

SISTEMAS OPERATIVOS I

UNIDAD IV

ADMINISTRACIÓN DE MEMORIA



INSTITUTO TECNOLÓGICO
DE MORELIA

Departamento de Sistemas y
Computación

Disponible en: www.benito.org.mx

M.C. Benito Sánchez Raya

sanchezraya@hotmail.com

CONTENIDO

1. Administración de memoria básica
2. Intercambio (Swapping)
3. Memoria virtual
4. Algoritmos para reemplazo de páginas
5. Modelado de algoritmos de reemplazo de páginas
6. Aspectos de diseño de los sistemas con paginación
7. Aspectos de implementación
8. Segmentación

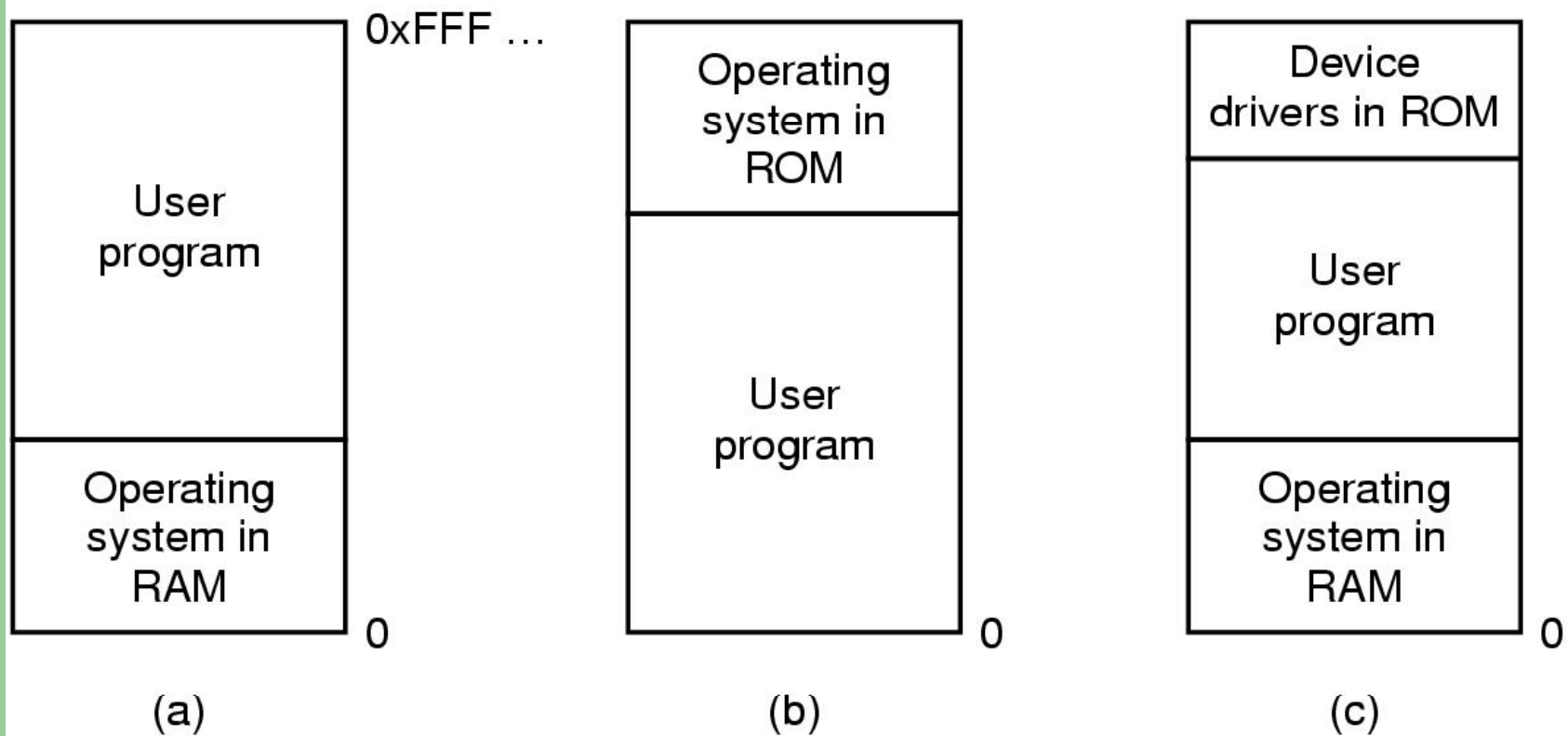
1. ADMINISTRACIÓN DE MEMORIA BÁSICA

- Jerarquías de memoria:
 - Caché → Pequeña, rápida, cara y volátil.
 - RAM → Mediana, regular costo, regular velocidad, y volátil.
 - D. Duro → Enormes, lentos, baratos y permanente.
- Parte del SO que administra la jerarquía de memoria se llama: ***Administrador de Memoria.***

- Administrador de memoria:
 - Mantener las áreas de memoria usadas y vacías.
 - Asignar memoria a los procesos cuando la requieren.
 - Liberarla cuando finalicen los procesos.
 - Administrar los intercambios entre la memoria principal y el disco.
 - Cuando la primera es pequeña, y no alberga a todos los procesos.
- Intercambio y paginación
 - Carga y descarga de procesos entre la memoria y el disco.

1.1. Monoprogramación sin intercambio ni paginación

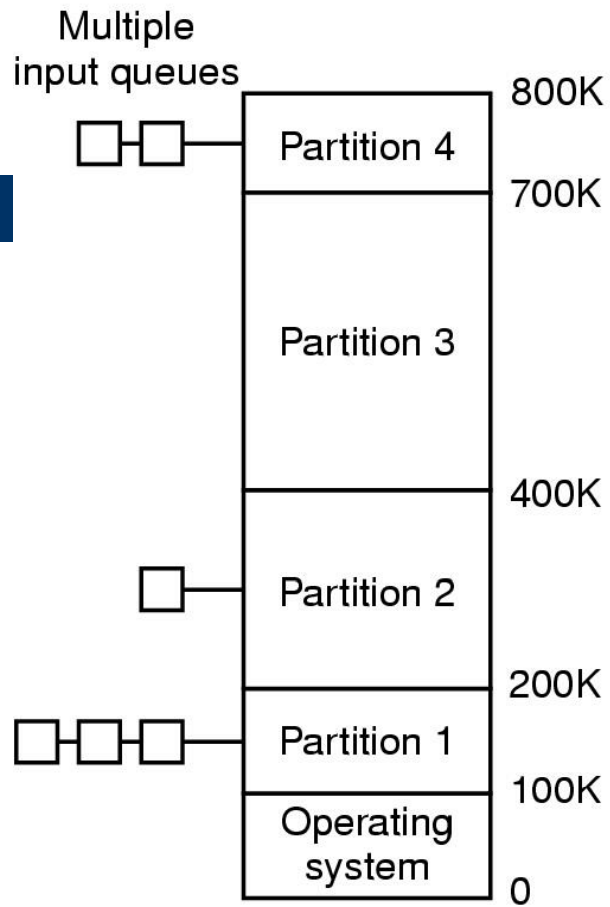
- Ejecutar un programa a la vez.
 - Compartir la memoria entre el programa y el SO.
- En la figura de la siguiente diapositiva se ilustran tres variaciones:
 - En la primera variante se usaba en mainframes y minicomputadoras.
 - La segunda, en sistemas de bolsillo y sistemas integrados.
 - La tercera, se empleo en PCs, era típico de MSDOS. En ROM esta el BIOS.
- Se tecllea un comando, el SO carga el programa de disco a memoria, lo ejecuta, al finalizar, se prepara para cargar un nuevo programa y sobrescribir la memoria



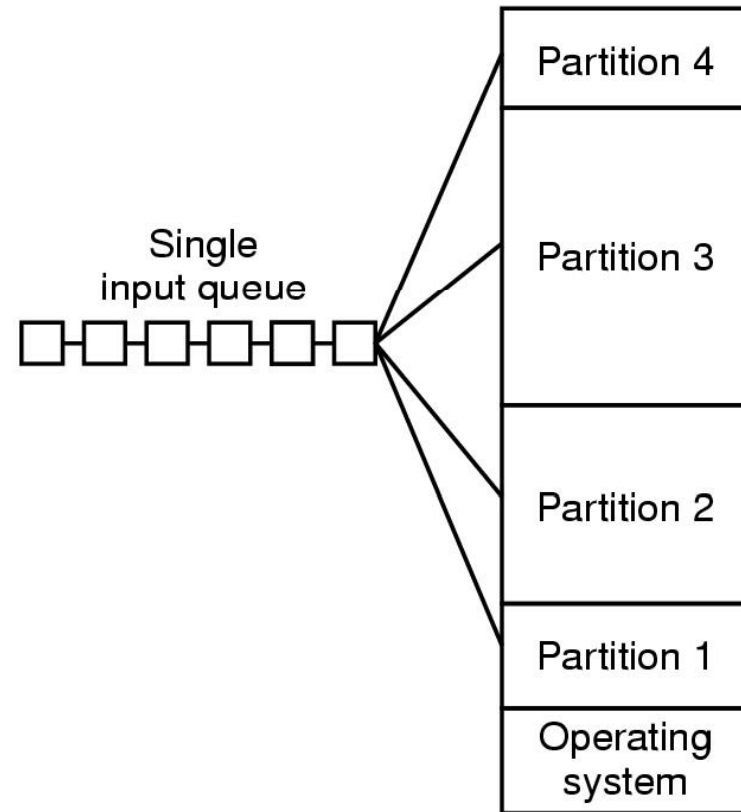
- a) SO en la parte baja de la RAM. b) SO parte alta de la ROM
 c) Controladores de dispositivos en ROM y el resto en RAM.

1.2. Multiprogramación con particiones fijas

- Ya existe la multiprogramación.
- Consiste en dividir la memoria en n particiones (iguales o desiguales)
 - Normalmente en forma manual y antes de arrancar el sistema.
- En la figura siguiente, en el inciso a):
 - Al llegar un trabajo se coloca en la cola de entrada de la partición más pequeña en la que cabe.
 - Por ser particiones fijas se desperdicia el espacio no ocupado por un trabajo (más pequeño el trabajo que la partición).
 - Desventaja: Puede haber una partición grande vacía mientras en la cola de una partición chica hay muchos trabajos formados, como las particiones 1 y 3 de la figura.



(a)



(b)

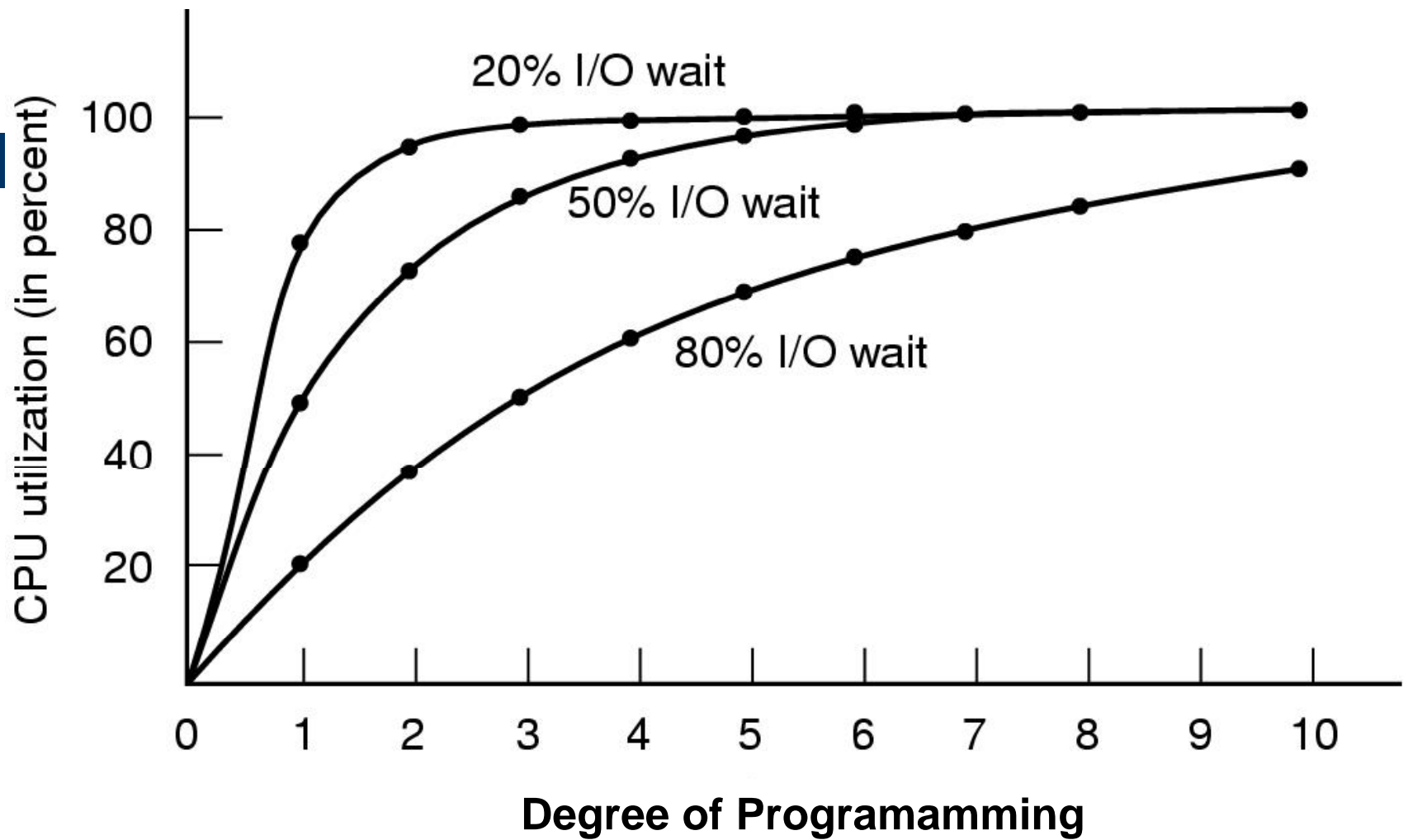
- a) Particiones de memoria fijas, con colas de entrada individuales.
- b) Particiones de memoria fijas, con una sola cola de entrada.

- En la figura siguiente, en el inciso b):
 - Otra opción es mantener una sola cola.
 - Al desocuparse una partición el trabajo al frente de la cola que quepa en esa partición la ocupará.
 - Es decir, ocupará una partición lo mas justa a su tamaño.
 - **Una variante sería:**
 - Escoger de la fila el trabajo que mas se acerque al tamaño de la partición.
 - Desventaja:
 - Discriminaría a los trabajos pequeños. Por que estos desperdiciarían espacio.
 - Cuando debería darles mejor servicio, porque normalmente son trabajos interactivos.

- Una estrategia sería:
 - Cada vez que se ignore un trabajo, del frente de la fila, se le da un punto; y después de n puntos ya no se le puede ignorar.
- Este sistema de particiones fijas se uso en el OS/360 de IBM
 - Se llamaba MFT (Multiprogramación con número fijo de tareas).
 - Ya no se usa.

1.3. Modelado de la Multiprogramación

- Consiste en ver el aprovechamiento de la CPU desde un punto de vista probabilístico.
- Supongamos que:
 - Hay n procesos en la memoria al mismo tiempo.
 - Un proceso pasa un porcentaje de tiempo p (en %) esperando a que terminen operaciones de E/S.
 - La probabilidad de que todos estén esperando E/S es de p^n .
 - Por lo tanto el aprovechamiento de la CPU esta dado por:
 - Aprovechamiento de la CPU = $1-p^n$.
 - A esto se le denomina *grado de multiprogramación*.





- Ejemplo:

- Si una computadora tiene 32 MB de RAM
- 16 son para el SO
- Los programas de usuario requieren 4 MB de memoria para ejecutarse.
- Con una espera de E/S del 80% para cada uno.
- Podríamos tener 4 programas cargados en memoria al mismo tiempo.
- Calcular su aprovechamiento:

– Aprovechamiento de la CPU = $1-p^n$.

- $1-0.8^4$
- $1-0.4096$
- 0.5904
- **59%**

Si aumentamos otros 16 MB de RAM, ¿Cuál sería el aprovechamiento del CPU?

- Solución:

- Serían 8 programas a la vez en memoria
- El aprovechamiento sería:
 - $1 - 0.8^8$
 - $1 - 0.167$
 - 0.833
 - **83%**
- Es decir, al aumentarle 16 de RAM sube el rendimiento en un 24%.

¿Y Si aumentamos otros 16 de RAM?

- Solución:
 - El aprovechamiento sería del 93%
 - Subiría en 10%.
- En base a este modelo se puede decir que la primera adición de RAM es buena inversión, pero la segunda no lo es tanto.
- La idea principal de este modelado es que la multiprogramación permite a los procesos usar la CPU cuando de otra manera estaría inactiva.

1.4 Análisis del desempeño de un sistema multiprogramado

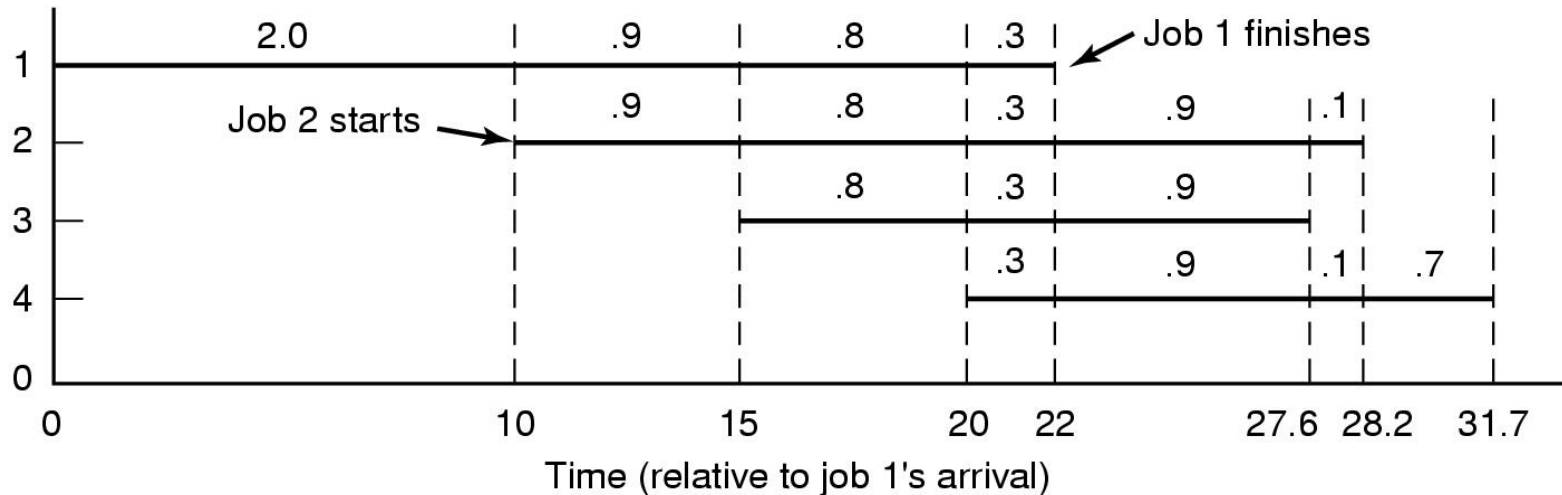
- Consideremos un área de cómputo con procesamiento por lotes:
 - Sus trabajos esperan 80% en E/S
 - En la figura siguiente inciso a)
 - Llegan 4 trabajos a ejecutarse, en el orden ya especificado y en esos tiempos, con su cantidad de minutos requeridos.
 - El primero llega a las 10:00 am, requiere 4 min. para ejecutarse.
 - Utiliza solo **12** seg de CPU por cada minuto en memoria.
 - El trabajo tendrá que permanecer en la memoria: **20 min.**
 - Para poder realizar 4 min. de CPU.
 - Aun sin tener competencia empleara ese tiempo.

Job	Arrival time	CPU minutes needed
1	10:00	4
2	10:10	3
3	10:15	2
4	10:20	2

(a)

	# Processes			
	1	2	3	4
CPU idle	.80	.64	.51	.41
CPU busy	.20	.36	.49	.59
CPU/process	.20	.18	.16	.15

(b)



(c)

- a) Llegada y requerimientos de 4 trabajos b) Aprovechamientos de la CPU
 c) Serie de sucesos, los números arriba de las líneas indican el tiempo de CPU en cada intervalo

- De 10:00 a 10:10 estará solo en el CPU:
 - Empleará **2** min de su tiempo.
- El aprovechamiento del CPU aumentaría de **20%** a **36%**. (calculados con $1-p^n$)
- Pensemos en calendarización round-robin
 - Cada uno recibe el 50% del tiempo del CPU, es decir, **0.18 min** de CPU de cada minuto real ($36\%/2$).
- La aparición del segundo trabajo solo le quita el **10%** del tiempo de CPU al primero (de 20 a 18 min).

- A las 10:15 llega el tercer trabajo
 - El trabajo 1 ha recibido: **2.9** min
 - El trabajo 2 ha recibido: **0.9** min.
 - Con multiprogramación de tres vías, cada trabajo recibe: **0.16** minutos de CPU por minuto real.
 - Aprovechamiento CPU = 48%
 - $48\% / 3 = 16\% = 0.16$ min
- El trabajo 1 termina en el tiempo 22, debido a la aparición del trabajo 2.

1.5. Reubicación y protección

- Cuando se enlaza (código+procedimientos+bibliotecas) un programa:
 - El enlazador requiere saber en que parte de la memoria iniciará el programa.
- Ejemplo:
 - Hay particiones de 100 K
 - El SO esta en los primeros 100 K
 - Un programa se carga en la partición 1
 - La primer línea del programa es una llamada a la biblioteca que se encuentra en la dirección 100 de dicha aplicación.
 - Esta dirección saltará a la dirección absoluta 100, cuando debería ser a la dirección $100+100$
 - Si se cargase en la partición 2, debería ser $200+100$, en lugar de 100.

- El análisis anterior es el problema de la reubicación
- Una posible solución sería que si el programa:
 - Se carga en la partición 1, sumar 100 a sus direcciones
 - Se carga en la partición 2, sumar 200 a sus direcciones
 - Así sucesivamente
- OS/MFT de IBM trabajaba bajo este esquema.



- La protección

- Consiste en proteger de accesos de algunos procesos a áreas de memoria que no les pertenecen.
- En este nivel todas las direcciones son absolutas.
 - De ahí el riesgo latente.

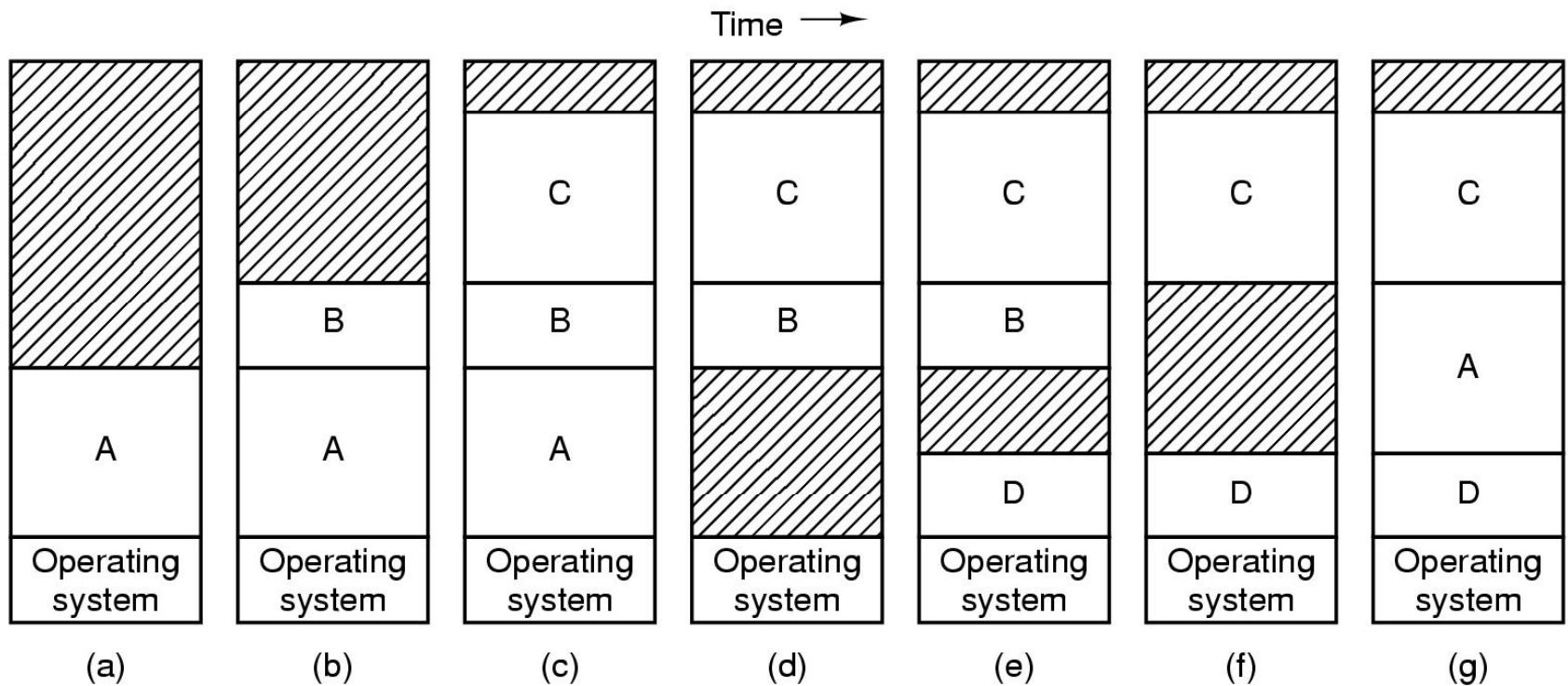
- IBM protegió sus equipos 360 de la siguiente manera:
 - Dividir la memoria en bloques de 2 KB.
 - Asignar códigos de protección de 4 bits a cada bloque.
 - En la palabra de estado del programa (PSW) se colocaban los 4 bits de protección del bloque del proceso en ejecución.
 - Solo el SO podía modificar los bits de la PSW.
 - El hardware de la IBM 360 detectaba cuando un proceso intentaba acceder a un bloque cuyos bits difieren de los de la palabra de estado (PSW).

- Otra solución a la reubicación y protección consistía:
 - Equipar la máquina con dos registros especiales de hardware:
 - Base y límite
 - Al calendarizar un trabajo
 - El registro base almacenaba la dirección donde iniciaba la partición en donde se cargo el trabajo.
 - El registro límite guardaba la longitud de la partición.
 - Cada que se genera una dirección se le suma el registro base.
 - El hardware protege de otros usuarios la modificación de los registros.

- Ejemplo:
 - Si el registro base tiene la dirección 100
 - Una llamada CALL 100, se transformaría en una llamada CALL 100+100.
 - Esto sin modificar la instrucción en si. Como era el caso de la primera opción.
 - También se debe estar verificando el registro límite para evitar desbordamientos.
- Una desventaja: Haría una suma y una comparación cada vez que se accese a la memoria
- La CDC 6600 usaba este esquema.
- Intel 8088 en IBM-PC usaba solo el registro base.

2. INTERCAMBIO (swapping)

- En un sistema de procesamiento por lotes es muy fácil organizar la memoria en particiones fijas.
- En sistemas compartidos no siempre se pueden tener en memoria los programas activos.
 - Los programas excedentes se mantienen en disco y se traerán en forma dinámica a la memoria para su ejecución.
- Esto se puede solucionar con:
 - Intercambio
 - Traer a la memoria un proceso entero y ejecutarlo por un rato y volver a guardarlo en disco.
 - Memoria virtual
 - Permite que los programas se ejecuten, aunque solo una parte de ellos este en memoria principal.



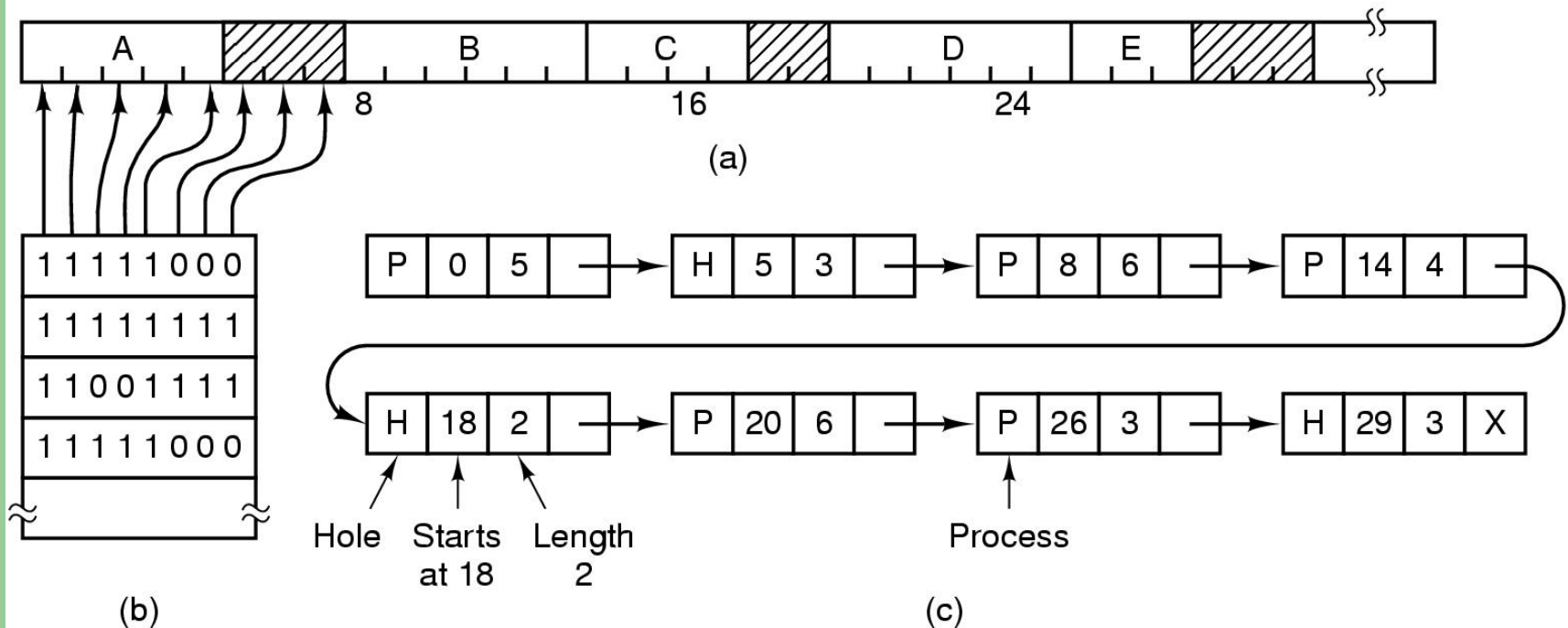
Cambios en la asignación de memoria por la carga y descarga de procesos.
Lo sombreado es memoria vacía.

- En a): Solo el proceso A esta en memoria
- Después se crean o se traen a memoria los procesos B y C.
- En d): A se intercambia al disco.
- Luego llega D y B sale.
- Al final se vuelve a traer a memoria A.
 - Ahora quedo en un lugar distinto, es preciso reubicar las direcciones que contiene, ya sea:
 - Por hardware (Durante la ejecución del programa)
 - Por software (En el momento del intercambio)

- Aquí las particiones son variables en cantidad y tamaño, conforme van llegando los procesos
 - Esto complica su control.
- Compactación de memoria:
 - Cuando el intercambio crea muchos huecos, se trata de crear un solo hueco pero de mayor tamaño, desplazándolo hacia abajo en la medida de lo posible.
 - No es común (requiere mucho tiempo CPU)

- ¿Cuánta memoria darle a un proceso?
 - Si los procesos son de tamaño fijo:
 - Se asigna lo que requiera.
 - Si los segmentos de datos de los procesos suelen crecer (ejemplo asignación dinámica)
 - Si hay un hueco adyacente, se le podrá asignar.
 - Moverse a un área de memoria mas grande.
 - Si no hay, bajar otros procesos a disco.
 - Si aun así no hay, tendrá que eliminarse.

2.1. Administración de memoria con mapas de bits



- a) Memoria con 5 procesos y tres huecos, zonas sombreadas vacías y con 0.
 b) Con mapa de bits c) Con listas enlazadas

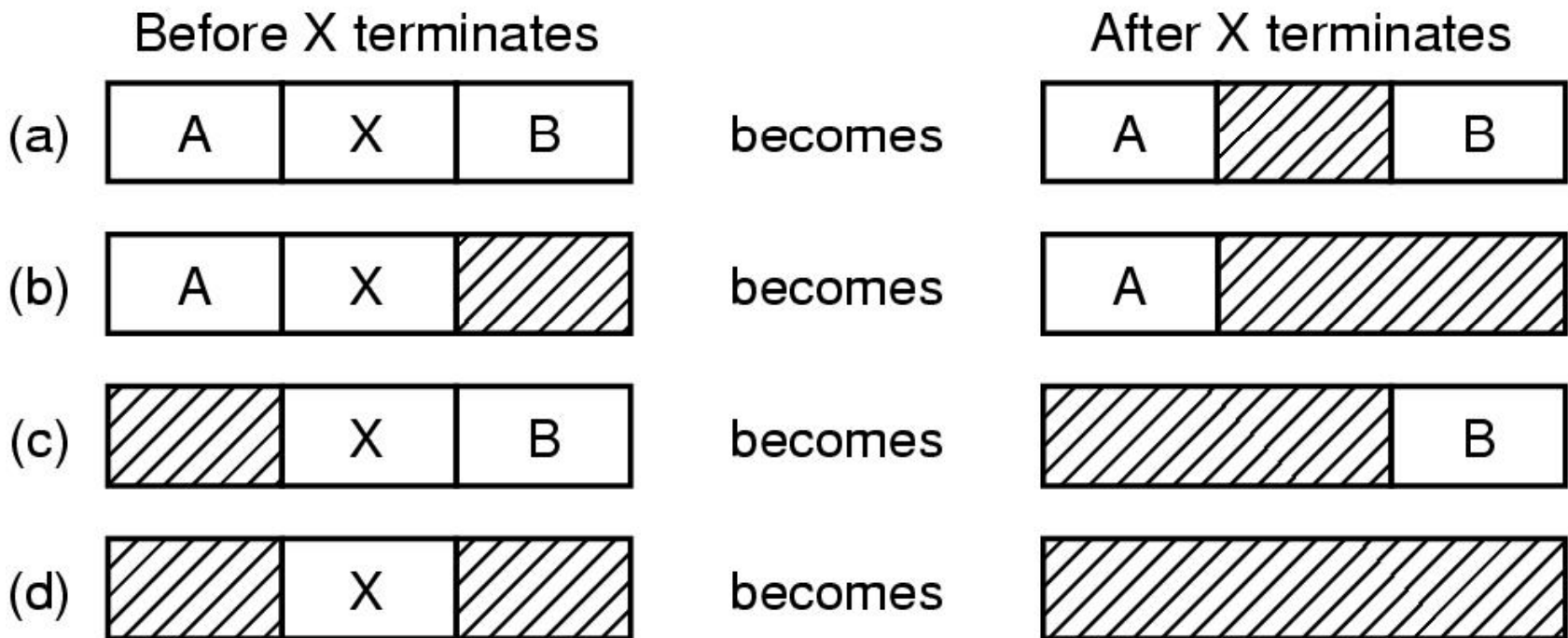
- Forma de administrar la memoria dinámica.
- La memoria se divide en unidades de asignación
- A cada unidad de asignación le corresponde un bit
- Bit en 0 desocupada, en 1 asignada.
- Tamaño de la unidad de asignación:
 - Tamaños pequeños, mayor mapa de bits
 - Tamaños grandes, mapa de bits pequeño.
 - Habría desperdicio de memoria por el proceso en su último bloque (sino es múltiplo de la unidad de asignación)

- El tamaño de asignación solo dependerá del total de memoria libre.
- Un inconveniente sería que al subir un proceso más a memoria, se tendría que recorrer todo el mapa de bits en busca de k bits consecutivos en 0, lo cual sería lento.

2.2. Administración de memoria con listas enlazadas

- Consiste en mantener una lista enlazada de segmentos de memoria asignados y libres.
- Un segmento puede ser:
 - Proceso o
 - Hueco
- Cada nodo de la lista:
 - Proceso o Hueco
 - Dirección de inicio
 - Longitud
 - Apuntador al siguiente nodo

- Cuando un proceso termina o se “baja” a disco, la actualización (4 combinaciones) de la lista es:



- Cuando un proceso se crea o se “sube” a memoria:
 - Algoritmo del primer ajuste:
 - Se explora la memoria hasta hallar un hueco lo suficientemente grande donde quepa.
 - El hueco se divide en dos partes (proceso y hueco restante, a menos que sea exacto)
 - Algoritmo del siguiente ajuste:
 - Similar al anterior, excepto que se explora la memoria a partir de donde se realizó el anterior ajuste, no lo hace desde el principio.
 - Suele ser más lento que el anterior.

- Algoritmo del mejor ajuste:

- Se explora toda la lista y se escoge el hueco más pequeño que alcance.
- Es más lento.
- Desperdicia más memoria que los dos anteriores. Porque deja huecos pequeños inservibles.

- Algoritmo del peor ajuste:

- Analiza toda la lista y toma el hueco más grande que encuentre.
- Para dejar huecos de tamaño aceptable y reutilizables.

- Algoritmo del ajuste rápido:

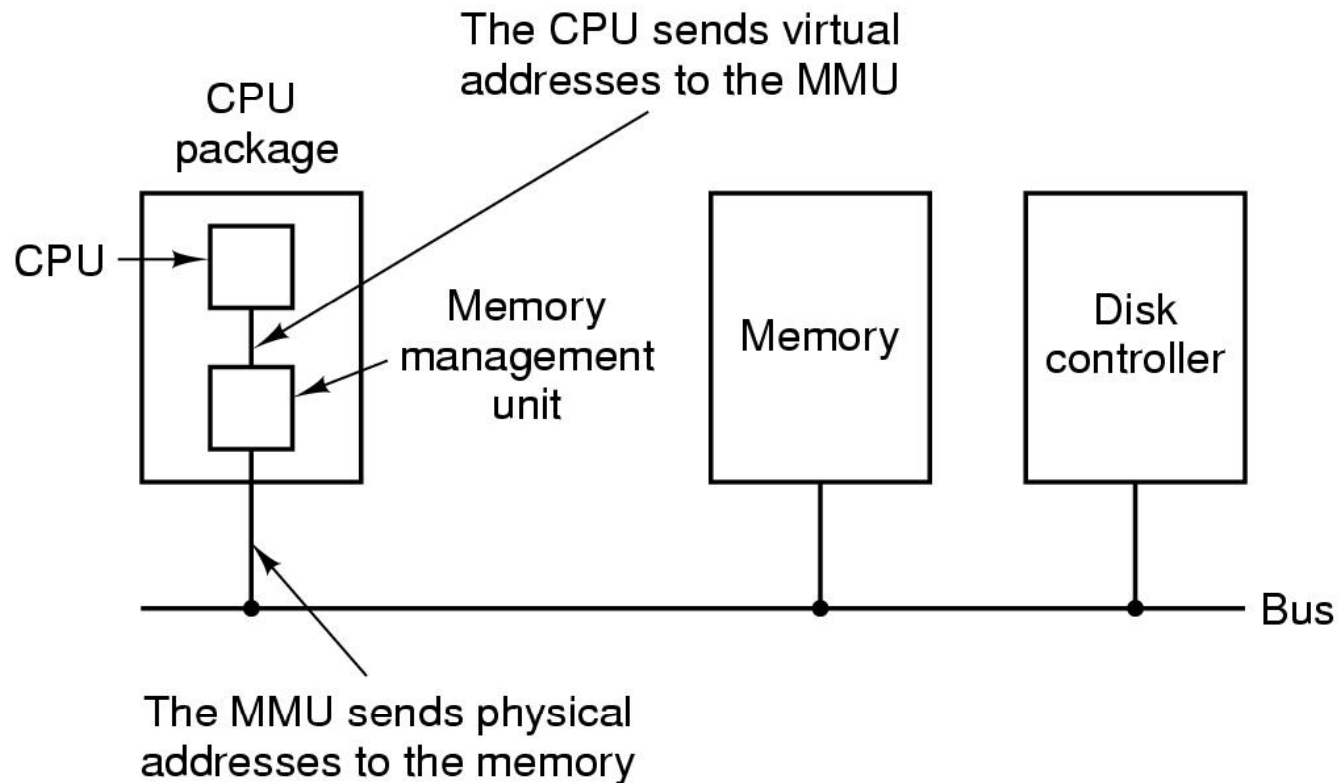
- Mantiene listas individuales para algunos de los tamaños que se solicitan en forma más común.
- Podría mantenerse una tabla de n elementos, donde habría apuntadores a cada una de las listas donde hay bloques de huecos de tamaños similares.
 - Ejemplo: A los de 4 KB, a los de 8 KB, etc, etc
- La localización de un hueco es muy rápida
- Pero es complejo y lenta la actualización cuando un proceso termina o se intercambia.
- Sino hay fusión la memoria pronto se fragmentará en muchos huecos pequeños inservibles.

3. MEMORIA VIRTUAL

- Al aparecer programas más grandes que la memoria, se optó por dividir el programa en fragmentos, llamados superposiciones (overlays)
 - La superposición 0 inicia la ejecución, termina y después se carga la 1, y así sucesivamente.
 - En ocasiones se podía tener más de una superposición en memoria.
 - El partir el programa se realizaba en forma manual.

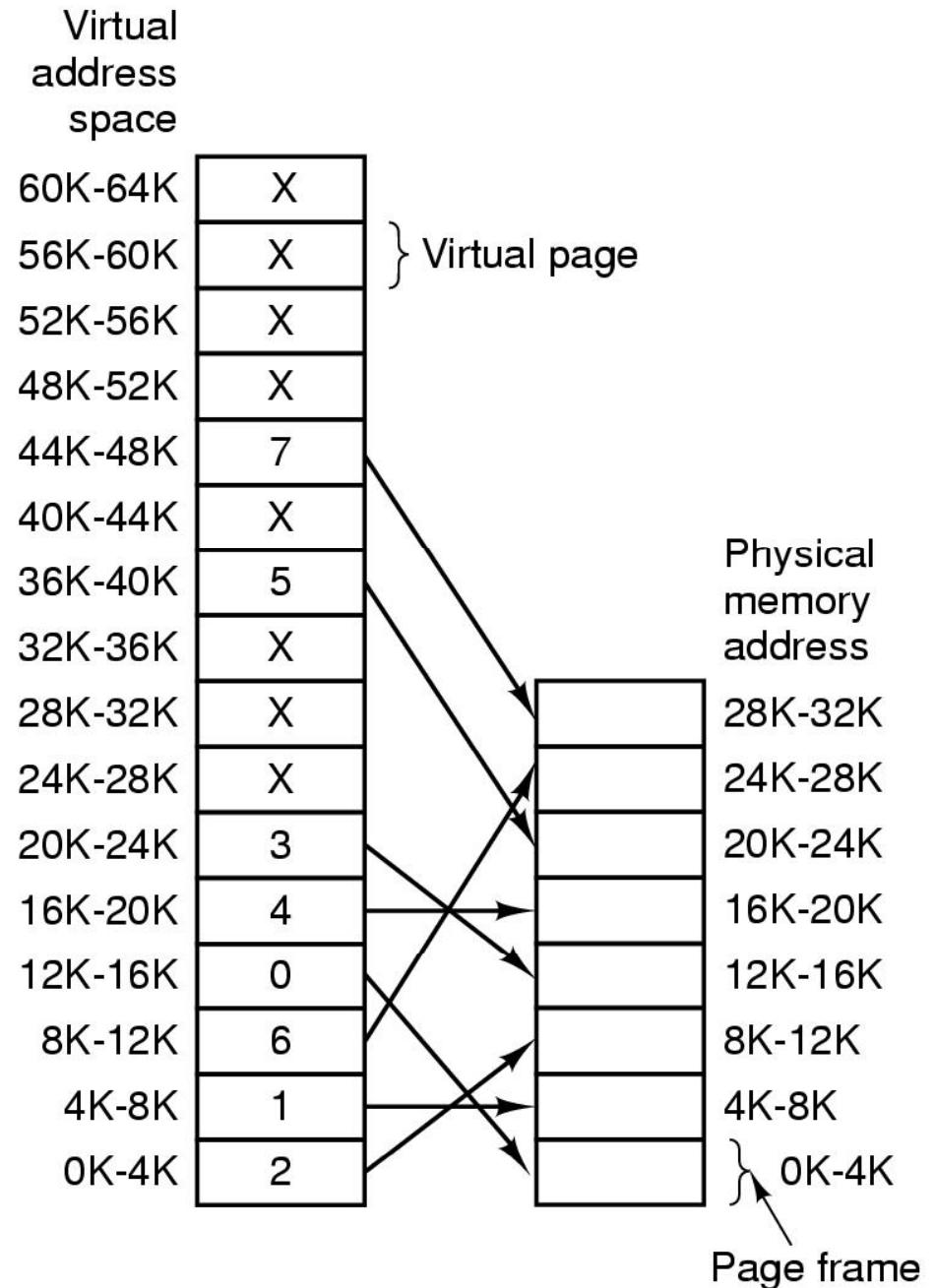
- La solución fue la memoria virtual.
 - El programa, sus datos y su pila pueden ser mayores que la memoria.
 - Se carga en memoria, lo que esté en uso en ese momento, y el resto a disco.
 - Intercambios constantes de fragmentos entre memoria y disco.
 - En multiprogramación:
 - Puede haber múltiples fragmentos de múltiples programas a la vez.

3.1. Paginación



- La CPU envía direcciones virtuales a la MMU
- La MMU envía direcciones físicas a la memoria.
- Las direcciones enviadas por el programa son direcciones virtuales, y constituyen el espacio de direcciones virtuales.
- Las direcciones virtuales no se envían directamente al bus de memoria sino a la MMU.
- La MMU establece una correspondencia entre las direcciones virtuales y las físicas.

- Página virtual vs marco de página
- Puede generar direcciones de 16 bis (de 0 a 64K)
- Físicamente tiene 32K de memoria física.
- Puede ejecutar programas de hasta 64 K
- Debe estar en disco el programa completo para subir parte de él cuando se requiera.
- Páginas y marcos de página deben ser del mismo tamaño.
- Páginas = Marco de pagina = 4K
- Hay 16 Páginas virtuales y 8 marcos de página.
- Las X indican que esas páginas no están presentes en memoria
- En hardware un bit de presente/ausente lo indica



- **Ejemplo: MOV REG, 0**

- Se envía la dirección virtual 0 a la MMU
- La MMU verifica que la 0 (0 a 4095) esta en el marco de página 2 (8192 a 12287).
- Así que la MMU coloca en el bus de dirección la dirección 8192.
- Así que MOV REG, 0 → 8192

- **Ejemplo: MOV REG, 8192**

- Se transforma a MOV REG, 24576

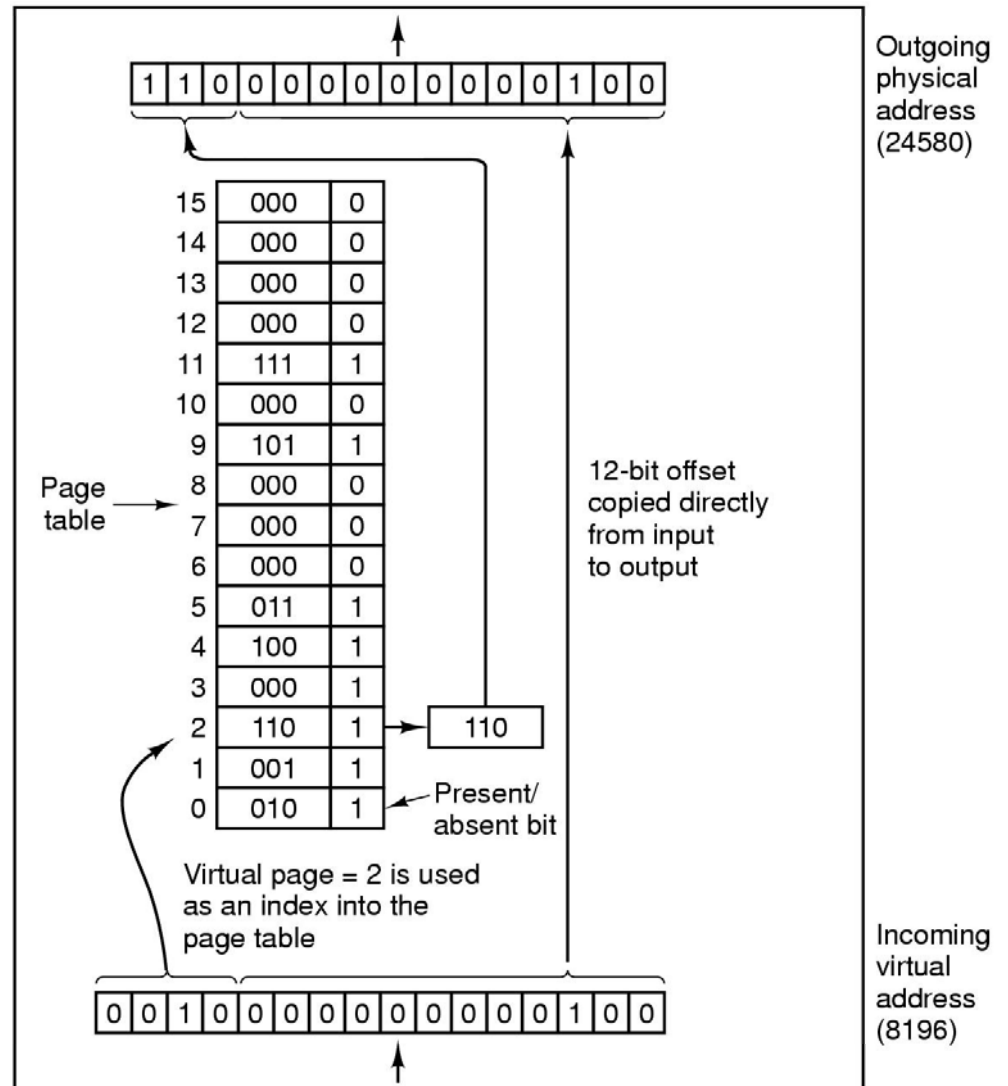
- **Ejemplo: MOV REG, 20500**

- Esta 20 bytes arriba del inicio de la pagina virtual 5
- Se transforma a MOV REG, 12308

● Ejemplo: MOV REG, 32780

- Una dirección 12 bytes arriba del inicio de la pagina virtual 8.
- La MMU detecta que no tiene correspondencia en memoria física.
 - La CPU salta al SO (interrupción), este salto se llama **fallo de página**.
 - El SO escoge un marco de página que no se este usando mucho y lo baja a disco.
 - En su lugar sube el marco de página requerido.
 - Modifica el mapa de direccionamiento
 - Reinicia la instrucción interrumpida.

- Ejemplo de dirección virtual: 8196 (0010000000000100)
- Se transforma empleando el mapa de la figura anterior.
- La dirección de 16 bits se divide en:
 - 4 → Núm de página
 - 12 → Desplazamiento
- Podríamos tener 16 páginas
- Podríamos direccionar 4096 bits de cada página.
- El num de página se usa como índice para consultar la *tabla de páginas*.
- Si bit presente/ausente es 1:
 - El marco de página hallado en la tabla de páginas se copia en los tres primeros bits del registro de salida junto con su desplazamiento (12 bits)
 - Los 15 bits constituyen la dirección física
 - Esto se coloca en el bus de memoria.



3.2. Tablas de páginas

- Problemas a resolver:
 1. Las tablas de páginas deben ser extremadamente grandes
 2. La transformación de direcciones debe ser rápida.

1. **Las tablas de páginas deben ser extremadamente grandes**

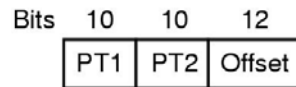
- Las computadoras modernas suelen tener 32 bits (4 GB) o más, con páginas de 4K, tendría millones de páginas (1048576).
- Una de 64 bits: 4611686018427387904 páginas

2. **La transformación de direcciones debe ser rápida**

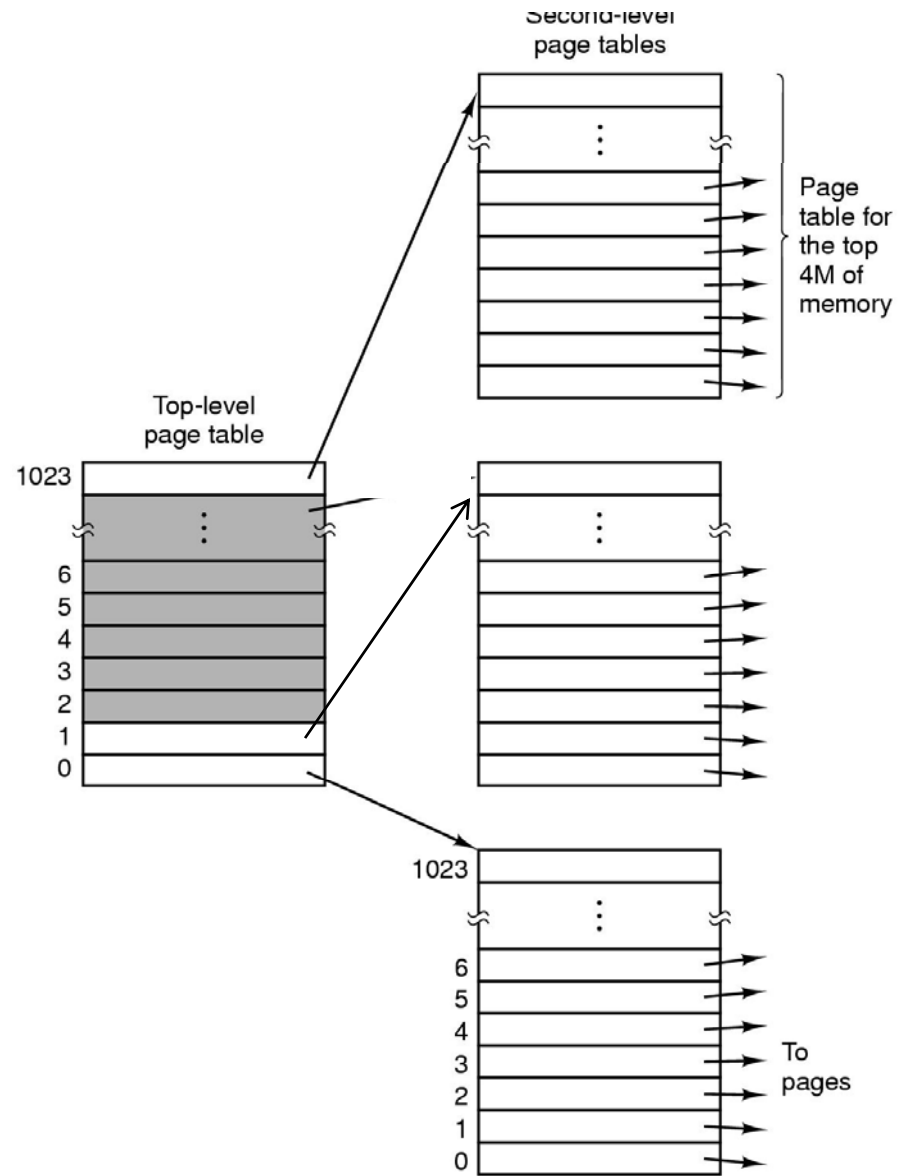
- Esta en función del punto anterior, aunado a que cada vez que se haga referencia a la memoria se requiere hacer la transformación.
- Por una instrucción típica se requiere en ocasiones hasta 4 accesos a memoria.
- La idea es no generar cuellos de botella.

3.2.1. Tablas de páginas multinivel

- Soluciona el problema de almacenar enormes tablas de páginas en la memoria todo el tiempo.
- Se usan tablas de páginas multinivel
- En el inciso a) de la siguiente figura:
 - Tenemos una dirección virtual de 32 bits.
 - Se divide en tres campos:
 - PT1 10 bits
 - PT2 10 bits
 - Despl 12 bits
- Los desplazamientos son de 12 bits (indica que las páginas son de 4K ($2^{12}/1024$))
- Hay un total de 2^{20} páginas (1,048,576)



(a)



(b)

- a) Dirección de 32 bits con dos campos de tabla de páginas
- b) Tablas de páginas de dos niveles.

- No requiere que las tablas de páginas estén todo el tiempo en memoria
- Ejemplo:
 - Un proceso requiere 12 MB
 - 4MB de la memoria baja para el texto
 - Los siguientes 4MB para los datos
 - Los 4MB superiores de la memoria para la pila
 - Entre los datos y la pila hay un hueco enorme que no se usa.

- En la figura anterior inciso b):
 - A la izquierda tenemos la tabla de paginas de primer nivel con 1024 entradas (campo PT1)
- Cuando se le presenta a la MMU una dirección virtual:
 1. Extrae el campo PT1
 2. Usa el valor anterior para consultar la tabla de páginas de 1er nivel (1024 entradas de 4 KB cada una = 4MB)
 3. El valor anterior contiene la dirección del número de marco de página donde esta almacenada una tabla de páginas de 2do nivel.

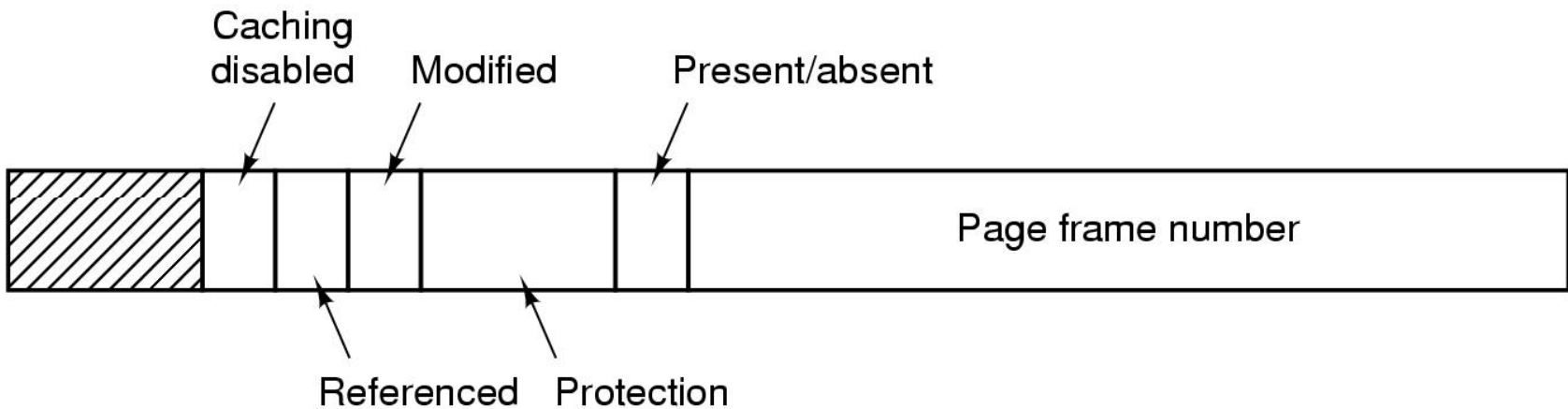
- Para el ejemplo anterior:
 - La entrada 0 del primer nivel contiene:
 - Tabla de páginas del texto del programa.
 - La entrada 1:
 - Tabla de páginas para los datos.
 - La entrada 1023:
 - Tabla de páginas para la pila.
- Las entradas sombreadas no se usan.
- PT1 nos lleva a la dirección donde quedaron los segmentos del programa (primer nivel, al inicio del segmento)
- La tabla del primer nivel nos lleva al inicio de cada segmento (código, datos, pila) del programa

- PT2 tiene el índice a la tabla de páginas de 2do nivel seleccionada (la dirección exacta en el segmento).
- Despl Tiene el desplazamiento que existe a partir del inicio de la página virtual correspondiente.
- El espacio de direcciones contiene más de un millón de páginas
 - Bajo este esquema solo se necesitan 4 tablas de páginas.
 - 1er nivel
 - Una
 - 2do nivel
 - 0 – 4MB, 4 – 8 MB, 4MB superiores

- También puede haber fallos de página.
- Se podrán manejar tablas de páginas de más de dos niveles
- Se estarían cargando/descargando constantemente los marcos de página a manipular.

3.2.2. Estructura de una entrada de tabla de página.

- El tamaño y estructura depende de la máquina.



- a) Numero de marco de página:
 - Es su identificador
- b) Bit de presente/ausente
 - Esta o no en memoria (fallo de página)
- c) Protección
 - Tres bits, para ver si se puede leer, escribir o ejecutar.
- d) Bit de Modificada
 - Encendido cuando se ha escrito algo en la página. Y poder actualizarse en disco si ha sido modificada, de lo contrario se desecha.
- e) Bit de Solicitada
 - Se enciende cuando se hace referencia a dicha página.
- f) No poner en caché
 - Uso en registros de dispositivos. Para no usar copias antiguas de algún comando.

3.3. Búferes de consulta para traducción

- Consideremos una instrucción que copia un registro en otro:
 - Sin paginación:
 - Sólo se efectúa una referencia a memoria.
 - Con paginación:
 - Se requieren referencias adicionales (para acceso a las tablas de paginación)
 - Reduce el desempeño del CPU.

- Normalmente se hacen muchas referencias a pocas páginas (y no al contrario)
- En base a esto se optó por:
 - Colocar un dispositivo hardware en las computadoras
 - Que almacene y traduzca direcciones virtuales a físicas sin pasar por la tabla de páginas.
 - Se llama Buffer de Consulta para Traducción (TLB) o memoria asociativa.
 - Normalmente esta dentro de la MMU y no pasa de 64 entradas.



- Ejemplo:

- Proceso cargado en las páginas virtuales 19, 20 y 21.
- Tiene códigos de protección para leer y ejecutar.
 - Datos en uso en páginas 129 y 130
 - Índices de cálculos en la página 140
 - La pila esta en las páginas 860 y 861

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

TLB para acelerar la Paginación

- **Funcionamiento:**

- Cuando se presenta una dirección virtual a la MMU.
 - Primero: El hardware verifica (en paralelo) si esta presente en la TLB.
 - Si no viola los bits de protección el marco de página se toma de la TLB sin recurrir a la tabla de páginas.
 - Si viola los bits de protección o no se encuentra en la TLB se produce un falló de página.
 - Cuando no esta la dirección virtual en la TLB
 - Se realiza una consulta ordinaria a la tabla de páginas.
 - Desaloja una entrada en la TLB y coloca la nueva dirección en ella, para futuros accesos.

● MANEJO DEL TLB POR SOFTWARE

- En máquinas SPARC, MIPS, Alpha, HP PA, etc
 - Administran la TLB por software
 - El SO carga explícitamente la TLB
 - Cuando no se encuentra una página en la TLB:
 - Se genera un fallo de página
 - Dejando que el SO resuelva el problema.
 - Debe hacerse con pocas instrucciones, porque los fallos TLB ocurren con más frecuencia que los de página.
 - Con esto se simplifica la MMU
 - Se dejan recursos para caché u otros recursos que mejoran el desempeño.
 - Se pueden reducir los fallos de TLB si se cargan anticipadamente las entradas (por probabilidades y predicciones)

3.4. Tabla de páginas invertidas

- Ejemplo:
 - Un espacio de direcciones de 32 bits (2^{32}), con 4096 (4K) bytes por página.
 - Requiere más de un millón (1048576) de entradas en la tabla de páginas.
 - Sólo sistemas recientes pueden manejar una tabla tan grande.
 - Ahora imaginemos una máquina de 64 bits, ocuparía una tabla de 30 millones de Gbytes. ¡No es factible!

- Las tablas de páginas anteriores requieren una entrada por cada página virtual.
- Una solución es la tabla de páginas invertidas:
 - Hay una entrada por cada marco de página en la memoria física, en lugar de una entrada por página virtual.
 - Ejemplo:
 - Direcciones virtuales de 64 bits con páginas de 4K y 256 MB de RAM.
 - Solo requiere 65536 entradas en la tabla.
 - La entrada indica que proceso y que página virtual esta en el marco de página correspondiente (proceso, marco).

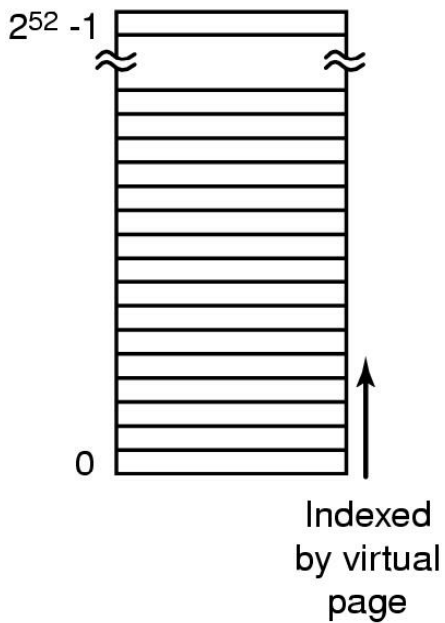


- Desventajas:

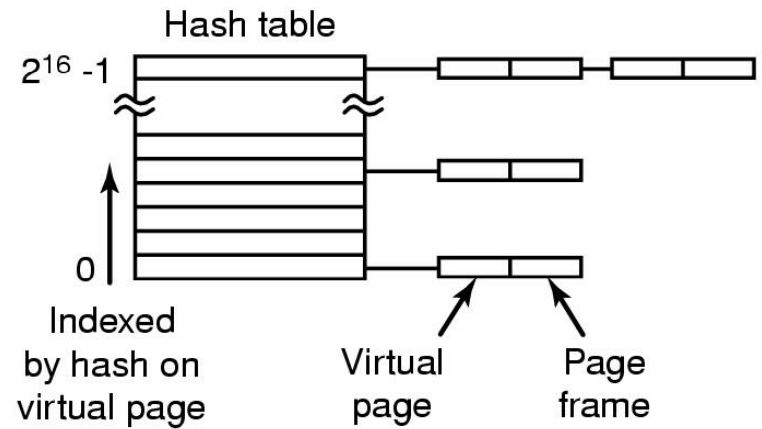
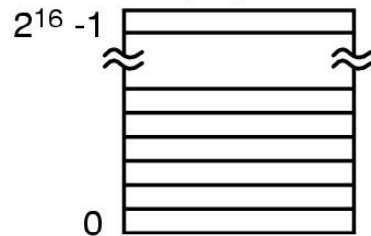
- La traducción de direcciones virtuales a físicas es mucho más difícil.
- Cuando el proceso n hace referencia a la página virtual p , el hardware no podrá utilizar p como índice para consultar la tabla de páginas.
 - Se hará una búsqueda en toda la tabla de páginas invertida, buscando la entrada (n,p)
- Esta búsqueda se hará cada vez que se haga referencia a la memoria.

- Una mejora a lo anterior sería:
 - Implementar una TLB en hardware y por software analizar los fallos de página.
 - La búsqueda podría ser a través de una tabla hash sobre la dirección virtual.
 - Las páginas virtuales en memoria, en cierto momento, con la misma llave hash estarán enlazadas.
 - Si la tabla hash tiene tantas ranuras como marcos de páginas, las cadenas tendrían una longitud promedio de una sola entrada.

Traditional page table with an entry for each of the 2^{52} pages



256-MB physical memory has 2^{16} 4-KB page frames



Comparación entre una tabla de páginas tradicional y una invertida

- Una vez hallado el número de marco, se coloca en el TLB el nuevo par (virtual, física)
- Se usan actualmente en algunos equipos IBM y HP.
- Se volverán más comunes en los equipos de 64 bits.

4. ALGORITMOS PARA REEMPLAZO DE PÁGINAS

- Al producirse un fallo de página:
 - El SO decidirá que marco de página bajará a disco, para en su lugar colocar el nuevo marco solicitado.
 - Si el marco a desalojar fue modificado, se reemplazará, de lo contrario simplemente se sobrescribirá.
 - El desempeño del sistema se mejoraría o empeoraría, dependiendo del algoritmo de reemplazo de paginas a emplear.

- Este problema también aplica en otras áreas:
 - Cache de memoria
 - Pueden o no modificarse
 - Servidor Web (proxy)
 - No se modifican

Algoritmos

1. Algoritmo óptimo de reemplazo de páginas
2. Algoritmo de reemplazo de páginas no usadas recientemente (NRU)
3. Algoritmo de reemplazo de páginas FIFO
4. Algoritmo de reemplazo de páginas de segunda oportunidad
5. Algoritmo de reemplazo de páginas tipo reloj
6. Algoritmo de reemplazo de páginas menos usadas recientemente (LRU)
7. Algoritmo de reemplazo de páginas no usadas frecuentemente (NFU)
8. Algoritmo del envejecimiento
9. Algoritmo de reemplazo de páginas de conjunto de trabajo
10. Algoritmo de reemplazo de páginas WSClock

1. Algoritmo óptimo de reemplazo de páginas

- Es el mejor, fácil de describir, pero difícil de implementar
- Las páginas se rotulan con el número de instrucciones que se ejecutarán antes de que se haga la primera referencia a ella.
- Al presentarse un fallo de página:
 - De las páginas que están en memoria, se reemplaza la que más instrucciones falten para referenciarla.
 - La idea es aplazar lo más que se pueda los fallos de página.
 - Es decir, se desaloja la página con el rótulo mayor.

- Este algoritmo no es factible ponerlo en practica (No se usa).
 - Al producirse un fallo de página, no hay forma de saber cuando se referenciará a cada página.
- Sin embargo, al implementar un simulador en la primera ejecución, para llevar la cuenta de todas las referencias a páginas, sería posible implementarlo en la segunda ejecución.
 - Usando la información recabada en la primera ejecución.
- Lo anterior servirá para medir el desempeño de algoritmos reales, al compararlo con el desempeño obtenido con el simulador.
- Cabe mencionar que el simulador sólo será válido para ese programa y sus mismas entradas.

2. Algoritmo de reemplazo de páginas no usadas recientemente (NRU)

- NRU → Not Recently Used
- Para la obtención de las estadísticas de uso, las computadoras con memoria virtual asocian a cada página dos bits de estado.
 - R → Se enciende cuando se referencia la página.
 - M → Se enciende cada que se escribe en la página.
- Los bits se actualizan cada vez que se haga referencia a la memoria.
 - Es fundamental que se enciendan por hardware y se apaguen por software.

- Si el hardware no cuenta con estos bits, pueden simularse en software:
 - Al cargar un proceso, todas sus entradas se marcarán como *no presentes*.
 - Al hacer referencia a una página, se produce un fallo de página y el SO encenderá el bit R, con permisos de solo lectura. Y reiniciará la instrucción.
 - Al querer escribir, habrá otro fallo de página, entonces se encenderá el bit M, cambiando a modo de lectura/escritura.
 - Frecuentemente el bit R se establece a 0 para distinguir las páginas que no han sido solicitadas últimamente.

- Al presentarse un fallo de página, el SO examina todas las páginas y las divide en cuatro categorías, en base a R y a M:
 - Clase 0: No solicitada, No modificada.
 - Clase 1: No solicitada, Modificada.
 - Clase 2: Solicitada, No modificada.
 - Clase 3: Solicitada, Modificada.
- La clase 1, ocurre cuando una interrupción apaga R de una clase 3.
- M no se debe apagar, porque sirve para reemplazar tablas modificadas.

- Este algoritmo desalojará al azar una página de clase menor que esté vacía.
 - Es mejor desalojar una página modificada que no ha sido referenciada, en lugar de una sin cambios pero con muchas referencias.
- Este algoritmo, es fácil de entender e implementar.
 - Moderadamente eficiente, es decir, es aceptable.

3. Algoritmo de reemplazo de páginas FIFO

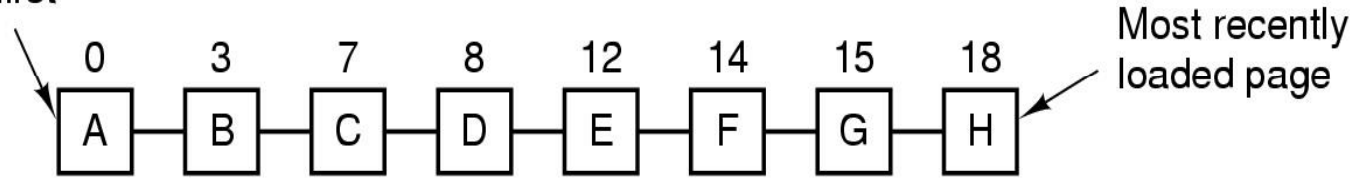
- El SO mantiene una lista de todas las páginas que actualmente están en memoria.
- Con la página más antigua al principio y la más recientemente cargada al final
- Al presentarse un fallo de página, se desaloja la página del principio de la lista y la nueva se anexa al final.
- Podría desalojar una página que se usa mucho.
- Nunca se usa FIFO, al menos en su forma pura.

4. Algoritmo de reemplazo de páginas de segunda oportunidad

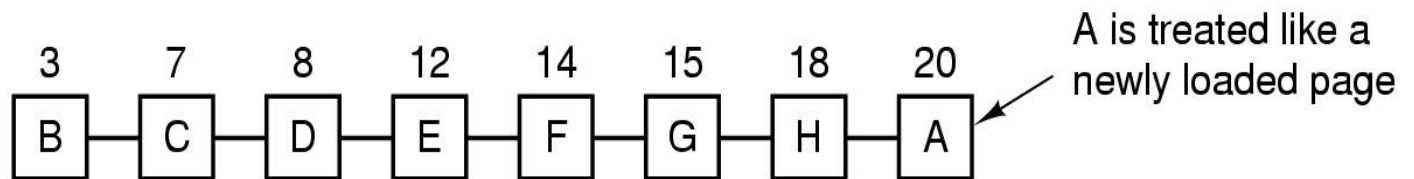
- Es una variante del FIFO
- Evita el problema de desalojos de páginas que se usan mucho.
- Funcionamiento:
 - Se examina el bit R de la página más antigua:
 - Si $R=0$, indica que, aparte de antigua no se usa mucho. Por lo tanto es reemplazada de inmediato.
 - Si $R=1$, se apaga el bit y la página es colocada al final como si acabara de llegar. Posteriormente se continúa la búsqueda.

- Este algoritmo busca una página antigua que no se use mucho.
- Si se ha hecho referencia a todas las páginas, el algoritmo se comporta como FIFO puro.
 - Tomando el principio de la lista, debido a que fue apagando(R) a todas las páginas.
- La lista analiza tiempos de llegada.
- Es ineficiente, porque constantemente cambia páginas de lugar.

Page loaded first



(a)



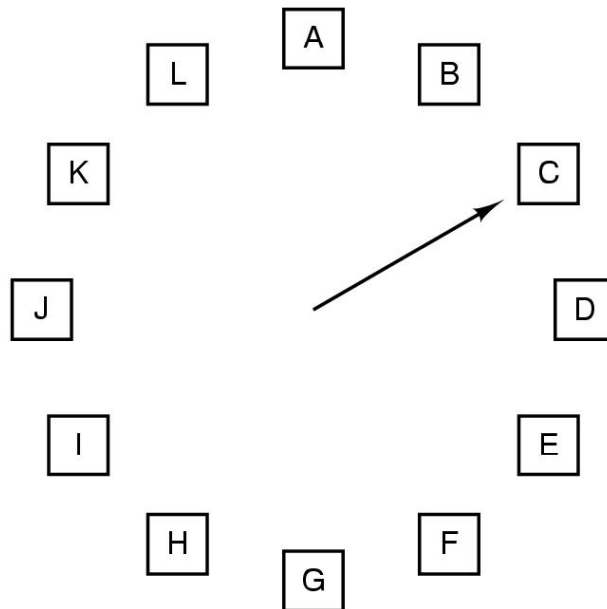
(b)

- a) Páginas en orden FIFO
b) Listas de página si hay un fallo de página en el tiempo 20 y el bit R de la página A está encendido. Los número de arriba de las páginas son sus horas de carga.

5. Algoritmo de reemplazo de páginas tipo reloj

- Esta variante mantiene las páginas en una lista circular parecida a un reloj.
 - En lugar de estar cambiando constantemente las páginas, como las versión anterior.
- Una manecilla apunta a la página más antigua
- Al presentarse un fallo de página:
 - Se examina la página a la que apunta la manecilla
 - Si $R=0$
 - Se desaloja
 - La nueva se inserta en su lugar
 - La manecilla se incrementa una posición
 - Si $R=1$
 - Se cambia a 0
 - La manecilla se adelanta a la siguiente página

- Se repite este proceso hasta hallar una página con $R=0$.
- La diferencia con el anterior es su implementación



When a page fault occurs,
the page the hand is
pointing to is inspected.
The action taken depends
on the R bit:

- R = 0: Evict the page
- R = 1: Clear R and advance hand

6. Algoritmo de reemplazo de páginas menos usadas recientemente (LRU)

- LRU → Least Recently Used
- Una aproximación al algoritmo óptimo se basa en que:
 - Es probable que las páginas que se han usado mucho en las últimas instrucciones se usarán mucho nuevamente
 - Sin embargo, las que no se han usado en cierto tiempo, seguirán sin usarse un tiempo más.
- La idea es desalojar la que tenga más tiempo sin usarse.

- Es factible, pero con un costo elevado.
 - Requiere mantener una lista enlazada de todas las páginas
 - Primero, las usadas más recientemente
 - Al final, las menos usadas recientemente
- El costo es debido a que la lista debe actualizarse cada vez que se hace referencia a la memoria.
 - Encontrar una página en la lista, borrarla y reinsertarla al frente, aún en hardware sería tardado.

LRU EN HARDWARE

- Primera opción:
 - Equipar al hardware con un contador (de 64 bits), C.
 - C se incrementará en forma automática, después de cada instrucción.
 - Cada entrada de las páginas debe tener un campo para vaciar el valor de C.
 - Después de cada referencia a la memoria, el valor actual de C se vacía a la entrada de la página que se está referenciando.
 - Al ocurrir un fallo de página, el SO examina todos los campos de la tabla de páginas, hasta encontrar el valor más pequeño, esta sería la página menos recientemente usada.

- Segunda opción:

- En una máquina con n marcos de página
- El hardware LRU mantendrá una matriz de $n \times n$ bits. Iniciándose con ceros.
- Cada vez que se referencia al marco de página k , el hardware enciende primero todos los bits de la fila k , y luego apaga todos los bits de la columna k .
- En cualquier instante la fila cuyo valor binario sea el más bajo, será la del marco menos recientemente usado.

	Page			
	0	1	2	3
0	0	1	1	1
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

(a)

	Page			
	0	1	2	3
0	0	0	1	1
1	1	0	1	1
2	0	0	0	0
3	0	0	0	0

(b)

	Page			
	0	1	2	3
0	0	0	0	1
1	1	0	0	1
2	1	1	0	1
3	0	0	0	0

(c)

	Page			
	0	1	2	3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0

(d)

	Page			
	0	1	2	3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	1
3	1	1	0	0

(e)

0	0	0	0
1	0	1	1
1	0	0	1
1	0	0	0

(f)

0	1	1	1
0	0	1	1
0	0	0	1
0	0	0	0

(g)

0	1	1	0
0	0	1	0
0	0	0	0
1	1	1	0

(h)

0	1	0	0
0	0	0	0
1	1	0	1
1	1	0	0

(i)

0	1	0	0
0	0	0	0
1	1	0	0
1	1	1	0

(j)

- **Ejemplo:**

- Haciendo referencia a páginas en el orden:
0,1,2,3,2,1,0,3,2,3

- Algoritmo no factible, porque no hay máquinas que lo traigan incorporado en hardware.

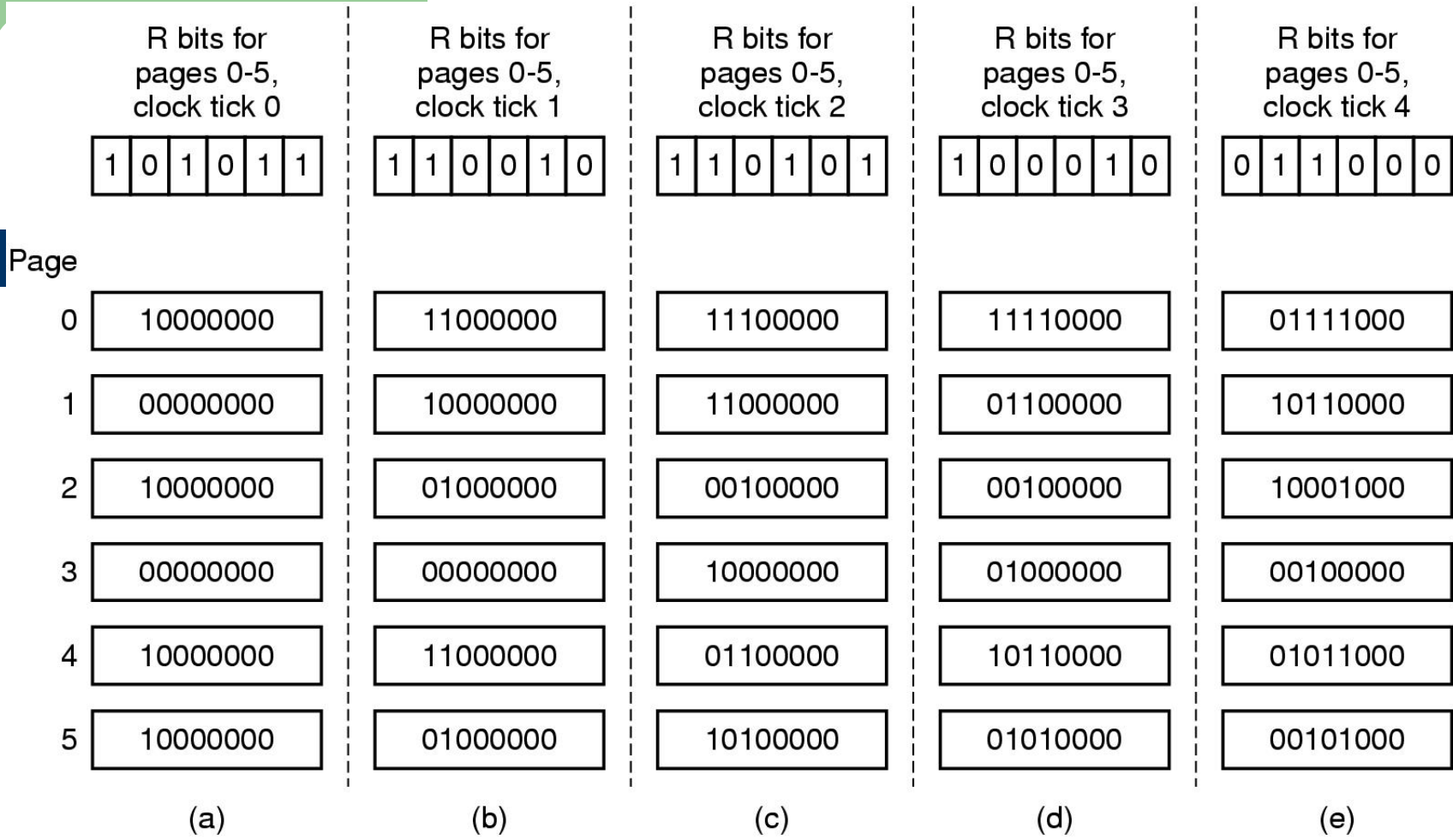
7. Algoritmo de reemplazo de páginas no usadas frecuentemente (NFU)

- NFU → Not Frequently Used
- Es un LRU(menos usada recientemente) implementado en software
- Requiere un contador de software para cada página, de valor inicial cero.
- Frecuentemente estará “limpiando” el bit R.
- En cada interrupción de reloj, el SO explora las páginas en memoria, para cada una el bit R (sea 0 o 1) se suma a su contador.

- Los contadores son un intento por llevar la cuenta de qué tanto se referencian las páginas
- Al ocurrir un fallo de página, se escoge la página con el contador más bajo.
- Inconvenientes:
 - Nunca olvida
 - En varias pasadas no funcionarían los contadores, porque se quedarían con cuentas acumuladas en las pasadas anteriores.
 - Esto haría que se pierda el objetivo de este algoritmo.

8. Algoritmo del envejecimiento

- También es un LRU(menos usada recientemente) en software
- Es el NFU (no usadas frecuentemente) modificado
- Igual que el anterior, cada página requiere un contador, y cada tick de reloj explora las páginas y suma los R bits al contador.
- Todos los contadores se desplazan un bit a la derecha, antes de sumar el bit R.
- Después el bit R se suma al bit de la extrema izquierda, en lugar de la derecha.



Se muestran seis paginas durante cinco ticks de reloj.
 Los ticks de reloj (a, b, c, d, e)

- Cuando ocurre un fallo de página, se desaloja la página cuyo contador es el más bajo.
- Es evidente que al no hacer referencia a una página, ésta tendrá n (de acuerdo a los ticks de reloj) ceros a la izquierda, por lo que será un valor pequeño.
- Cuando dos páginas tienen el mismo contador(sobre todo ceros), se toma al azar cualquiera de ellas.

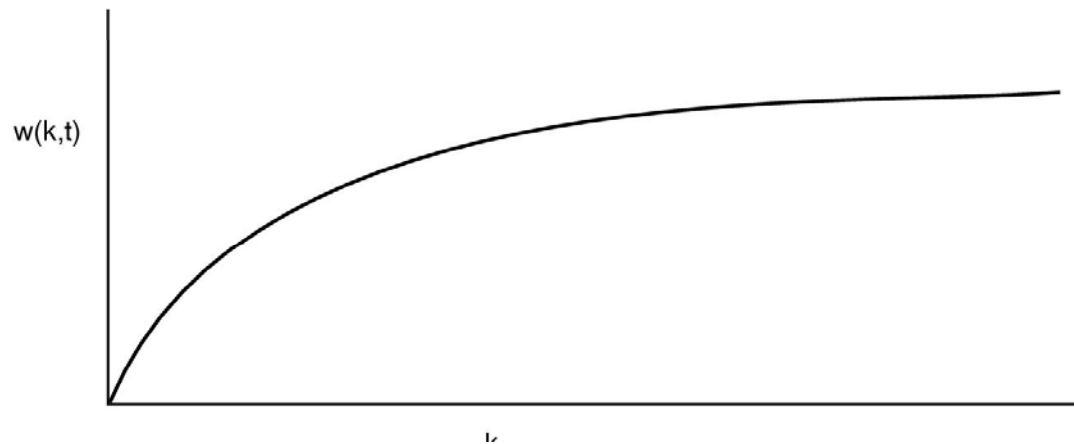
9. Algoritmo de reemplazo de páginas de conjunto de trabajo

- Es la forma más pura de paginación
- Los procesos inician sin páginas en la memoria
- Al tratar de obtener la primer instrucción, se producirá un fallo de página; y así respectivamente.
- Después de cierto tiempo, el proceso tiene casi todas sus páginas en memoria, y su ejecución se estabiliza con pocos fallos de página.
- A lo anterior se le llama **paginación por demanda**.

- Por fortuna, la mayoría de los procesos, en cualquier momento de su ejecución sólo referencia a ciertas páginas suyas, no es común que en una pasada requiera todas sus páginas.
- **Conjunto de trabajo.** Es el conjunto de páginas que en cierto momento está usando un proceso.
- Si todo el conjunto de trabajo está en memoria, no habría fallos de página, al menos en esa pasada.
- Si se tiene memoria reducida para almacenar el conjunto de trabajo, habría muchos fallos de página y se ejecutará con lentitud.

- **Programa hiperpaginado (thrashing).** Es un programa que cada pocas instrucciones causa fallos de página.
- Es común que se vacíe un proceso completo a disco para cargar a otro proceso. En este caso cuando se vuelva ejecutar, es como si empezara de nuevo.
- **Modelo de conjunto de trabajo o prepaginación.** Es cuando el SO trata de cargar anticipadamente el conjunto de trabajo, para evitar demasiados fallos de página y agilizar la ejecución de procesos.

- El conjunto de trabajo cambia con el tiempo (en cada pasada).
- La mayoría de los programas, una vez estabilizado, tienen accesos de manera aleatoria a un número reducido de páginas. Es decir tiene un comportamiento asintótico.



- Debido a dicho comportamiento, es posible hacer una conjetura respecto a qué páginas se necesitarán al reiniciar el proceso.
 - De acuerdo al momento en que se suspendió la última vez.
- Para este algoritmo es necesario que el SO sepa qué páginas tiene en memoria en cada momento, y cuales no lo están.
- El conjunto de trabajo, será el conjunto de páginas que se usaron en las k referencias más recientes.

- Se usa el tiempo de ejecución
 - Por ejemplo el conjunto de trabajo sería el conjunto de páginas empleadas durante los últimos 100 milisegundos de tiempo de ejecución, ese sería el valor para k .
- **Tiempo virtual actual.** Tiempo efectivo de CPU que ha consumido un proceso desde que inició su ejecución.
- Por lo que **se redefine el conjunto de trabajo.** Es el conjunto de páginas a las que ha hecho referencia durante los últimos t segundos de tiempo virtual.

- La idea sería hallar una página que no este en el conjunto de trabajo y desalojarla.
- Cada entrada de la página contendría:
 - Tiempo aproximado del último uso de la página, R, M, etc.
- Funcionamiento:
 - El hardware se encarga de encender los bits R y M.
 - Una interrupción de reloj periódica, causa la ejecución de software que apaga en cada tick de reloj el bit R.
 - En cada fallo de página, la tabla de páginas se explora en busca de una apropiada para desalojar.



– Al buscarla:

- Se examina R , si $R=1$.

- Se escribe el tiempo virtual en el campo “tiempo del último uso” en la tabla de páginas. Para indicar que la página se estaba usando cuando ocurrió el fallo.
- Como se acaba de solicitar, es obvio que pertenece al conjunto de trabajo y no es candidata al desalojo.

- Si $R=0$

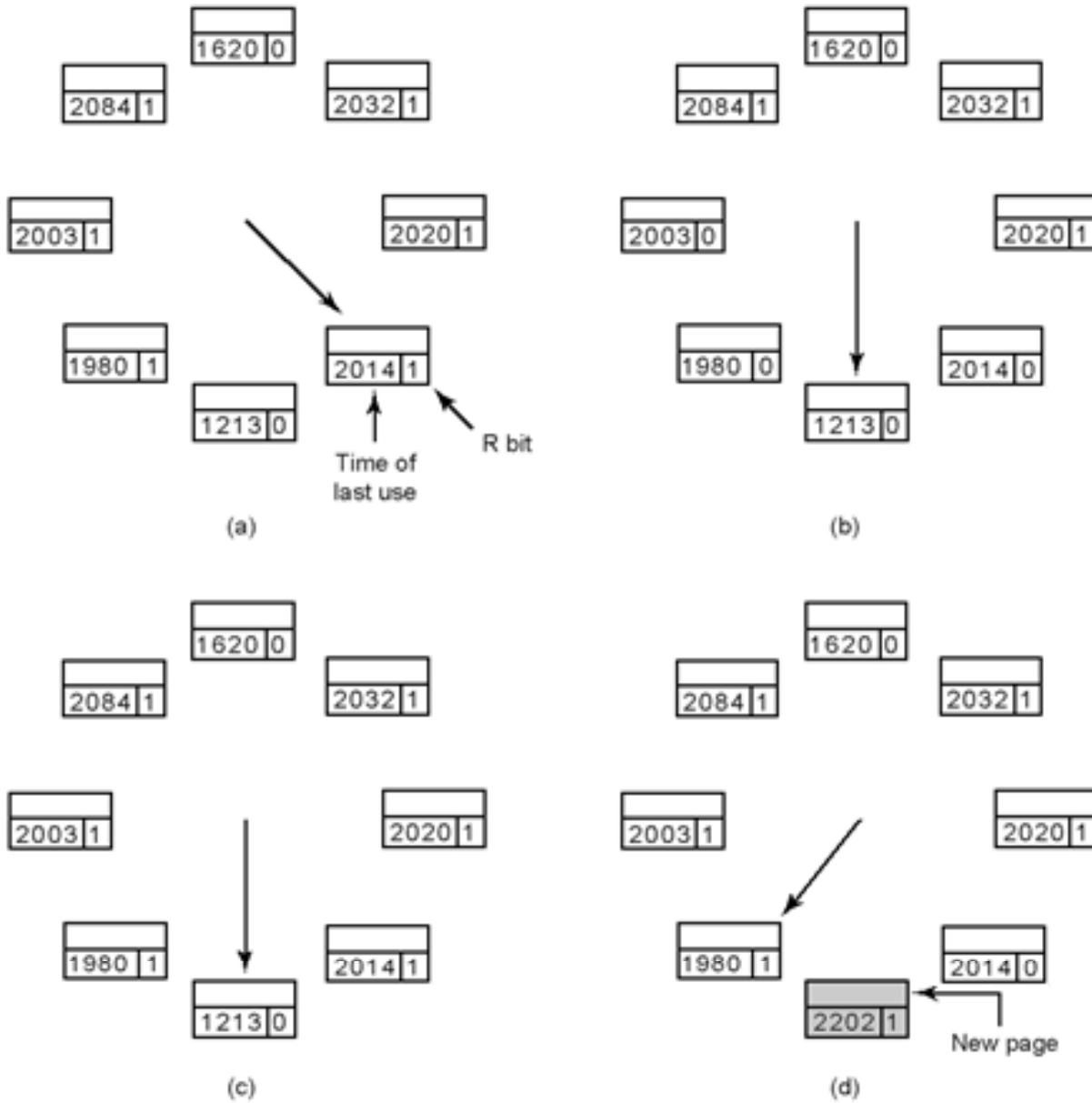
- Indica que no se ha hecho referencia a ella en este tick y puede ser candidata.
- Para verificarlo: Se calcula su edad y se compara con t .
Edad = tiempo virtual actual (proceso) – tiempo de último uso (marco)
- Si edad $> t$, ya no es parte del conjunto y se desaloja.
- Si edad $\leq t$, todavía pertenece al conjunto de trabajo y se le perdona la vida de momento, pero se toma nota de la que tenga mayor edad.
- Si hay varias con $R=0$, se desaloja la de mayor edad.

- Si todas son $R=1$, todas se han solicitado en el tick actual, y se desalojará una al azar.
- Este algoritmo es engorroso
 - Porque cada vez que hay un fallo de página, se debe explorar toda la tabla de páginas para encontrar una candidata.

10. Algoritmo de reemplazo de páginas WSClock

- Se basa en los algoritmos anteriores de reloj y de conjunto de trabajo.
- Algoritmo usado ampliamente en la práctica.
 - Por su sencillez de implementación y buen desempeño.
- Se emplea una lista circular de marcos de página.
- Al inicio la lista esta vacía.
- Al cargarse la primera página ésta se añade a la lista, y así sucesivamente, formando un anillo.
- Cada entrada tiene: tiempo del último uso, R, M, etc.

2204 Current virtual time



a) y b) cuando R=1
c) y d) cuando R=0

- Cuando hay un fallo de página se examina primero la página de la manecilla:
 - Si $R=1$
 - La página se usó en el tick actual. No es candidata.
 - Se apaga el bit R .
 - Se adelanta la manecilla y se repite el algoritmo
 - Ver incisos $a \rightarrow b$

– Si $R=0$

- Si la edad $> t$ y la página ésta limpia (no modificada)
 - Indica que ya no es del conjunto de trabajo y que no es necesario respaldarla. La nueva página se colocaría en este marco (ver incisos c \rightarrow d).
- Si la edad $> t$ y la página no esta limpia
 - No se podría desalojar (a menos que se respalde)
 - Para evitar una conmutación de procesos, se calendariza la escritura en disco, y la manecilla se adelanta y el algoritmo continua su análisis.
 - ¡Podría haber una página limpia y vieja!, para usar de inmediato.

- Si la manecilla da la vuelta completa, habría dos opciones:

a) Se calendarizó al menos una escritura:

- Simplemente seguirá dando vueltas, hasta que alguna termine su escritura y quede limpia.
- La limpia se desalojará.

b) No se calendarizó ninguna escritura:

- Indicará que todas las páginas están en el conjunto de trabajo
- Se usará al azar cualquier marco limpio. En el peor de los casos, se desalojará cualquiera.
 - Guardándola previamente en disco.

Conclusión de los algoritmos:

	Algoritmo	Funcionamiento	Observaciones
1	Óptimo	Reemplaza la página que se solicitará al último. No se puede saber.	No se usa Sólo para comparar
2	NRU	Divide las páginas en 4 clases, se toma la de la clase menor	Fácil, pero burdo
3	FIFO	Conserva el orden de carga en una lista	Puede desalojar páginas en uso
4	Segunda oportunidad	Tipo FIFO. Verifica si se está usando la página antes de desalojarla.	Muy lento
5	Reloj	Tipo 2da oportunidad. Implementación más eficiente	Realista

	Algoritmo	Funcionamiento	Observaciones
6	LRU	Excelente, pero requiere hardware especial.	Difícil de implementar
7	NFU	Intento de LRU en software	No es muy bueno
8	Envejecimiento	Buena aproximación al LRU en software	Buen desempeño
9	Conjunto de trabajo	Desempeño razonable	Implementación complicada
10	WSClock	Buen desempeño e implementación eficiente	Eficiente y realista

REFERENCIA:

- Sistemas Operativos Modernos, Segunda Edición
TANENBAUM
Prentice Hall