Universidad del País Vasco
Euskal Herriko Unibertsitatea
GASTEIZKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE VITORIA-GASTEIZ

6 Carga y ubicación de programas en memoria. Direccionamiento físico y virtual.

Introducción a los Sistemas Operativos, 2023-2024

Pablo González Nalda

Depto. de Lenguajes y Sistemas Informáticos
EU de Ingeniería de Vitoria-Gasteiz,
UPV/EHU

19 de marzo de 2024

Contenidos de la presentación

Contenidos

Conceptos básicos de la gestión del espacio

Conceptos básicos de la gestión del espacio:

Recursos: se deben repartir recursos: memoria, almacenamiento y uso de elementos (CPUs).

Ordenador multiprocesador: conjunto de procesadores con un espacio de direccionamiento de memoria física: SMP (*Shared-memory Multi-Processor*)

Unidad de direccionamiento: cantidad de memoria que señala una dirección de memoria, normalmente un *byte (B)*.

Particionado variable o fijo: los elementos que se asignan a diferentes procesos o cuentas usuarias son de tamaño uniforme o no.

Gestión del espacio

Capacidad de direccionamiento

$$E = 2^b \cdot d$$

(E memoria total, b bits y unidad de direccionamiento de d bytes)

Gestión del espacio

La gestión del espacio implica definir cuál es la cantidad mínima de memoria para asignar o liberar. Particionado variable o fijo (lista de bloques o mapa de bits)

Mapas de bits:

Particionado fijo un bit por partición
Particionado variable Lista de elementos y lista de huecos para representar los elementos y el espacio libre.

Políticas de asignación de huecos:

First/next fit primer hueco, siguiente
Best fit el más pequeño que quepa
Worst fit el más grande libre

`https://thumbsup2life.blogspot.com/2011/02/best-fit-first-fit-and-worst-fit-memory.html`

Gestión de memoria

En la memoria: Sistema Operativo, librerías compartidas de carga dinámica, y programas.

Tareas del SO:

- Carga y ubicación
- Gestión de más de un programa en memoria
- Carga bajo demanda de código compartido durante la ejecución: DLL
- Programas que no caben completamente en memoria
- Memoria compartida entre programas
- Gestión de la memoria libre

Mecanismos para la gestión de memoria

- Permanentes o no (en ejecución se mueven fuera de RAM)
- Contiguos o no (programa ordenado, consecutivo en RAM)
- Enteros o no (partes del programa fuera de RAM)

Combinaciones:

Intercambio de procesos No permanentes

Paginación y segmentación No contiguos

Enlace dinámico y memoria virtual No contiguos y no enteros

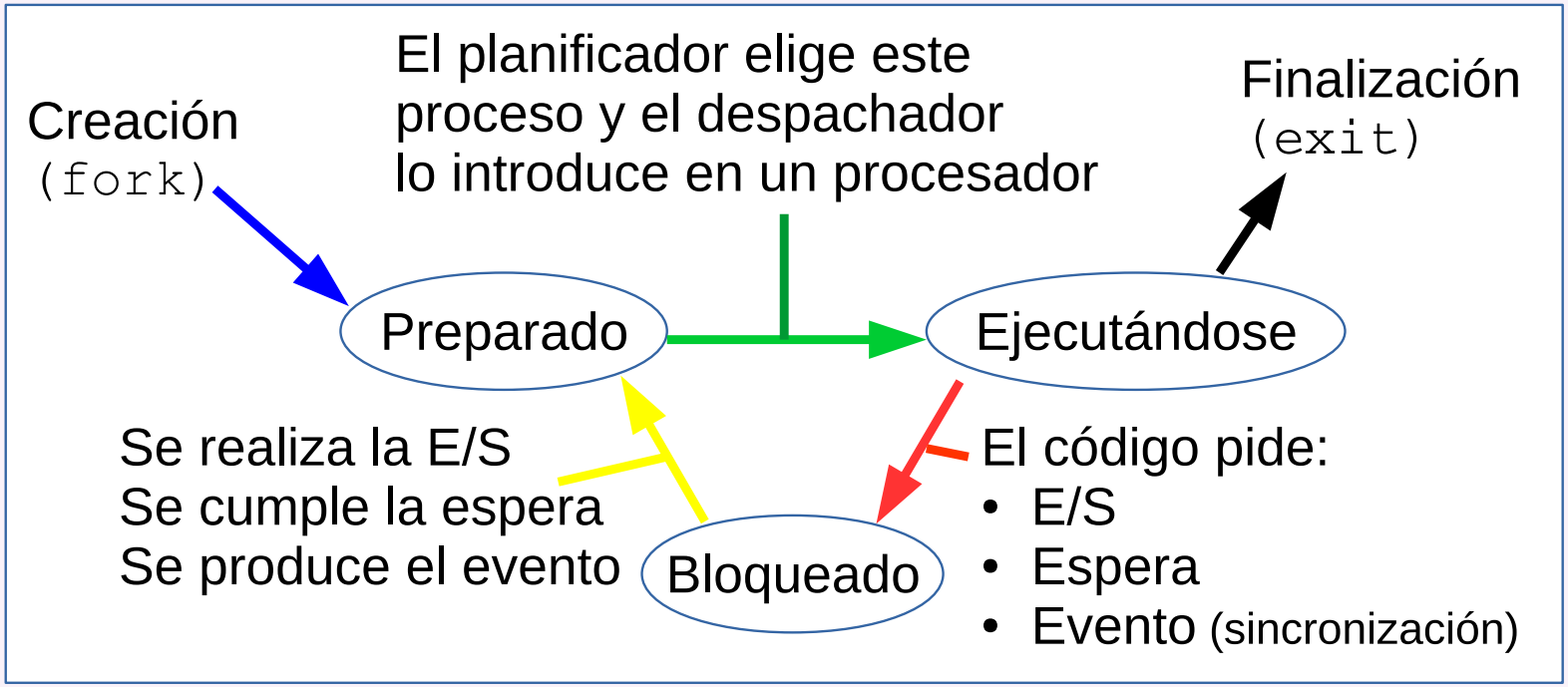
Evaluación del rendimiento de la gestión de memoria

Temporal Pérdida de eficiencia temporal en la aplicación de la política, con o sin hardware adicional

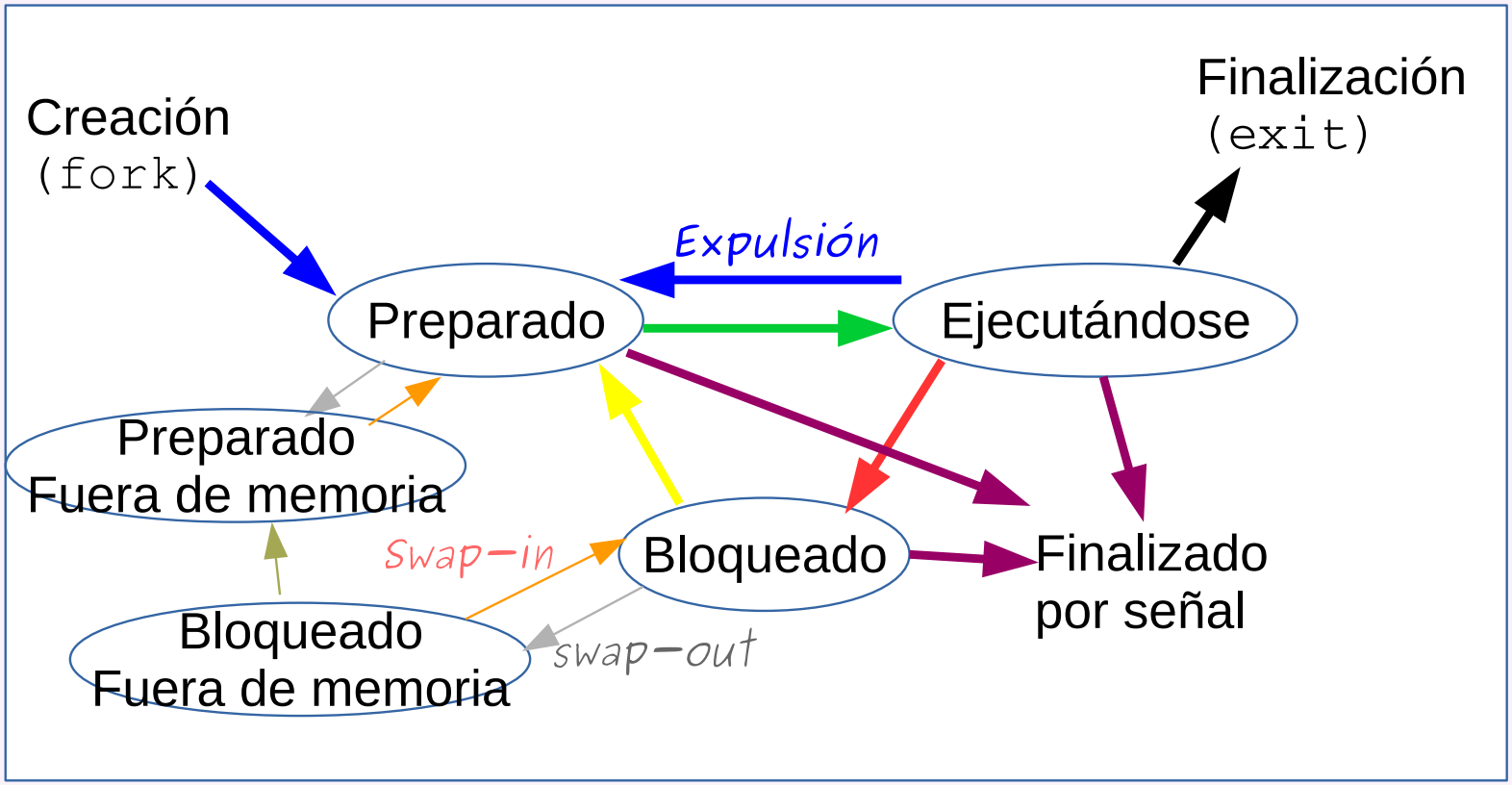
Espacial Eficiencia en el uso del espacio de memoria (fragmentación, compactación)

Contenidos

Estados y cambio de contexto (avance del tema 7)



Estados y cambio de contexto (avance del tema 7)



Intercambio de procesos o *swapping* de procesos

Mover programas de RAM a almacenamiento (partición o fichero *swap*) durante el tiempo de ejecución para dejar espacio libre en RAM. Aplicable a procesos que no están en ejecución (esperan E/S, eventos o recurso de CPU).

En inglés: *swapper, swap-in, swap-out*

Bloqueado y preparado tanto dentro como *fuera de memoria* $\Rightarrow$ planificación a medio plazo

Criterios de planificación

- bloqueado o preparado
- prioridad del proceso
- tamaño del programa, más hueco pero más movimiento
- tiempo en memoria

Solución de problemas

Existe pendiente una operación de DMA sobre un búfer ubicado en la memoria de programa.

Si ese programa se elige para sacar de la RAM:

- impedir *swap-out* del proceso con DMA pendientes
- E/S sobre búferes del sistema (residentes)

Contenidos

Métodos para tener programas más grandes que la RAM disponible

Los métodos para tener programas más grandes que la RAM disponible:

- solapamientos (*overlays*)
- librerías de enlace dinámico (DLL o .so (*shared object*))
- memoria virtual

Librerías de Enlace Dinámico, DLL / .so

El mecanismo del enlace dinámico necesita una llamada al sistema que cargue las rutinas y que gestione la memoria (`mmap`).

Beneficios actuales de las DLL (en combinación con el uso de memoria virtual):

- tamaño en disco y en RAM: una repetición de la rutina
- sólo se carga una vez para todos los programas, mayor rapidez de carga
- se cargan cuando se necesitan, los programas se inician más rápidamente
- se actualizan más fácilmente los programas si se actualizan la librerías

Contenidos

Carga de un programa en Memoria. Cargador

Funciones del Cargador:

- Lee el fichero ejecutable
- Lo ubica en un espacio libre de la memoria:
  1. Busca un espacio consecutivo libre en memoria
  2. Copia el código del programa
  3. Copia los datos Inicializados
  4. Reserva espacio para la pila y las memoria dinámica
- Comienza con la primera instrucción del programa:
  1. Inicializa los registros de la CPU (PC, PSW, SP, ...).
- El SO tiene que llevar contabilidad/gestión del uso de la memoria:
  - Conocer el espacio libre
  - Zona de memoria que ocupa un programa
  - Debe liberar esa memoria cuando el programa termine

Reubicación en tiempo de carga


```
1  int Global1, x, Resul;
   main()
   {
4  Global1 = 35;
   x = 3;
   Resul = Global1 + x*x;
7  escribir(Resul);
   }
   void escribir(int R)
10 {
   printf("El resultado es %d\n", R);
   }
```

```
   2000: Global1:
   2004: x:
3  2008: resul:
   PROC main
   2012: PUSH r30
6  2016: MOV r30,r31
   2020: MOV r1, 35
   2024: ST r1, 2000
```

Reubicación en tiempo de carga


```
2000: Global1:
2004: x:
2008: resul:
PROC main
2012: PUSH r30
2016: MOV r30,r31
2020: MOV r1, 35
2024: ST r1, 2000
2028: MOV r1, 3
2032: ST r1, 2004
2036: LD r2, 2000
2040: LD r3 ,2004
2044: MULT r4, r3,r3
2048: ADD r5, r2,r4
2052: ST r5, 2008
2056: PUSH r5
2060: CALL 2132
...
;;;;
main
2128: RET
PROC escribir
2132: PUSH r30 ;; escribir
```

Inconvenientes de la ubicación estática

No tiene sentido pensar en la ubicación estática en tiempo de compilación, porque hace incompatibles los programas para su ejecución concurrente.

Inconvenientes de la ubicación estática en tiempo de carga:

1. Tiempo invertido en el proceso de reubicación ralentiza la fase de carga.
2. El programa una vez reubicado no se puede mover (es estático). Puede generar fragmentación de la memoria.

Podríamos solventar estos problemas si la reubicación se realizara en el momento de acceder a memoria, en cada referencia.

Reubicación dinámica

La reubicación dinámica consiste en calcular en cada acceso a memoria RAM la dirección que corresponde a una dirección lógica del programa.

El resultado es que cada programa gestiona todo su espacio de direcciones y el hardware recoloca cada una de sus partes en la RAM.

Contenidos

Paginación

Paginación: asignación de memoria no contigua, sólo es necesario tener memoria libre.

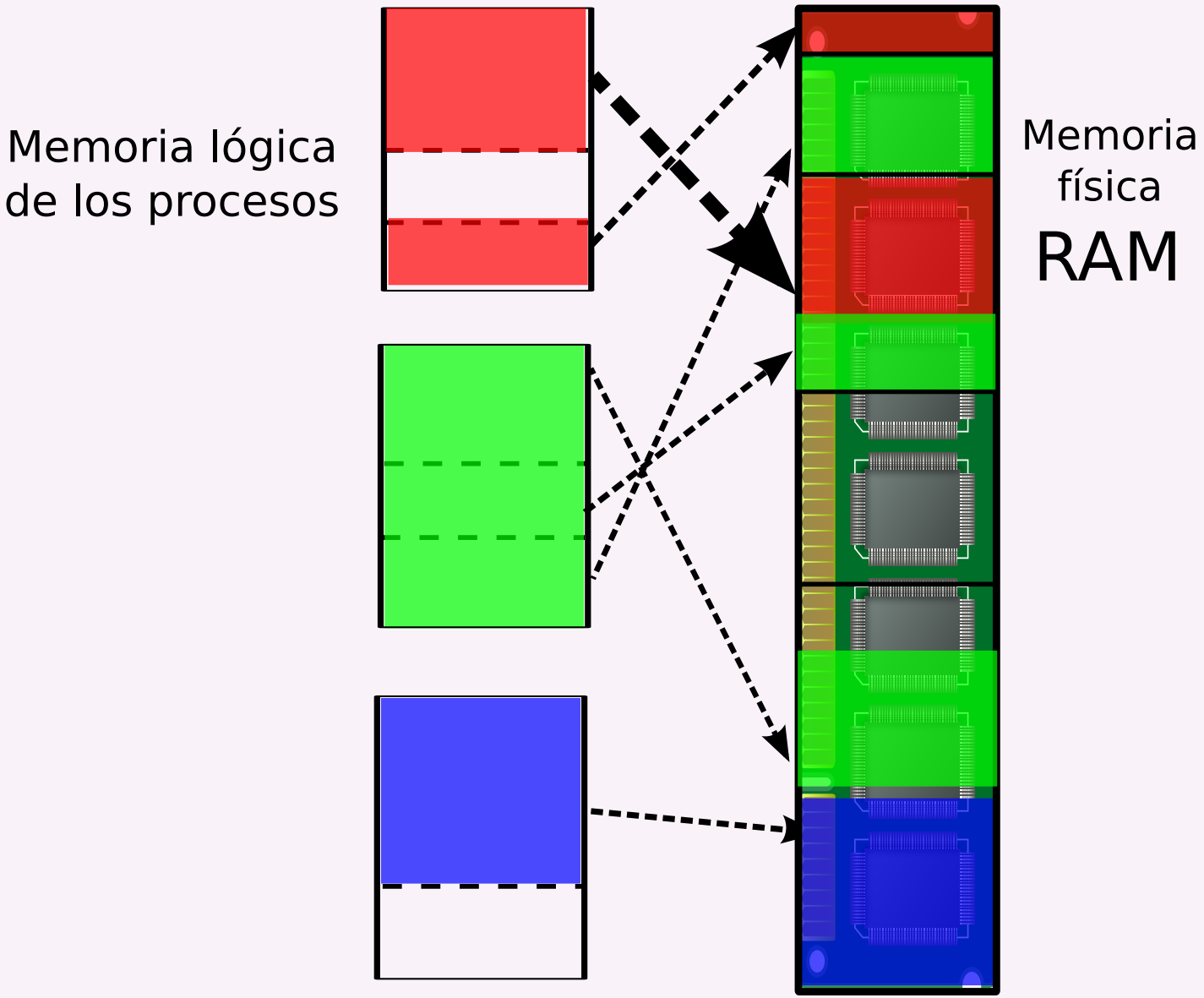
Se dividen los programas en trozos del mismo tamaño, llamados páginas.

Direccionamiento virtual: una *dirección lógica (@l)* se traduce en una *dirección física (@f)* en tiempo de ejecución, cada vez que se referencia una posición de memoria del programa (para lectura o escritura, tanto código como datos). Es una *reubicación dinámica*.

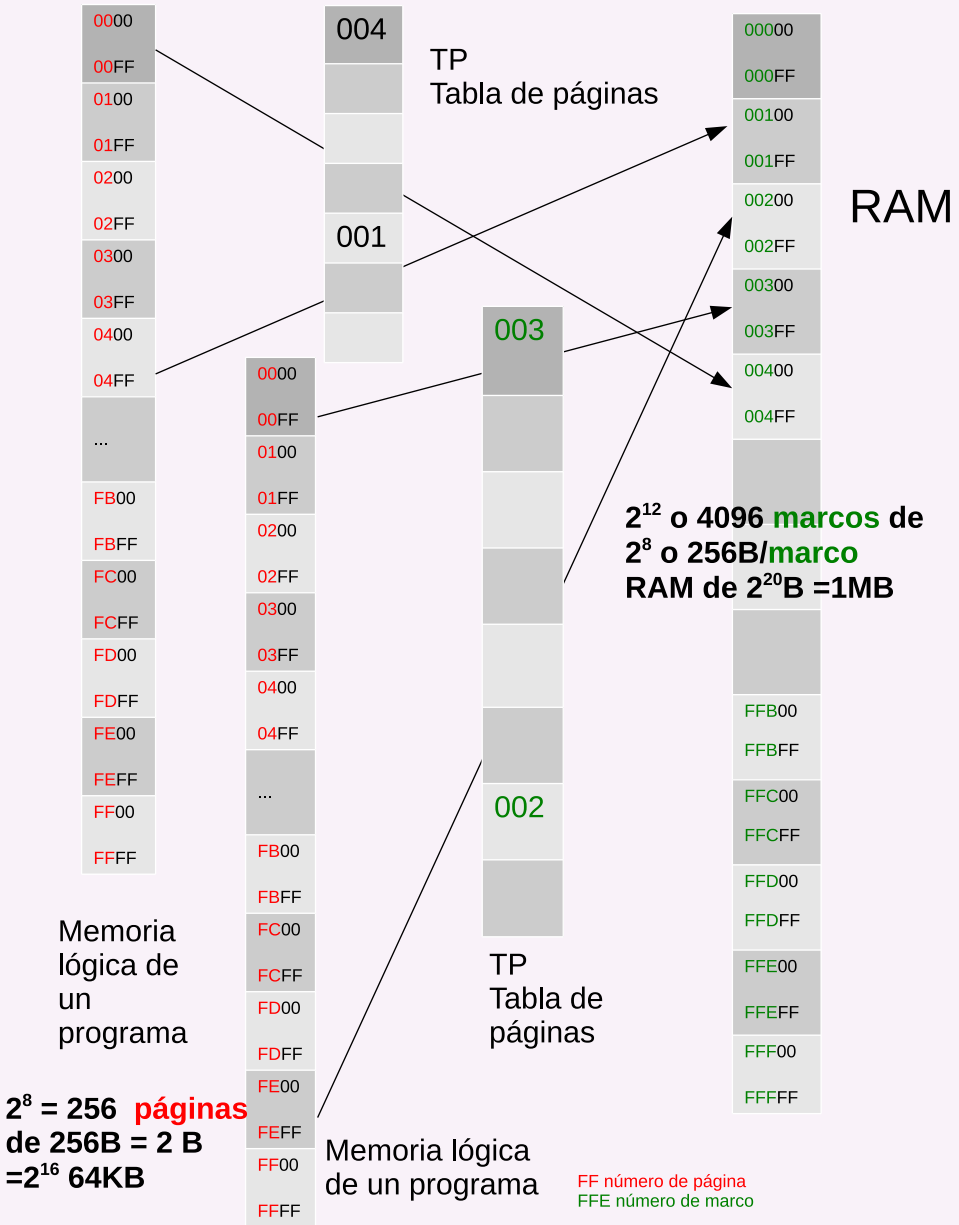
Los programas pueden usar el direccionamiento virtual para compartir código o datos para ahorrar memoria y tiempo de carga.

paginación

Esquema: programas no contiguos en memoria



Esquema: paginación, de páginas a marcos



Esquema de traducción de página a marco

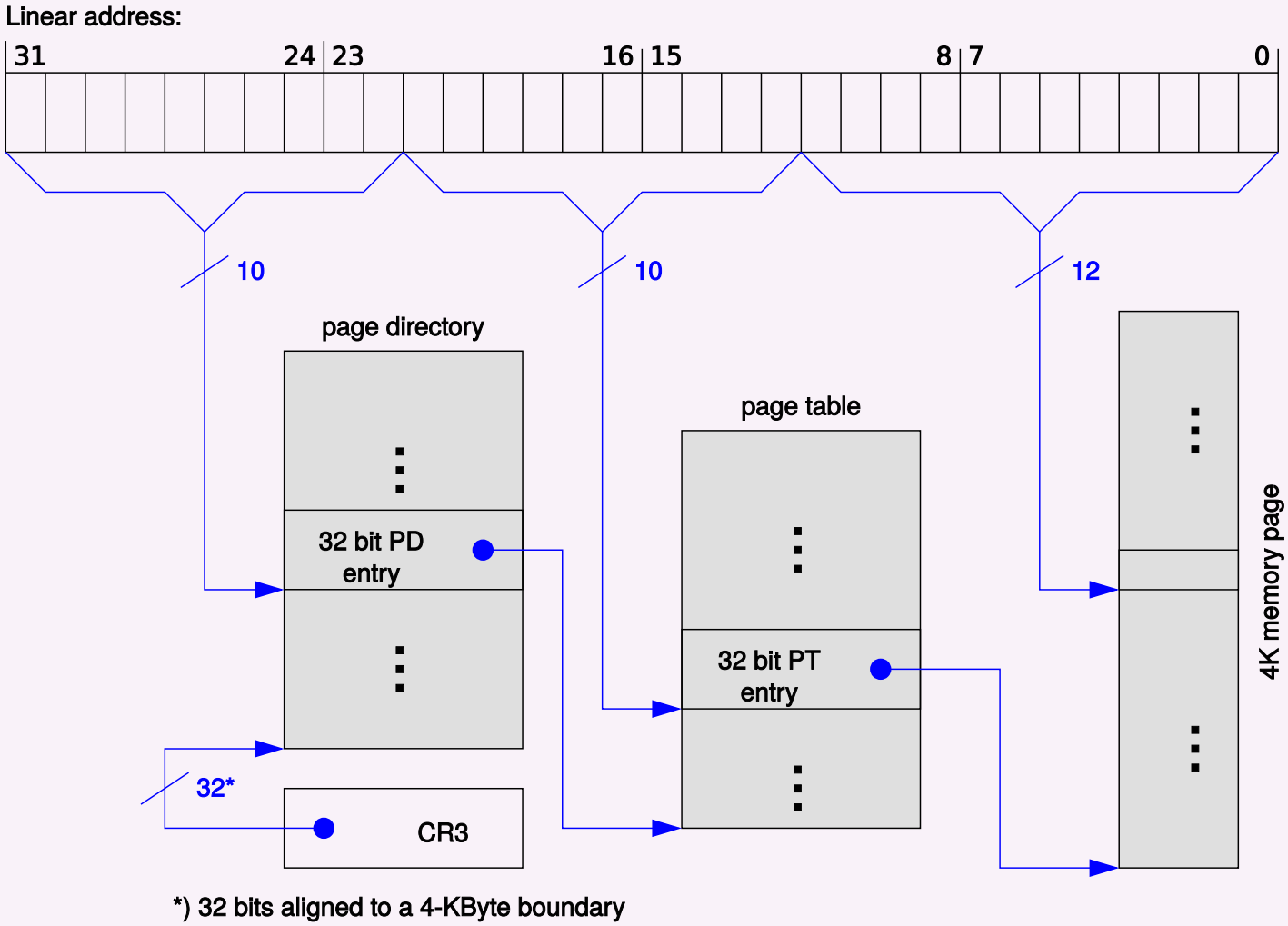
El problema es que, si usamos una sola tabla de páginas por proceso, tenemos direcciones de 32 bits y por tanto una memoria de $2^{32}B$ y páginas de 4KB:

$$\frac{2^{32}B}{4KB/p} = \frac{2^{32}B}{2^{12}B/p} = 2^{20}p$$

es decir, un millón largo de entradas en la tabla de páginas.

Por eso se hace una tabla en 2 niveles: el primero tiene una tabla de 1024 entradas, que nos permite encontrar qué rangos de direcciones tienen páginas en uso. El segundo tiene tablas de 1024 entradas que nos indican las páginas usadas.

Esquema de traducción de página a marco



`Paginación lineal.` Hay muchos más esquemas en la Red: buscad imágenes *`memory address translation`*

Soporte hardware

Hardware para traducción de direcciones: MMU (unidad de gestión de memoria en la CPU).

TLB (Translation Lookaside Buffer): caché de traducciones para evitar un acceso a memoria para consultar la tabla de páginas

Por cada página hay varios bits, de sólo lectura y de no ejecutable en cada entrada de la Tabla de Páginas de un proceso.

Contenidos

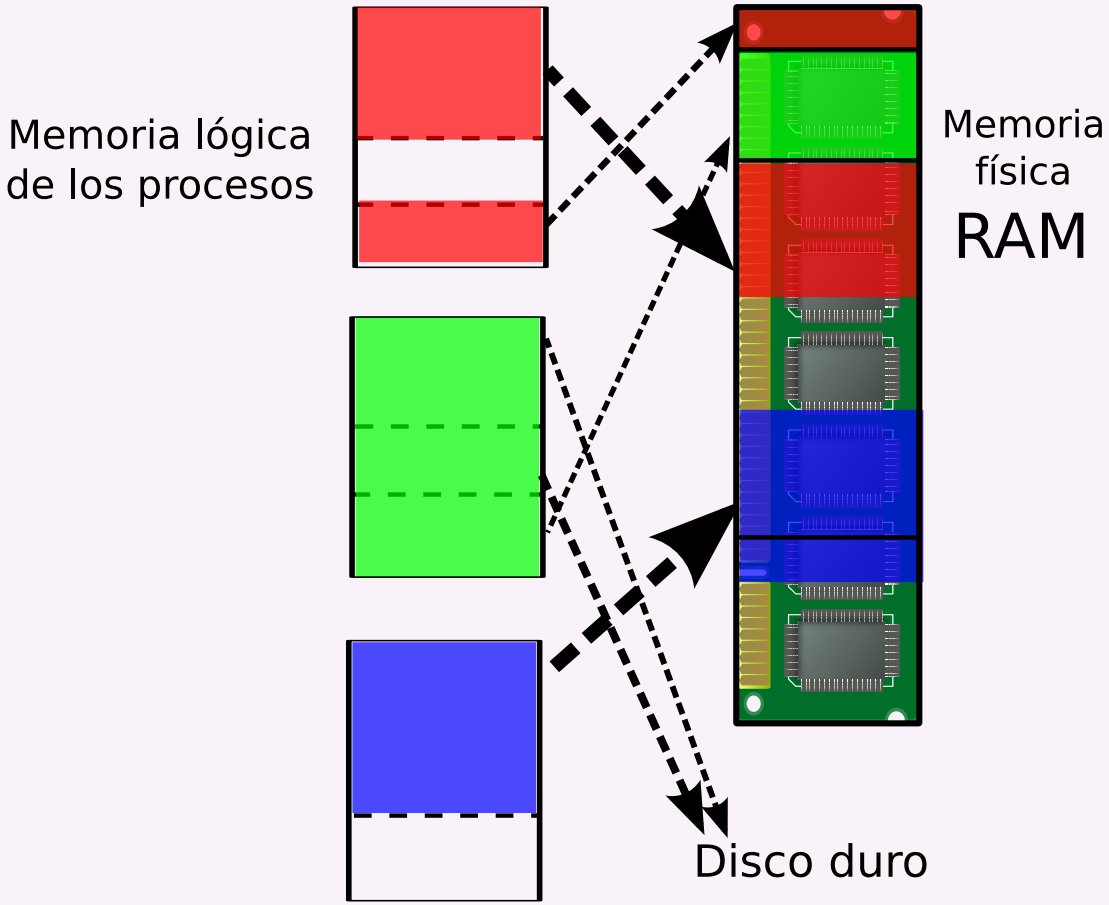
Memoria virtual

La Memoria Virtual (MV) es un sistema de paginación en el que sólo un *subconjunto de las páginas* está en memoria. El resto de las páginas están en almacenamiento en la partición *swap* (puede ser un fichero *swap*) o en el fichero ejecutable.

Ventajas:

- protección de memoria, un programa no puede acceder a la memoria de otro
- menor latencia, no hay que cargar el programa entero para empezar a ejecutarlo
- se pueden ejecutar más programas con menos memoria por programa a costa de usar almacenamiento
- total independencia de los programas con respecto al hardware

Esquema de la memoria virtual



Se ubican en los ejecutables las páginas del código y en *swap* los datos en memoria del programa

Soporte hardware

Aparte del hardware necesario para la paginación, para la MV se necesita:

- bit de validez V: para cada página hay un bit en la tabla de páginas que indica si la página está en memoria o no.
- interrupción de fallo de página (FP): cuando se *referencia* un byte de una página, la MMU consulta el bit de validez y si es 0 ejecuta la rutina de atención al FP para cargar la página en memoria.
- Opcionalmente, espacio en almacenamiento en disco.

Carga de programas

Carga de un programa cuando se usa memoria virtual:

- El cargador inicia la tabla de páginas
- carga ciertas páginas y marca el bit V a 1 para cada una
- anota las posiciones en disco del resto de las páginas (no cargadas) del ejecutable

Estrategias de carga:

- carga por demanda, según se van referenciando
- prepaginación de un cierto número de páginas, dependiendo de los recursos disponibles y del tamaño del programa

Procedimiento de acceso a RAM

Para cualquier acceso a memoria:

- decodificar la @ lógica y consultar la tabla de páginas
- si la dirección física no es correcta, *core dump*
- si no, si V es 1, acceso al marco indicado en la TP
- si no, V es 0 por lo que hay fallo de página (FP)

Para poder atender el FP se puede necesitar que se revierta (*rollback*) la ejecución de las instrucciones de Lenguaje Máquina hasta un estado consistente (procesadores segmentados o `pipelined`)

Rutina de atención al FP

La rutina de atención al fallo de página (FP) consiste en:

- si hay marcos libres se carga la página
- si no, el algoritmo de reemplazo elige una *página víctima* que quizás haya que escribir en disco
- el proceso se bloquea hasta que termina la carga de página por DMA, y pasa a ejecutarse otro proceso
- al terminar la carga se actualiza la TP y el bit V, y el proceso pasa a preparado

Rendimiento de la memoria virtual

- Sin FP, hay que cargar todo el ejecutable al principio (equivalente a n FP)
- localidad espacial $\Rightarrow$ muchos accesos a pocas páginas antes de FP
- mientras se atiende el FP se ejecuta otro proceso

Localidad temporal y espacial

Necesitamos dos conceptos:

Localidad temporal

Un programa suele cumplir una localidad temporal, dado que usa repetidamente el mismo trozo de código durante un cierto tiempo.

Localidad espacial

Un programa suele cumplir una localidad espacial porque si usa cierto trozo de código es más probable que se vaya a usar un trozo cercano en el binario, por la forma de compilarlo.

Políticas de reemplazo de páginas

Algoritmos de reemplazo de páginas
Page replacement algorithm

Criterios para la elección de *página víctima*:

- minimizar número de fallos de página (FP), explotando la localidad temporal de los programas
- sencillez en procesamiento y hardware

Cada programa tiene su secuencia de referencias a memoria, por lo que el salto de una secuencia a otra rompe la localidad

FIFO

La política FIFO es elegir la página que lleva más tiempo cargada en memoria (aunque sea usada a menudo). Al no usar localidad no produce buenos resultados, porque se puede elegir como *página víctima* una página muy usada que se cargó la primera.

Política óptima

La óptima es seleccionar la página que va a tardar más tiempo en ser referenciada. Implica conocer el comportamiento futuro del programa.

Política *Usada menos recientemente* (*Least recently used*, LRU)

LRU aprovecha la localidad temporal, al suponer que las páginas que se acaban de usar es más probable que se usen en un futuro cercano, y las no usadas en mucho tiempo no se usarán.

La información sobre cada página hay que modificarla en cada referencia, mientras que FIFO lo hace sólo en la carga.

Implementación (muy costosa):

- lista ordenada en función de la última referencia
- contador global de referencias y un registro por marco para almacenar el valor del contador en el momento del acceso

Aproximación a LRU

En el momento del acceso a la página se activa su *bit de referencia R*.

Cuando se produce la interrupción del reloj, al final del intervalo de tiempo entre *ticks de reloj* en la rutina de atención al reloj se actualiza el contador del marco con el valor del contador de ticks y se pone su bit de referencia R a 0.

Con esta aproximación se pierde precisión al no ordenar las páginas accedidas por última vez en el mismo tick.

Otros elementos para tener en cuenta

Hay otros elementos que tener en cuenta para mejorar la efectividad de estos algoritmos: el bit de *modificada*, para marcar las páginas en las que se ha escrito y por tanto necesitan ser escritas en el *swap*. Es una operación más costosa que elegir una página con código, que no es necesario escribirla.

Memoria virtual y multiprogramación

Si el sistema es multiprogramado, las secuencias de referencias se intercalan y la localidad temporal se rompe. Una página que se ha referenciado recientemente no se va a usar pronto porque se está ejecutando otro programa.

Rango de asignación

El rango de asignación es el conjunto de páginas que se tienen en cuenta al aplicar el algoritmo de reemplazo.

Rangos de asignación

Rangos de asignación:

global: referencias de todos los programas en una secuencia única. Se desecharán más probablemente las páginas de procesos que no están en ejecución.

local: el algoritmo de reemplazo se aplica sobre las páginas del proceso que ha sufrido el fallo de página.
Es necesario repartir, asignar (a_i) los marcos (M) entre los procesos (N) para alojar parte de todas las páginas del proceso (s_i, siendo $S = \sum s_i$ todas las páginas necesarias):

- igual número de páginas por proceso $a_i = M/N$
- igual proporcional al uso de memoria $a_i = \frac{s_i}{S} \cdot M$

Solución intermedia: definir máximos y mínimos.

Conjunto de trabajo o *working set*

Working Set

El conjunto de trabajo o *working set* (WS), es el subconjunto de páginas de un programa que va a recibir *casi todas* las referencias que genere el programa en un futuro cercano.

Por tanto, si se mantiene el WS en memoria, va a haber pocos fallos de página. Por lógica, el rango de asignación es local, la página víctima se elige de entre las del programa.

Se limita el grado de multiprogramación máximo al que permite tener todos los WS en memoria:

$$\sum_{i=1}^{N} tam(WS_i) \leq M$$

Cálculo del *Working Set*

De la definición de WS deducimos que hay que hacer una traza de las últimas referencias de un proceso en un tiempo Δ o *ventana* de tamaño Δ.

El caso más simple es que la ventana esté compuesta de las páginas con el bit de referencia activado en el último tick de reloj. Si se aplica envejecimiento se puede mejorar la búsqueda del WS.

Con la carga de páginas por demanda podría haber demasiados fallos de página y se soluciona con la prepaginación si no se llega a un número de páginas del proceso en memoria (buscará la carga del WS entero).

Fijación de páginas en memoria

Es necesario hacer una página fija en memoria (no se puede elegir como víctima) mediante un *bit de fijación* si:

- se espera una operación de DMA sobre un búfer que es parte de esa página
- esa página acaba de cargarse por FP y su proceso todavía no la ha usado

En el primer caso también se soluciona usando búferes de E/S que gestione el sistema operativo, como en el intercambio de procesos.

Copy-on-write

Inmediatamente después de un `fork` todas las páginas se comparten por padre e hijo y se marcan con un bit denominado *bit Copy-on-write* (copia cuando se escriba). Las tablas de páginas apuntan a las mismas páginas de código hasta que un proceso sufre un `exec`.

En el momento de referenciar para escritura una página con el bit activado se duplica la página para que cada programa mantenga sus datos propios, dejando de compartir.

Contenidos

¿Más preguntas?

¿Más preguntas?

Universidad del País Vasco
Euskal Herriko Unibertsitatea
GASTEIZKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE VITORIA-GASTEIZ

6 Carga y ubicación de programas en memoria. Direccionamiento físico y virtual.

Introducción a los Sistemas Operativos, 2023-2024

Pablo González Nalda

Depto. de Lenguajes y Sistemas Informáticos
EU de Ingeniería de Vitoria-Gasteiz,
UPV/EHU

19 de marzo de 2024