T3-Memoria

Licencia

- Este documento puede contener partes de las transparencias de la asignatura Sistemas Operativos del plan de estudios 2003 de la Facultat d'Informàtica de Barcelona
- Este documento puede contener partes de las transparencias que se proporcionan con el libro:
  - Operating Systems Conceps 8th edition. Silberschatz, Galvin and Gagne ©2009

Índice

Memoria física vs. Memoria lógica
Espacio de direcciones de un proceso
Asignación de direcciones a un proceso
Tareas del Sistema operativo en la gestión de memoria
Soporte del hardware a la gestión de memoria

CONCEPTOS

Memoria física vs. Memoria lógica

- CPU sólo puede acceder directamente a memoria y registros
  - Instrucciones y datos deben cargarse en memoria para poder referenciarse
  - Carga: reservar memoria, escribir en ella el programa y pasar la ejecución al punto de entrada del programa
- Tipos de direcciones:
  - Referencia emitida por la CPU: @ lógica
  - Posición ocupada en memoria: @ física
  - No tienen por qué coincidir si el **SO** y el **HW** ofrecen soporte para la **traducción**
    - Los sistemas de propósito general actuales lo ofrecen

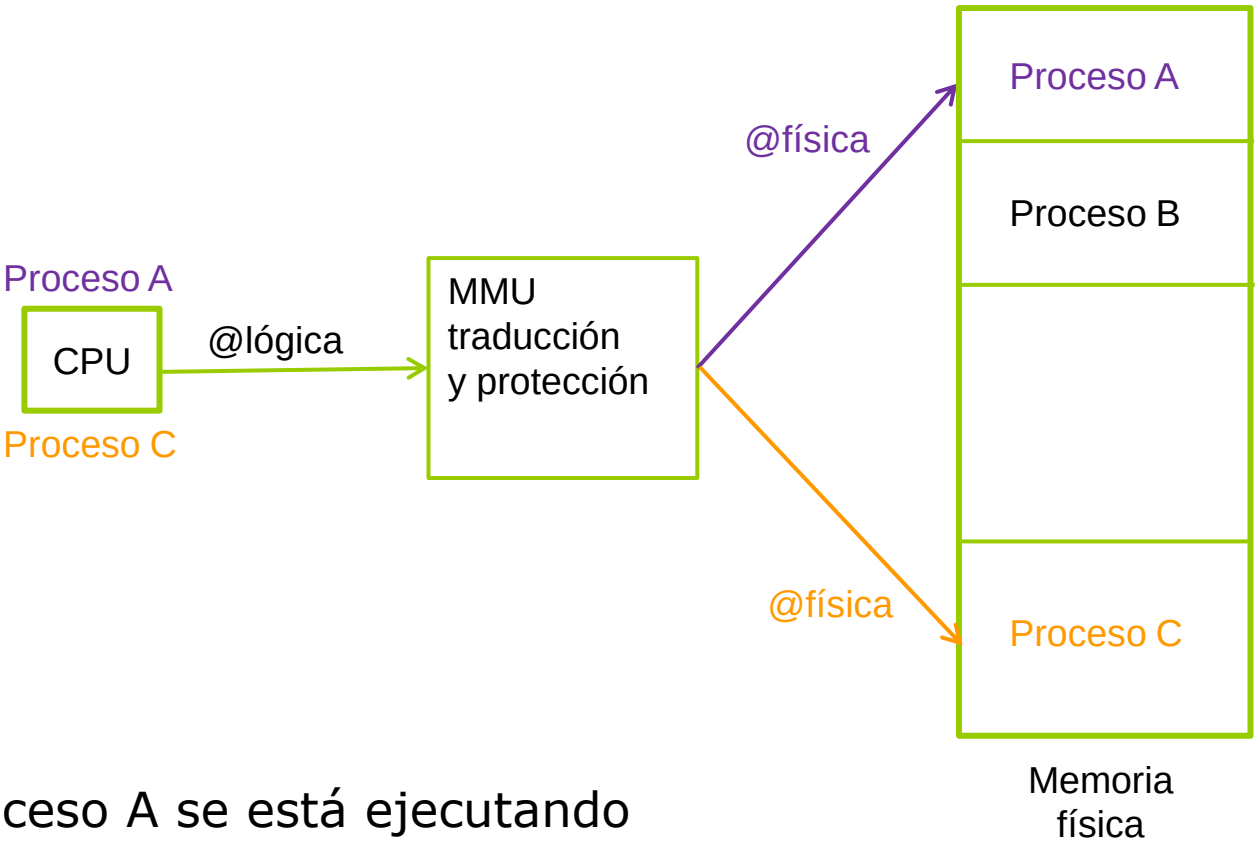
Espacio de @ de un proceso

- Espacio de **direcciones del procesador**
  - Conjunto de @ que el procesador puede emitir
- Espacio de **direcciones lógicas de un proceso**
  - Conjunto de @ lógicas que un proceso puede referenciar
- Espacio de **direcciones físicas de un proceso**
  - Conjunto de @ físicas asociadas al espacio de direcciones lógicas del proceso
- Correspondencia entre @ lógicas y @ físicas
  - Fija
    - Espacio de @ lógicas == Espacio de @ físicas
  - Mecanismo de **traducción en tiempo de ejecución**
    - **Colaboración entre HW y SO**
      - HW ofrece el mecanismo
        - Memory Management Unit (MMU)
      - SO lo configura

Sistemas multiprogramados

- Sistemas multiprogramados
  - Varios programas cargados en memoria física simultáneamente
  - Facilita la ejecución concurrente y simplifican el cambio de contexto
    - **1 proceso en CPU pero N procesos en memoria física**
    - Al hacer cambio de contexto no es necesario cargar de nuevo el proceso que ocupa la cpu
  - SO debe **garantizar protección** de la memoria física
    - Cada proceso sólo debe acceder a la memoria física que tiene asignada
    - **Colaboración entre SO y HW**
      - MMU ofrece el mecanismo para detectar accesos ilegales
      - SO configura la MMU
  - Al hacer cambio de contexto el SO debe actualizar la MMU con la información del nuevo proceso

Sistemas multiprogramados



1-Proceso A se está ejecutando
2-Cambio de contexto a C

Asignación de @ a un programa (I)

- OPCIÓN 1: **Asignación** de @ a instrucciones y datos **en tiempo de compilación**
  - Código absoluto
    - @ física == @ lógica → No hay traducción en tiempo de ejecución
  - Ubicación estática
    - Siempre mismas posiciones en memoria
    - Se fijan esas posiciones en tiempo de compilación
      - Conflictos en sistemas multiprogramados
        - Determinar posiciones que ocupará en memoria física en tiempo de compilación

Asignación de @ a un programa (II)

- OPCIÓN 2: **Asignación** de @ a instrucciones y datos **en tiempo de carga**
  - Reubicación estática: **se reescribe el espacio de @ lógico en tiempo de carga**
  - @ físicas == @ lógicas → No hay traducción en tiempo de ejecución
  - Facilita carga en sistemas multiprogramados pero es poco flexible

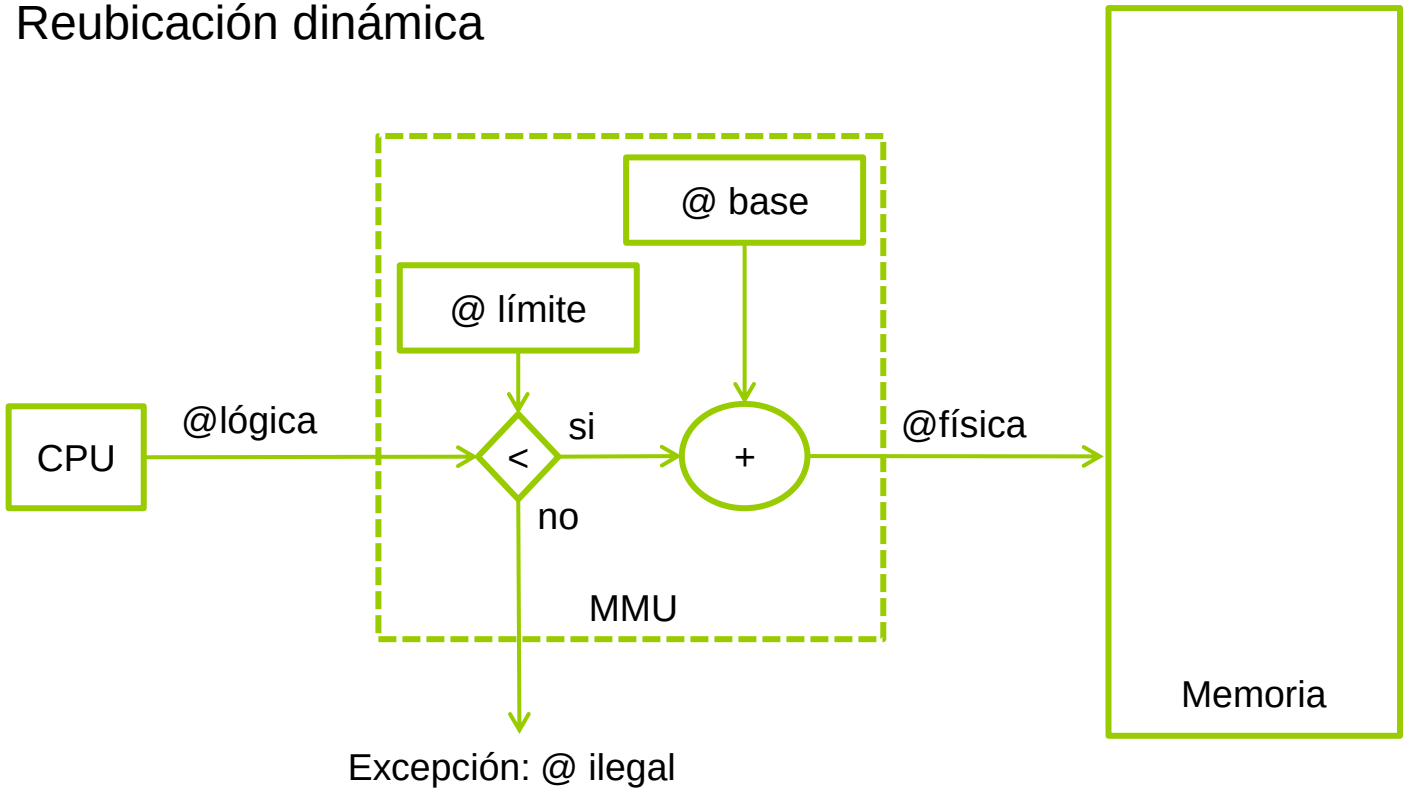
- OPCIÓN 3: **Asignación** de @ a instrucciones y datos **en tiempo de ejecución**
  - @ físicas != @ lógicas → requiere traducción en tiempo de ejecución
  - Procesos pueden cambiar de posición en memoria sin modificar su espacio lógico de @
  - Ejemplo: reubicación dinámica
    - Espacio de @ lógico y físico contiguo
    - @ lógicas relativas al inicio del programa

Soporte HW (I)

- Complejidad del HW necesario depende de la complejidad de la gestión que ofrece el SO
- Como mínimo ofrece soporte a la traducción y a la protección pero puede ser necesario para otras tareas de gestión
- SO es el responsable de configurar la MMU con los valores correspondientes al proceso en ejecución
  - Qué @ lógicas son válidas y con qué @ físicas se corresponden
  - Asegura que cada proceso sólo tiene asociadas sus @ físicas
- Soporte a la traducción y a la protección entre procesos
  - MMU recibe @ lógica y usa sus estructuras de datos para traducirla a la @ física correspondiente
    - Si la @ lógica no está marcada como válida o no tiene una @ física asociada genera una excepción para avisar al SO
- SO gestiona la excepción en función del caso
  - Por ejemplo, si la @ lógica no es válida puede eliminar al proceso

Soporte HW (II): ejemplo

- Reubicación dinámica



Tareas del SO en la gestión de memoria

- Carga de programas en memoria
- Memoria dinámica
- Soporte a la asignación de memoria
- Soporte a la traducción de direcciones
- Soporte a la protección y compartición de memoria entre procesos
- Servicios para la optimización del uso de memoria
  - COW
  - Memoria virtual
  - Prefetch

Carga de un programa
Memoria dinámica
Asignación de memoria
Soporte a la traducción de direcciones
Soporte a la protección y compartición de memoria entre procesos

SERVICIOS BÁSICOS DEL SO

Servicios básicos: carga de programas (I)

- Ejecutable debe estar en memoria para ser ejecutado
- SO debe
  - Interpretar el formato del ejecutable
  - Preparar el esquema del proceso en memoria lógica y **asignar memoria** física
    - **Inicializar estructuras de datos** del proceso
      - Descripción del espacio lógico
        - Qué @ lógicas son válidas
        - Qué tipo de acceso es válido
      - Información necesaria para configurar la MMU cada vez que el proceso pasa a ocupar la CPU
    - **Inicializar MMU**
  - **Leer secciones del programa** del disco y escribir memoria
  - Cargar registro **program counter** con la dirección de la instrucción definida en el ejecutable como **punto de entrada**
- Optimizaciones
  - Carga bajo demanda
  - Librerías compartidas y enlace dinámico
- En Linux se provoca cuando un proceso muta (**exec)**

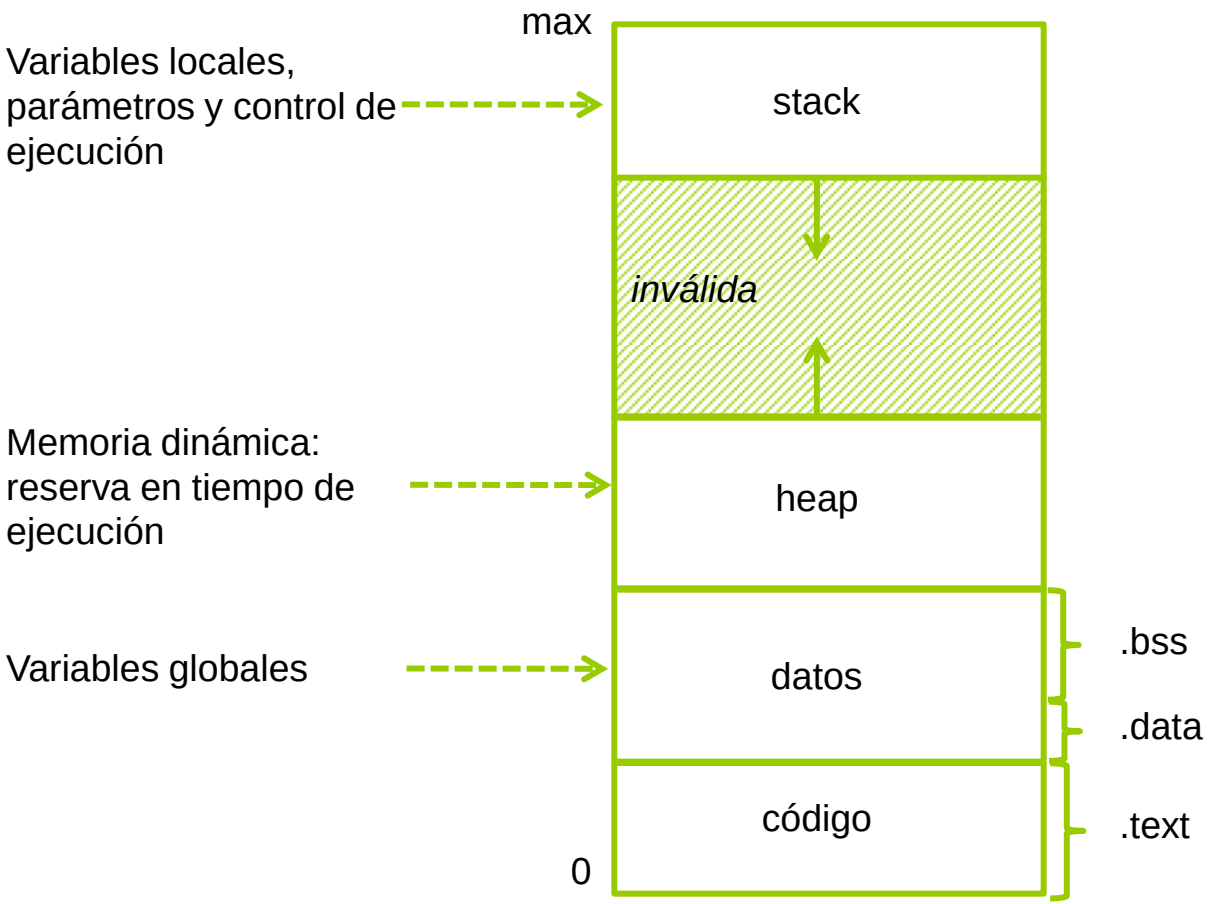
Servicios básicos:carga de programas (II)

- Formato del ejecutable
  - Cabecera del ejecutable define las secciones: tipo, tamaño y posición dentro del binario (podéis probar *objdump –h programa*)
  - Existen diferentes formatos de ejecutable
    - ELF (*Executable and Linkable Format)*: es el más extendido en sistemas POSIX

Algunas secciones por defecto de un ejecutable ELF	
.text	código
.data	datos inicializados
.bss	datos sin valor inicial
.debug	información de debug
.comment	información de control
.dynamic	información para enlace dinámico
.init	código de inicialización del proceso (contiene la @ de la 1ª instrucción)

Servicios básicos: carga de programas (III)

- Preparar el esquema del proceso en memoria lógica
  - Esquema habitual



Servicios básicos: carga de programas (IV)

- Optimizaciones: **carga bajo demanda**
  - **Una rutina no se carga hasta que se llama**
  - Se aprovecha mejor la memoria ya que no se cargan funciones que no se llaman nunca (por ejemplo, rutinas de gestión de errores)
  - Se acelera el proceso de carga
  - Hace falta un mecanismo que detecte si las rutinas no están cargadas. Por ejemplo:
    - SO:
      - Registra en sus estructuras de datos que esa zona de memoria es válida y de dónde leer su contenido
      - En la MMU no le asocia una traducción
    - Cuando el proceso accede a la @, la MMU genera una excepción para avisar al SO de un acceso a una @ que no sabe traducir
    - SO comprueba en sus estructuras que el acceso es válido, provoca la carga y reanuda la ejecución de la instrucción que ha provocado la excepción

Servicios básicos: carga de programas (V)

- Optimizaciones: **librerías compartidas y enlace dinámico**
  - **Se retrasa el enlace hasta el momento de ejecución**
  - Ahorra espacio en disco
    - Los binarios no contienen el código de las librerías dinámicas
  - Ahorra espacio en memoria
    - Los procesos pueden compartir la zona en memoria que contiene el código de las librerías comunes
      - SO lo debe tener en cuenta al implementar el mecanismo de protección
  - Facilita la actualización de los programas para que usen las nuevas versiones de las librerías de sistema
    - No hace falta regenerar ejecutables
  - Mecanismo
    - Binario contiene el código de una rutina *stub*
      - Comprueba si algún proceso ya ha cargado la rutina de la librería compartida y la carga si no es así
      - Substituye la llamada a sí misma por la llamada a la rutina de la librería compartida

Memoria dinámica (I)

- Estructuras de datos cuyo tamaño depende de parámetros de la ejecución
  - Fijar el tamaño en tiempo de compilación no es adecuado
    - Se desaprovecha memoria
    - O se tiene error de ejecución por no haber reservado suficiente
- Los SO ofrecen llamadas a sistema para validar nuevas regiones de memoria en tiempo de ejecución: **memoria dinámica**
  - Se almacena en la zona heap del espacio lógico de @
- Implementación
  - SO actualiza su estructura de datos con la nueva región
  - Puede retrasar el momento de asignar @ físicas hasta que se intente escribir en la región
    - Se asigna temporalmente una zona inicializada con 0 para resolver lecturas
  - Actualiza la MMU en función de la política de asignación de memoria que siga
  - El interfaz puede definir que la región está inicializada con 0 o no

Memoria dinámica (II)

- Linux sobre Pentium
  - Interfaz tradicional de Unix poco amigable
    - brk y sbrk
    - Permiten modificar el límite del heap
    - Programador es responsable de controlar posición de cada variable en el heap
  - También se puede usar la llamada a sistema mmap
    - Pensada para mapear ficheros en memoria y acceder a ellos a través del espacio lógico de direcciones
    - Se puede usar para pedir memoria dinámica *anónima* (zona de memoria no respaldada por fichero)

Memoria dinámica (III)

- Librerías de soporte a la programación optimizan y facilitan la gestión del heap
  - Librería de C: malloc, calloc, realloc free…
    - Validan la nueva región y devuelven la @ lógica inicial
    - La región validada puede ser mayor que la pedida por el programador aunque la librería registra qué zona es la que está usando el programador
    - Mantiene listas de regiones libres en la zona de heap validada para intentar satisfacer peticiones sin recurrir al sistema
    - **Modifica el puntero límite del heap sólo si es necesario** para satisfacer una petición o si es posible reducir el tamaño del heap
    - **Programador debe liberar la región cuando ya no sea necesaria (free)**
      - Cuando el programador libera una región se decide si simplemente pasa a formar parte de la lista de regiones libres o si es adecuado reducir el tamaño del heap

Memoria dinámica (IV): ejemplos

- Qué diferencias a nivel de *heap* observáis en los siguientes ejemplos?
  - Ejemplo 1:

```
...
new = sbrk(1000);
...
```

  - Ejemplo 2:

```
...
new = malloc(1000);
...
```

- Cambia el tamaño del *heap* en los dos casos?

Memoria dinámica (V): ejemplos

- Qué diferencias a nivel de *heap* observáis en los siguientes ejemplos?
  - Ejemplo 1:

```
...
ptr = malloc(1000);
...
```

  - Ejemplo 2:

```
...
for (i = 0; i < 10; i++)
    ptr = malloc(100);
...
```

- Se reservan las mismas posiciones de memoria lógica?

Memoria dinámica (VI): ejemplos

- Qué errores contienen los siguientes fragmentos de código?
- Código 1:

```
...
for (i = 0; i < 10; i++)
    ptr = malloc(SIZE);

// uso de la memoria
// ...

for (i = 0; i < 10; i++)
    free(ptr);
...
```

- Código 2:

```
int *x, *ptr;


...
ptr = malloc(SIZE);
...
x = ptr;
...
free(ptr);

printf(..., *x);
```

- El código 2 produce error siempre?

Servicios básicos: asignación de memoria (I)

- Se ejecuta cada vez que un proceso necesita memoria física:
  - En linux: creación (fork), mutación del ejecutable (exec), uso de memoria dinámica, implementación de alguna optimización (carga bajo demanda, memoria virtual, COW…).
- Pasos
  - **Seleccionar memoria física libre** y marcarla como ocupada en las estructuras de datos del SO
  - Actualizar MMU con el mapeo @ lógicas - @ físicas
    - Necesario para implementar la **traducción de direcciones**
- Posible problema: fragmentación de memoria

Servicios básicos: asignación de memoria (II)

- Fragmentación de memoria: memoria que está libre pero no se puede usar para un proceso
  - **Fragmentación interna**: memoria asignada a un proceso aunque no la necesita
  - **Fragmentación externa**: memoria libre y no asignada pero no se puede asignar por no estar contigua
    - Se puede evitar compactando la memoria libre si el sistema implementa asignación de @ en tiempo de ejecución
      - Costoso en tiempo

Servicios básicos: asignación de memoria (III)

- Primera aproximación: **asignación contigua**
  - Espacio de @ físicas contiguo
    - Todo el proceso ocupa una partición que se selecciona en el momento de la carga
  - Poco flexible y dificulta aplicar optimizaciones (como carga bajo demanda)
- **Asignación no contigua**
  - Espacio de @ físicas no contiguo
  - Aumenta flexibilidad
  - Aumenta la granularidad de la gestión de memoria de un proceso
  - Aumenta complejidad del SO y de la MMU
- Basada en
  - **Paginación** (particiones fijas)
  - **Segmentación** (particiones variables)
  - Esquemas combinados
    - Por ejemplo, segmentación paginada

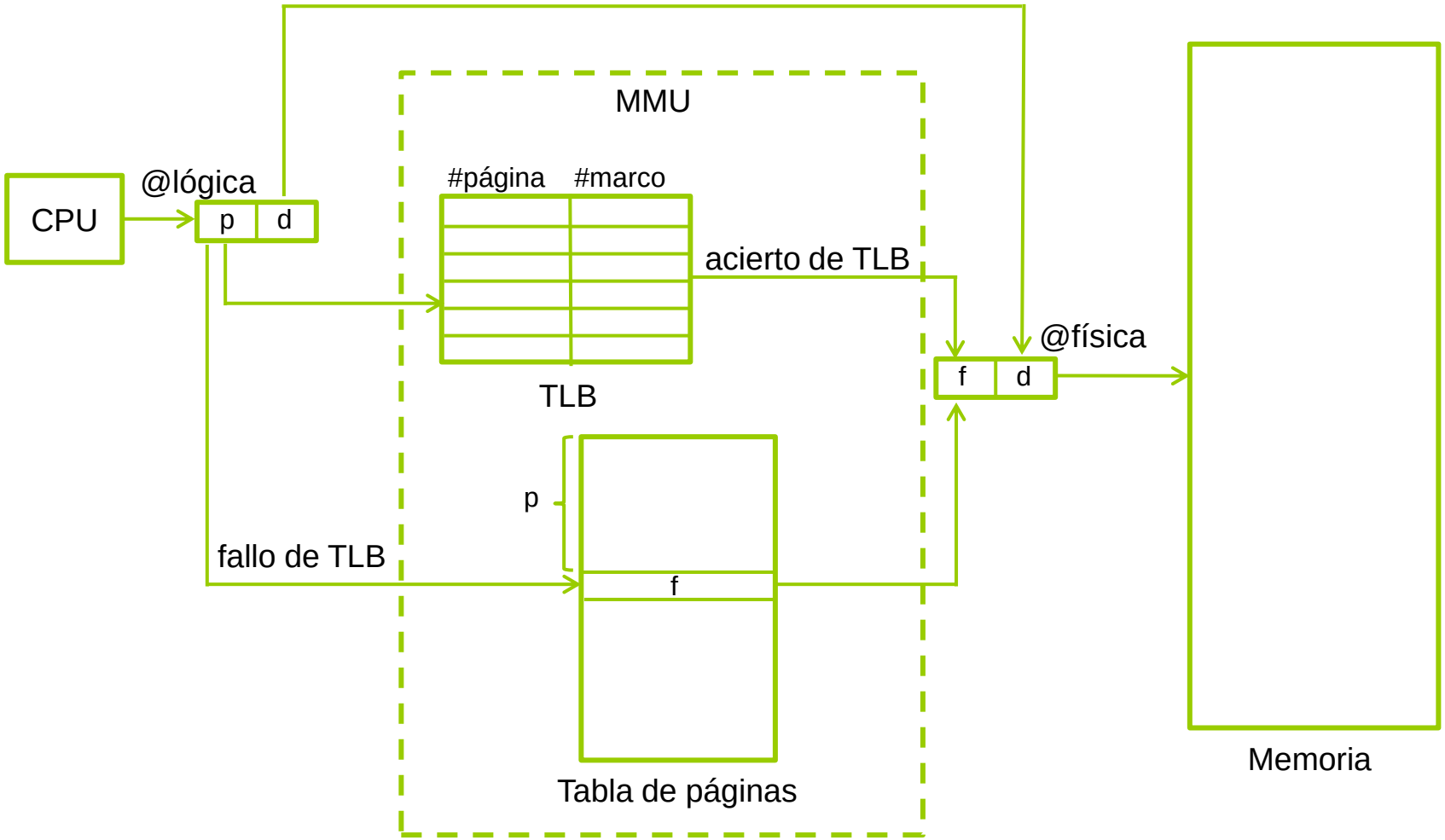
Servicios básicos:asignación de memoria (IV)

- **Esquema basado en paginación**
  - Espacio de @ lógicas dividido en particiones de tamaño fijo: **páginas**
  - Memoria física dividida en particiones del mismo tamaño: **marcos**
  - Asignación
    - Para cada página del proceso buscar un marco libre
      - Lista de marcos libres
    - Puede haber fragmentación interna
  - Cuando un proceso acaba la ejecución devolver los marcos asignados a la lista de libres
  - **Página: unidad de trabajo del SO**
    - Facilita la carga bajo demanda
    - Permite especificar protección a nivel de página
      - Facilita la compartición de memoria entre procesos

Servicios básicos:asignación de memoria (V)

- MMU
  - **Tabla de páginas**
    - Para mantener información a nivel de página: validez, permisos de acceso, marco asociado, etc….
    - Una entrada para cada página
    - Una tabla por proceso
  - Suele guardarse en memoria y SO debe conocer la @ base de la tabla de cada proceso (por ejemplo, guardándola en el PCB)
  - Procesadores actuales también disponen de TLB (*Translation Lookaside Buffer*)
    - Memoria asociativa de acceso más rápido en la que se almacena la información para las páginas *activas*

Servicios básicos:asignación de memoria(VI)



Servicios básicos:asignación de memoria (VII)

- Tamaño de página potencia de 2
  - Tamaño muy usado 4Kb (2^12)
  - Influye en
    - Fragmentación interna y granularidad de gestión
    - Tamaño de la tabla de páginas
- Tamaño de cada tabla de páginas suponiendo páginas de 4Kb

	Espacio lógico de procesador	Número de páginas	Tamaño TP
Bus de 32 bits	2^32	2^20	4MB
Bus de 64 bits	2^64	2^52	4PB

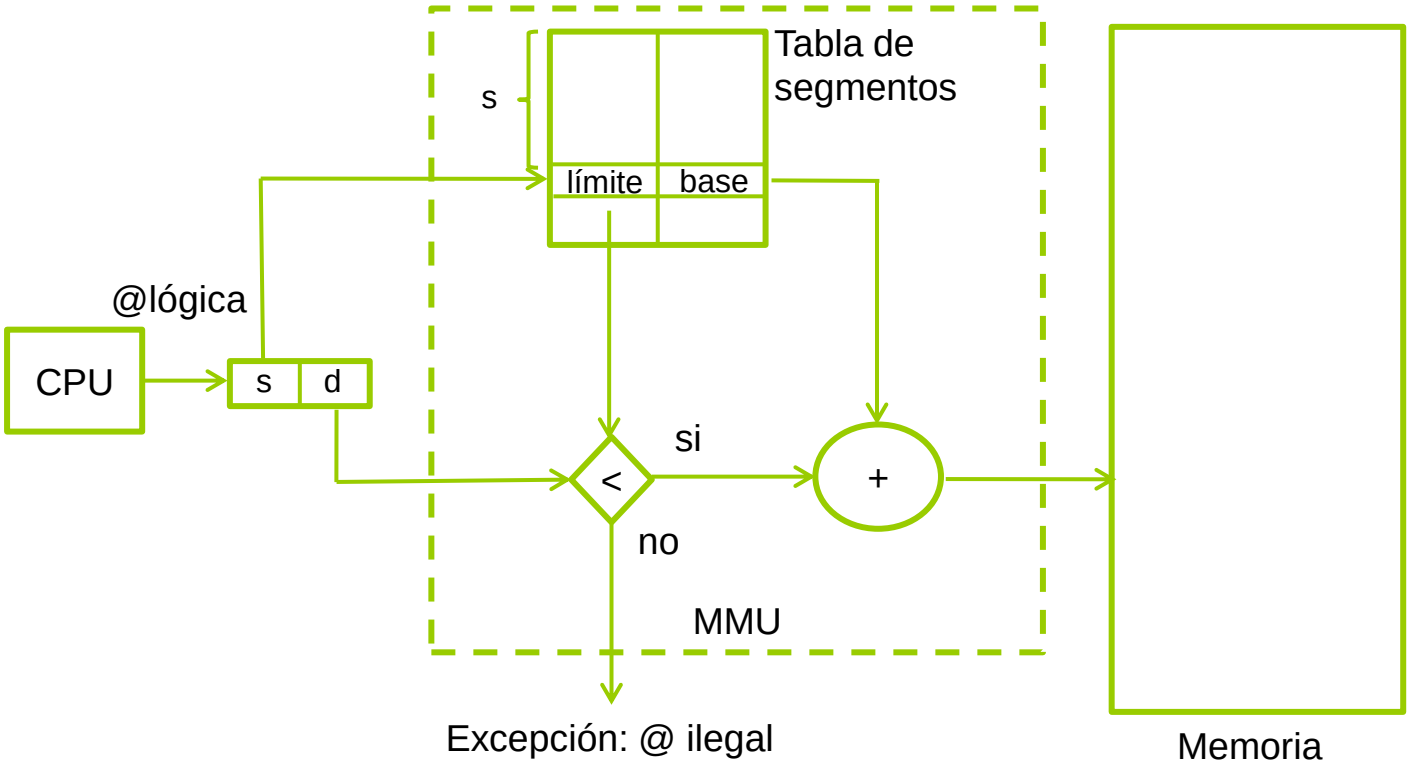
- Esquemas para reducir el espacio ocupado por las TP: TP multinivel
  - TP dividida en secciones y se añaden secciones a medida que crece el espacio lógico de direcciones

Servicios básicos:asignación de memoria (VIII)

- **Esquema basado en segmentación**
  - Se divide el espacio lógico del proceso teniendo en cuenta el tipo de contenido
    - Aproxima la gestión de memoria a la visión de usuario
  - Espacio de @ lógicas dividido en particiones de tamaño variable: **segmentos**
    - Como mínimo un segmento para el código y otro para la pila y los datos
    - Las referencias a memoria que hace el programa están formadas por un segmento y el desplazamiento dentro del segmento
  - Memoria física libre contigua forma una partición disponible
    - Cada partición es de un tamaño diferente
  - Asignación
    - Para cada segmento del proceso
      - Busca una partición en la que quepa el segmento
      - Posible políticas: first fit, best fit, worst fit
      - Selecciona la cantidad de memoria necesaria para el segmento y el resto continúa en la lista de particiones libres
    - Puede haber fragmentación externa
  - Cuando un proceso acaba la ejecución devolver la porción asignada a la lista de libres

Servicios básicos:asignación de memoria (IX)

- MMU
  - Tabla de segmentos
    - Para cada segmento: @ base y tamaño
    - Una tabla por proceso



Servicios básicos:asignación de memoria (X)

- Esquemas combinados: segmentación paginada

  - Espacio lógico del proceso dividido en segmentos
  - Segmentos divididos en páginas
    - Tamaño de segmento múltiplo del tamaño de página
    - Unidad de trabajo del SO es la página

Servicios básicos: traducción

- Traducción de @lógica a @ física se hace por HW
- SO debe configurar la MMU para que contenga las @ físicas que se corresponden con las @lógicas del proceso en ejecución
  - Inicialización al **asignar nueva memoria**
  - En el **cambio de contexto**
    - Para el proceso que abandona la CPU: si aún no ha acabado la ejecución almacenar en las estructuras de datos del proceso la información necesaria para reconfigurar la MMU cuando vuelva a ocupar la CPU
    - Para el proceso que pasa a ocupar la CPU: configurar la MMU
      - Pasar a utilizar las traducciones del proceso actual (si es necesario invalidar la TLB)
- SO debe gestionar las excepciones generadas por la MMU cuando ésta no es capaz de hacer la traducción
  - Permite implementar por ejemplo carga bajo demanda

Servicios básicos: protección

- El espacio físico de un proceso debe estar protegido del resto de procesos o incluso de accesos del propio proceso sobre @ no válidas
  - Al configurar la MMU el SO garantiza que un proceso sólo accede a las páginas físicas que tiene él asociadas

- También se puede implementar protección contra tipos de accesos no deseados
  - Por ejemplo, escribir en una zona de código
  - Soporte HW: debe ofrecer un modo de registrar tipos de accesos permitidos (por ejemplo: lectura, escritura, ejecución, o una combinación de ellos)
    - Por ejemplo, sistemas basados en paginación asocian a cada entrada de la tabla de página los permisos asociados
  - Cuando un proceso accede a una @ lógica la MMU comprueba que el tipo de acceso está permitido para es @. Si no lo está, genera excepción para que la gestione el SO
  - Permite implementar optimización COW

Servicios básicos: compartición (I)

- Compartición de memoria entre procesos
  - Se puede especificar a nivel de página o de segmento
  - Para procesos que ejecutan el mismo código no es necesario varias copias en memoria física (no es modificable)
    - **Librerías compartidas**
  - Los SO proporcionan llamadas a sistema para que un proceso cree zonas de memoria en su espacio lógico que sean compartibles y para que otro proceso la pueda mapear en su espacio de memoria
    - **Memoria compartida** como mecanismo de **comunicación entre procesos**
  - El resto de memoria es **privada** para un proceso y nadie la puede acceder

Servicios básicos: compartición (II)

- Compartición de memoria entre threads
  - Todos los threads pueden acceder a todo el espacio lógico de la tarea a la que pertenecen
  - Cosas a tener en cuenta en la programación con threads
    - Cada thread tiene su pila propia donde el compilador reserva espacio para sus variables locales, parámetros, y control de su ejecución
    - Aunque es accesible por todos los threads (un acceso a esas direcciones no genera una excepción) hay que tener en cuenta la visibilidad de las variables

Servicios básicos: compartición (III)

- Problemas en la compartición de memoria
  - Recurso compartido accedido concurrentemente por varios flujos de ejecución (procesos o threads)
  - **Condición de carrera** (race condition)
    - El resultado de la ejecución depende del orden en el que se alterna el uso de la cpu entre los flujos
    - Puede dar resultados incoherentes

```
/* variable compartida */ int primero =  1;
```

```
cmp primero, 0
je else
```

```
/* Proceso 1 */
if (primero){
        primero --;
        /* tarea uno */
} else {
        /* tarea dos */
}
```

```
/* Proceso 2 */
if (primero){
        primero --;
        /* tarea uno */
} else {
        /* tarea dos */
}
```

Posibles Resultados	Tarea 1	Tarea 2
Resultado 1	Proceso 1	Proceso 2
Resultado 2	Proceso 2	Proceso 1
Resultado 3	**Proceso 1 y Proceso 2**	**Ninguno**

Servicios básicos: compartición (IV)

- Región crítica
  - Zona de código que con condición de carrera que pueden provocar resultados incoherentes
    - Acceso a variables compartidas que son modificables
  - Garantizar acceso en exclusión mutua
    - Sólo un proceso ejecuta código dentro de la región al mismo tiempo
    - Aunque haya cambio de contexto
- SO ofrece mecanismos para implementar acceso en exclusión mutua (por ejemplo, *semáforos*)
  - Permite que el programador marque inicio y fin de región crítica
    - Inicio: si ya hay algún proceso en la región el proceso se bloquea, si no el SO marca que hay alguien y permite que el proceso continúe
    - Fin: SO libera la región crítica
  - Responsabilidad del programador hacerlo bien
    - Evitar inanición: todos los procesos tienen que poder acceder en algún momento
    - Evitar abrazos mortales: provocados por regiones críticas anidadas
    - Maximizar concurrencia

Servicios básicos: compartición (V)

```
/* variable compartida */ int primero =  1;
/* declarar e inicializar mutex */
```

	/* Proceso 1 */	/* Proceso 2 */	
	Inicio_Mutex	**Inicio_Mutex**	
Región Crítica	if (primero){	if (primero){	Región Crítica
	primero --;	primero --;	
	Fin_Mutex	**Fin_Mutex**	
	/* tarea uno */	/* tarea uno */	
	} else {	} else {	
	Fin_Mutex	**Fin_Mutex**	
	/* tarea dos */	/* tarea dos */	
	}	}	

COW

Memoria virtual

Prefetch

SERVICIOS PARA LA OPTIMIZACIÓN DEL USO DE MEMORIA

Optimizaciones: COW

- COW (*Copy on write*)
  - **Retrasar el momento de la copia mientras sólo se acceda en modo lectura**
    - Se puede evitar la copia física si los procesos sólo van a usar la región para leer
  - Se puede aplicar
    - Dentro de un proceso
    - Entre procesos (por ejemplo, fork de Linux)
  - Implementación
    - Al hacer la copia
      - En la estructura de datos que describe el espacio lógico del proceso el SO marca la región destino con los permisos de acceso reales
      - En la MMU el SO marca la región destino y la región fuente con permiso sólo de lectura
      - En la MMU el SO asocia a la región destino las direcciones físicas asociadas también a la región fuente
    - Si un proceso intenta acceder a una de las dos regiones MMU genera excepción y SO la gestiona haciendo la copia real
      - Reserva direcciones físicas nuevas para la región destino, copia la información, actualiza la MMU para las dos regiones y reanuda la instrucción que ha provocado la excepción y que ahora podrá completar el acceso
      - Si el sistema está basado en paginación el COW se aplica página a página

Optimizaciones: Memoria Virtual (I)

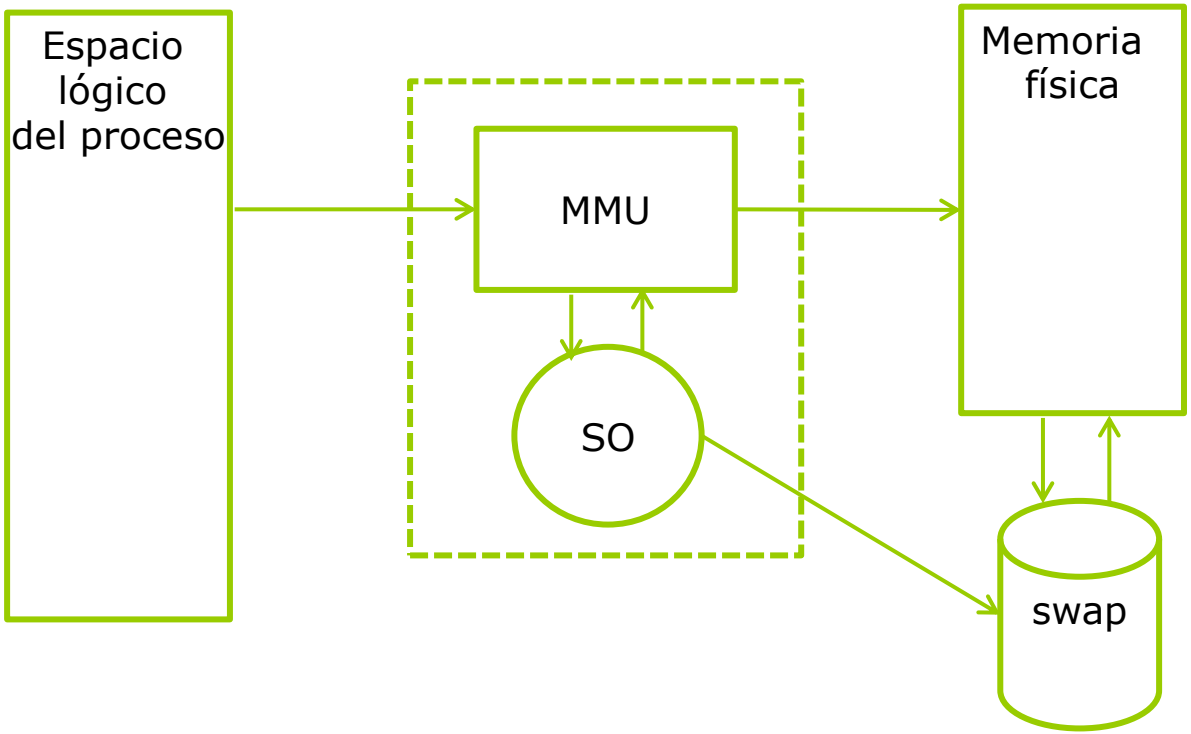
- Memoria virtual
  - Extiende la idea de la carga bajo demanda
  - Objetivo
    - Reducir la cantidad de memoria física asignada a un proceso en ejecución
      - **Un proceso realmente sólo necesita memoria física para la instrucción actual y los datos que esa instrucción referencia**
    - Aumentar el grado de multiprogramación
      - Cantidad de procesos en ejecución simultáneamente

Optimizaciones: Memoria Virtual (II)

- Primera aproximación: intercambio (*swapping*)
  - Idea: sólo hace falta tener en memoria el proceso activo (el que tenía la CPU asignada)
    - Si el proceso activo necesitaba más memoria física que la disponible en el sistema se puede expulsar temporalmente de memoria alguno de los otros procesos cargados (*swap out*)
    - Almacén secundario o de soporte (*backing storage*):
      - Dispositivo de almacenaje en el que se guarda el espacio lógico de los procesos a la espera de volver a ocupar la CPU
        - Mayor capacidad que la que ofrece la memoria física
      - Típicamente una zona de disco: espacio de intercambio (*swap area*)
    - Estado de los procesos: no residentes (*swapped out)*
    - Al asignar la cpu a un proceso no residente es necesario cargarlo en memoria de nuevo antes de permitir que reanude la ejecución
      - Ralentiza la ejecución
  - Evolución de la idea
    - Evitar expulsar de memoria procesos enteros para minimizar la penalización en tiempo de la ejecución
    - Se puede aprovechar la granularidad que ofrece la paginación

Optimizaciones: Memoria Virtual (III)

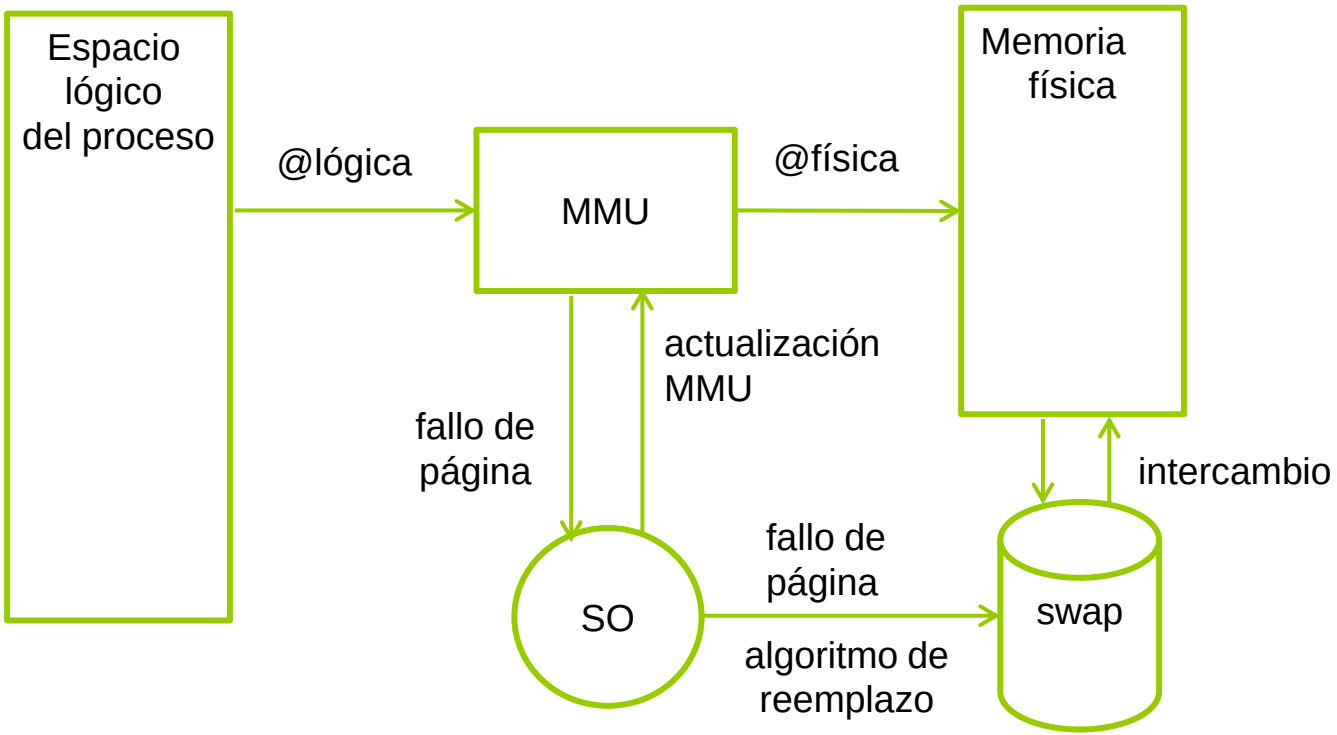
- Memoria virtual basada en paginación
  - **Espacio lógico de un proceso está distribuido entre memoria física** (páginas residentes) **y área de swap** (páginas no residentes)



Optimizaciones: Memoria Virtual (IV)

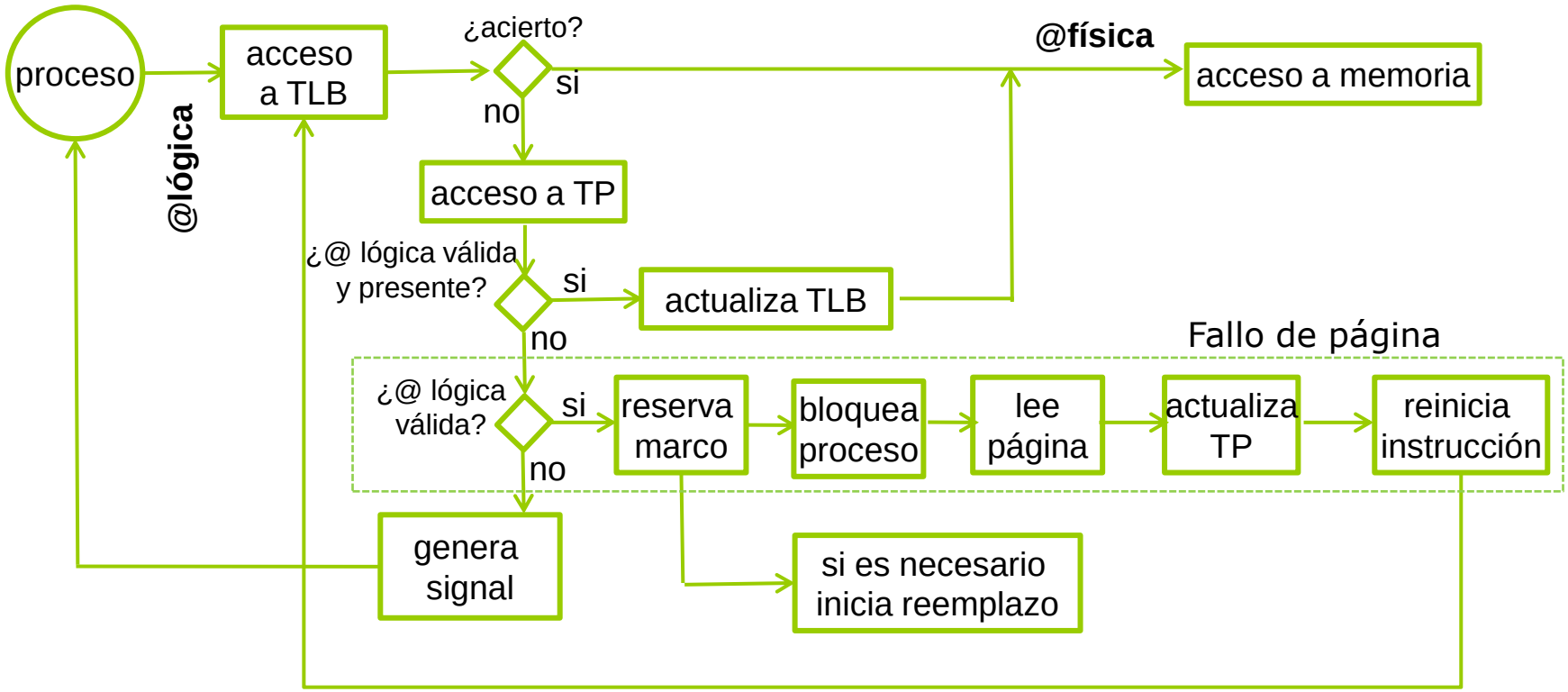
- **Reemplazo de memoria**: cuando SO necesita liberar marcos
  - Selecciona una página *víctima* y actualiza la MMU eliminando su traducción
  - Guarda su contenido en el área de swap para que se pueda recuperar
  - Asigna el marco ocupado a la página que se necesita en memoria
- Cuando se accede a una página guardada en el área de swap
  - MMU no puede hacer la traducción: genera excepción
    - **Fallo de página**
  - SO
    - Comprueba en las estructuras del proceso que el acceso es válido
    - Asigna un marco libre para la página (lanza el reemplazo de memoria si es necesario)
    - Localiza en el área de swap el contenido y lo escribe en el marco
    - Actualiza la MMU con la @física asignada

Optimizaciones: Memoria Virtual (V)



Optimizaciones: Memoria Virtual (VI)

- Pasos en el acceso a memoria



Optimizaciones: Memoria Virtual (VII)

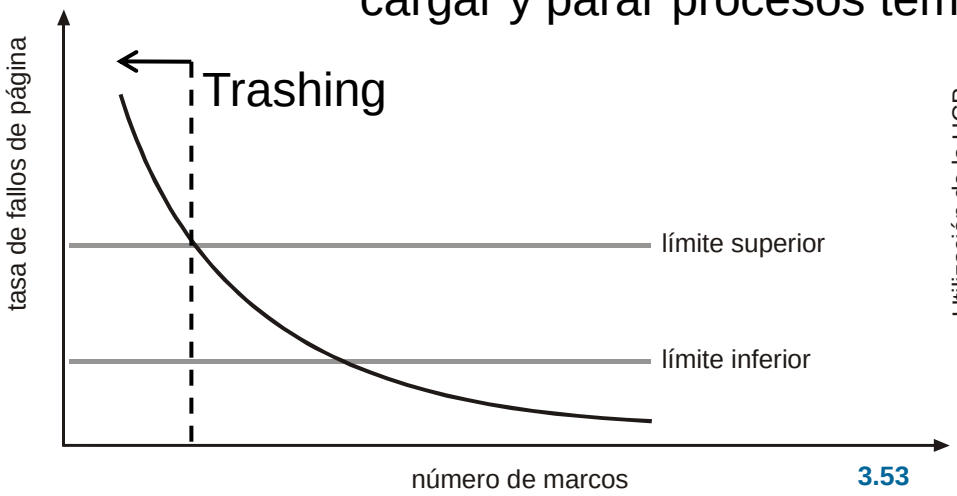
- Efectos del uso de la memoria virtual
  - La suma de los espacios lógicos de los procesos en ejecución puede ser mayor que la cantidad de memoria física de la máquina
  - El espacio lógico de un proceso también puede ser mayor que la memoria física disponible
  - Acceder a una página no residente es más lento que acceder a una página residente
    - Excepción + carga de la página
    - Importante minimizar el número de fallos de página

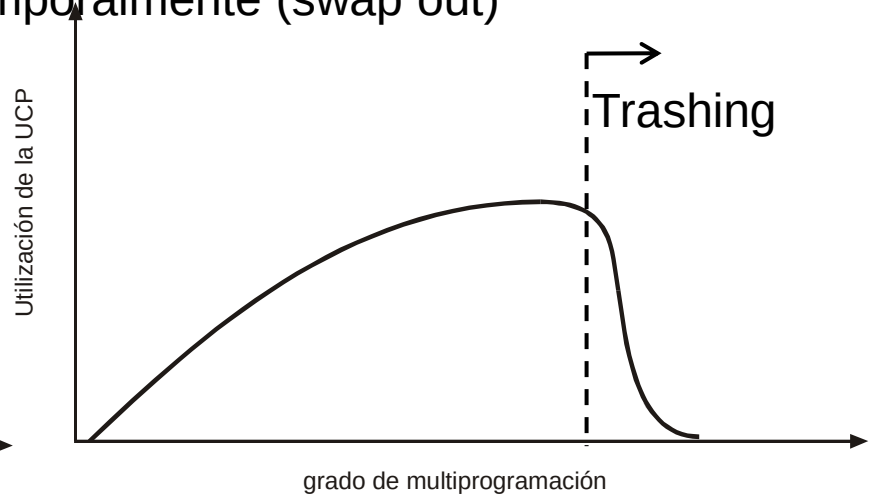
Optimizaciones: Memoria Virtual (VIII)

- Modificaciones en el SO
  - Añadir las estructuras de datos y los algoritmos para gestionar el área de swap
    - Asignación, liberación y acceso
  - Algoritmo de reemplazo
    - ¿Cuándo se ejecuta? ¿Cómo se seleccionan las páginas víctimas? ¿Cuántas páginas víctimas en cada ejecución del algoritmo?
    - Objetivo: minimizar el número de fallos de página y acelerar su gestión
      - **Intentar seleccionar las víctimas entre las páginas que ya no se necesitan o que se va a tardar más tiempo en necesitar**
        - » Ejemplo: Least Receantly Used (LRU) o aproximaciones
      - Intentar que siempre que se da un fallo de página haya un marco disponible
- Modificaciones en la MMU: depende de los algoritmos de gestión de memoria virtual.
  - Por ejemplo, algoritmo de reemplazo puede necesitar un bit de referencia por página

Optimizaciones: Memoria Virtual (IX)

- Sobrepaginación (*thrashing*)
  - Proceso en thrashing
    - **Invierte más tiempo en el intercambio de memoria que avanzando su ejecución**
    - No consigue mantener simultáneamente en memoria el conjunto mínimo de páginas que necesita para avanzar
  - Se debe a que se ha sobrecargado el sistema de memoria
    - Detección: controlar tasa de fallos de página por proceso
    - Tratamiento: controlar el número de procesos que se permiten cargar y parar procesos temporalmente (swap out)





Optimizaciones: Memoria prefetch

- Objetivo: minimizar número de fallos de página
- Idea: anticipar qué páginas va a necesitar el proceso en el futuro inmediato y cargarlas con anticipación
- Parámetros a tener en cuenta:
  - Distancia de prefetch: con qué antelación hay que cargar las páginas
  - Número de páginas a cargar
- Algoritmos sencillos de predicción de páginas
  - Secuencial
  - Strided

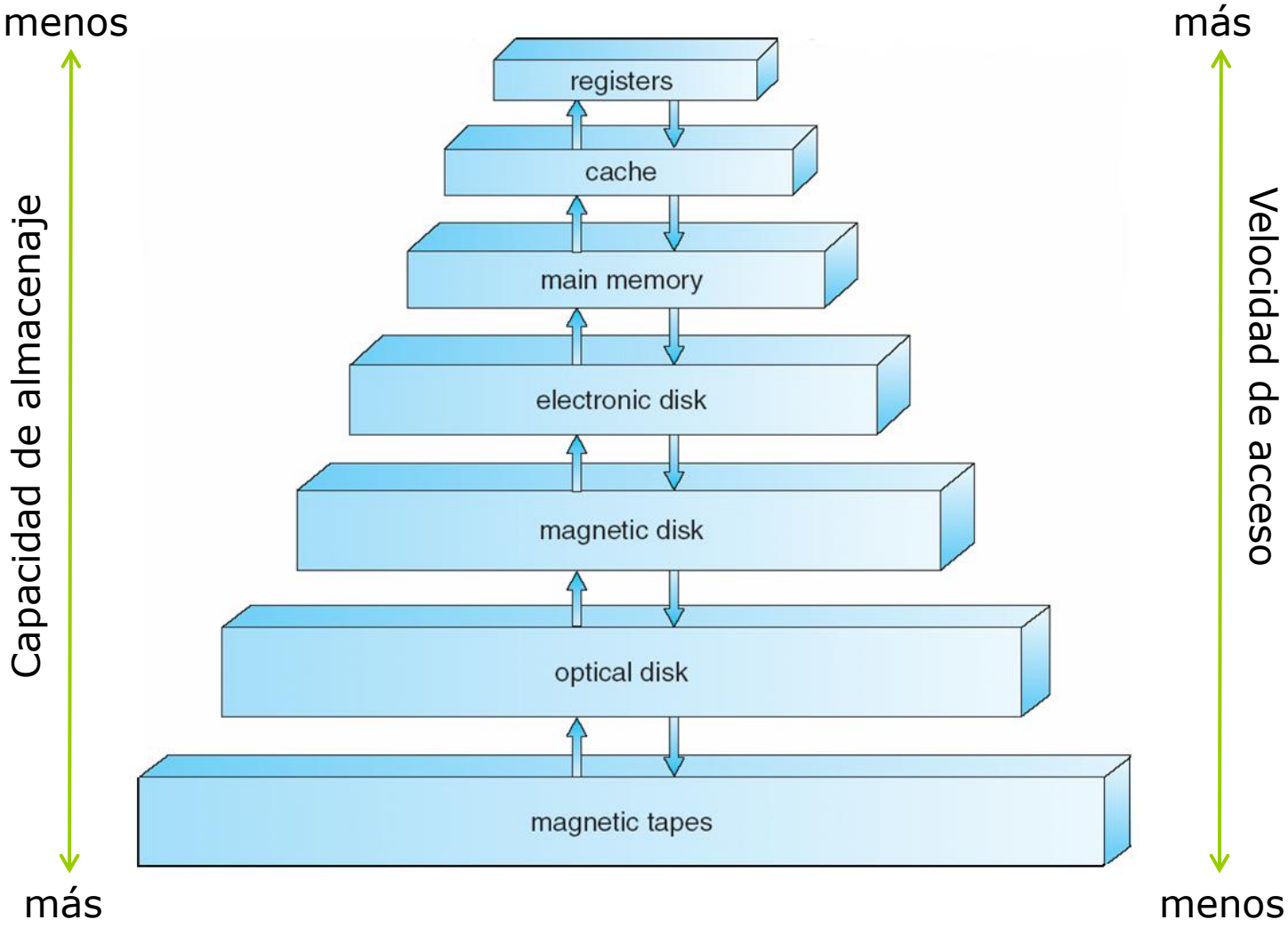
Resumen: Linux sobre Pentium

- Memoria virtual basada en segmentación paginada
  - Tabla de páginas multinivel (2 niveles)
    - Una por proceso
    - Guardadas en memoria
    - Registro de la cpu contiene la @ base de la TP del proceso actual
  - Algoritmo de reemplazo: aproximación de LRU
    - Se ejecuta cada cierto tiempo y cuando el número de marcos libres es menor que un umbral

- Implementa COW
- Carga bajo demanda
- Soporte para librerías compartidas
- Prefetch simple (secuencial)

- Llamada a sistema **exec:** provoca la **carga** de un nuevo programa
  - Inicialización del PCB con la descripción del nuevo espacio de direcciones, asignación de memoria, …
- Creación de procesos (**fork**):
  - Inicialización del PCB con la descripción de su espacio de direcciones (copia del padre)
  - Se utiliza COW: hijo comparte marcos con padre hasta que algún proceso los modifica
  - Creación e inicialización de la TP del nuevo proceso
    - Se guarda en su PCB la @ base de su TP
- Planificación de procesos
  - En el **cambio de contexto** se actualiza en la MMU la @ base de la TP actual y se invalida la TLB
- Llamada a sistema **exit**:
  - Elimina la TP del proceso y libera los marcos que el proceso tenía asignados (si nadie más los estaba usando)

Jerarquía de almacenamiento



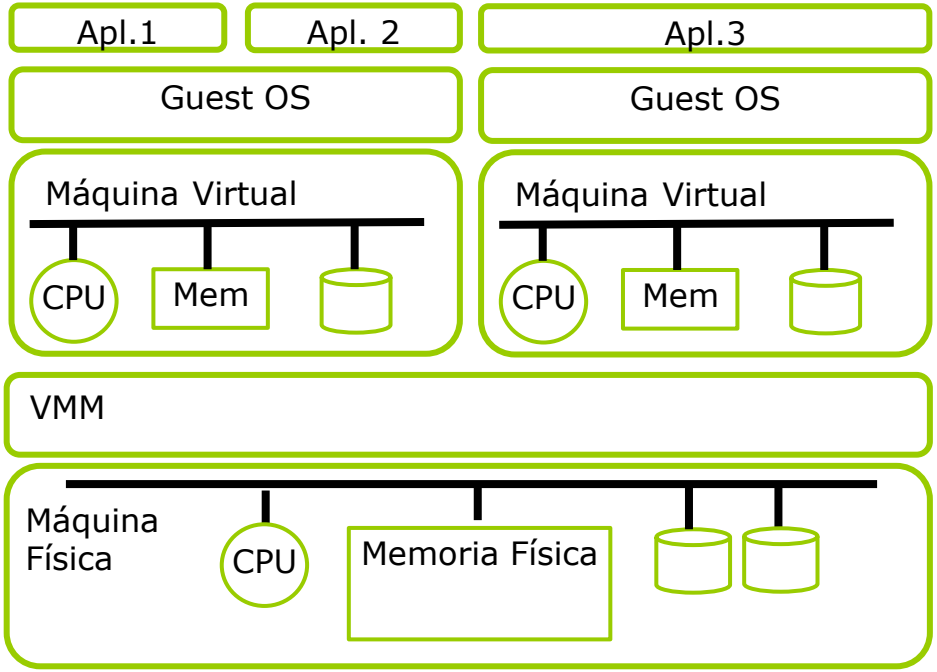
Entornos basados en virtualización

SOPORTE A DIFERENTES ARQUITECTURAS/ENTORNOS

Entornos basados en virtualización(I)

- Virtualización
  - Técnica que permite ocultar los recursos físicos del software que los están usando
- Máquina virtual
  - Contenedor software que puede correr su propio sistema operativo (*guest OS*) y aplicaciones como si fuera una máquina física
  - Aísla el software que ejecuta del resto de software que se está ejecutándo en la misma máquina física
  - Posibles operaciones sobre una máquina virtual: encender, parar, suspender, migrar, modificar recursos asignados…
- *Virtual Machine Monitor (VMM)* o *Hypervisor* gestiona el reparto de recursos entre máquinas virtuales

Entornos basados en virtualización(II)



Entornos basados en virtualización (III)

- Facilita
  - Compartición de máquina física garantizando seguridad y fiabilidad
  - Consolidación de recursos: ahorro energético y maximización de la utilización de la máquina
  - Administración del entorno de ejecución de cada aplicación
    - Adaptado a las necesidades de la aplicación
    - Uniformiza la visión de los recursos físicos
      - Facilita reproducir el entorno de ejecución en cualquier plataforma física
      - Cloud computing

Entornos basados en virtualización (IV)

- Algunos tipos de virtualización
  - Emulación Hardware
    - Interpreta cada instrucción de lenguaje máquina
    - Se utiliza en el desarrollo de sistemas
    - Ejecución lenta
    - Ejemplo: Bochs
  - Paravirtualization
    - guest OS modificado para interactuar con el VMM
    - Mejora rendimiento
    - Ejemplo: xen
  - *Full virtualization* o virtualización nativa
    - guest OS no necesita ninguna modificación
      - VMM intercepta código cuando es necesario
    - Ejemplo: vmware

Entornos basados en virtualización (V)

- Traducción de direcciones
  - Nuevo nivel de direcciones
    - Dirección de máquina: dirección física real
    - Dirección física: abstracción que representa las direcciones *físicas* de la máquina virtual
    - Dirección lógica: direcciones referenciadas por los programas
  - Guest OS mantiene relación entre direcciones físicas y direcciones lógicas y VMM mantiene en la MMU la relación entre direcciones de máquina y direcciones físicas
  - VMM gestiona las modificaciones de la MMU
    - Interceptando el código del guest OS (full virtualization) o cooperando con él (paravirtualization)

Entornos basados en virtualización (VI)

- Protección de acceso entre máquinas virtuales
  - VMM la garantiza
    - Asigna marcos (direcciones de máquina) a cada máquina virtual
    - Sólo él puede modificar la MMU
- Reemplazo de memoria
  - VMM puede necesitar reclamar memoria de las VM si se ha sobredimensionado la memoria física asignada a cada una
    - Guest OS es el que realmente sabe la memoria usada por cada aplicación
    - Técnica de ballooning
      - Cargar un módulo en el guest OS que consuma memoria forzando a que se ejecute el algoritmo de reemplazo del guest
      - Direcciones de máquina asociadas a las direcciones físicas liberadas son las que el VMM reclamará