eliminar región
  - Implícitamente al terminar el proceso o por solicitud del programa en t. de ejecución (p.ej. desproyectar un archivo)
- Cambiar tamaño de la región
  - Implícitamente para la pila o por solicitud del programa para el heap
- Duplicar región
  - Operación requerida por el servicio FORK de POSIX

# Índice

- Generalidades
- Objetivos del sistema de gestión de memoria
- Requerimientos para manejo de memoria
- Particionamiento de memoria
- Paginación y segmentación
- Memoria virtual
- Archivos proyectados en memoria

# Objetivos del Gestor de Memoria

- S.O. 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
  - Proporcionar protección entre procesos
  - Permitir que procesos comparten memoria
  - Dar soporte a las regiones del proceso
  - Maximizar el grado de multiprogramación
  - Proporcionar a los procesos mapas de memoria muy grandes

# Índice

- Objetivos del sistema de gestión de memoria
- Requerimientos para manejo de memoria
- Particionamiento de memoria
- Paginación y segmentación
- Memoria virtual
- Archivos proyectados en memoria

# Requerimientos para manejo de memoria

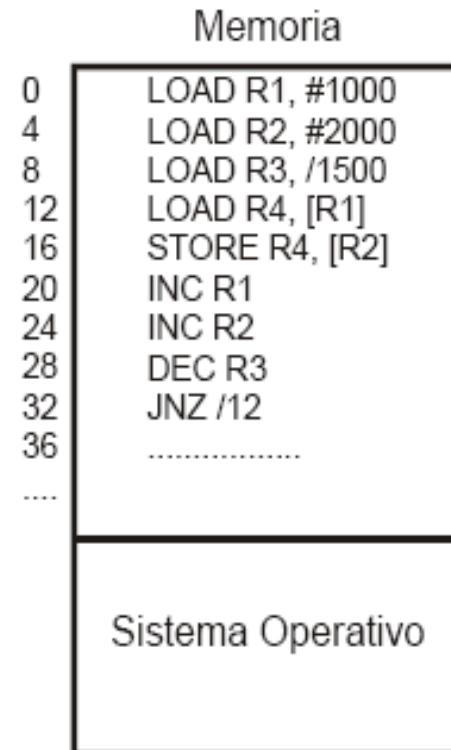
- Espacios Lógicos Independientes
  - No se conoce posición de memoria donde un programa ejecutará
  - Código en ejecutable genera referencias entre 0 y N
  - Ejemplo: Programa que copia un vector

Fichero Ejecutable	
0	
4	Cabecera
...	
96	
100	LOAD R1, #1000
104	LOAD R2, #2000
108	LOAD R3, /1500
112	LOAD R4, [R1]
116	STORE R4, [R2]
120	INC R1
124	INC R2
128	DEC R3
132	JNZ /12
136	.....

Vector destino a partir de dirección 2000  
Tamaño del vector en dirección 1500  
Vector origen a partir de dirección 1000

# Requerimientos para manejo de memoria

- **Ejecución en S.O. Monoprogramado**
  - S.O. en direcciones más altas
  - Programa se carga en dirección 0
  - Se le pasa el control al programa



# Requerimientos para manejo de memoria

- **Reubicación**
  - Necesaria en S.O. con multiprogramación:
    - Reubicar: Traducir direcciones lógicas a físicas
  - Dir. lógicas: direcciones de memoria generadas por el programa
  - Dir. físicas: direcciones de mem. principal asignadas al proceso
  - Función de traducción:
    - $\text{Traducción}(IdProc, dir\_lógica) \rightarrow dir\_física$
  - Reubicación crea espacio lógico independiente para proceso
  - S.O. debe poder acceder a espacios lógicos de los procesos
  - Ejemplo: Programa tiene asignada memoria a partir de 10000
    - Sumar 10000 a direcciones generadas
  - Dos alternativas:
    - Reubicación hardware o software

# Requerimientos para manejo de memoria

- **Protección**
  - Monoprogramación: Protección del SO
  - Multiprogramación: Además procesos entre sí
  - Traducción debe crear espacios disjuntos
  - Necesario validar todas las direcciones que genera el programa
    - La detección debe realizarla el hardware del procesador
    - El tratamiento lo hace el SO
  - En sistemas con mapa de E/S y memoria común:
    - Traducción permite impedir que procesos accedan directamente a dispositivos de E/S

# Requerimientos para manejo de memoria

- **Compartimiento de Memoria**

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

Mapa proceso 1



Mapa proceso 2



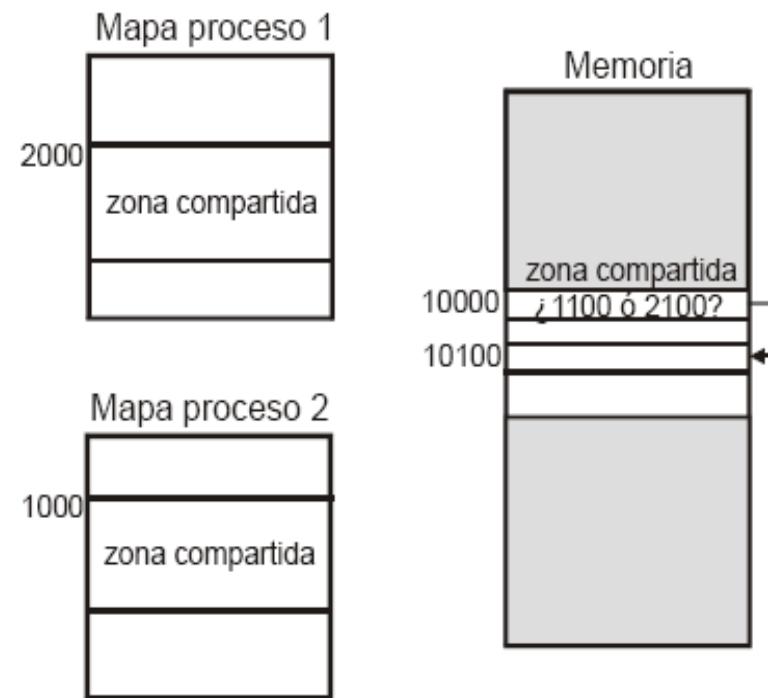
Memoria



# Requerimientos para manejo de memoria

- **Problemas al Compartir Memoria**

- Si posición de zona compartida contiene referencia a otra posición de la zona
- Ejemplo con zonas de código:
  - Zona compartida contiene instrucción de bifurcación a instrucción dentro de la zona
- Ejemplo con zonas de datos:
  - Zona contiene una lista con punteros



# Requerimientos para manejo de memoria

- **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:
    - Detectar accesos no permitidos a una región
    - Detectar accesos a huecos
    - Evitar reservar espacio para huecos
  - S.O. debe guardar una tabla de regiones para cada proceso

# Requerimientos para manejo de memoria

## ■ Maximizar Rendimiento

- Reparto de memoria maximizando grado de multiprogramación
- Se “desperdicia” memoria debido a:
  - “Restos” inutilizables (fragmentación)
  - Tablas requeridas por gestor de memoria
- Menor fragmentación -> Tablas más grandes
- Compromiso: Paginación
- Uso de memoria virtual para aumentar grado de multiprogramación

Aprovechamiento de memoria óptimo e 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

# Requerimientos para manejo de memoria

- **Mapas de Memoria muy Grandes para Procesos**
  - Procesos necesitan cada vez mapas más grandes
    - Aplicaciones más avanzadas o novedosas
  - Resuelto gracias al uso de memoria virtual
  - Antes se usaban *overlays*:
    - Programa dividido en fases que se ejecutan sucesivamente
    - En cada momento sólo hay una fase residente en memoria
    - Cada fase realiza su labor y carga la siguiente
    - No es transparente: Toda la labor realizada por programador

# Índice

- Objetivos del sistema de gestión de memoria
- Requerimientos para manejo de memoria
- Particionamiento de memoria
- Paginación y segmentación
- Memoria virtual
- Archivos proyectados en memoria

# Particionamiento de memoria

- **Particionamiento fijo.**

- Se asignan tamaños fijos a bloques en memoria.
- Un programa puede ser muy grande para un bloque.
- Se desperdicia espacio si un proceso es más pequeño que un bloque.



(a) Equal-size partitions

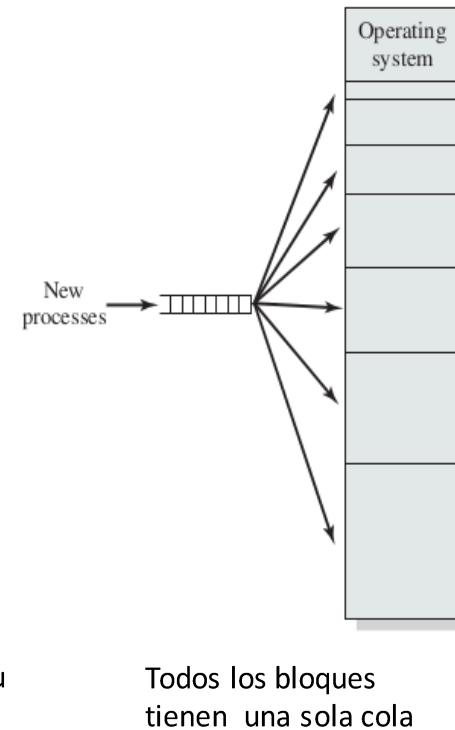
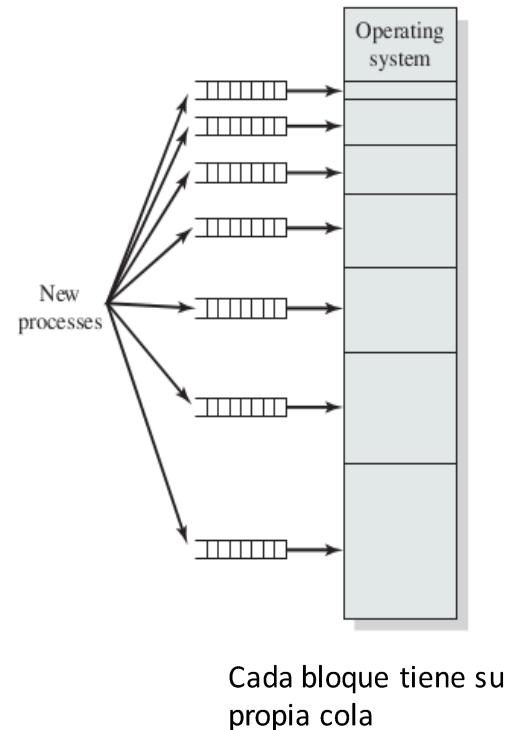


(b) Unequal-size partitions

# Particionamiento de memoria

- **Particionamiento fijo.**

- Algoritmo de asignación.
  - Tamaños fijo. Algoritmo simple.
  - Tamaños variables.
    - Se asigna un proceso al bloque más pequeño en el que encaje. Cada bloque tiene su cola.
    - Se asignan mediante una cola. Bloques grandes pueden ser poco usados.



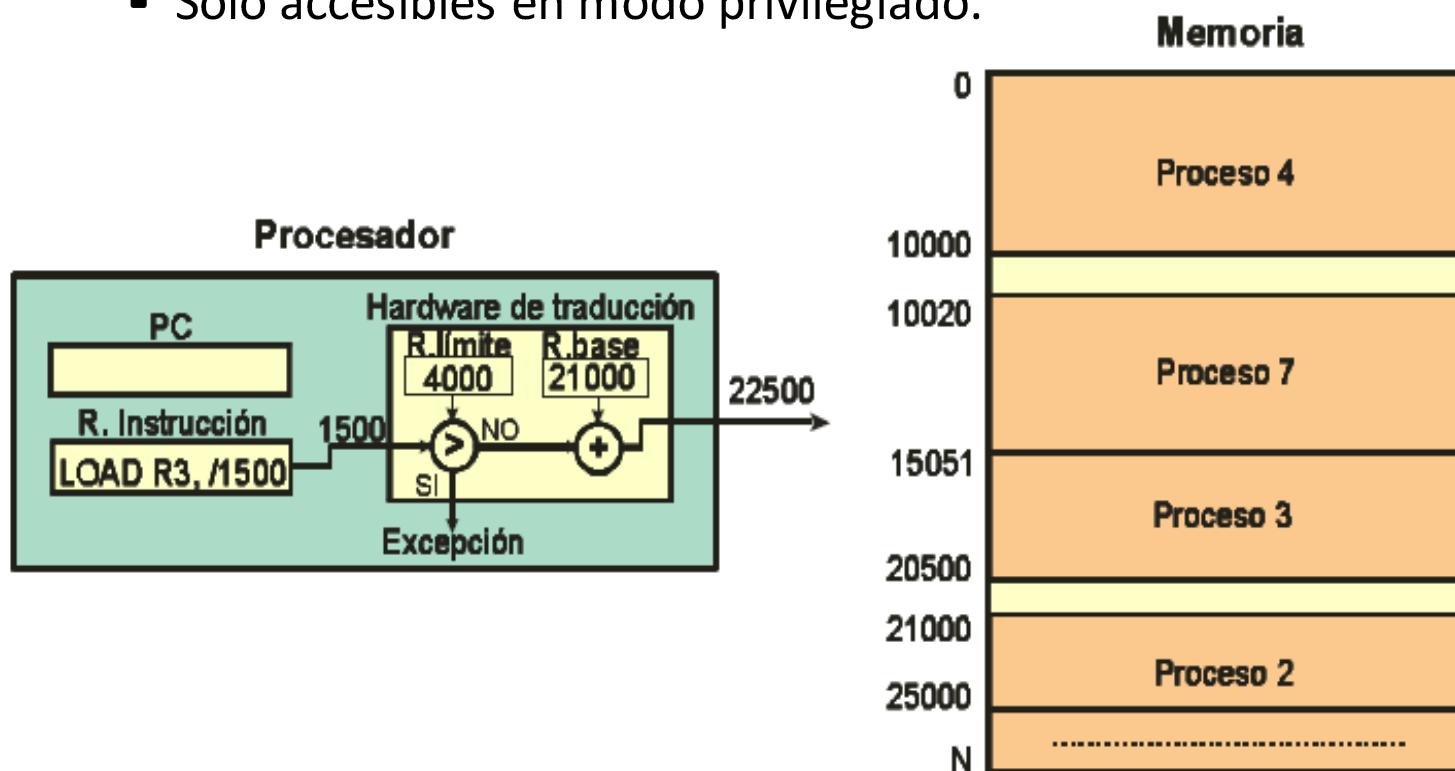
# Particionamiento de memoria

- **Particionamiento dinámico.**
  - Número de bloques variable con tamaños variables.
  - Se le asigna a un proceso la cantidad de memoria que necesita.
  - Presenta fragmentación externa.

# Particionamiento de memoria

- **Asignación Contigua**

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



# Particionamiento de memoria

- **Asignación Contigua**

- S.O. mantiene información sobre:
  - Copia del valor de regs. valla de cada proceso en su BCP
  - En el contexto S.O. carga en regs. valor adecuado
  - 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

# Particionamiento de memoria

- **Fragmentación Externa**
  - Espacio de memoria que queda entre bloques asignados a procesos.
- **Fragmentación Interna.**
  - Situación en la que la memoria asignada puede ser ligeramente mayor que la requerida, por ejemplo, cuando se redondea la cantidad de memoria asignada a un proceso.

# Particionamiento de memoria

- **Política de Asignación de Espacio**
  - ¿Qué hueco usar para satisfacer una petición?
  - Posibles políticas:
    - Primer ajuste: Asignar el primer hueco con tamaño suficiente
    - Mejor ajuste: Asignar el menor hueco con tamaño suficiente
      - Lista ordenada por tamaño o buscar en toda la lista
    - Peor ajuste : Asignar el mayor hueco con tamaño suficiente
      - Lista ordenada por tamaño o buscar en toda la lista
    - Primer ajuste es más eficiente y proporciona buen aprovechamiento de la memoria
  - Estrategia más sofisticada: Sistema *Buddy*
    - Listas de huecos con tamaños potencias de 2

# Particionamiento de memoria

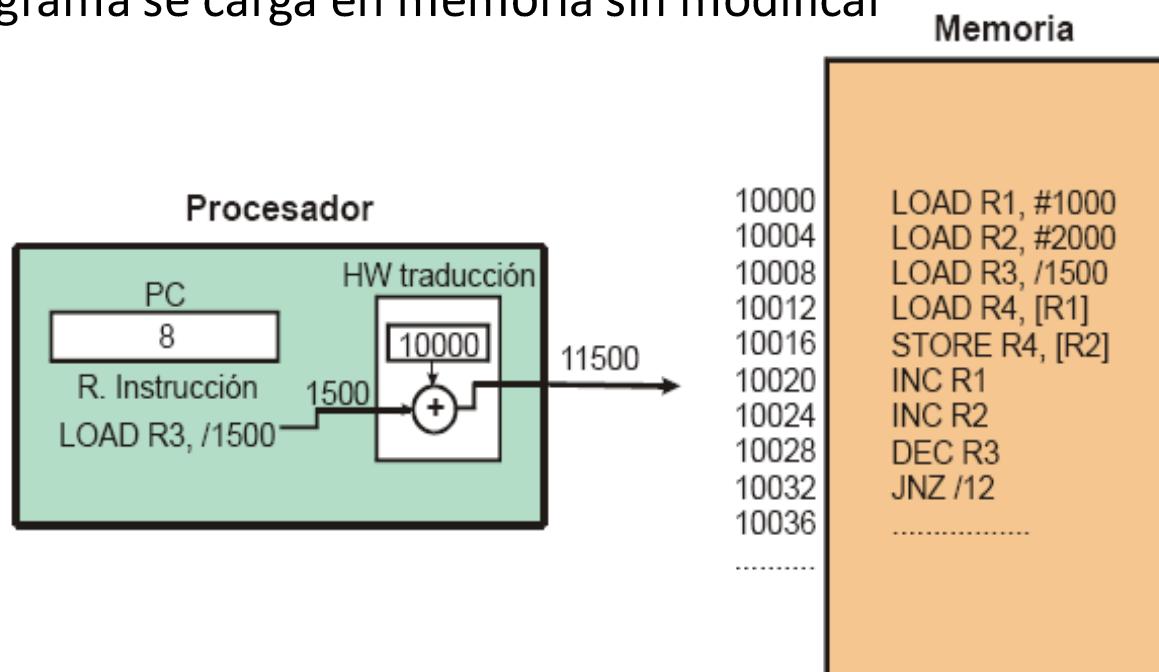
- **Operaciones sobre regiones con asignación contigua**
  - Al crear proceso se le asigna zona de memoria de tamaño fijo
    - Suficiente para albergar regiones iniciales
    - Con huecos para permitir cambios de tamaño y añadir nuevas regiones (p.ej. bibliotecas dinámicas o pilas de threads)
  - Difícil asignación adecuada
    - Si grande se desperdicia espacio, si pequeño se puede agotar
  - 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 y detectar accesos erróneos o desbordamiento de pila

# Particionamiento de memoria

- **Valoración del Esquema de Asignación Contigua**
  - ¿Proporciona las funciones deseables en un gestor de memoria?
    - Espacios independientes para procesos:
      - mediante registros valla
    - Protección:
      - mediante registros valla
    - Compartir memoria:
      - no es posible
    - Soporte de regiones:
      - no existe
      - se reserva espacio para huecos
    - Maximizar rendimiento y mapas grandes
      - mal aprovechamiento de memoria por fragmentación externa
      - no permite memoria virtual

# Particionamiento de memoria

- **Reubicación Hardware**
  - Hardware (MMU) encargado de traducción
  - S.O. se encarga de:
    - Almacena por cada proceso su función de traducción
    - Especifica al hardware qué función aplicar para cada proceso
  - Programa se carga en memoria sin modificar



# Particionamiento de memoria

## ▪ Reubicación Software

- Traducción de direcciones durante carga del programa
  - Programa en memoria distinto del ejecutable
- Desventajas:
  - No asegura protección
  - No permite mover programa en tiempo de ejecución



# Particionamiento de memoria

- **Intercambio**
  - ¿Qué hacer si no caben todos los programas en mem. principal?
  - Uso de intercambio (*swapping*)
  - *Swap*: partición de disco que almacena imágenes de procesos
  - *Swap out*:
    - Cuando no caben en memoria procesos activos, se expulsa proceso de memoria copiando imagen a *swap*
    - Diversos criterios de selección del proceso a expulsar
      - P.ej. Dependiendo de prioridad del proceso
      - Preferiblemente un proceso bloqueado
      - No expulsar si está activo DMA sobre mapa del proceso
    - No es necesario copiar todo el mapa (ni código ni huecos)

# Particionamiento de memoria

- **Intercambio**

- *Swap in:*
  - Cuando haya espacio en memoria principal, se lee proceso a memoria copiando imagen desde *swap*
  - También cuando un proceso lleva un cierto tiempo expulsado
    - En este caso antes de *swap in*, hay *swap out* de otro
- Asignación de espacio en el dispositivo de *swap*:
  - Con preasignación: se asigna espacio al crear el proceso
  - Sin preasignación: se asigna espacio al expulsarlo
- Usado en primeras versiones de UNIX
- Solución general → Uso de esquemas de memoria virtual
  - Aunque se sigue usando integrado con esa técnica

# Índice

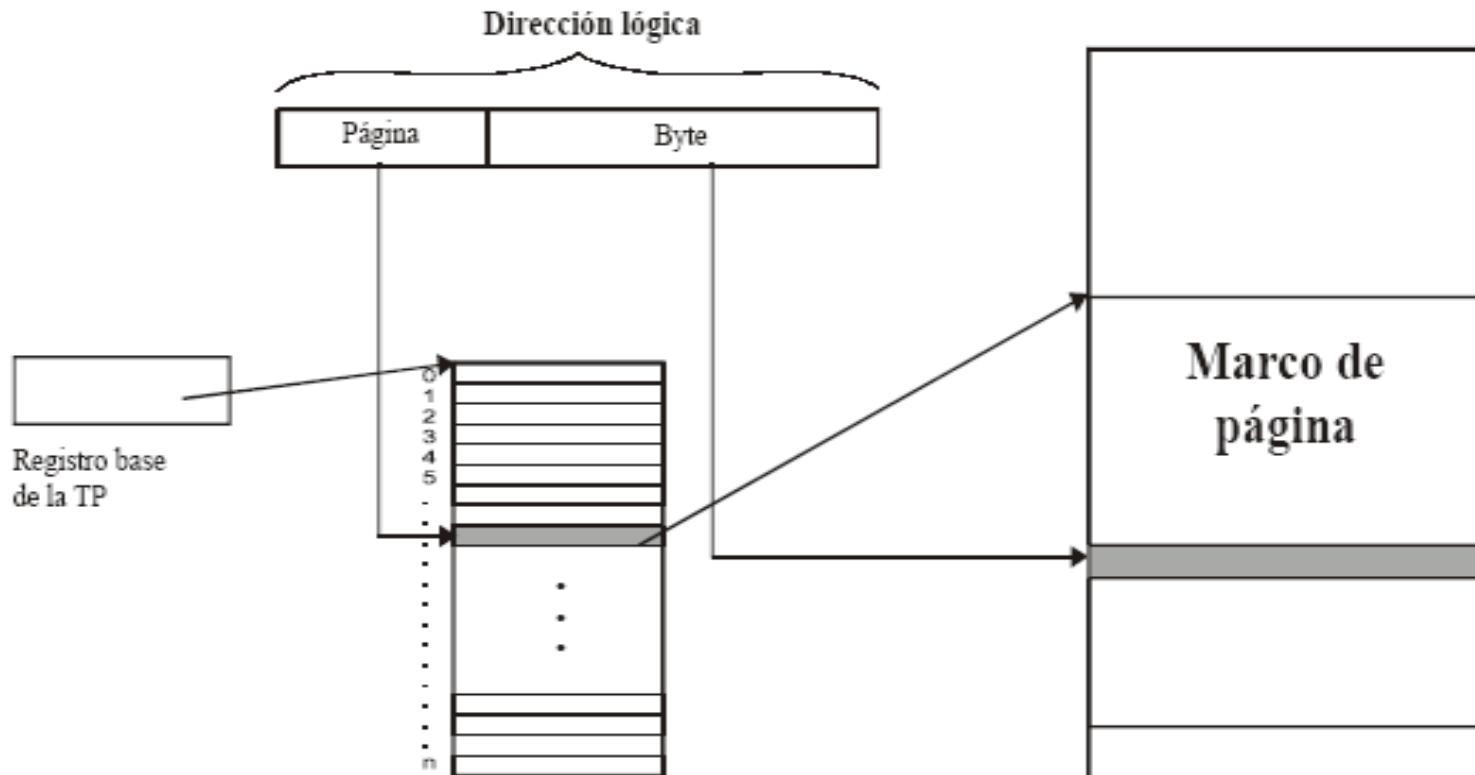
- Generalidades
- Objetivos del sistema de gestión de memoria
- Requerimientos para manejo de memoria
- Particionamiento de memoria
- Paginación y segmentación
- Memoria virtual
- Archivos proyectados en memoria

# Paginación y segmentación

- **Paginación: Aspectos Hardware**

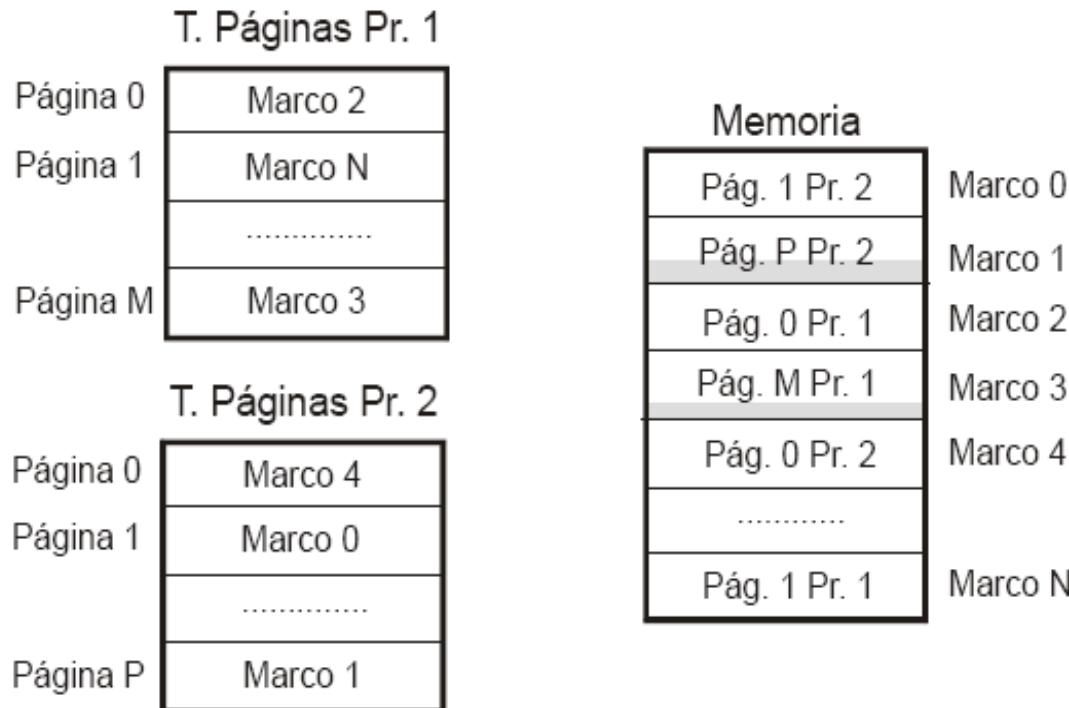
- Asignación no contigua.
- Unidad: página (tamaño potencia de 2)
- Mapa de memoria del proceso dividido en páginas
- Memoria principal dividida en marcos (tam. marco=tam. página)
- dir. lógica: nº página + desplazamiento
- Tabla de páginas (TP):
  - Relaciona cada página con el marco que la contiene
- MMU usa TP para traducir direcciones lógicas a físicas.
- Típicamente usa 2 TPs:
  - TP usuario: p.ej. dir. lógicas que empiezan por 0
  - TP sistema: p.ej. dir. lógicas que empiezan por 1
    - Sólo se permite usar estas direcciones en modo sistema
- Si mapa de E/S y memoria común: dir. lógica → dir. de E/S

# Paginación y segmentación



Esquema de Traducción

# Paginación y segmentación



- Fragmentación Externa en Paginación
  - mem. asignada > mem. requerida
    - – puede desperdiciarse parte de un marco asignado

# Paginación y segmentación

- **Contenido de Entrada de TP**
  - Número de marco asociado
  - Información de protección: RWX
    - Si operación no permitida → Excepción
  - Bit de página válida/inválida
    - Si se accede → Excepción
    - Usado en mem. virtual para indicar si página presente
  - Bit de página accedida (*Ref*)
    - MMU lo activa cuando se accede a esta página
  - Bit de página modificada (*Mod*)
    - MMU lo activa cuando se escribe en esta página
  - Bit de desactivación de cache:
    - se usa cuando la entrada corresponde con direcciones de E/S

# Paginación y segmentación

- **Tamaño de la Página**
  - Condicionado por diversos factores contrapuestos:
    - Potencia de 2 y múltiplo del tamaño del bloque de disco
    - Mejor pequeño por:
      - Menor fragmentación
      - Se ajusta mejor al conjunto de trabajo
    - Mejor grande por:
      - Tablas más pequeñas
      - Mejor rendimiento del dispositivo de E/S
  - Compromiso (entre 2K y 16K)

# Paginación y segmentación

- **Gestión del S.O.**
  - S.O. mantiene una TP por cada proceso
    - En c. contexto notifica a MMU cuál debe usar
  - S.O. mantiene una única TP para el propio S.O.
    - Proceso en modo sistema accede directamente a su mapa y al del SO
  - S.O. mantiene tabla de marcos:
    - estado de cada marco (libre o ocupado, ...)
  - S.O. mantiene tabla de regiones por cada proceso
  - Mucho mayor gasto en tablas que con asignación contigua
    - Es el precio de mucha mayor funcionalidad

# Paginación y segmentación

- **Valoración de la Paginación**
  - ¿Proporciona las funciones deseables en un gestor de memoria?
    - Espacios independientes para procesos:
      - mediante TP
    - Protección:
      - mediante TP
    - Compartir memoria:
      - entradas corresponden con mismo marco
    - Soporte de regiones:
      - bits de protección
      - bit de validez: no se reserva espacio para huecos
    - Maximizar rendimiento y mapas grandes
      - permite esquemas de memoria virtual

# Paginación y segmentación

- **Implementación de TP**

- TPs se mantiene normalmente en memoria principal
- 2 problemas: eficiencia y gasto de almacenamiento
- Eficiencia:
  - Cada acceso lógico requiere dos accesos a memoria principal
    - a la tabla de páginas + al propio dato o instrucción
  - Solución: *cache* de traducciones → TLB
- Gasto de almacenamiento:
  - Tablas muy grandes
  - Ejemplo: páginas 4K, dir. lógica 32 bits y 4 bytes por entrada
    - Tamaño TP:  $2^{20} * 4 = 4\text{MB}/\text{proceso}$
  - Solución:
    - Tablas multnivel
    - Tablas invertidas

# Paginación y segmentación

- **Translation Look-aside Buffer (TLB)**
  - Memoria asociativa con info. sobre últimas páginas accedidas
    - *cache* de entradas de TP correspondientes a estos accesos
  - 2 alternativas:
    - Entradas en TLB no incluyen información sobre proceso
      - Invalidar TLB en cambios de contexto
    - Entradas en TLB incluyen información sobre proceso
      - Registro de UCP debe mantener un ident. de proceso actual
  - Gestionada por HW:
    - MMU consulta TLB: Si fallo usa la TP en memoria
    - “Casi” transparente al S.O.
      - Volcar a TP en c.contexto para actualizar *Ref y Mod*
      - Invalidar, si entradas no incluyen información del proceso
  - Diseño alternativo: TLB gestionada por SW

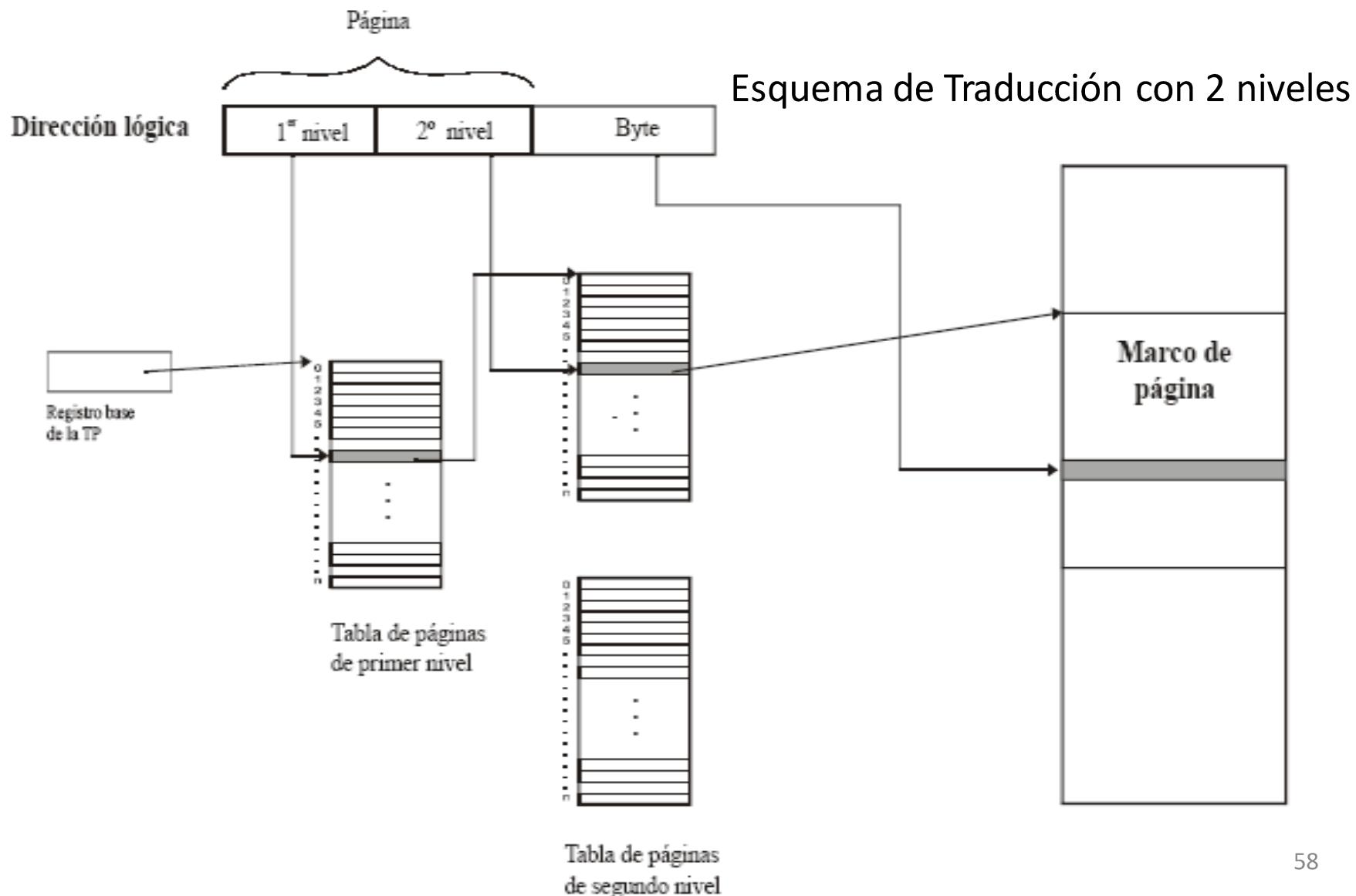
# Paginación y segmentación

- **TLB gestionada por SW**
  - Organización alternativa bastante usada actualmente
    - Traspasar al S.O. parte del trabajo de traducción
  - MMU no usa tablas de páginas, sólo consulta TLB
  - S.O. mantiene TPs que son independientes del HW
  - Fallo en TLB → Excepción.
  - S.O. se encarga de:
    - Buscar “a mano” en TP la traducción
    - Rellenar (con posible reemplazo) TLB con la traducción
  - Proporciona flexibilidad en diseño de S.O. pero menor eficiencia

# Paginación y segmentación

- **TP Multinivel**
  - 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
    - 1 acceso lógico →  $M + 1$  accesos a memoria
    - Uso de TLB
  - Si todas las entradas de una TP son inválidas
    - No se almacena esa TP
    - Se pone inválida la entrada correspondiente de la TP superior

# Paginación y segmentación



# Paginación y segmentación

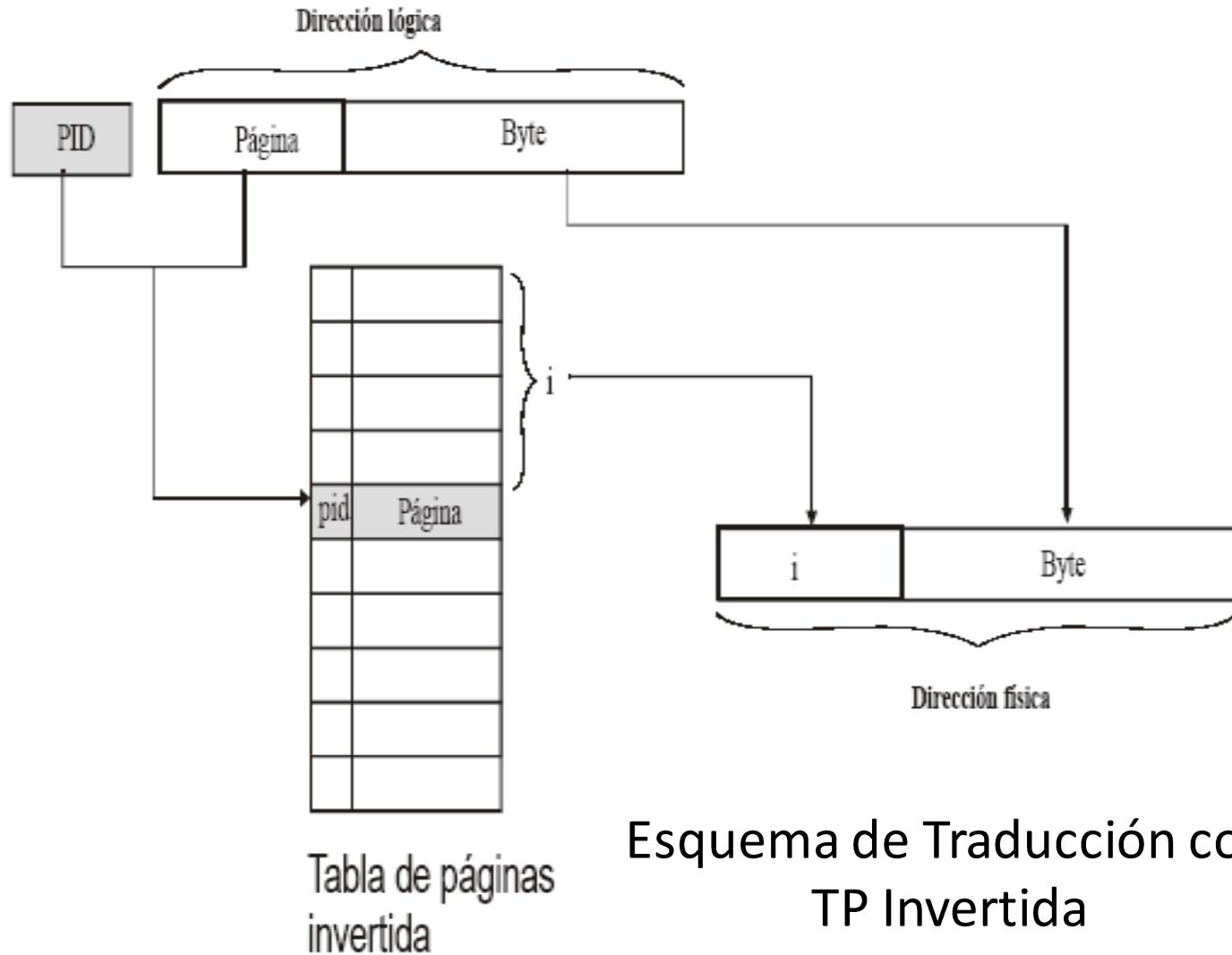
- **Ventajas de Tabla Multinivel**
  - Si proceso usa una parte pequeña de su espacio lógico
    - Ahorro en espacio para almacenar TPs
  - Ejemplo: Proceso que usa 12MB superiores y 4MB inferiores
    - 2 niveles, páginas de 4K, dir. lógica 32 bits (10 bits por nivel) y 4 bytes por entrada
    - Tamaño:  $1 \text{ TP N1} + 4 \text{ TP N2} = 5 * 4\text{KB} = 20\text{KB}$  (frente a 4MB)
  - Ventajas adicionales:
    - Permite compartir TPs intermedias
    - Sólo se requiere que esté en memoria la TP de nivel superior
    - TPs restantes pueden estar en disco y traerse por demanda

# Paginación y segmentación

- **TP Invertida**

- Procesadores actuales espacio lógico enorme (dirs. de 64 bits)
  - TPs muy grandes incluso usando multinivel
- Possible solución alternativa: Uso de TPs invertidas
  - Una entrada por cada marco indica página almacenada en él
    - Tamaño de TP proporcional a memoria principal
  - Es necesario guardar núm. de página e id. de proceso
- Procedimiento de traducción:
  - MMU usa TLB convencional
  - Si fallo en TLB → MMU busca traducción en TP invertida
- Para evitar búsqueda secuencial en TP invertida:
  - Se organiza como una tabla *hash*
  - Difícil implementar compartición
- Hay que tener en cuenta que aunque TP pequeña, S.O. debe guardar info.  
de páginas no residentes

# Paginación y segmentación

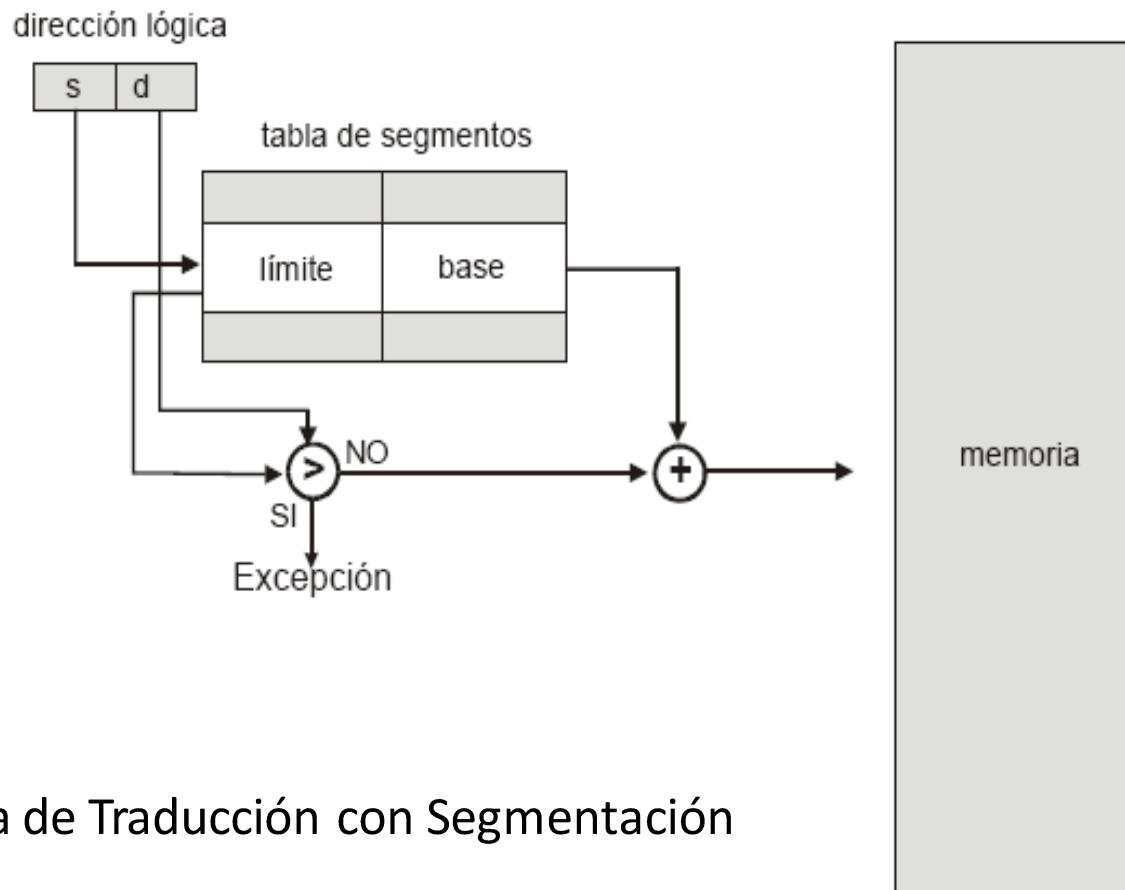


# Paginación y segmentación

- **Segmentación**

- Esquema HW que intenta dar soporte directo a las regiones
- Generalización de reg base y límite: 1 pareja por cada segmento
- Dirección lógica: núm. de segmento + dirección en el segmento
- MMU usa una tabla de segmentos (TS)
- S.O. mantiene una TS por proceso
  - en c.contexto notifica a MMU cuál debe usar
- Entrada de TS contiene (entre otros):
  - r. base y límite del segmento
  - protección: RWX
- Fragmentación externa: segmento es la unidad de asignación
- S.O. mantiene información sobre estado de la memoria:
  - Estructuras de datos que identifiquen huecos y zonas asignadas

# Paginación y segmentación



Esquema de Traducción con Segmentación

# Paginación y segmentación

- **Valoración de la Segmentación**

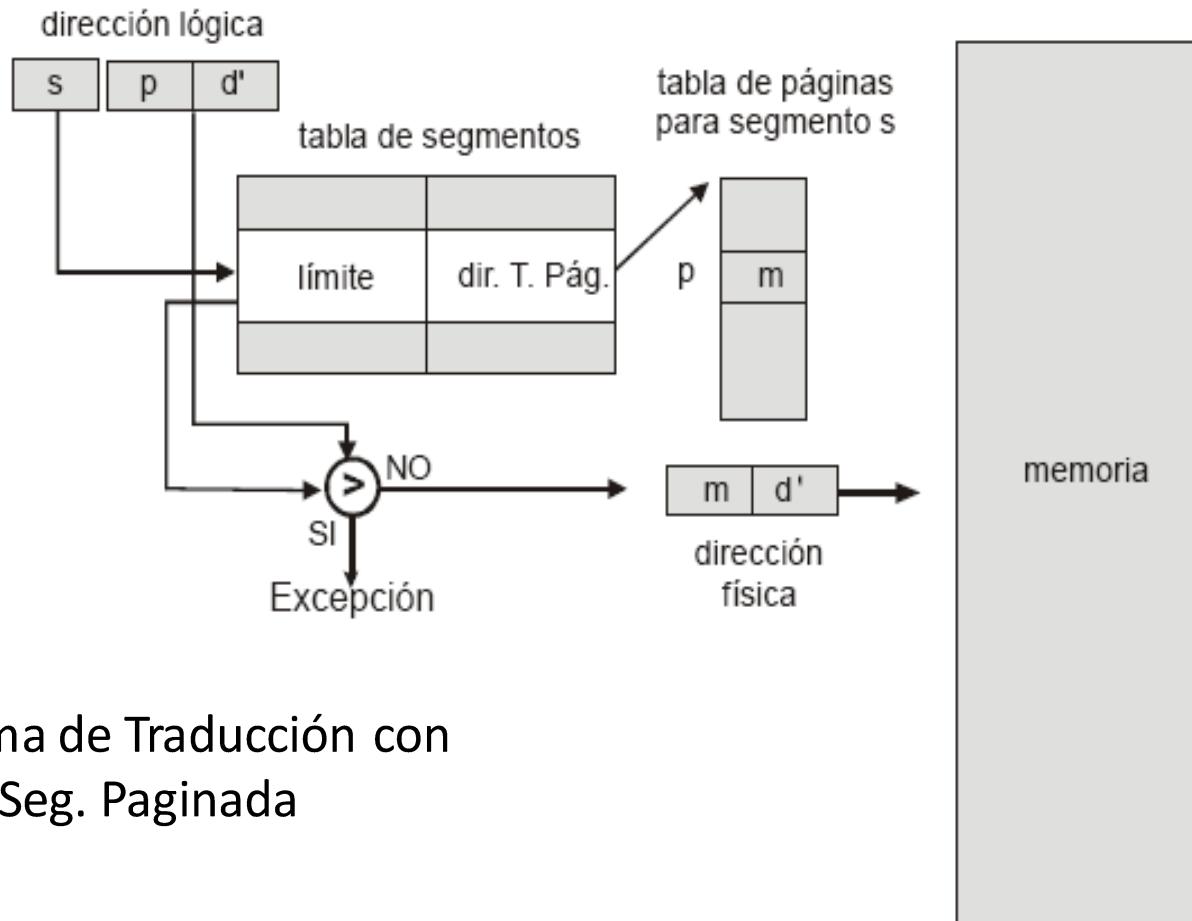
- ¿Proporciona las funciones deseables en un gestor de memoria?
  - Espacios independientes para procesos:
    - mediante TS
  - Protección:
    - mediante TS
  - Compartir memoria:
    - compartir segmentos: misma entrada en diferentes TS
  - Soporte de regiones:
    - bits de protección
  - Maximizar rendimiento y mapas grandes
    - presenta fragmentación externa
    - no facilita esquemas de memoria virtual debido a tamaño variable de segmentos
- Esquema apenas usado

# Paginación y segmentación

- **Segmentación Paginada**

- Entrada en TS apunta a una TP para el segmento
- “Lo mejor de los dos mundos”
- Segmentación:
  - Soporte directo de segmentos
  - Facilita operaciones sobre regiones:
    - Establecer protección → Modificar sólo una entrada de TS
    - Definir compartición de segmento → entradas de TS apuntando a la misma TP de segmento
- Paginación:
  - Asignación no contigua de segmento
  - Fragmentación interna

# Paginación y segmentación



Esquema de Traducción con  
Seg. Paginada

# Paginación y segmentación

- **Valoración de la Seg. Paginada**

- ¿Proporciona las funciones deseables en un gestor de memoria?
  - Espacios independientes para procesos:
    - mediante TS
  - Protección:
    - mediante TS
  - Compartir memoria:
    - compartir segmentos: misma entrada en diferentes TS
  - Soporte de regiones:
    - bits de protección
  - Maximizar rendimiento y mapas grandes
  - permite esquemas de memoria virtual
- Frente a paginación:
  - Facilita al SO gestión de regiones pero requiere HW más complejo

# Índice

- Generalidades
- Objetivos del sistema de gestión de memoria
- Requerimientos para manejo de memoria
- Particionamiento de memoria
- Paginación y segmentación
- Memoria virtual
- Archivos proyectados en memoria

# Memoria Virtual

- M. virtual: S.O. gestiona niveles de M. principal y M. secundaria
  - Transferencia de *bloques* entre ambos niveles
  - De M. secundaria a principal: por demanda
  - De M. principal a secundaria: por expulsión
- Aplicable porque procesos exhiben proximidad de referencias
  - Procesos sólo usan parte de su mapa en intervalo de tiempo
  - M. virtual: intentar que parte usada (*conjunto de trabajo*) resida en m. principal (*conjunto residente*)
- Dirección lógica  $\leftrightarrow$  Dirección virtual
- Beneficios:
  - Aumenta el grado de multiprogramación
  - Permite ejecución de programas que no quepan en mem. ppal
- No adecuada para sistemas de tiempo real
- Normalmente basada en paginación

# Memoria Virtual

- **Paginación por Demanda**
  - Segmentación pura no adecuada para memoria virtual
    - tamaño de segmentos variable
  - Paginación y segmentación paginada sí lo son:
    - Bloque transferido → Página
    - M. virtual + Paginación → Paginación por demanda
  - Estrategia de implementación: Uso del bit de validez
    - Página no residente se marca como no válida
    - En acceso: Excepción de fallo de página
      - S.O. trae la página correspondiente de mem. secundaria
      - S.O. debe diferenciar entre pág. no residente y pág. inválida
  - Prepaginación: Traer páginas por anticipado (no por demanda)
    - En fallo de página se traen además otras páginas que se considera que necesitará el proceso
  - Beneficiosa dependiendo de si hay acierto en la predicción

# Memoria Virtual

- **Tratamiento del Fallo de Página**

- Tratamiento de excepción (dirección de fallo disponible en reg.)
  - Si dirección inválida → Aborta proceso o le manda señal
  - Si no hay ningún marco libre (consulta T. marcos)
    - Selección de víctima (Alg. de reemplazo): pág P marco M
      - » Marca P como inválida
    - Si P modificada (bit *Mod* de P 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 válida referenciando a M
      - Pone M como ocupado en T. marcos (si no lo estaba)
  - Fallo de página puede implicar dos operaciones en disco

# Memoria Virtual

- **Políticas de Administración de Memoria Virtual**
  - Política de reemplazo:
    - ¿Qué página reemplazar si hay 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

# Memoria Virtual

- **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

# Memoria Virtual

- **Algoritmos de Reemplazo - Algoritmo Óptimo**
  - Criterio: página residente que tardará más en accederse
  - Irrealizable
  - Versión local y global
  - Interés para estudios analíticos comparativos

# Memoria Virtual

- **Algoritmos de reemplazo - Algoritmo FIFO**
  - Criterio: página que lleva más tiempo residente
  - Fácil implementación:
    - 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
  - Anomalía de Belady
    - Se pueden encontrar ejemplos que al aumentar el número de marcos aumenta el número de fallos de página

# Memoria Virtual

- **Algoritmos de reemplazo - Algoritmo del Reloj (o 2<sup>a</sup> 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<sup>a</sup> 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

# Memoria Virtual

- **Algoritmos de reemplazo - Algoritmo LRU**
  - Criterio: página residente menos recientemente usada
  - Por proximidad de referencias:
    - Pasado reciente condiciona futuro próximo
  - Sutileza:
    - En su versión global: menos recientemente usada en el tiempo lógico de cada proceso
  - Difícil implementación estricta (hay aproximaciones):
    - Precisaría una MMU específica
  - Posible implementación con HW específico:
    - En entrada de TP hay un contador
    - En cada acceso a memoria MMU copia contador del sistema a entrada referenciada
    - Reemplazo: Página con contador más bajo

# Memoria Virtual

- **Buffering de Páginas**

- Peor caso en tratamiento de fallo de página:
  - 2 accesos a dispositivo
- Alternativa: mantener una reserva de marcos libres
- Fallo de página: siempre usa marco libre (no reemplazo)
- Si número de marcos libres < umbral
  - “demonio de paginación” 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:
  - fallo de página la recupera directamente de la lista (no E/S)
  - puede arreglar el comportamiento de algoritmos “malos”

# Memoria Virtual

- **Retención de Páginas en Memoria**
  - Páginas marcadas como no reemplazables
  - Se aplica a páginas del propio S.O.
    - S.O. 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

# Memoria Virtual

- **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:
  - Por ejemplo: instrucción MOVE /DIR1, /DIR2 requiere un mínimo de 3 marcos:
    - instrucción y dos operandos deben estar residentes para ejecutar la instrucción

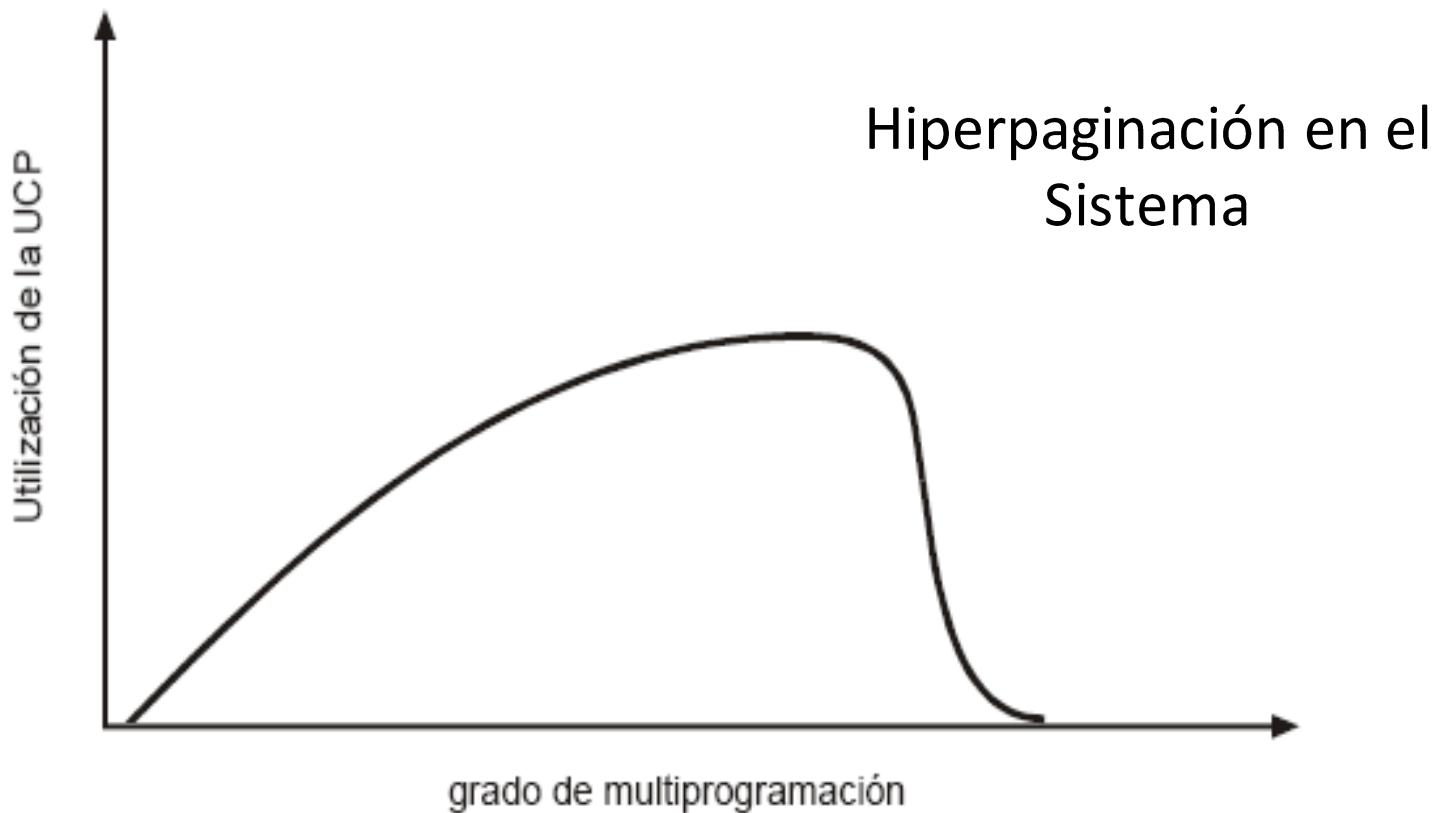
# Memoria Virtual

- **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

# Memoria Virtual

- **Hiperpaginación (Trashing)**
  - Tasa excesiva de fallos de página de un proceso o en el sistema
  - Con asignación fija: Hiperpaginación en Pi
    - Si conjunto residente de Pi < conjunto de trabajo Pi
  - Con asignación variable: Hiperpaginación en el sistema
    - Si nº marcos disponibles < S conjuntos de trabajo de todos
    - Grado de utilización de UCP 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?

# Memoria Virtual



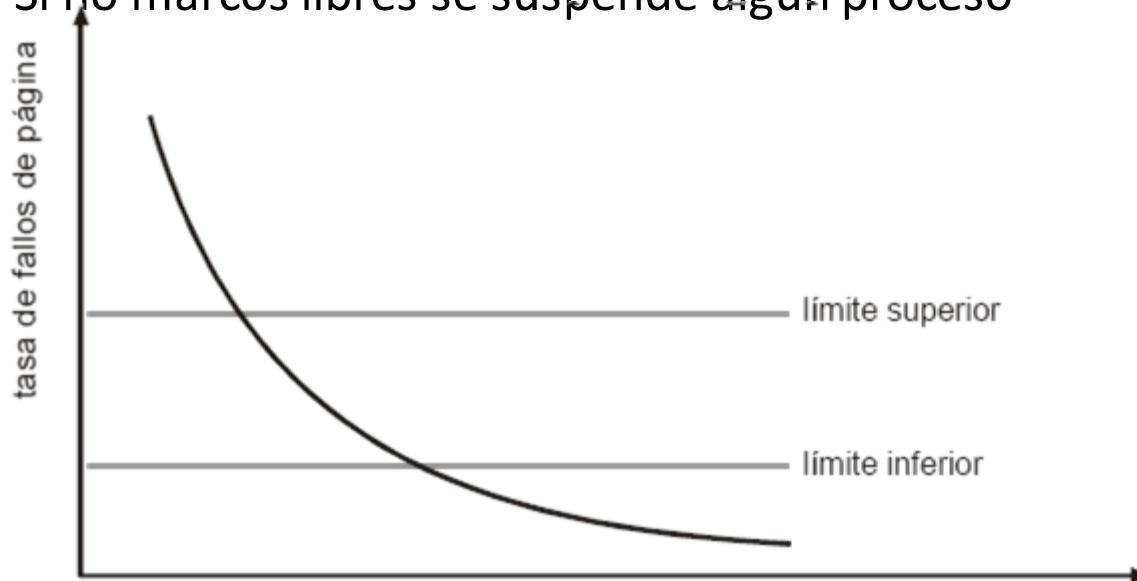
# Memoria Virtual

- **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

# Memoria Virtual

- **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



# Memoria Virtual

- **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

# Memoria Virtual

- **Dispositivo de Paginación (Swap)**
  - Preasignación de espacio
    - Cuando se crea una región se le asigna espacio en *swap*
    - Al expulsar una página siempre tiene espacio reservado
    - Permite detectar sincronamente la falta de espacio de *swap*
  - Sin preasignación de espacio
    - No se asigna espacio en *swap* al crear la región
    - En primera expulsión se le asigna espacio en *swap*
    - Página que nunca se expulsa no gasta espacio en *swap*
    - Mejor aprovechamiento de espacio de almacenamiento
  - Mientras no se modifique la página no va a *swap*
  - Regiones compartidas no usan normalmente el *swap*
  - Algunos sistemas permiten añadir dispositivos de paginación dinámicamente e incluso usar archivos como *swap*
    - Acceso a archivos más lento que acceso a dispositivos

# Memoria Virtual

- **Operaciones sobre Regiones con M. Virtual**
  - Creación de región
    - Al crear mapa inicial o por solicitud posterior
  - Liberación de región
    - Al terminar el proceso o por solicitud posterior
  - Cambio de tamaño de región
    - Del heap o de la pila
  - Duplicado de región
    - Operación requerida por el servicio FORK de POSIX

# Memoria Virtual

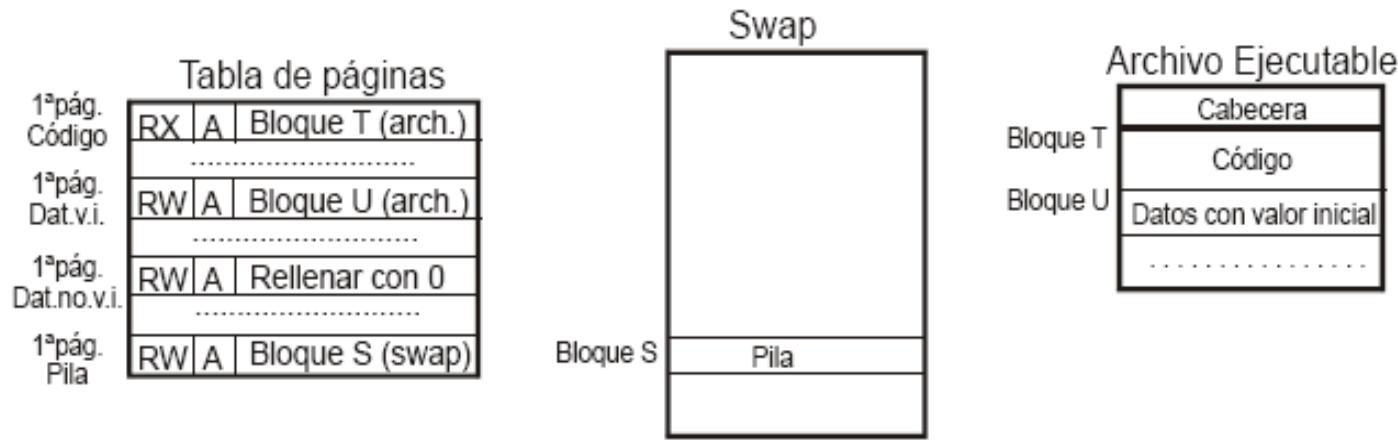
- **Creación de Región**

- Nueva región no se le asigna m. principal (carga por demanda)
- Págs. de región se marcan como inválidas pero válidas para S.O.
- S.O. actualiza t. de regiones y entradas de t. de págs. Asociadas
  - Debe buscar hueco en mapa para asignar a nueva región
- Dependiendo del tipo de soporte:
  - Soporte en archivo
    - Páginas se marcan como *Cargar de archivo (CA)*
    - Se almacena dirección del bloque del archivo correspondiente
  - Sin soporte
    - Por seguridad, págs. se marcan como *Rellenar con ceros (RC)*
    - Fallo de página no implica lectura de dispositivo
- Si privada y preasignación de *swap*, se reserva espacio de *swap*
- Pila es “especial”: debe contener argumentos del programa
  - Típicamente se copian args. en bloque(s) de *swap*

# Memoria Virtual

- **Creación de Región**
  - Una vez creada región, cuando se expulsa página modificada
    - Si la región es privada se escribe página en *swap*
      - Si no preasignación de swap en 1<sup>a</sup> expulsión se reserva espacio
      - Posteriores fallos se sirven de ese bloque de *swap*
    - Si la región es compartida se escribe página en soporte
      - Todos los fallos se sirven del soporte
  - En creación de mapa inicial (EXEC en POSIX):
    - Se crean regiones según sus características
      - Código: CA, Compartida, RX
      - Datos v. inicial: CA, Privada, RW
      - Datos sin v. inicial: RC, Privada, RW
      - Pila inicial: contenido inicial en swap
    - Huecos se marcan como inválidos también para S.O.

# Memoria Virtual



Creación del Mapa Inicial

# Memoria Virtual

- **Liberación de Región**
  - Actualizar tabla de regiones para eliminar región
  - Marcar como inválidas páginas asociadas
  - Si región privada, se libera espacio de *swap*
  - La liberación puede deberse a:
    - Solicitud explícita (p.ej. desproyección de región)
    - Finalización del proceso (EXIT en POSIX)
    - EXEC de POSIX libera mapa actual del proceso antes de construir nuevo mapa vinculado a ejecutable

# Memoria Virtual

- **Cambio de Tamaño**
  - Si disminuye, similar a liberación pero sólo parte afectada
    - Ajusta t. regiones, marca págs. como inválidas y libera swap
  - Si aumenta tamaño:
    - Comprobar no solapamiento
    - Fijar nuevas páginas como no residentes y con mismas características que otras páginas de la región
    - Si preasignación: reserva espacio en *swap* para nuevas págs.
  - Expansión del heap
    - Solicitudada por programa mediante servicio de S.O.
    - Marca páginas correspondientes como RC, privadas, RW
  - Expansión de pila más compleja: es “automática”

# Memoria Virtual

- **Expansión de la Región de Pila**
  - Expansión “automática”
    - Programa disminuye valor de SP y accede a zona expandida
      - Fallo de página
  - Tratamiento de fallo de página:
    - Si dirección realmente inválida
      - Si dirección < SP → Aborta proceso o le manda señal
      - Si no → Expansión de pila
  - Expansión de pila:
    - S.O. verifica que no hay solapamiento
    - Marca páginas como RC, privadas, RW
    - Si preasignación: reserva espacio en *swap*
  - Sutileza: Siempre debe haber al menos una página inválida entre pila y región más cercana

# Memoria Virtual

- **Duplicado de una Región**
  - Necesario para FORK de UNIX
    - Duplicar regs. privadas del padre y compartir las no privadas
  - Copia de región de un proceso en mapa de otro
    - Operación costosa: se debe copiar contenido
  - FORK sería muy ineficiente
    - Además, en la mayoría de los casos el mapa duplicado del hijo apenas se usa ya que después viene un EXEC
  - Optimización: *copy-on-write* (COW)
    - Se comparte una página mientras no se modifique
    - Si un proceso la modifica se crea una copia para él
    - “Duplicado por demanda”

# Memoria Virtual

- **Copy-on-Write**
  - Implementación de COW
    - Se comparten páginas de regiones duplicadas pero:
      - se marcan de sólo lectura y con bit de COW
      - primera escritura → Fallo de protección → copia privada
    - Puede haber varios procesos con misma región duplicada
      - Existe un contador de uso por página
      - Cada vez que se crea copia privada se decrementa contador
      - Si llega a 1, se desactiva COW ya que no hay duplicados
  - FORK con COW
    - Se comparten todas las regiones
    - Las regiones privadas se marcan como COW en padre e hijo
  - Resultado de la optimización del FORK:
    - En vez de duplicar espacio de memoria sólo se duplica la TP

# Índice

- Generalidades
- Objetivos del sistema de gestión de memoria
- Requerimientos para manejo de memoria
- Particionamiento de memoria
- Paginación y segmentación
- Memoria virtual
- Archivos proyectados en memoria

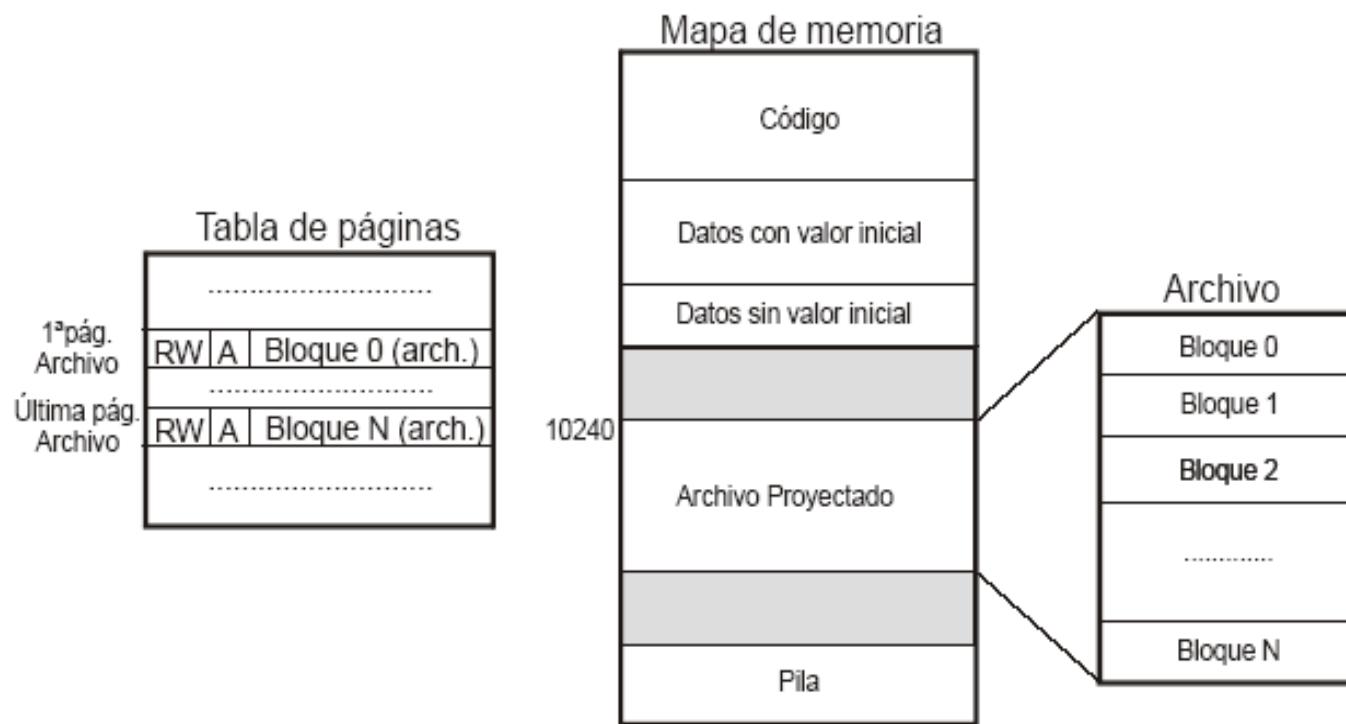
# Archivos Proyectados en Memoria

- Generalización de memoria virtual
  - Entradas de TP referencian a un archivo de usuario
- Programa solicita proyección de archivo (o parte) en su mapa
  - Puede especificar protección y si privada o compartida
- S.O. rellena entradas correspondientes con:
  - No residente, CA
  - Privada/Compartida y Protección según especifica la llamada
- Cuando programa accede a posición de memoria asociada a archivo proyectado, está accediendo al archivo

# Archivos Proyectados en Memoria

- Forma alternativa de acceso a archivos frente a *read/write*
  - Menos llamadas al sistema
  - Se evitan copias intermedias en cache del sistema de archivos
  - Se facilita programación ya que una vez proyectado se accede a archivo como a estructuras de datos en memoria
- Se usa para carga de bibliotecas dinámicas
  - La zona de código se proyecta como compartida
  - La zona de datos con valor inicial se proyecta como privada

# Archivos Proyectados en Memoria



<b>Technique</b>	<b>Description</b>	<b>Strengths</b>	<b>Weaknesses</b>
<b>Fixed Partitioning</b>	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.
<b>Dynamic Partitioning</b>	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.
<b>Simple Paging</b>	Main memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames.	No external fragmentation.	A small amount of internal fragmentation.
<b>Simple Segmentation</b>	Each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.	No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning.	External fragmentation.
<b>Virtual Memory Paging</b>	As with simple paging, except that it is not necessary to load all of the pages of a process. Nonresident pages that are needed are brought in later automatically.	No external fragmentation; higher degree of multiprogramming; large virtual address space.	Overhead of complex memory management.
<b>Virtual Memory Segmentation</b>	As with simple segmentation, except that it is not necessary to load all of the segments of a process. Nonresident segments that are needed are brought in later automatically.	No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support.	Overhead of complex memory management.