Pep3 - Memoria

Gestión de memoria

Requerimientos de un Sistema de Administración de Memoria

1. Re-ubicación:

- Capacidad del proceso para moverse entre memoria física.
- El programador no sabe dónde será ubicado su programa en RAM cuando este se carga y se ejecuta.
- Durante la ejecución el proceso puede ser swapeado a disco y vuelto a ser cargado en una ubicación distinta de memoria.
- Una vez que un programa se ha llevado al disco, sería bastante limitante tener que colocarlo en la misma región de memoria principal donde se hallaba anteriormente, cuando éste se trae de nuevo a la memoria. Por el contrario, podría ser necesario **reubicar** el proceso a un área de memoria diferente.

2. Protección:

- Los programas de otros procesos no deben ser capaces de referenciar sin permiso posiciones de memoria de un proceso, tanto en lectura como escritura. (Excepto en memoria compartida como hebras)
- Referencias a memoria generadas por un proceso deben comprobarse en tiempo de ejecución por el hardware. SO no puede anticipar todas las referencias que un programa hará y si pudiese sería demasiado costoso.
- Hardware provee los mecanismos de protección ¿Cómo?

3. Compartición:

- Se debe permitir que los procesos compartan áreas de memoria
- Es mejor que compartan memoria a que cada proceso tenga una copia particular de un mismo dato
- sincronización: El sistema de gestión de memoria debe permitir el acceso controlado a áreas de memoria compartidas sin comprometer la protección esencial.

Preguntas para final de la clase
¿Se podría decir que: La memoria virtual permite la traducción de las direcciones de memoria para poder almacenar en memoria principal o secundaria?

4. Organización Local/Lógica

- Los programas son organizados como módulos. por ejemplo: módulo de datos. módulo de código.
- Los módulos se pueden compilar independientemente uno de otros
- Se puede asignar distintos grados de protección como read-only o executy-only
- De manera lógica se puede dividir la memoria con técnicas de paginación y segmentación

5. Organización Física

- Memoria se organiza en niveles (Mayor jerarquía, acceso más rápido pero menor memoria)
- Caché > Ram > Disco
- Es tarea del SO mover instrucciones y datos de los procesos de un nivel a otro

Esquemas de particionamiento de memoria física RAM

- Gestión de memoria -> Objetivo: Traer procesos a memoria principal para ser ejecutados
- Esto implica uso de técnicas de **Particionamiento Fijo** y **Dinámico** de la memoria física lo que a su vez... (Descontinuadas hoy en día)

1. Particionamiento Fijo

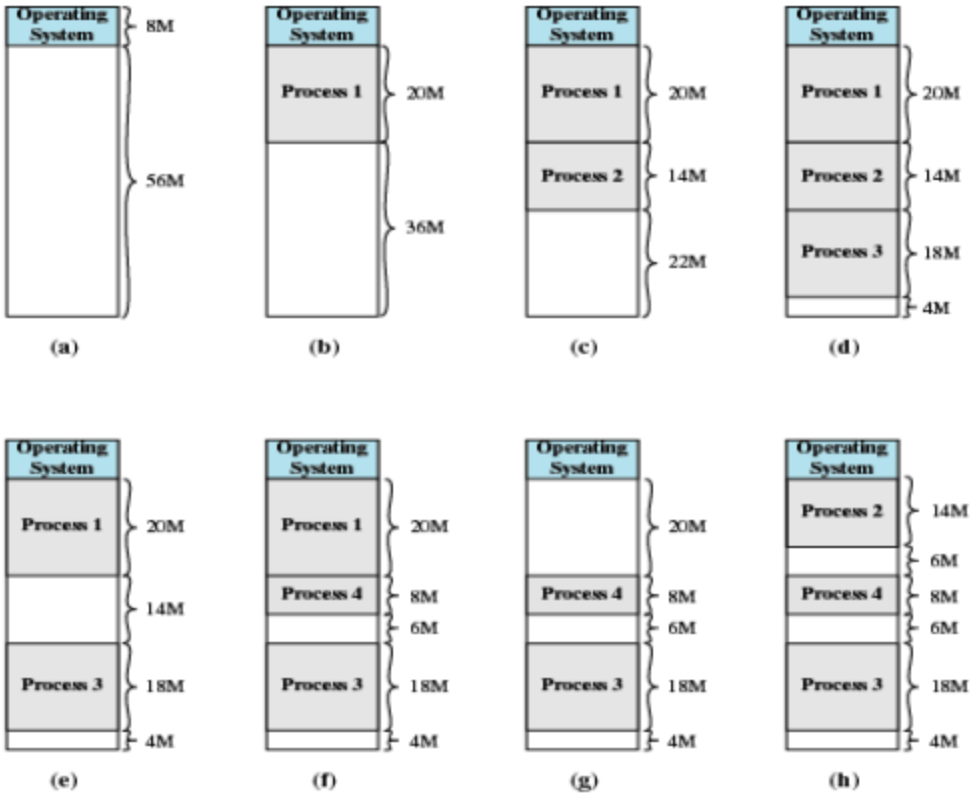
- La memoria se divide en particiones del **mismo tamaño** (o de distinto tamaño igualmente).
- Si el *tamaño_proceso* < *tamaño_partición* y existe una partición libre entonces el proceso puede ser cargado en memoria física.
- Si todas las particiones están ocupadas, el SO podría swapear a disco un proceso.
- Problemas:
  - ¿Qué pasa si el proceso no cabe?
    - Overlays
    - Particiones de distinto tamaño pero fijas
  - **Fragmentación Interna:** Espacio dentro de la partición no usada por el proceso (se busca minimizarla)
- Particionamiento fijo de la MF sin MV no es usado en sistemas modernos

Ventajas/Desventajas:

- Simple de implementar
- Número de particiones es fijo y determinado en tiempo de booteo lo que limita la multiprogramación
- Procesos pequeños harán uso ineficiente del espacio
- Podría existir un proceso > a la partición más grande

2. Particionamiento dinámico

- Si existe suficiente memoria se crea una partición del mismo tamaño del proceso que se quiere cargar
- Procesos van ocupando solo el espacio que necesitan, pero se va generando **fragmentación externa**
- Se asume que se carga el proceso completo en memoria



- Notamos que eventualmente la memoria se llena de espacios pequeños o fragmentación externa
- Se podría usar compactación para reparar esto pero es costoso y lento

Algoritmos de Posicionamiento en Particionamiento dinámico

1. Best-fit:

- Usa el bloque de memoria (partición) más pequeño entre los suficientemente grandes para alojar el proceso
- Peor rendimiento, genera alta fragmentación externa

2. Worst-fit:

- Usa el bloque de memoria (partición) más grande entre los suficientemente grandes para alojar el proceso

3. First-fit:

- Busca desde el comienzo la primera partición que pueda alojar el proceso.
- El más veloz, generalmente produce los mejores resultados

4. Next-fit:

- Busca desde el *último espacio alocado* la primera partición que pueda alojar el proceso.
- Rendimiento marginalmente peor que First Fit

Método Buddy

- Inicialmente toda la memoria se considera de tamaño 2^U
- Genera fragmentación interna

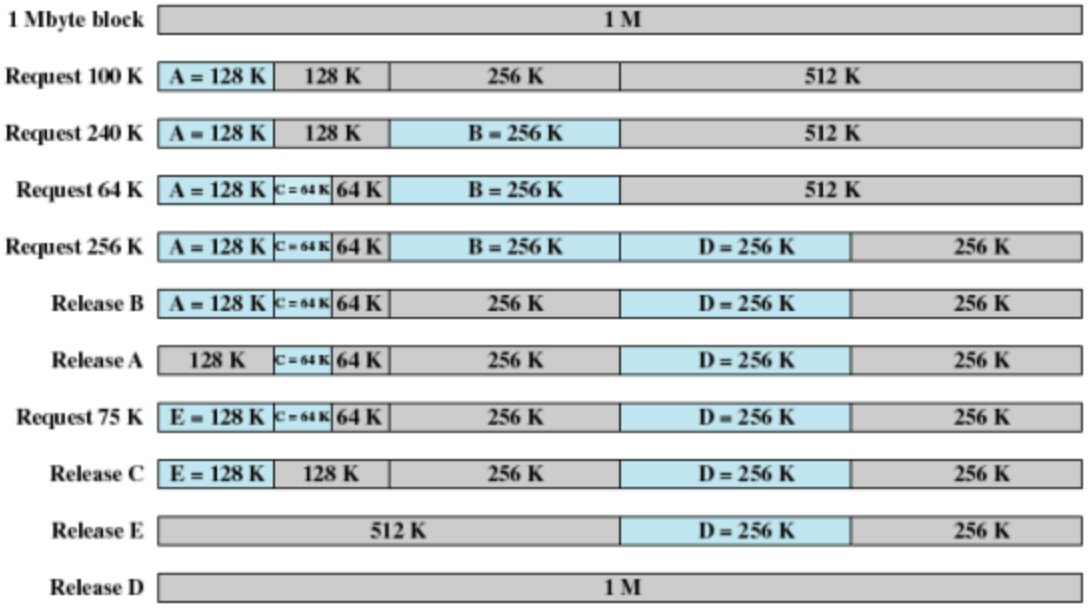


Figure 7.6 Example of Buddy System

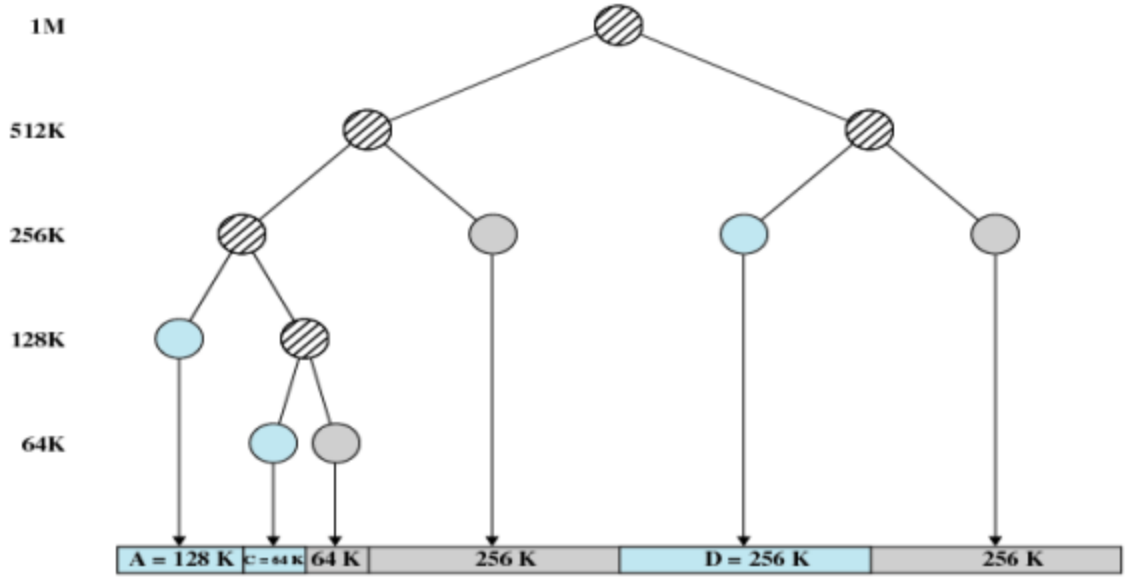


Figure 7.7 Tree Representation of Buddy System

Tipos de direcciones

cuando nos referimos a memoria virtual no hablamos de un espacio de almacenamiento, más bien de un sistema de redireccionamiento.

1. Lógica/Virtual

- Referencia a una dirección de memoria independiente de la ubicación real en memoria física
- Se debe traducir en dirección física

2. Física

- Dirección absoluta en memoria principal

3. Relativa

- Tipo especial de dirección lógica, la dirección se expresa como una Ubicación relativa a algún punto conocido

Paginación

- Se parece a particionamiento fijo
- La memoria física se divide en bloques de igual tamaño llamados **marcos**

- En memoria virtual se divide los procesos en **páginas**
- Se busca asignar páginas a marcos
- El SO mantiene una tabla de páginas para cada proceso, que mapea desde pagina virtual a número de marco en memoria principal
- Permite que el proceso no necesariamente se tenga que cargar completo y permite que se pueda cargar discontinuamente
- Tabla de páginas del proceso D guarda el número de marco en el que quedó
    - bit de control permite saber si la página está cargada o no en un marco de mem principal

Direccionamiento con paginación

Logical Addres

$$\text{Logical Addres} = \ \#\text{Page} + \text{Offset}$$

- Direcciones de 16 bits
- Tamaño de página $1Kb = 2^{10}\ \text{Bytes} = 1024\ \text{Bytes}$
- Por tanto, offset = 10 bits y los 6 restantes son para número de página
- Cantidad de páginas = 2^6 = 64 paginas de 1Kb cada una
- Offset está dado por tamaño de página
- Bits restantes lo que sobre
- Cantidad de pagina

$$\text{Tamaño de página} = 2^{\text{Offset}}$$

$$\text{Bits de Offset} = \log_2(\text{tamaño de página})$$

$$\text{Bits de \#Página} = \text{Total Bits} - \text{Bits de offset}$$

$$\text{Cantidad de Páginas} = 2^{\text{Bits de \#Página}}$$

$$\text{Cantidad de marcos} = \frac{\text{Tamaño memoria física}}{\text{Tamaño de Página}}$$

- Mi dirección lógica se compone por el número de página y offset. El número de página hace referencia a la tabla de paginas del proceso la cual contendrá el número de marco.
- Al juntar este número de marco con el offset se obtiene la dirección física

Segmentación

- Particiona la memoria en segmentos lógicos desde le punto de vista del programador
- Segmentos son de largo variable y sufren de fragmentación externa
- Generalmente existe un tamaño máximo de segmento
- tabla de segmentación: guarda direcciones físicas donde comienza el segmento

Direccionamiento con segmentación

- Dada mi dirección lógica

$$\text{Logical Addres} = \ \#\text{Segmento} + \text{Offset}$$

$$\text{Tamaño de segmento max} = 2^{\text{Offset}}$$

$$\text{Dirección Física} = \text{Offset} + \text{Direccion base física}$$

- Tabla de segmento: Guarda longitud del segmento y dirección base física del segmento
- Mediante el Número Segmento se va a la tabla de segmentos del proceso para encontrar la dirección base física del segmento
- Se compara el offset con la longitud de segmento y si $\text{Offset} \geq \text{Longitud del segmento}$ entonces la dirección no es válida.
- Si la dirección es válida entonces se obtiene la dirección física

Carga y enlazamiento de programa

Memoria Virtual

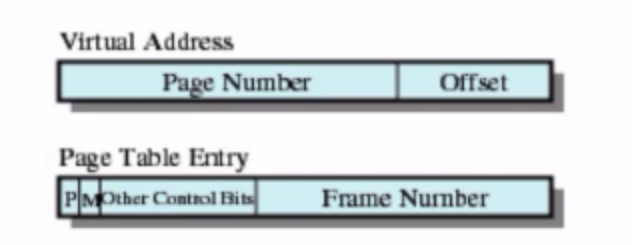
Principio de Localidad

- Cuando un proceso accede a una cierta dirección de memoria es muy probable que la próxima referencia sea la siguiente dirección

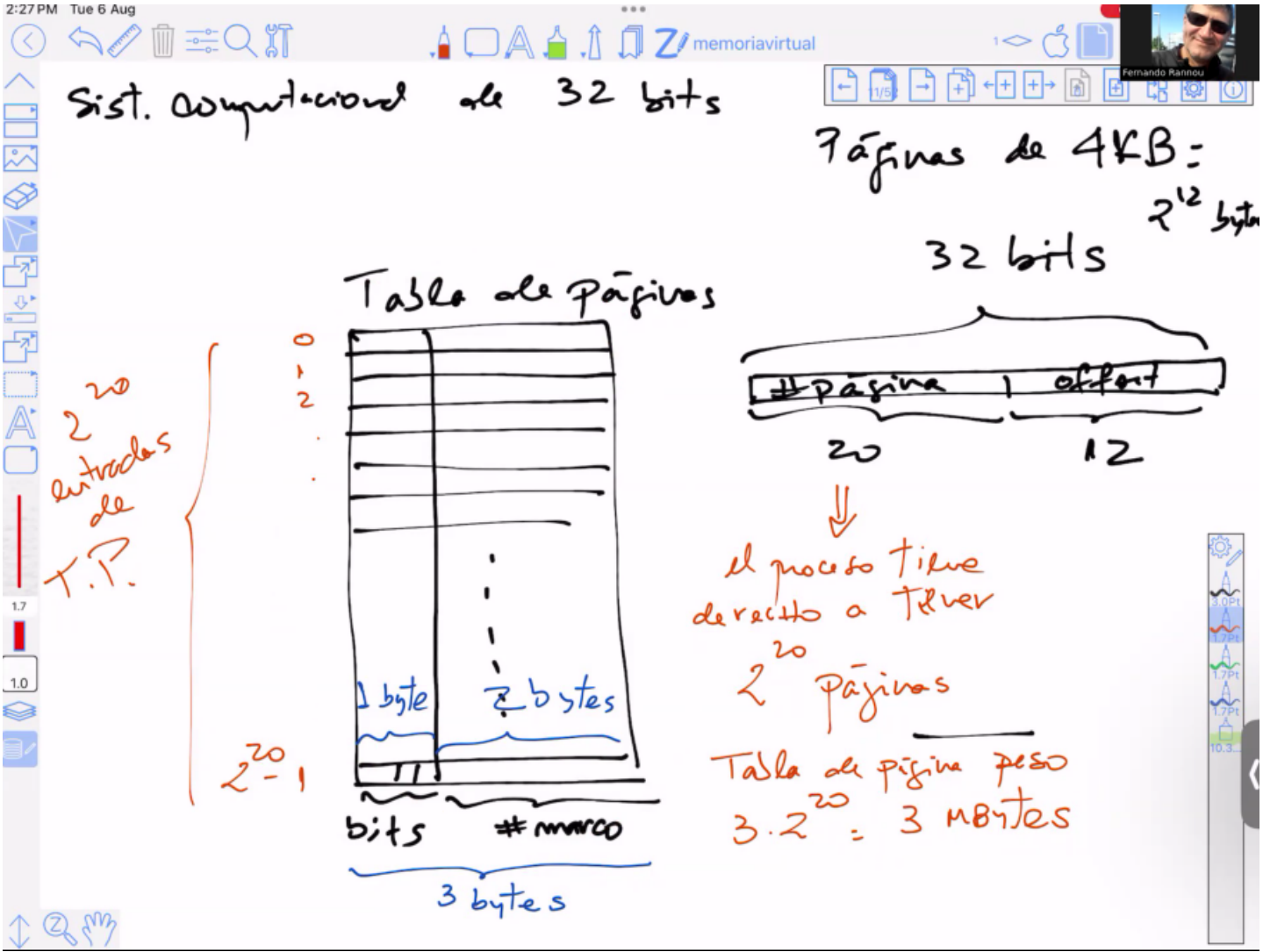
MV con Paginación

Entrada de Tabla de Página:

- Se usa un bit de presencia P para indicar si la página está en memoria ppal o no
- Se usa un bit de modificación M para indicar si la página he sido modificada o no desde la última vez que se trajo a memoria ppal
- Si bit M no ha sido seteado entonces no es necesario escribir la página a disco
- $\text{P} + \text{M} + \text{Other control bits} + \#\text{Marco}$
- Tiene tamaño variable y depende del tamaño del proceso
- Se guarda la tabla de páginas en memoria principal
- Se puede tener más páginas en un proceso que el total de marcos ya que el proceso puede ser más grande que toda la memoria principal



$$\text{Peso Tabla de Página} = \text{Cantidad de Bytes por Entrada} \cdot \text{Cantidad de Entradas}$$

- Esto sería lo que pesa en RAM la tabla de página de UN solo proceso
- Recordar que la unidad mínima de memoria es el byte, por lo tanto no puedo direccionar cosas como 1 bit.
  - $$\text{Cantidad de marcos que usa la TP} = \frac{\text{Peso tabla de página}}{\text{Peso de Página}}$$
- 

- ¿En cuántos marcos cabe esta tabla de página?

Esquema de tabla de dos (o más) niveles

- Solo tendré en memoria principal aquellas tablas de página que esté utilizando
- **Tabla raíz** que me diga donde está cargada la tabla 0,1,2,etc... de la tabla de páginas

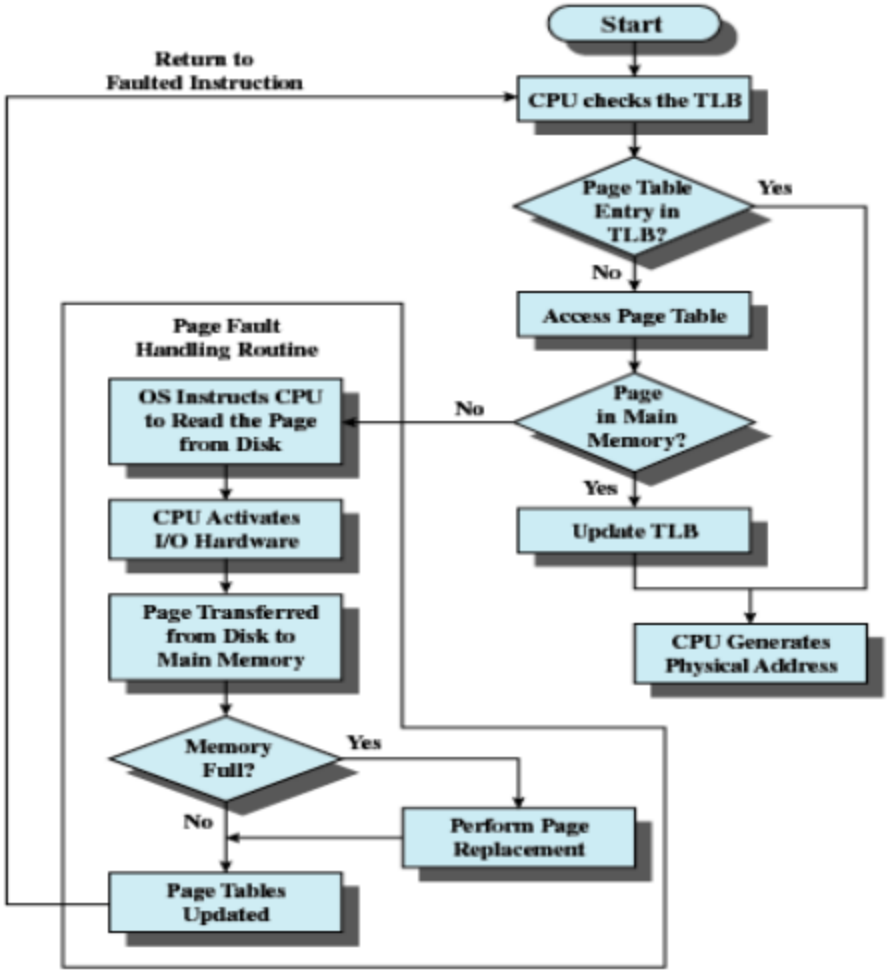
- Entrada de tabla raíz se compone por $\mathrm{P} + \mathrm{M} + \#\mathrm{Marco}$ donde esté cargada la página X de la TP
- Tabla raíz siempre está en memoria principal, esto resuelve el problema del espacio pero aumenta mucho el tiempo de acceso a la memoria, cada acceso a una tabla sería un acceso a memoria
- Una forma de disminuir este tiempo de acceso es mediante...

Tabla Invertida de Páginas

- Existe solo una tabla invertida de página para todo el sistema
- Tamaño de la tabla de página se vuelve proporcional al tamaño de memoria física en vez de depender del tamaño de direccionamiento
- Cada entrada contiene:
- $$\#Pagina + \text{Process ID} + \text{Bits de Control} + \text{Puntero a prox página}$$
- Hay una entrada de TPI por cada marco en memoria
- Aun así genera dos acceso a memoria física y para esto se propone la TLB

TLB: Translation Lookaside Buffer

- Almacena el mapeo de página -> marco de las páginas más recientemente usadas
- Cada entrada de TLB se compone por un $\#Pag$ y $\#Marco$
- Hit de TLB: Encuentro el mapeo
  - Si no está en TLB y voy a TP a buscarlo
    - Si no está en TP -> Page Fault
      - Voy a buscarlo a memoria secundaria
      - Se bloquea proceso hasta encontrarlo y actualiza la TP
    - Si está en TP se actualiza la TLB y se genera la dirección física
  - Si está en TLB se genera dirección física



Tamaño de páginas

- Página pequeña $\rightarrow$ Menor fragmentación interna
- Página pequeña $\rightarrow$ tablas más grandes
- Página muy grande $\rightarrow$ Se pierde principio de localidad

MV con Segmentación

- Particiona la memoria virtual y física en segmentos de tamaño variable
- Da soporte a la compartición entre procesos
- Permite programas que se modifican o recompilan de forma independiente

Entrada de tabla de segmento

$$P + M + Otrosbitsdecontrol + Longitud + DirBasesegmento$$

MV Con Segmentación Paginada

- Segmentos de tamaño variable con páginas de tamaño fijo
- Dirección Virtual = $\#Segmento + \#Página + Offset$
- $\#Seg$ me indica donde inicia el segmento en dirección física
- y con el $\#Pag$ conozco a cuál página del segmento quiero acceder
- finalmente con el offset conozco el marco que debo consultar en memoria principal

Políticas de Memoria VIrtual

1. Política de Recuperación/Fetch

- Determina cuándo una página se trae a la memoria principal
- Paginación bajo demanda: Una página se trae a memoria cuando se hace referencia una posición en dicha página
- Prepaginación: Se traen varias páginas diferentes a las que ha causado el Page Fault.

2. Política de Ubicación

- Determina en qué parte de la memoria real van a residir las porciones de la memoria de un proceso
- Cuando se usa segmentación es importante ya que afecta la fragmentación externa
- En paginación no es tan importante, pues la fragmentación sería la misma independiente del lugar donde se cargue

3. Política de Reemplazo

- Objetivo: Página que será reemplazada deberá tener la menor probabilidad de volver a referenciarse en un futuro
- Bloqueo de marcos: es posible dejar marcos bloqueados para no reemplazar las páginas.

Algoritmos de reemplazo

1. Óptimo

- Se fija en el futuro, lo que hace que este algoritmo no sea posible implementar pues no se puede tener certeza de este
- Tomará como reemplazo la página para la cual la siguiente referencia se encuentra más lejos

2. Least Recently Used (LRU)

- Se fija en el pasado

- Selecciona la página que **no** ha sido referenciada durante más tiempo

3. First In First Out (FIFO)

- Se fija en los que ya están y ve el que lleva más tiempo, este, lo reemplaza.