Gestión de memoria

Sistemas Operativos (SO)

Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

Licencia Creative Commons

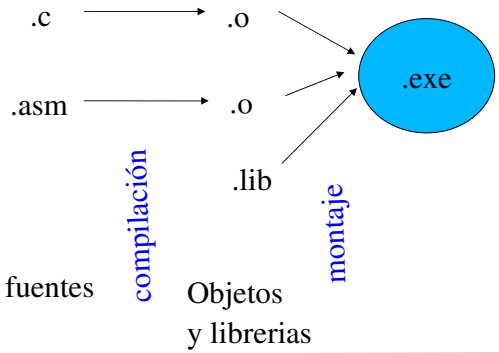
Licencia Creative Commons

Contenido

- Evolución: de posiciones fijas a paginación
- Memoria virtual
- Soporte del SO y HW
- Políticas y Algoritmos

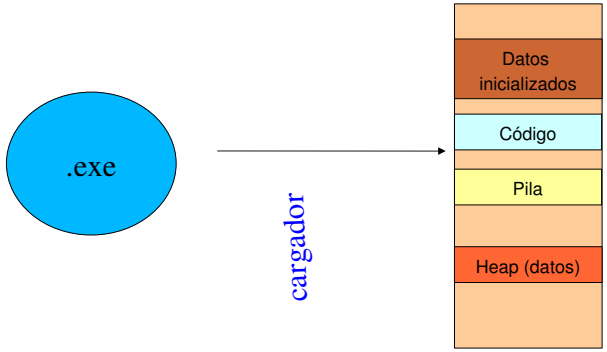
Compilar – montar – ejecutar

- En fase de compilación y montaje se genera un ejecutable
  - Tiene una cabecera dónde indica qué es código, datos inicializados, memoria necesaria para pila y datos no inicializados



Cargador

- El programa cargador coge un ejecutable y lo situa en memoria
  - En principio, todo su código y datos son puestos en memoria.
  - También se reserva espacio para pila y datos no inicializados



Definición: espacios de memoria

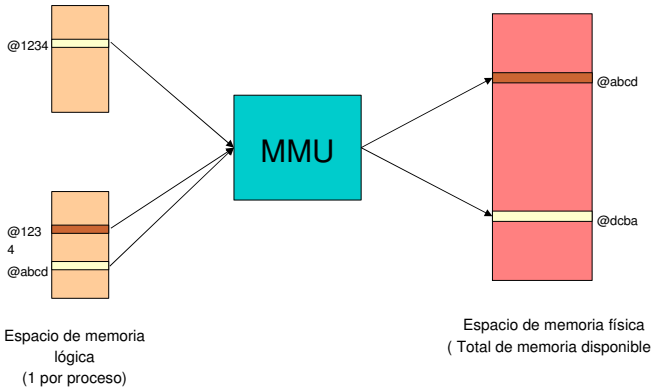
- Espacio lógico del procesador
  - Rango de direcciones que puede acceder un procesador
  - Depende del bus de @
- Espacio lógico del proceso
  - Espacio que ocupa un proceso en ejecución
  - Las direcciones que lanza un procesador cuando quiere acceder a datos/código/pila del proceso
  - RELATIVAS
- Espació físico
  - Direcciones de memoria física

De direcciones lógicas a direcciones físicas

- Las direcciones que genera un procesador cuando está ejecutando un proceso són lógicas
  - Relativas a una dirección 0, igual para todos
- Estos datos se guardan en posiciones físicas de memoria
- Hace falta traducir de direcciones lógicas a físicas
  - MMU: *memory management unit*. El hardware necesario para producir esta traducción
- Puede haber más cosas
  - Swap, memoria virtual

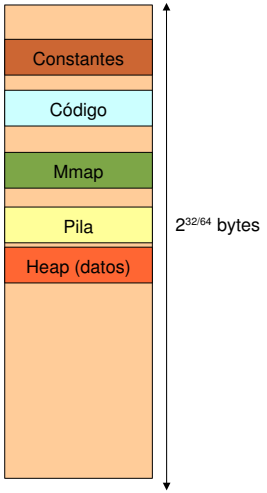
MMU

- MMU (Memory Management Unit)
  - Unidad encargada de traducir las @ lógicas a @ físicas



Espacio de direcciones lógico

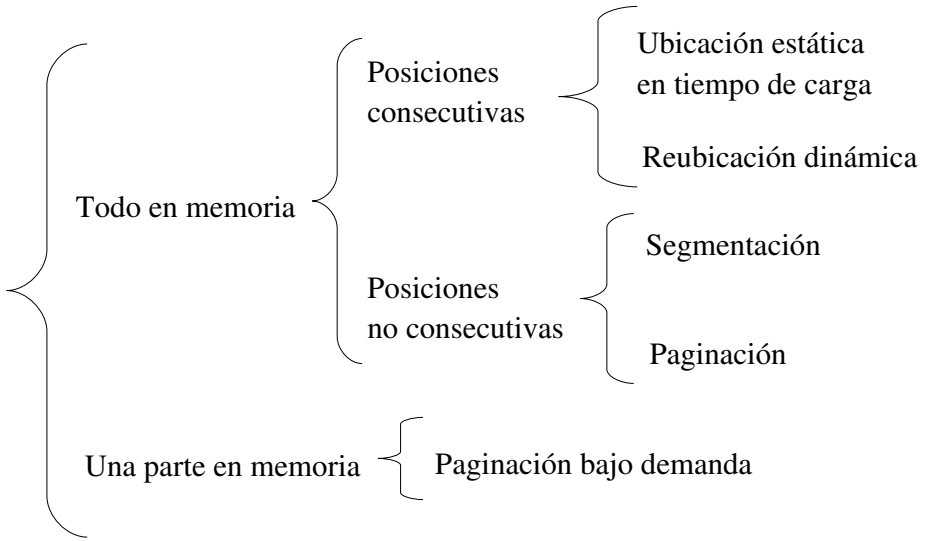
- Constantes, Código
  - Solo lectura
  - Tamaño fijo
- Heap
  - Datos dinámicos globales
  - Tamaño cambia durante la vida del proceso
    - *malloc / free*
- *Pila*
  - *Datos privados a cada thread*
  - *Tamaño cambia durante la vida del proceso*
    - *Implícitamente*
- Mmap
  - Objetos (p.ej. ficheros) accesibles desde el espacio lógico



Un sólo proceso

- En ciertos SO antiguos sólo podíamos tener un proceso en memoria (DOS, por ejemplo)
  - MMU muy sencilla, sólo un registro para proteger espacio de SO
- Los SO de propósito general actuales son multiproceso
- Los SO específicos no necesariamente
  - El SO de la SONY PSP (MBX) admite un solo proceso, igual que la Nintendo NS
  - MMU sólo protección de la zona de sistema

Multiproceso: Carga de ejecutables en memoria



Todo en memoria, posiciones consecutivas

- Ventajas:
  - MMU muy simple (registro base + registro longitud para detectar accesos inválidos; @física=@lógica+reg_base)
- Inconvenientes
  - Hay que buscar un agujero de tamaño adecuado donde poner el programa
  - Ubicación estática en tiempo de carga (una vez en un lugar, no se mueve) o bien
  - Reubicación dinámica: si necesito memoria se mueve entero a una zona especial llamada área de swap. A la vuelta debo encontrarle un hueco adecuado.
  - Se produce fragmentación

Todo en memoria, posiciones no consecutivas

- Problema: buscar un hueco adecuado, cosa que produce fragmentación
- Solución: dividirlo en trozos
  - Un número fijo de trozos, de tamaño variable (segmentación)
  - Un número variable de trozos de tamaño fijo (paginación)
- Ventaja: mejor gestión del espacio
- Inconveniente: MMU más compleja
  - Necesita un registro base para cada bloque, lo que es un número conocido en caso de segmentación (registros de segmento) o un número desconocido en caso de paginación

Parte en memoria

- Si hago paginación, necesito tener todas las páginas en memoria?
  - Pérdida de espacio de un bien preciado: la memoria
- Paginación bajo demanda
  - Sólo traigo las páginas solicitadas
- Complica aún más la MMU!

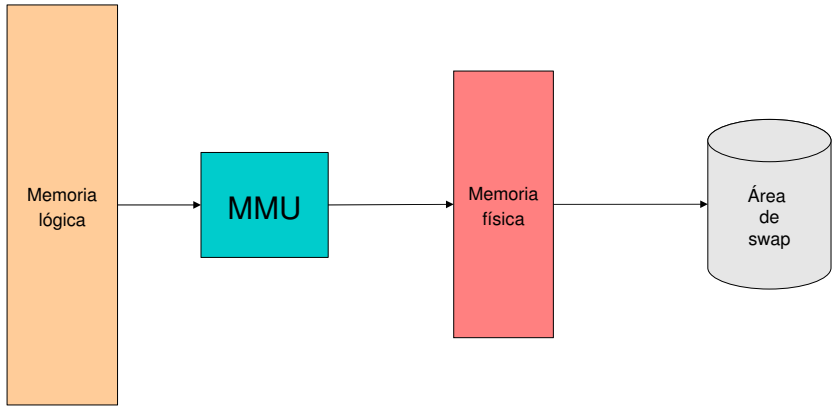
Modelos de memoria

- Uniprogramado/Multiprogramado
  - Nº de programas en memoria a la vez: 1 / Más de 1
- Residente/No residente
  - Información ha de estar en memoria toda la ejecución
- Inmóvil/ Móvil
  - La traducción de @ lógica a física es siempre la misma
- Contigua/No contigua
  - Direcciones @ lógicas contiguas son @ físicas contiguas
- Entero/No entero
  - El programa ha de estar completamente en memoria física para ejecutarse

Modelos de memoria (II)

- Memoria virtual
  - Multiprogramado, no residente, móvil, no contiguo, no entero
  - Paginación, Segmentación
  - Paginación segmentada, segmentación paginada,
  - Paginación bajo demanda
- SO
  - entero
    - SO sencillo
    - Uso ineficiente de la memoria
  - no entero, partes residentes
    - SO más complejo
      - Código y datos esenciales (gestión de memoria, RSI, ...) siempre residentes

Memoria virtual



Espacio de direcciones usado por los procesos ($2^{32/64}$ direcciones)

Total de memoria disponible en el sistema

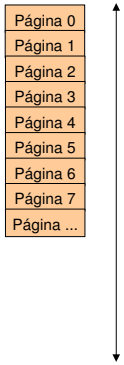
Almacenamiento de memoria lógica que no cabe en memoria física

Responsabilidades del HW i del SO

- Hardware
  - Traducción de @ lógicas a @ físicas
  - Detección de problemas
    - fallo de páginas
    - acceso inválido
    - falta de privilegios
- SO
  - Resolución de problemas
  - Gestión del espacio libre/ocupado

Paginación

- Dividir el espacio de memoria lógico y físico en bloques de igual tamaño.
  - Cada bloque es una página
    - Unidad mínima de asignación y de trabajo
    - 4Kb – 1Mb
  - Simplifica la gestión del SO
  - Reduce el tamaño de las estructuras
  - El HW hace la traducción de una página lógica a una página física



Gestión del espacio ocupado

- Tabla de páginas
  - 1 por proceso
    - En memoria residente
      - si esta en RUN
    - Forma parte del contexto de un proceso
  - Cada entrada representa una página lógica del proceso
    - Nº página física (si tiene una asociada)
    - bits de validez, presencia, referencia, modificación (dirty bit)
    - protecciones
  - Consultada por el HW
    - para hacer las traducciones
    - detectar problemas

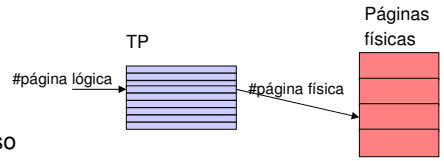


Tabla de paginas (II)

- Espacio de memoria de 32 bits
  - Páginas de 4KB ⇒ 2^{20} páginas x 32 bits/entrada ⇒ 4MB
  - Tablas de páginas multinivel
    - Cada nivel (excepto el último) guarda las páginas de la TP del siguiente nivel
    - El último nivel guarda las páginas del proceso de usuario
    - Reduce la porción de tabla de páginas que ha de estar residente
      - Típicamente únicamente una página (De 4Mb a 4kb!!!)

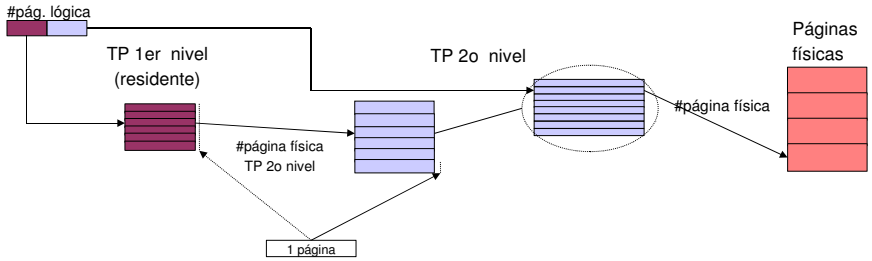
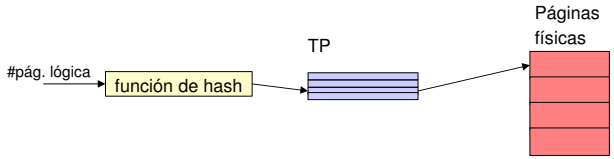


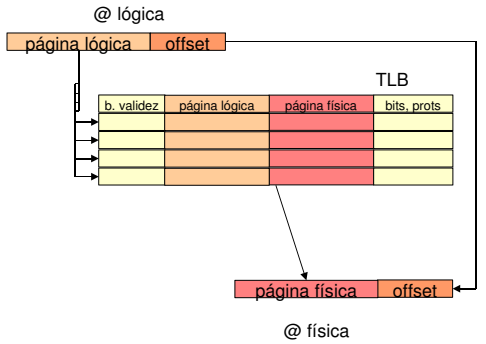
Tabla de paginas (III)

- Espacio de memoria de 64 bits
  - Páginas de 4KB ⇒ 2^{52} páginas x 32 bits/entrada ⇒ 4 PB!!!
  - Tablas de páginas invertidas
    - Solo existen entradas para aquellas páginas válidas
    - Tabla de hash con tantas entradas como páginas físicas



Translation Lookaside Buffer

- Demasiados accesos a memoria para traducir una @lógica
- Solución: TLB
  - cache totalmente asociativa de pequeño tamaño
  - Guarda entradas de la TP
  - entradas gestionadas por el SO
  - se ha de invalidar al cambiar de proceso
    - bit de validez



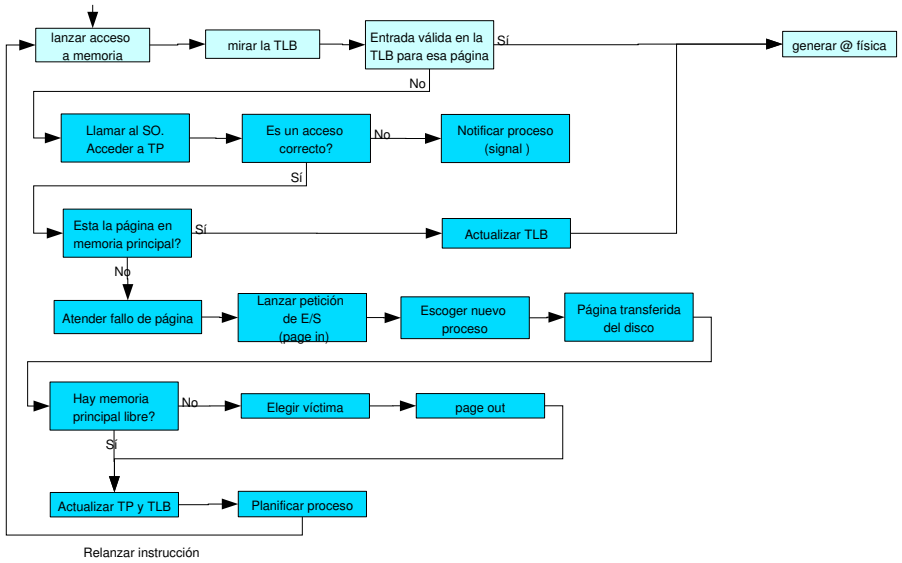
Fallo de página

- Se produce cuando se intenta acceder a una página que no esta en memoria física
  - Al hacer obtener una instrucción
  - Al leer los operandos
  - Al escribir los resultados

add @x, @y, @z

- Soluciones
  - interrumpir la ejecución, salvar el estado, solucionar , restaurar estado, continuar
  - eliminar instrucción, solucionar, reejecutar

Diagrama de tratamiento de páginas



Tamaño de página

- ¿Cómo influye el tamaño de página? (4Kb – 16Mb)
  - tamaño de la TP
  - fragmentación interna
  - localidad
  - grado de multiprogramación
  - tiempo de fallo de página
  - fallos de TLB
- En general depende mucho del programa en sí
  - hay sistemas que permiten definir el tamaño de página por proceso

Protecciones

- Proteger páginas contra operaciones inválidas
  - Bits de protección (rwx) por página
  - Comprobados por el HW
- Proteger memoria contra acceso por otros procesos
  - Memoria virtual!!!
  - Un proceso solo puede acceder a sus @ lógicas
    - solo puede acceder a las páginas que están en su TP

Swap

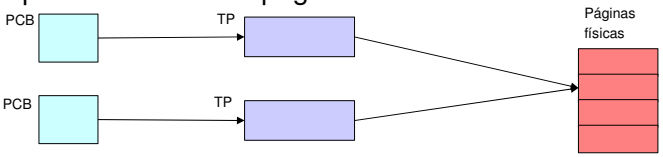
- Zona de almacenamiento secundario para la memoria lógica
  - Extiende la memoria disponible
  - Transparente al usuario
- Espacio reservado, todo junto en disco
  - No utilizable para nadie más
- Dos posibilidades:
  - estático:
    - toda información que este en memoria física también está en el swap
  - dinámico:
    - solo aquellos datos que no están en memoria física están en el swap

Gestión del espacio libre

- Mapa de bits
  - Sencillo
  - Difícil encontrar huecos
  - Rápido!
  - Ocupa poco espacio
- Lista de libres
  - Organizada por zonas libres y ocupadas
  - Fácil encontrar huecos
  - Gestión compleja

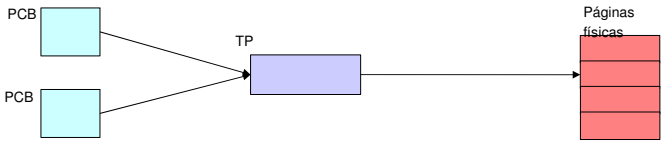
Memoria Compartida

- Páginas compartidas (procesos)
  - Páginas @lógicas en diferentes procesos (o el mismo) que apuntan a la misma página física



Espacio de direcciones compartido (flujos,clones)

- Usan la misma TP ----> Todas las páginas son las mismas



Copy on write

- Cuando creamos un nuevo proceso con fork()
  - hay que copiar los contenidos del padre al hijo, es decir duplicar las páginas
    - muchas páginas no cambiarán nunca (código, exec)
- Idea: seamos vagos :)
  - Inicialmente solo copiamos la tabla de páginas
    - A las páginas con permisos de escritura se les quita y se marcan como CoW
  - Mientras haya lecturas todo va bien
  - Cuando haya una escritura
    - Si es una página CoW se crea una copia para el proceso y se le da permisos de escritura (lo hace el SO)
  - Al final cada proceso tiene propias las páginas que usa para escribir y comparte el resto

Políticas de gestión de memoria

- Política de busqueda
  - ¿Cuándo se trae un página a memoria principal?
- Política de colocación
  - ¿Dónde colocamos la página? No importa, ya que son del mismo tamaño
- Política de reemplazo
  - ¿Qué página eliminamos si nos hace falta hacer sitio?
- Política de limpieza
  - ¿Cuándo escribimos los contenidos de una página modificada?
- Control de la carga
  - Grado de multiprogramación

Política de búsqueda

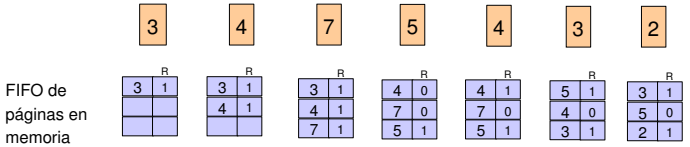
- Bajo demanda
  - Cuando se pide y no está en memoria principal se va a buscar
- Prepaginación
  - Cuando se carga el programa se cargan varias páginas
  - Objetivo: reducir el tiempo de arranque
- Prefetching
  - Cuando se trae una página traer además páginas contiguas
  - Objetivo: obtener beneficios de la localidad espacial

Políticas de reemplazo

- Cuando todas las páginas de memoria están ocupadas y necesitamos una
  - escoger una víctima
  - paginarla (page out) si se ha modificado
  - traer la página del swap (page in)/ crear una nueva página
- Diferentes políticas
  - FIFO
    - Fácil de implementar
    - No tiene en cuenta la localidad temporal
  - LRU (Least Recently Used)
    - Excelente algoritmo (prácticamente óptimo)
    - Difícil de implementar

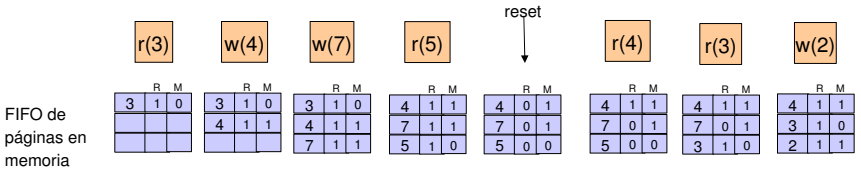
Políticas de reemplazo (II)

- Segunda oportunidad
  - Mejora sobre FIFO
  - Si el bit R esta a 1 se coloca al final de la cola en lugar de elegir la página y se pasa al siguiente



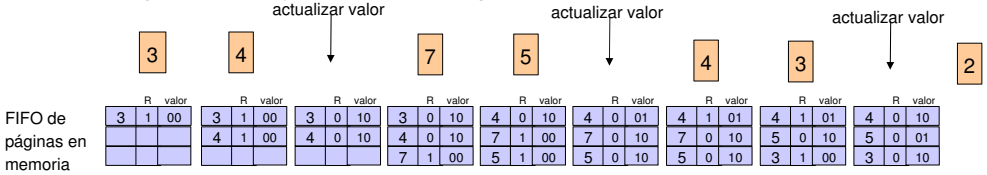
Políticas de reemplazo (III)

- NRU (Non Recently Used)
  - Se basa en los bits de modificado y referencia
    - ¬R,¬M > ¬R,M > R,¬M > R,M
  - Cada cierto tiempo se resetea el bit R
  - En caso de empate: FIFO
  - Simple, Bastante eficiente



Políticas de reemplazo (III)

- Envejecimiento
  - Cada página tiene un número de n bits que determina su valor (R+n bits)
    - se escoge la página con número más bajo
  - Cada cierto tiempo:
    - valor = (valor de R << n) + (valor actual >> 1)
  - Algoritmo muy eficiente, se aproxima a LRU



Políticas de reemplazo (III)

- Page Buffering
  - Técnica para combinar con las anteriores
  - Lista de páginas libres
    - Sólo páginas que no hayan sido modificadas
    - Contiene páginas que han sido “liberadas” pero que no han abandonado la memoria
    - Si se necesita un página se coge de esta lista
    - Si un proceso vuelve a referenciar un página antes de que sea descartada se rescata de la lista
  - Lista de páginas modificadas
    - Páginas “liberadas” pero que se han de guardar porque han sido modificadas
    - Se escriben de forma conjunta cada cierto tiempo
    - Pasan a la lista de libres

Política de limpieza

- Bajo demanda
  - Cuando es seleccionada como víctima
- Prelimpieza (Precleaning)
  - Limpia las páginas modificadas cada cierto tiempo sincronizándolas con el swap
    - Reduce el tiempo de *page out* posteriormente ya que las páginas están “limpias”
    - Se puede sincronizar varias páginas contiguas reduciendo el tiempo total de transferencia

Pageout daemon

- Proceso de sistema
  - Se ejecuta cuando el sistema no está ocupado
  - Se encarga de la prelimpieza
  - Mantener el número de páginas libres en el sistema a un cierto número
    - No liberarlas del todo hasta que haga falta realmente (Page Buffering)

Control de la carga

- Si el grado de multiprogramación
  - es bajo --> baja utilización del procesador
  - es alto --> poca memoria física por proceso --> trashing
- Trashing: Situación anómala del sistema
  - El sistema está más tiempo atendiendo fallos de página que haciendo trabajo útil
- Detección
  - Monitorizar la utilización del subsistema de memoria
    - Frecuencia de fallos de página
    - Páginas escritas a swap / segundo
    - ...
  - Si el parámetro supera un cierto un umbral --> trashing

Control de la carga (II)

- Solución
  - Invocar el planificador a medio plazo
    - Disminuir el grado de multiprogramación temporalmente
      - Swapear (swap out) procesos enteros
    - Cuando se normalice la situación
      - Traer los procesos swapeados (swap in)
  - Que procesos?
    - Procesos menos prioritarios
    - Procesos con mayor número de fallos de página
    - Procesos más grandes (en número de páginas)
    - ...

Llamadas a Sistema

- Obtener memoria
  - Marcar página(s) como válida(s) en TP
  - (Reservar página(s) en el área de swap)
  - Marcar página(s) como ocupada(s) (bitmap / lista)
- Liberar memoria
  - Marcar página(s) como inválida(s) en TP
  - (Liberar espacio en el área de swap)
  - Invalidar entrada(s) de la TLB
  - Marcar página(s) como libre(s)

RSI memoria

- Motivos que generan una interrupción y respuesta del SO
  - Violación de protecciones
    - avisar al proceso (SIGSEGV)
  - Página no presente
    - fallo de página ⇒ page in
  - Página no válida
    - Si era un acceso a la pila

      dar más páginas a la pila
    - sino

      avisar al proceso (SIGSEGV)

UNIX

- int brk(void *end_data_segment)
  - Fija el limite superior en *end_data_segment*
    - *Para pedir más memoria subimos el limite*
    - *Para liberar memoria bajamos el limite*
  - *No está pensado para ser usado por los programadores de aplicaciones directamente*
    - *demasiado simple*

Librerías

- Se utilizan librerías que gestionan la memoria
  - Normalmente asociadas al lenguaje de programación
    - C: malloc/remalloc/free
    - C++: new/delete
    - Java: new/(garbage collector)
  - Una petición a la librería no siempre incrementa/decrementa la memoria del proceso
    - ¿Cual es el tamaño mínimo que puede incrementarse/decrementarse la memoria de un proceso?
- Libc
  - void * malloc (int tamaño)
  - void free (void *free)
  - Muchas implementaciones diferentes