Administración de Memoria y CPU

UNIDAD 1 - PARTE 3

Prof. **Iván Jiménez Utiel**

Contents

1 Características de la administración de Memoria.

1.1 Puntos clave en el Memory Management (Gestión de Memoria)

Definición 1.1 (Administrar la memoria)**.** Es seguir la pista de qué partes de la memoria están en uso y cuáles no, con el fin de poder asignar memoria a los procesos cuando la necesiten. Y así recuperar esa memoria cuando dejen de necesitarla.

Definición 1.2 (Memoria Virtual)**.** Es un esquema de ubicación de almacenamiento en el que la memoria secundaria puede ser accedida como si fuera parte de la Memoria Principal. Su tamaño es limitado por el SO y por la cantidad de memoria secundaria disponible.

Definición 1.3 (Espacio de direcciones)**.** El rango de direcciones de memoria disponibles para el proceso.

Término	Descripción
Marco	Bloque de memoria de longitud fija (memoria RAM)
Página	Bloque de memoria de longitud fija (alojada en mem. secundaria)
Segmento	Bloque de memoria de longitud variable (en mem. secundaria)

(a) Las referencias de memoria son direcciones lógicas dinámicamente traducidas a direcciones físicas en tiempo de ejecución → Un proceso puede ser intercambiado en memoria principal ocupando diferentes memorias en diferentes tiempos durante la ejecución.

(b) Un proceso puede ser partido en diferentes piezas que no necesariamente se tienen que ubicar de manera contigua en memoria principal.

(c) No es necesario que todos esos pedazos (páginas, segmentos) de un proceso estén en memoria durante la ejecución → si no está (fallo de página/segmento) se trae de memoria secundaria.

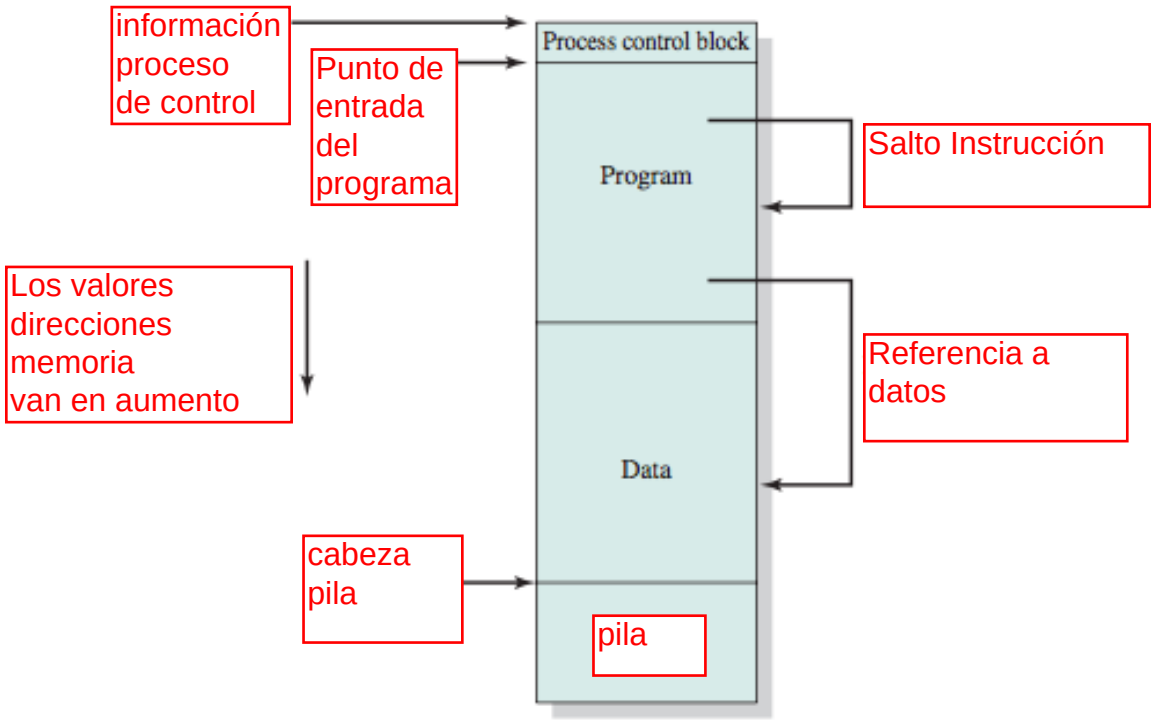


Figure 7.1 Addressing Requirements for a Process

2 Particionamiento

- Particionamiento Fijo
- Particionamiento Dinámico

2.1 Particionamiento Fijo

- Particiones de igual tamaño → Cualquier proceso con un tamaño ≤ al tamaño de partición puede ser cargado en una partición libre.
- El SO puede intercambiar un proceso a otra partición → si no está en estado Ready o Running.
- Si un programa no cabe en una partición → se buscaría lugar según algoritmo, pero si no, no se ejecuta
- **Memoria principal insuficiente** → un hueco pequeño puede ocupar una partición grande dejando espacios inutilizables → **fragmentación interna**.

Particiones de mismo tamaño, distinto y ejemplo overlay (de izquierda a derecha):

Figure 7.2 **Example of Fixed Partitioning of a 64-Mbyte Memory**

Ejemplo de particionamiento fijo de una memoria de 64 MiB. Se puede entender perfectamente con tamaños mayores de 1, 2, 4, 8 GiB, etc. Puesto que el funcionamiento es el mismo.
La fragamentación interna se produce cuando, por ej., un proceso de 1MiB ocupa una partición de 8MiB. Se desaprovechan 7MiB porque se asigna todo el espacio de esa partición.

- Se puede ver que la elección de particiones distintas ayuda a mejorar la ubicación de procesos de tamaño distinto → se reduce la **fragmentación interna.**
- Aunque persisten problemas de rendimiento:
  - Número de procesos activos limitados por el número de particiones.
  - Los procesos de tamaño pequeño no utilizan de manera eficiente el espacio en memoria → se mejora con la elección de particiones diferentes.

2.2 Particionamiento Dinámico

- Las particiones son de longitud y número variables.
- Los procesos se cargan en memoria principal y se le asigna exactamente tanta memoria como necesite.
- Siempre quedan pequeños huecos → **Fragmentación Externa** → Se puede resolver mediante **compactación**.
- **Compactación:** El SO mueve los procesos que están contiguos para quitar los espacios → Se gasta mucho tiempo y procesamiento de CPU.

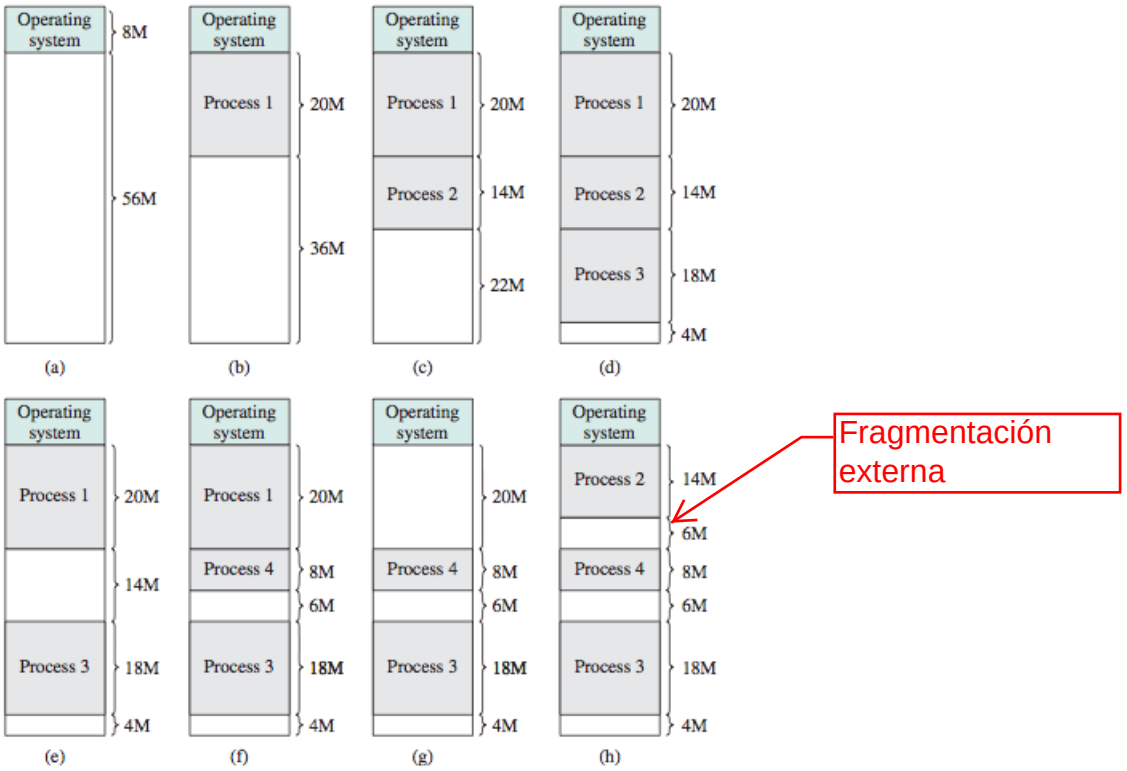


Figure 7.4 **The Effect of Dynamic Partitioning**

- El SO es el que decide que bloque libre asigna al proceso y lo hace mediante tres algoritmos:
  - First Fit (Primer ajuste): Selecciona el primer bloque disponible de tamaño suficientemente grande → suele ser el más eficiente.
  - Best Fit (Mejor ajuste): Selecciona el bloque disponible de tamaño más próximo solicitado.
  - Next Fit (Siguiente ajuste): Desde la última ubicación y elige el bloque disponible suficientemente grande. → Necesista compactación frecuente.

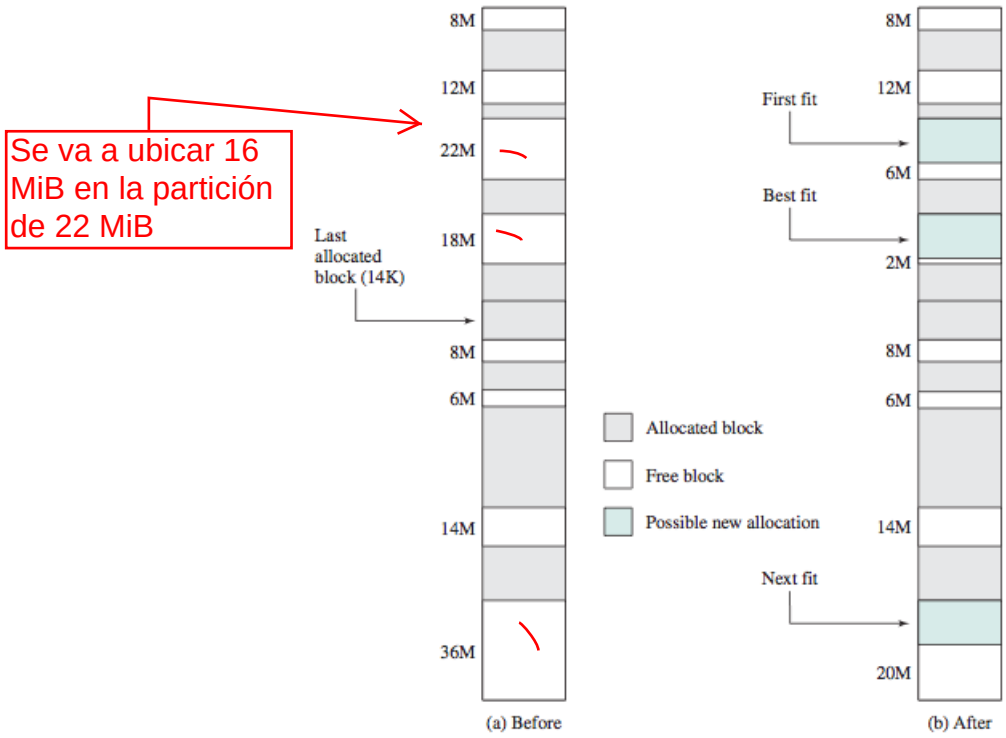


Figure 7.5 **Example Memory Configuration before and after Allocation of 16-Mbyte Block**

3 Memoria Virtual

(a) Memoria Real: Memoria principal, RAM.

(b) Memoria Virtual: Memoria en disco → Permite multiprogramación efectiva y libera al usuario de costosas gestiones (overlays).

- **Trashing:** Estado en el cual el sistema gasta mucho tiempo intercambiando (**swapping**) pedazos de procesos más que ejecutándolos.
- Evitar *trashing*: el SO intenta adivinar qué partes son menos usadas en el futuro → historia reciente.

(c) **Principio de localidad:** El programa y los datos referenciados dentro de un proceso tienen a concentrarse → Sólo unas cuantas piezas de un proceso serán necesarias en un corto espacio de tiempo ⇒ Las predicciones de la Memoria Virtual suelen ser eficientes.

(d) Se pude observar en la siguiente figura 8.1 el comportamiento de las referencias de páginas de memoria por un proceso → Se puede observar que existen algunas que son más referenciadas.

- El Hardware debe soportar paginación y segmentación.
- El SO debe permitir gestionar el movimiento de páginas y/o segmentos entre memoria secundaria y principal.

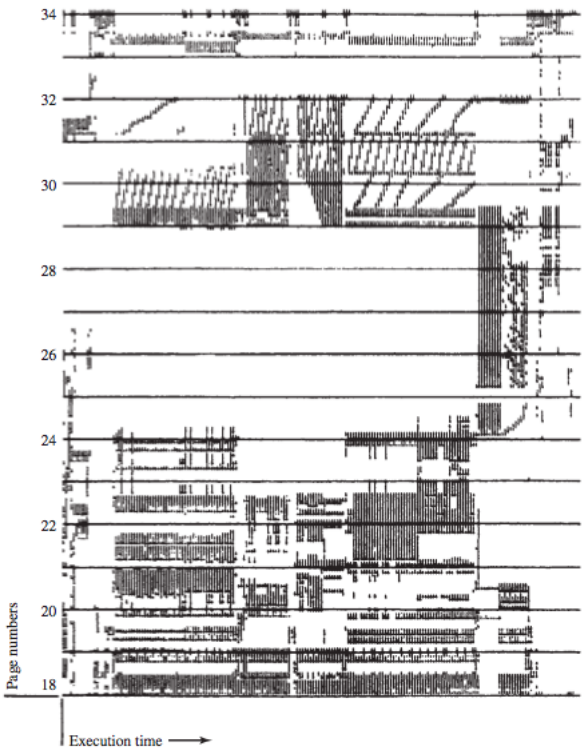


Figure 8.1 **Paging Behavior**

Durante el tiempo de vida de un proceso, las referencias se reducen a un subconjunto de páginas.

3.1 Paginación

- Cada proceso tiene su propia tabla de páginas.
- Cada entrada de la tabla de páginas contiene el número de marco de la página que le corresponde en memoria principal.
- Los pedazos de un proceso se llaman *pages* o páginas mientras que los de memoria son llamados *frames* o marcos.
- Hay dos bits extra que indican:
  - Si la página está en memoria o no.
  - Si los contenidos de la página han sido alterados desde que se cargaron.

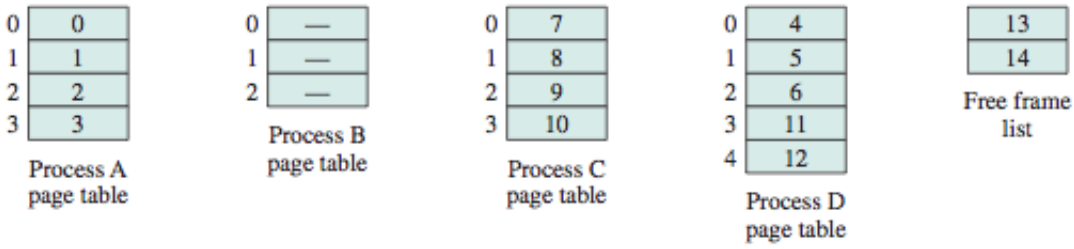


Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

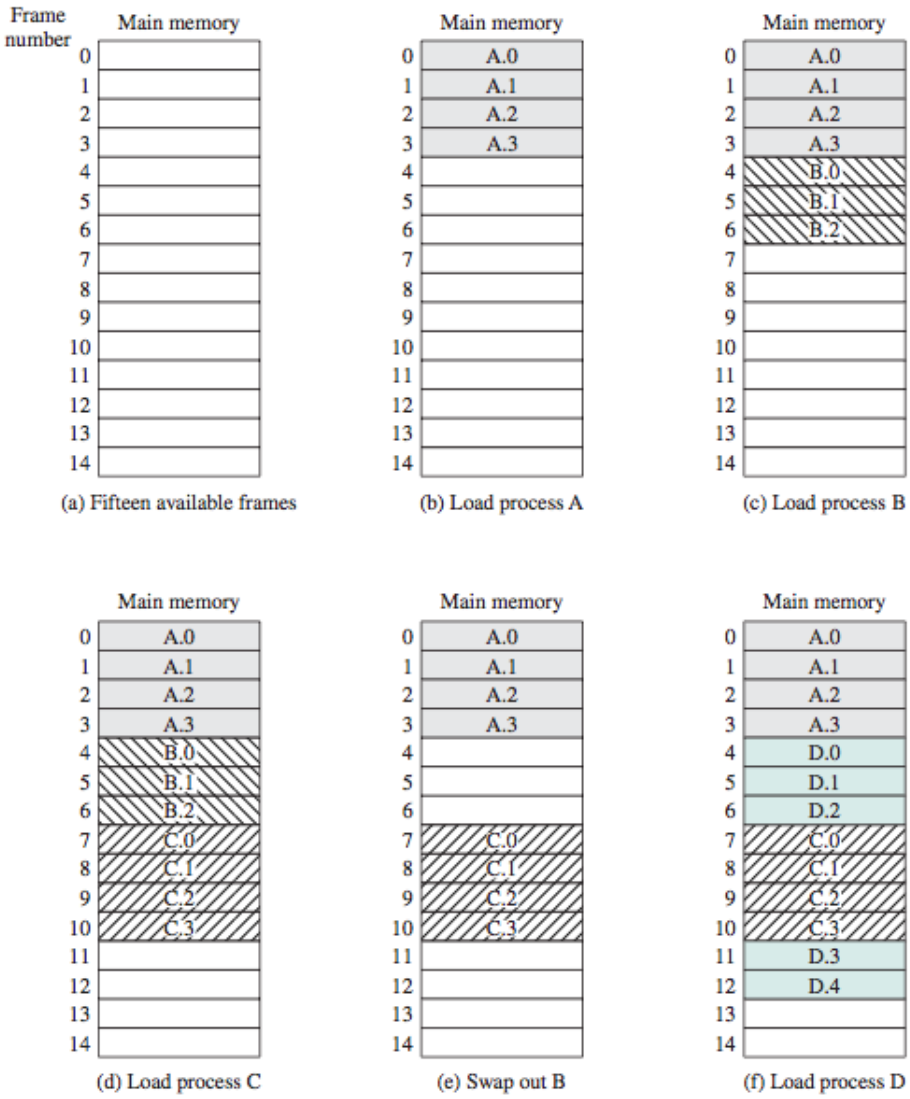
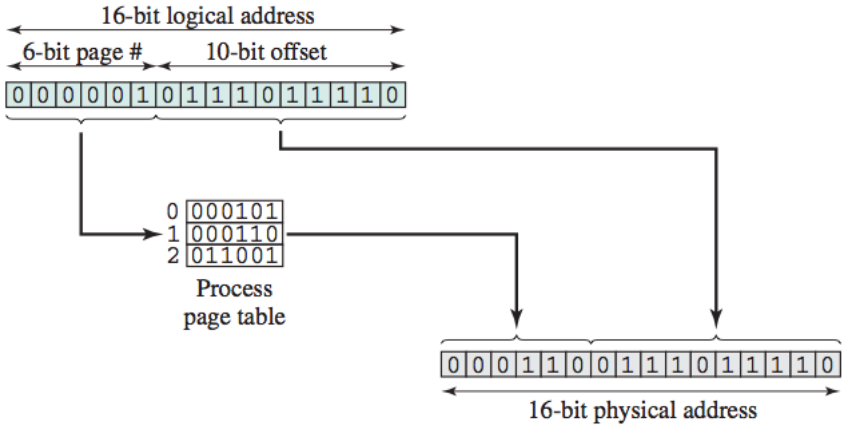


Figure 7.9 Assignment of Process to Free Frames



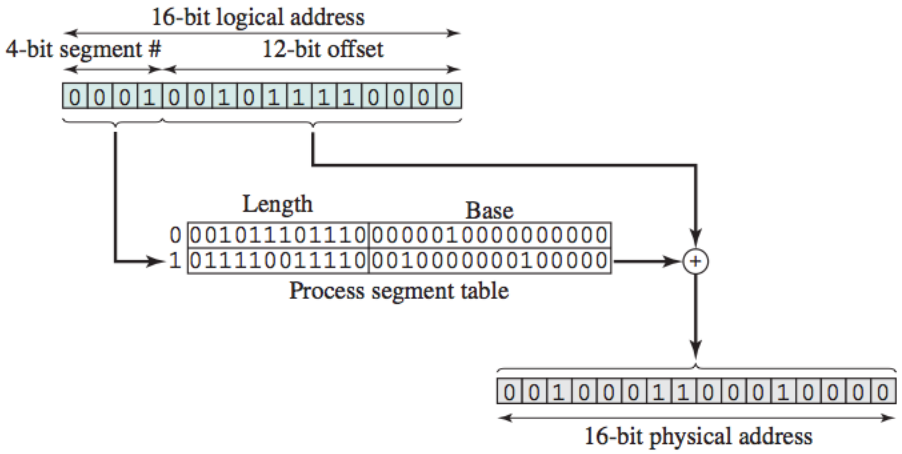
(a) Paging

(e) En el ejemplo anterior: tenemos la dirección lógica 0000010111011110 y que se compone de:

- El número 1 de la página y un offset de 478.
- Suponemos que esta página reside en el marco 6 (000110) en memoria principal.
- Luego la dirección física es → marco de página 6, offset 478 = 0001100111011110

3.2 Segmentación

- Un programa puede ser dividido en varios segmentos.
  - Segmentos de diferentes longitudes.
  - Hay un máxima longitud de segmento.
- Direccionamiento consiste en dos partes:
  - Un número de segmento y on offset (desplazamiento).
- **Segmentación → similar al particionamiento dinámico**



(b) Segmentation

- Resumen de las técnicas de particionamiento, paginación y segmentación:

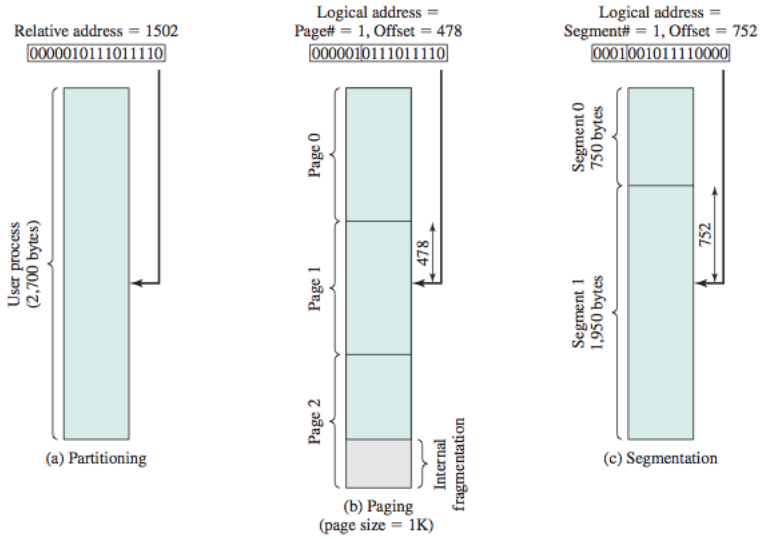


Figure 7.11 **Logical Addresses**

- En nuestro ejemplo, tenemos la dirección lógica: 0001001011110000, la cual tiene:
  - Segmento número 1, offset 752.
  - Suponemos que este segmento reside en memoria principal comenzando en la dirección física: 0010000000100000.
  - Con lo que la dirección física es: 0010000000100000 + 001011110000 = 0010001100010000

3.3 Algoritmos de planificación de memoria

Definición 3.1. Política de reemplazo: cuando todos los marcos en memoria principal son ocupados y si es necesario traer una nueva página, la política de reemplazo determina qué página en memoria va a a ser reemplazada.

- **¿Qué página es reemplazada?**
- **¿Cómo se determina?**

Existen algoritmos básicos que se utilizan para la selección de una página a reemplazar:

- Óptimo
- Menos recientemente usado (LRU - Least resently used)
- First-in-First-Out (FIFO)

Los veremos con el siguiente ejemplo de implementación de estas políticas en referencia a un programa que se ejecuta y va necesitando la siguiente secuencia de páginas:

- Páginas referenciadas: 2 3 2 1 5 2 4 5 3 2 5 2
- Esta secuencia significa que la primera página referenciada es 2, la segunda 3, y así sucesivamente.

(a) Algoritmo Óptimo:

- Para el reemplazo selecciona la página que más tiempo va transcurrir para la próxima referencia.
- Es imposible tener un conocimiento perfecto de eventos futuros.
- El algoritmo óptimo produce tres fallos de página después de que el marco se haya rellenado (Figura 8.15).

(b) LRU:

- Reemplaza la página que no ha sido referenciada en el tiempo más largo.
- Por el principio de localidad → página que menos probable sea referenciada in el futuro próximo.
- Difícil de implementar → una solución es poner una etiqueta para cada página con el tiempo de la última referencia → mayor sobrecarga.
- En el ejemplo de la Figura 8.15 es tan bueno como el Óptimo. (4 fallos de página).

(c) FIFO

- En el ejemplo tenemos 6 fallos de página.
  - Nótese que LRU reconoce que las páginas 2 y 5 son más frecuentemente utilizadas que otras, mientras que FIFO no.

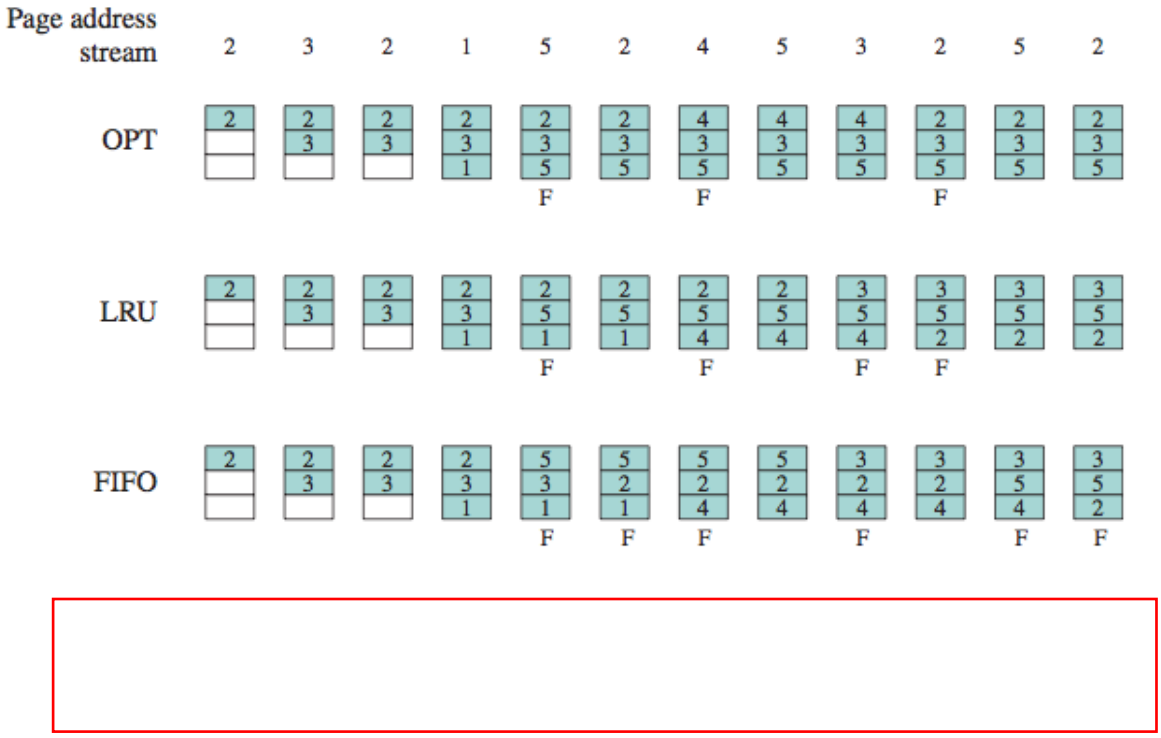


Figure 8.15 **Behavior of Four Page Replacement Algorithms**

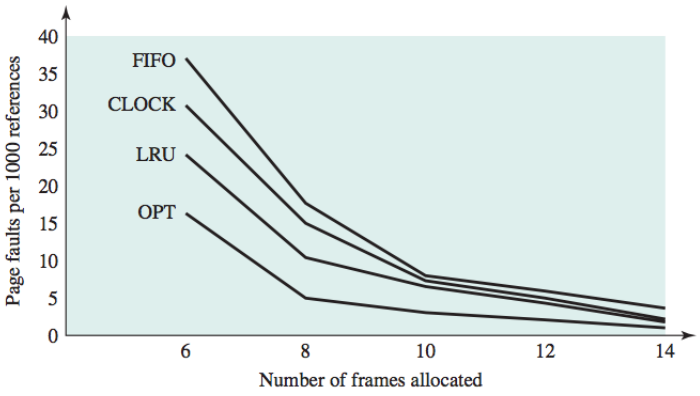


Figure 8.17 **Comparison of Fixed-Allocation, Local Page Replacement Algorithms**

4 Scheduling o planificación del procesador

Definición 4.1. El objetivo de la *planificación del procesador* es asignar los procesos que serán ejecutados por el procesador sobre un tiempo determinado → su objetivo es conseguir un *tiempo de respuesta mínimo,* rendimiento (*throughput*) y *eficiencia del procesador*.

(a) Objetivos de la Planificación:

- Compartir el tiempo entre los procesos.
- Prevenir la Inanición de los procesos.
- Utilizar de forma eficiente el procesador.
- Tener baja sobrecarga.
- Priorizar los procesos cuando sea necesario (deadlines).

(b) **Tipos de Scheduler o Planificador:**

- Planificador a Corto Plazo
  - Conocido como *dispatcher.*

  - Es el que más frecuentemente se ejecuta.
  - Invocado cuando un evento ocurre (I/O interrupts, signals, OS Calls).
- Planificador a Medio Plazo
  - Es parte de la función de decisión de intercambio basadas en la gestión del grado de multiprogramación. cuántos procesos/hilos deben estar en memoria (cola preaparados)
- Planificador a Largo Plazo
  - Determina qué programas son admitidos en el sistema para procesarlos (FCFS, RR).
  - Controla el grado de multiprogramación.
  - A más procesos → menor porcentaje de tiempo para cada proceso.

(c) **Modos de Decisión:**

no apropiativo
- **Non-preemptive:** Una vez que el proceso está en estado de ejecución, continuará hasta que termine o se bloque por una I/O.

expropiativo
- **Preemptive:** Los procesos de ejecución pueden ser interrumpidos cuando un nuevo proceso llega o de forma periódica.

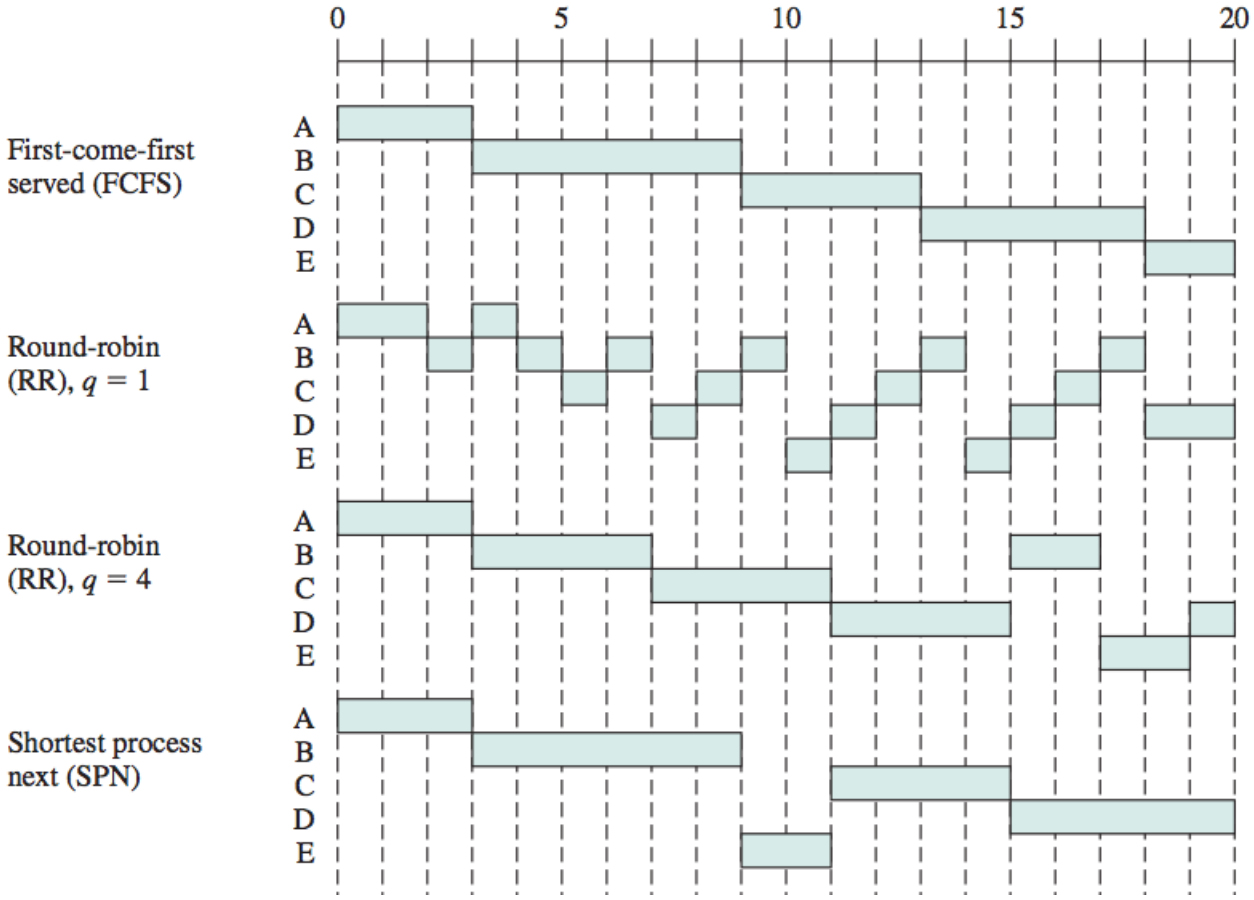
4.1 Algoritmos de Planificación de CPU

Consideremos el siguiente conjunto de procesos (procesos batch) en la siguiente tabla:

Table 9.4 Process Scheduling Example

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

- **First-Come-First-Served**
  - Los process van llegando a la cola de Ready por orden de llegada → Non-preemptive.
  - En el ejemplo se observar que cuando el proceso actual acaba, el más largo de la cola de Ready es seleccionado.
  - Un proceso corto puede esperar mucho tiempo antes de que sea ejecutado.
  - Se favorecen los procesos largos.
  - Es eficiente cuando hay pocos procesos.



- **SPF/SJF (Short Process/Job First)**
  - Pre-emptive.
  - Se seleccionan los procesos con el tiempo de procesamiento más corto.
  - Los procesos cortos son elegidos antes que los largos → Posible inanición.
- **Round-Robin:**
  - Utiliza una política preemptive mediante un reloj → es también conocido como *rebanada de tiempo* porque cada proceso tiene un pequeño pedazo de tiempo (**quantum**).
  - Cada interrupción de reloj (marcada por el quantum) se genera a intervalos periódicos.
  - Cuando una interrupción ocurre (cada quantum de tiempo) el proceso que se estaba ejecutando pasa a la cola de Ready y entra el siguiente.
- **Scheduling en Linux/Unix:**
  - Es una cola multinivel que utiliza RR con colas de prioridades.
  - Quantum de 1 segundo.
  - La prioridad está basada en tipos de procesos e históricos de ejecución.
  - Tanto Windows como MacOs también utilizan colas multinivel.

4.2 Fórmulas:

- Utilización del procesador: $P_u = Proc_{ocupado}/(Proc_{ocupado} + Proc_{desocupado})$ *100
- Throughput: $T = N°ProcesosCompletos/TiempoTotal$ <= 1
- Turnaround Time: $Tat = T_{proceso_fin} - T_{proceso_llegada}$
- Tiempo de respuesta: $Tr = T_{inicio_ejecucin} - T_{proceso_llegada}$

5 Taller de Laboratorio

6 Bibliografía

References

"Fundamentos de Sistemas Operativos". Capítulos 3,4,5. Gunnar Wolf y otros. Creative Commons

Métricas
Unidades a manejar
Para hablar de planificación del procesador, no se utilizan tiempos estándar (s, ms, ns), sino que:
Tick: Un tiempo mínimo dado durante el cual se puede realizar trabajo útil. Medida caprichosa y arbitraria.
* En Windows, un tick dura entre 10 y 15 ms.
* En Linux (2.6.8 en adelante), dura 1 ms.
Quantum: Tiempo mínimo, expresado en ticks, que se permitirá a un proceso el uso del procesador.
 * En Windows, 2 a 12 ticks (esto es, 20 a 180ms).
* En Linux, 10 a 200 ticks (10 a 200ms)