DCC

DEPARTAMENTO DE CIENCIA
DE LA COMPUTACIÓN

IIC2343

Arquitectura de Computadores

Clase 12 - Multiprogramación

Profesor: Germán Leandro Contreras Sagredo

Objetivos de la clase

- Conocer el concepto de multiprogramación.
- Entender la diferencia entre una memoria física y virtual, así como sus usos en la práctica.
- Entender los mecanismos que permiten la ejecución de múltiples procesos.
- Realizar ejercicios que consoliden los conocimientos anteriores.

Hasta ahora...

- Ya construimos un computador que tiene todo lo necesario para que pueda ser operado y programado, además de implementar lo necesario para que se comunique con otros dispositivos.

- Además, vimos cómo mejorar los accesos a memoria a través de una jerarquía y, puntualmente, una memoria caché.

Ahora, veremos mejoras en lo que respecta a la ejecución de **más de un programa**.

Multiprogramación

La multiprogramación corresponde a una forma básica de procesamiento “paralelo”, que se realiza a partir de **un único procesador**. Por lo mismo, el concepto de “paralelismo” no aplica al 100% en este caso.

Este se basa en la **ejecución intercalada** de los programas del computador en intervalos de tiempo definidos. Estos intervalos suelen ser muy pequeños, por lo que para el usuario **se percibe como paralelismo real**.

Multiprogramación - Composición de un programa

¿Qué es lo que define a un programa?

- **Memoria**
  - Código, variables, *stack* (**por programa**).
  - Queremos tener **múltiples programas** en memoria.
- **Estado de procesamiento (CPU)**
  - *Program Counter*, *Stack Pointer*, *flags* o *condition codes*, registros.
  - Queremos que la CPU maneje **múltiples estados**.

Un programa en ejecución es lo que conocemos como **proceso**.

Multiprogramación - Manejo de memoria

¿Cómo compartimos la memoria entre programas?

- **Solución simple:** Entregar un segmento de memoria fijo a cada programa. Esto, no obstante, trae consigo muchos problemas:
  - Debemos conocer con anticipación las direcciones que le corresponden a cada programa.
  - No existe protección de acceso entre programas.
  - Habría que establecer un tamaño fijo para cada programa, lo que dificulta su ejecución y programación.

- **Solución real y efectiva:** **Memoria virtual**

Memoria virtual

Corresponde a un espacio de memoria direccionable **que es traducido al espacio direccionable de la memoria principal** (física).

De esta forma, en la práctica los programas hacen uso de direcciones virtuales que, a través de una unidad intermediaria llamada ***Memory Management Unit* (MMU)**, son traducidas a direcciones físicas de la memoria principal.

Dos programas distintos pueden acceder a una misma dirección de memoria a nivel de código (un mismo literal) que será traducida a una dirección física diferente para cada uno de ellos.

Memoria virtual

Como el espacio virtual y físico son distintos, pueden a su vez tener **tamaños distintos**.

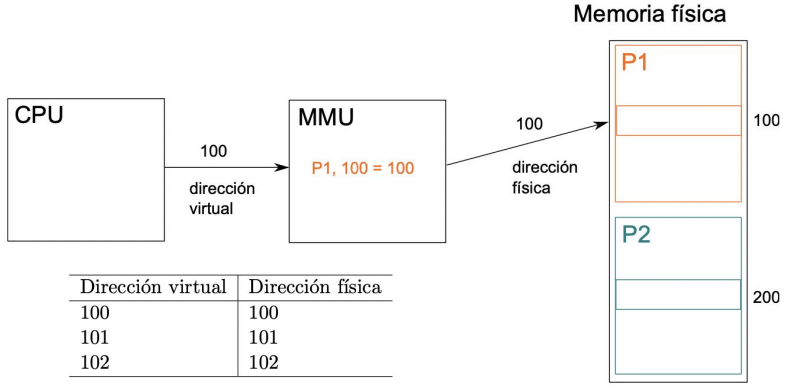
En la práctica, lo normal es que el espacio de memoria virtual sea **mayor** que el de espacio de memoria física. Esto permite que los programas puedan usar más espacio del que el computador podría permitir en una primera instancia.

No obstante lo anterior, si los programas de un computador no requirieran de mucha memoria, se podría asignar un espacio de direccionamiento virtual menor.

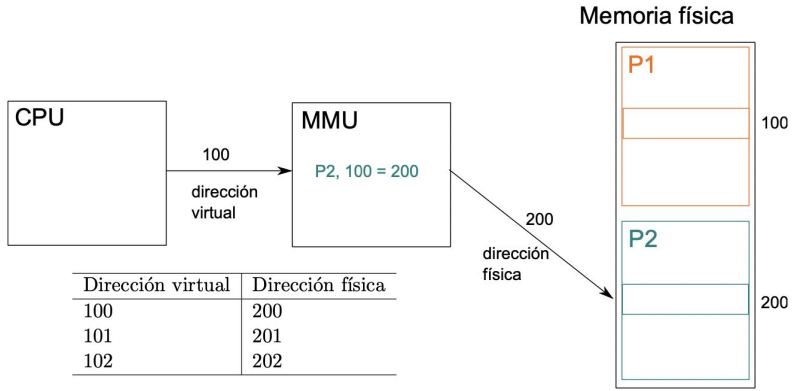
Memoria virtual - Esquema general

- En este ejemplo, los procesos P1 y P2 tratan de acceder a la misma dirección virtual 100.

- La MMU se encarga de traducirlos a espacios físicos diferentes. Por eso necesita tanto la dirección virtual como el **identificador del proceso** que solicita la dirección.



Dirección virtual	Dirección física
100	100
101	101
102	102



Dirección virtual	Dirección física
100	200
101	201
102	202

Memoria virtual - Traducción de direcciones

¿Cómo se lleva a cabo la traducción de una dirección virtual a una dirección física?

- La MMU debe contar con una **tabla** que posea la asociación entre direcciones virtuales y físicas, como se ilustró en el ejemplo.

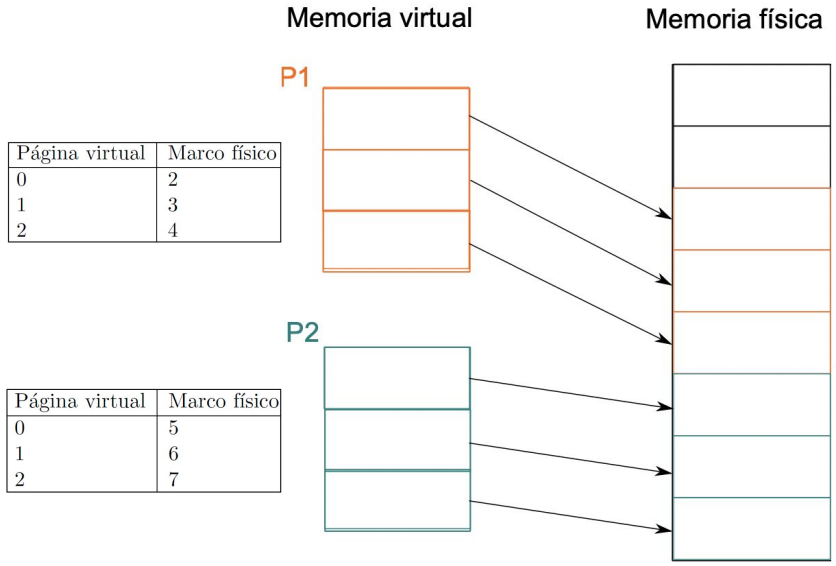
- No podemos guardar esta asociación directamente: **Cada programa** necesitaría una asociación virtual-física para todas las direcciones de memoria disponibles, lo que **podría ocupar todo el espacio de la memoria principal**.

Memoria virtual - Traducción de direcciones

- **Solución: Paginación.**

- El espacio de memoria virtual se divide en **bloques contiguos*** llamados **páginas**. Cada página será mapeada a un **marco** de la memoria física, bloque contiguo de la memoria principal del mismo tamaño.

* Similar a los conceptos de línea y bloque de memoria vistos para el esquema de caché.



Mapeo de las páginas de los procesos P1 y P2 a marcos físicos de la memoria, incluyendo sus tablas de página. En estas, solo almacenamos en memoria la asociación entre una página y un marco, lo que ocupa menos espacio.

Memoria virtual - Traducción de direcciones

- Para llevar a cabo la traducción de una dirección virtual en un esquema de páginas, hacemos uso de **los bits** de esta para separar **el número de página** del ***offset* de la palabra buscada dentro de la página**.

- **Ejemplo:** Si tenemos un espacio direccionable virtual de 9 bits, un espacio direccionable físico de 8 bits y poseemos un tamaño de página de 32 bytes (2^5 bytes):
  - Usamos **5 bits para ubicar una palabra en una página**.
  - Si tomamos la dirección **virtual** 43:
    43 = **000101011** → **Nº de página = 0001**, ***Offset* = 01011**

Memoria virtual - Traducción de direcciones

- Siguiendo el ejemplo anterior, si contamos con la siguiente asociación entre páginas y marcos:

Página	Marco
0000	100
0001	101
0010	110
...	...

Entonces, realizamos la siguiente traducción:
000101011 → 0001 (página) = 101 (marco)

Dirección física: 10101011

Cabe destacar dos cosas del ejemplo:
- Pasamos de una dirección de 9 bits a otra de 8 en este esquema, que es la dirección física.
- Del ejercicio asumimos que cada dirección almacena una palabra de 1 byte. Entonces, tenemos una memoria virtual de 2^9 bytes y una memoria física de 2^8 bytes.

Memoria virtual - Traducción de direcciones

¿Por qué los bits más significativos señalan el número de página?

Porque sigue la misma lógica que la obtención de *tags* en una memoria caché: Si tenemos 2^9 direcciones virtuales y tamaños de página de 2^5 bytes, entonces:
2^9 direcciones $\div$ 2^5 direcciones/página $\rightarrow$ **2^4 páginas**

Esto equivale a realizar 5 *shift rights* sobre la dirección virtual:

$b_8b_7b_6b_5b_4b_3b_2b_1b_0$ $\rightarrow$ $b_8b_7b_6b_5$: **Dirección de una página**
$b_4b_3b_2b_1b_0$: **Dirección de una palabra en una página**

Memoria virtual - Traducción de direcciones

¿Por qué los bits más significativos señalan el número de marco?

Misma lógica: Si tenemos 2^8 direcciones físicas y tamaños de marco de 2^5 bytes (ya que debe coincidir con el tamaño de página), entonces:
2^8 direcciones ÷ 2^5 direcciones/marco → **2^3 marcos**

Esto equivale a realizar 5 *shift rights* sobre la dirección física:

$b_7b_6b_5b_4b_3b_2b_1b_0$ → $b_7b_6b_5$: **Dirección de un marco físico**
$b_4b_3b_2b_1b_0$: **Dirección de una palabra en un marco**

Memoria virtual - Tabla de páginas

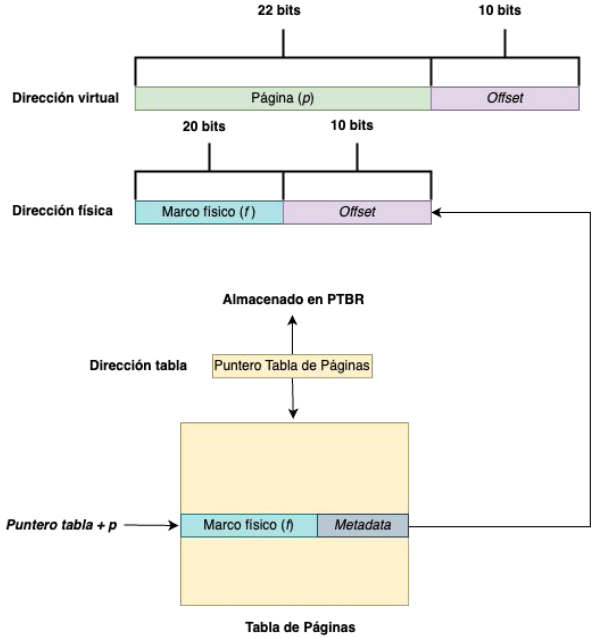
- Al crear un proceso a partir de la ejecución de un programa, a este se le asigna una **tabla de páginas** que se encuentra en la sección protegida de la memoria. La dirección de esta tabla se almacena en un registro especial: ***Page Table Base Register*** **(PTBR)**.

- El índice de cada entrada (**PTE**, ***P**age **T**able **E**ntry*) representa el **número de página** y en estas se almacenan los siguientes datos:
  - **Bits de marco físico**, utilizados para llevar a cabo la traducción de la dirección.
  - **Bits de *metadata***, que entregan información sobre la asociación entre la página y un marco físico.

Memoria virtual - Tabla de páginas

- Cuando un proceso accede a una dirección virtual, se obtiene su número de página y, a partir de la dirección almacenada en el registro PTBR, se revisa la entrada correspondiente en la tabla de páginas.

- Antes de realizar la traducción directa, **es necesario revisar los bits de *metadata***.



Ejemplo de traducción para direcciones virtuales de 32 bits, direcciones físicas de 30 bits y tamaño de página de 1KB.

Memoria virtual - Tabla de páginas

Para efectos de este curso, existen dos bits de *metadata* que será de interés considerar:

- ***Valid bit***: Indica si la entrada en la tabla de páginas es válida o no. Si no es válida, esto significa que la página **no posee un marco físico asociado**.

- ***Present bit***: Indica, para una entrada válida, si el contenido de la página está en un marco físico (“presente”) o si está en el ***swap file***.

Memoria virtual - *Page fault*

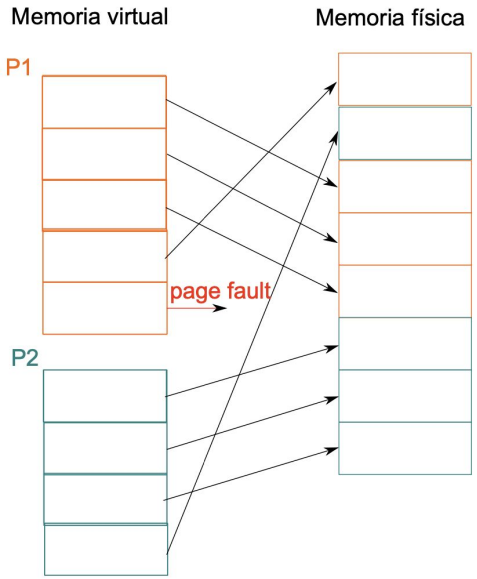
Al momento de buscar el marco físico para una página, incurrimos en un ***page fault*** si:

- La página no posee un marco físico asociado (*Valid bit* = 0).
- La página posee un marco físico, pero se encuentra en el *swap file* (*Valid bit* = 1, *Present bit* = 0).

Veremos a continuación cada caso.

Memoria virtual - *Page fault* por página no asignada

- Cuando un proceso es creado, el sistema operativo suele asignarle **solo algunos** marcos físicos a sus páginas con el fin de optimizar espacio.
- Si se accede a una dirección de una página sin marco físico, se genera un ***page fault*** y se le asigna uno si es que existen marcos disponibles.
- **¿Qué pasa si no hay marcos disponibles?**



Ejemplo de *page fault* sin marcos físicos disponibles en la memoria principal.

Memoria virtual - *Page fault* por página no asignada

El sistema operativo puede manejar el caso de no poseer espacio en la memoria principal de dos maneras:

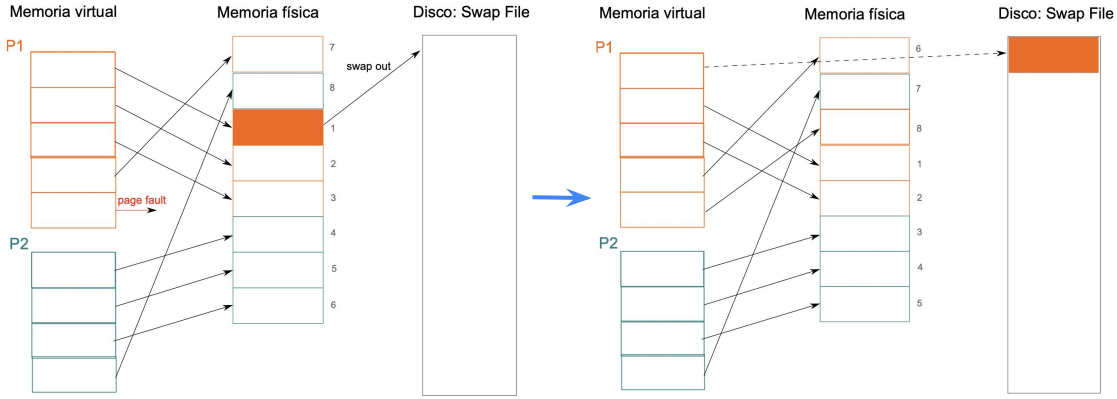
- Arrojar error de ejecución por falta de espacio (Ejemplo: `malloc()` `ENOMEM`).

- Realizar ***swapping*****:** Copiar el contenido de un marco físico según un criterio *cualquiera* a un ***swap file*** que se encuentra en **el siguiente nivel de jerarquía de memoria**, para posteriormente asociar la página al nuevo marco físico recientemente liberado.

Memoria virtual - *Page fault* por página no asignada

- El criterio para elegir el marco a ser copiado depende del sistema operativo. Al igual que con la memoria caché, se ocupan políticas como FIFO, LFU, LRU, entre otras.

- Se busca **reducir la cantidad de *swaps*** ya que estos involucran la transferencia de datos de una memoria con latencias mayores a las de la caché y memoria principal.



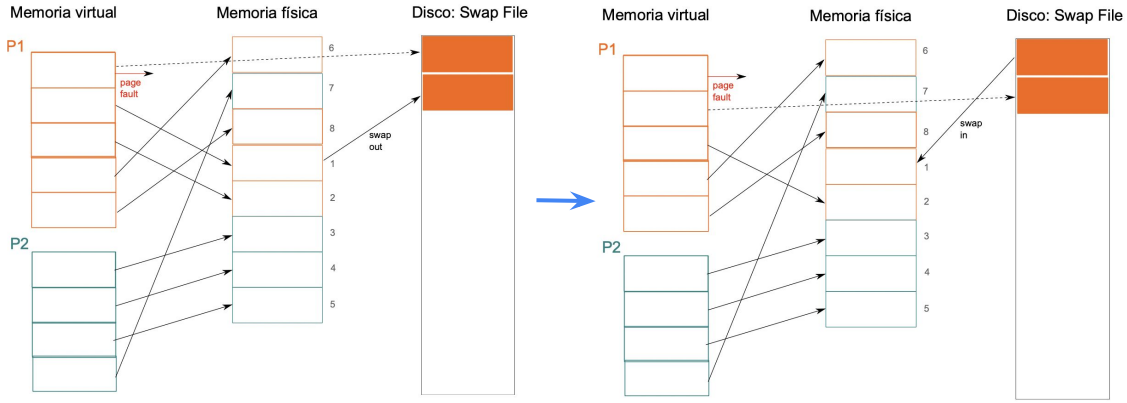
Página virtual	Marco físico	Validez	Disco
0	2	1	1
1	3	1	0
2	4	1	0
3	0	1	0
4	2	1	0
5	x	0	0
6	x	0	0
7	x	0	0

Ejemplo de *swapping*: El proceso P1 realiza *swap* del marco físico asociado a la página 0 y dicho marco ahora se asocia a la página 4. El *Valid bit* y *Present bit* ("Validez" y "Disco" respectivamente) de la primera entrada de la tabla de páginas de este proceso representan este caso.

Memoria virtual - *Page fault* por *swap file*

- Cuando un proceso posee una página válida (*Valid bit* = 1) pero esta se encuentra en el swap file (*Present bit* = 0), también se genera un ***page fault***.

- En este caso, independiente de que existan o no marcos físicos disponibles, es necesario realizar un ***swap in***: copiar en un marco físico el contenido de una página almacenada en el *swap file*.



Página virtual	Marco físico	Validez	Disco
0	3	1	0
1	3	1	1
2	4	1	0
3	0	1	0
4	2	1	0
5	x	0	0
6	x	0	0
7	x	0	0

Ejemplo de *swap-in*: El proceso P1 genera un *page fault* de una página que está en el *swap file*. Primero hace *swap out* del marco asociado a la página 1 y luego, en el marco liberado, hace el *swap in* para copiar el contenido de la página 0. El sistema operativo se encarga de actualizar los bits de *metadata* para reflejar este cambio.

Memoria virtual - Accesos a memoria

Ya con todo el esquema anterior claro: ¿cuántos accesos a memoria se requieren para obtener un dato de la memoria principal?

Al menos dos:

- Un acceso a la memoria principal para acceder a la tabla de páginas para realizar la traducción a una dirección física.
- El acceso a la jerarquía de memoria para el dato deseado.

¿Cómo podemos mejorar esto?

Memoria virtual - Accesos a memoria

Agregamos otra memoria caché: ***Translation Lookaside Buffer*** **(TLB)**.

- Guarda las traducciones de página que van siendo accedidas por cada proceso.
- Memoria relativamente pequeña (entre 16 y 512 entradas).
- Al ser más pequeñas, suelen ser *fully associative* (costo razonable para recorrer todas las entradas).
- Al igual que con las caché, puede tener niveles (L1, L2, L3).

Memoria virtual - Accesos a memoria

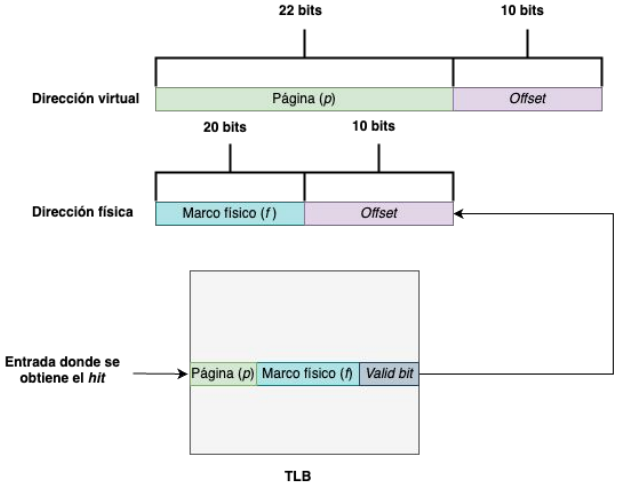
¿Qué almacena una entrada de la TLB?

- Al ser una caché *fully associative*, el *tag* a utilizar es la porción de bits de la dirección virtual que no es parte del *offset*. En este caso, **corresponde a los bits del número de página**.
- El contenido a almacenar en la entrada corresponde a lo mínimo requerido para realizar la traducción, *i.e.* **los bits de marco físico**.
- Al igual que con cualquier caché, requiere de **un bit de validez** para tener certeza de que el contenido de la entrada es válido.

Memoria virtual - Accesos a memoria

- Al ser *fully associative*, hay que recorrer todas las entradas de la TLB hasta encontrar una donde el *tag* coincida con el número de página de la dirección. Se acepta esto por sobre el uso de una caché *N-way* por tener pocas entradas (entre 16 y 512).

- En estricto rigor no es necesario almacenar los bits de *metadata* de la tabla de páginas ya que los marcos físicos accedidos más recientemente suelen estar en memoria y no en disco, pero existen implementaciones donde se incluyen.



Ejemplo de traducción para direcciones virtuales de 32 bits, direcciones físicas de 30 bits, tamaño de página de 1KB y uso de una TLB.

Memoria virtual - Accesos a memoria

¿Qué pasa si dos procesos acceden a la TLB?

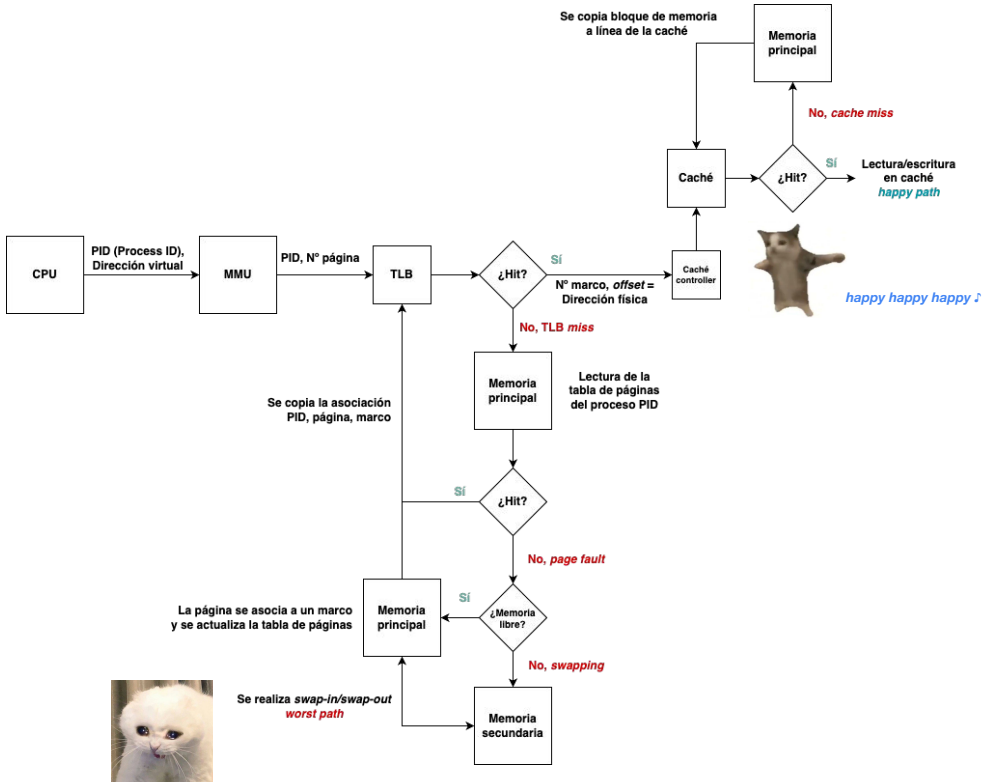
Cuando cambia el proceso en ejecución, hablamos de un **cambio de contexto**. En este caso, las entradas de la TLB son inválidas para el nuevo proceso en ejecución. Esto se puede resolver de dos formas:

1. ***Flush*:** Se "vacía" la caché (*Valid bit* = 0 en todas las entradas).
2. **ASID** (***A****ddress* ***S****pace* ***Id****entifier*): Identificador del proceso al que pertenece la traducción. Se almacena en cada entrada de la TLB y usa menos bits que el **PID** (***P****rocess* ***Id****entifier*), pero poseen asociación directa de igual forma.

Memoria virtual - Accesos a memoria

Con esto en consideración, el proceso completo de un acceso a memoria se observa en el siguiente diagrama.

Se apunta a que la mayoría de accesos sigan el ***happy path*** para disminuir el tiempo de ejecución de los procesos.



Memoria virtual - Accesos a memoria

Una mejora que se puede realizar para disminuir el *overhead* del *swapping* (y así los tiempos de acceso en el *worst path*) es la adición del ***dirty bit***.

Este bit se agrega a la *metadata* de cada PTE e indica si el marco físico ha sido modificado o no. Si no es el caso (*dirty bit* = 0), entonces el *swap out* de un marco físico **no se realiza**, se asume que su contenido sigue igual en el *swap file*. Como esto involucra la transferencia de datos con la memoria secundaria, el ahorro de tiempo es **sustancial**.

Memoria virtual - Tamaño de tabla de páginas

¿Tiene algún problema el esquema de paginación?

Supongamos que tenemos un esquema con:

- Palabras de memoria de 1B.
- Direcciones virtuales de 32 bits (2^{32}B = 4GB).
- Direcciones físicas de 30 bits (2^{30}B = 1GB).
- Tamaño de página de 1KB (2^{10}B → 10 bits de *offset*).
- #Bits de *metadata* = 4 bits (varía por implementación).

¿Cuál es el tamaño de la tabla de páginas de un proceso?

Importante: Para efectos prácticos:

- 1 byte = 1B
- 2^{10}B = 1KB
- 2^{20}B = 2^{10} KB = 1MB
- 2^{30}B = 2^{10} MB = 1GB
- 2^{40}B = 2^{10} GB = 1TB

Memoria virtual - Tamaño de tabla de páginas

Tamaño de tabla de páginas: `#PTEs * sizeof(PTE)`

- #Bits de página = Bits dir. virtual - Bits *offset* = 32 - 10 = 22
- #Bits de marco = Bits dir. física - Bits *offset* = 30 - 10 = 20
- `#PTEs` = Cantidad de páginas = $2^{\text{\#Bits de página}} = 2^{22}$
- `sizeof(PTE)` = #Bits de marco + #Bits *metadata*
  = 20 + 4 = 24 bits = 3B
- Tamaño de tabla de páginas = $2^{22} * 3B$ = **12MB**

¿Tiene algo de malo este tamaño de tabla de páginas?

Memoria virtual - Tamaño de tabla de páginas

Problemas del esquema anterior

- **La tabla de páginas no cabe dentro de una página (12MB > 1KB)**
  Generalmente se busca que quepa de forma que esta pueda contenerse por completo en un marco físico.

- **Se ocupan 12MB de espacio por proceso en ejecución**
  Si tenemos, por ejemplo, 50 procesos en ejecución, ocuparemos 600MB de espacio (más del 50% de la memoria de 1GB) **solo en tablas de páginas**.

¿Cómo podemos mejorar esto?

Memoria virtual - Tamaño de tabla de páginas

Solución simple: Aumentar el tamaño de páginas hasta que la tabla quepa completamente en una.

Si en el esquema anterior aumentamos el tamaño de página a 256KB (18 bits de *offset*), tendremos 2^{14} páginas y entradas de 2B, lo que se traduce en un tamaño de tabla de páginas de **32KB**.

Esta es una solución *naive*: Aumentar el tamaño de página implica aumentar la cantidad de memoria **mínima** que utiliza cada proceso. Si ocupan menos memoria, esta será desaprovechada y se generará **fragmentación interna**.

Memoria virtual - Tamaño de tabla de páginas

Solución real: Esquemas de paginación alternativos.

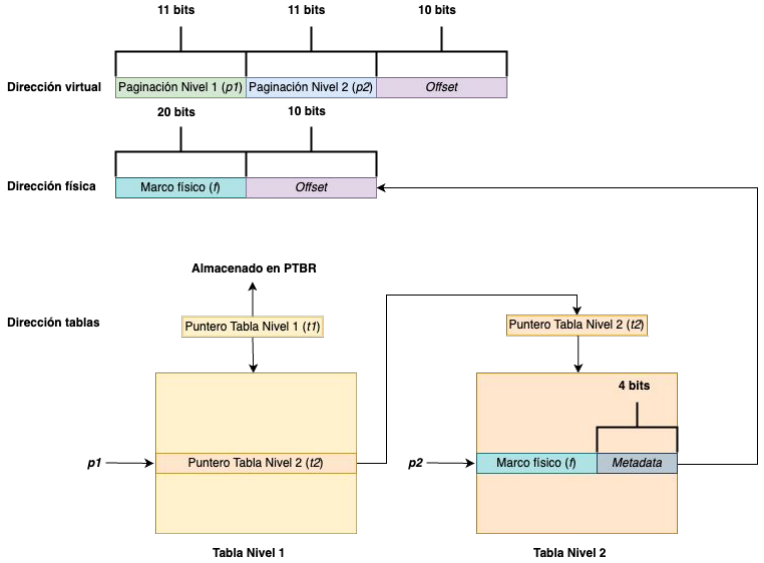
Existen dos tipos de esquema de paginación alternativos que resuelven el problema del tamaño de una tabla de páginas:

- Tabla de páginas multinivel } El más utilizado en la práctica.
- Tabla de páginas invertida

Memoria virtual - Tabla de páginas multinivel

- Se tiene una tabla de **primer nivel** cuyas entradas son **punteros a tablas de segundo nivel** (direcciones **físicas**).

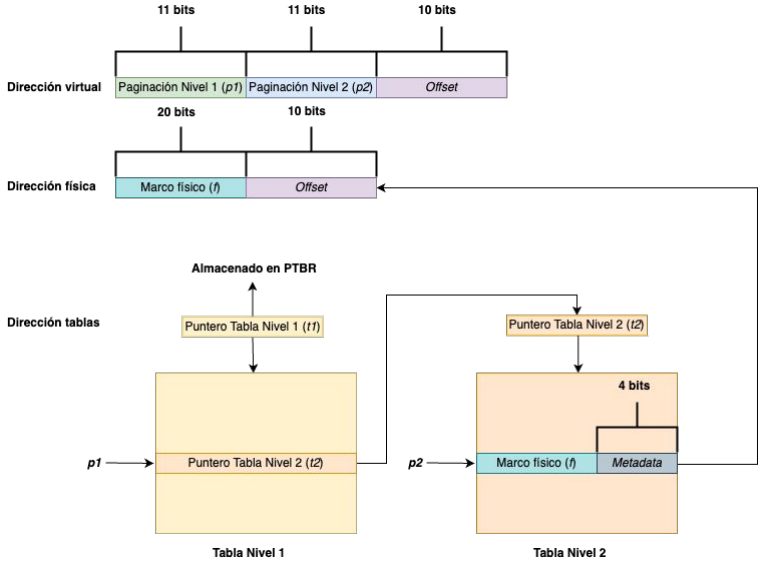
- En un esquema de paginación de dos niveles, las tablas de segundo nivel poseerán la asociación de marco físico y los bits de *metadata* para evaluar la validez de la entrada.



Ejemplo de paginación de 2 niveles para el esquema ejemplificado anteriormente.

Memoria virtual - Tabla de páginas multinivel

- Cada proceso siempre utilizará su tabla de primer nivel, con un tamaño igual a: 2^{11} * 30b = 7.5KB.
- Las tablas de segundo nivel se inicializan en memoria **solo cuando se accede a sus direcciones**. Su tamaño es: 2^{11} * 3B = 6KB.
- El uso de memoria mínimo es, entonces: 7.5KB + 6KB = **13.5 KB**

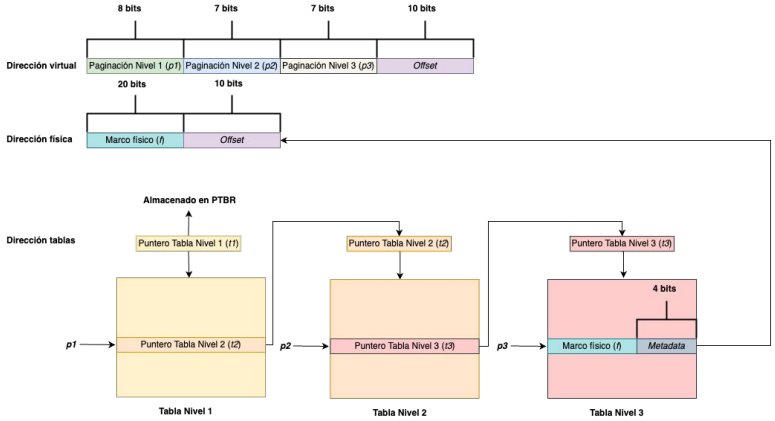


Ejemplo de paginación de 2 niveles para el esquema ejemplificado anteriormente.

Memoria virtual - Tabla de páginas multinivel

- Podemos extender este esquema a tres niveles con los siguientes tamaños de tabla por nivel:
  - Nivel 1: 2^8 * 30b = 0.9375KB
  - Nivel 2: 2^7 * 30b = 0.46875KB
  - Nivel 3: 2^7 * 3B = 0.375KB

- Entonces, tenemos un uso de memoria por tabla de páginas por proceso mínimo de:
  0.9375KB + 0.46875KB + 0.375KB = **1.78125KB**

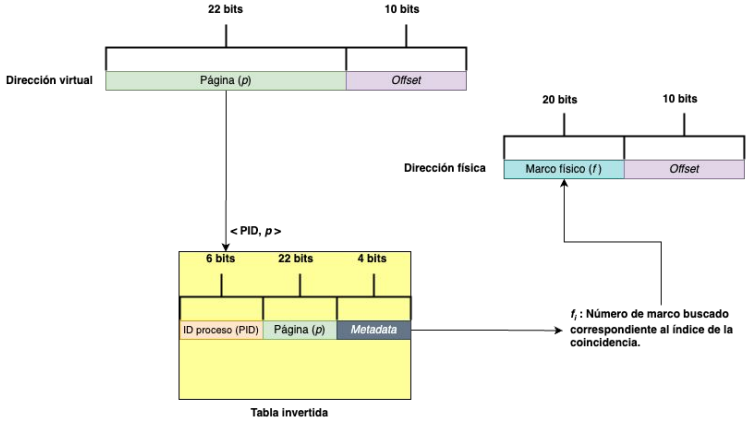


Ejemplo de paginación de 3 niveles para el esquema ejemplificado anteriormente.

Memoria virtual - Tabla de páginas invertida

- En vez de tener una tabla de páginas por proceso, tenemos una **única tabla compartida por todo proceso** que almacena la asociación **<PID, Nº página>**.

- Se tienen **tantas entradas como marcos físicos**. El índice donde se encuentre la combinación <PID, Nº página> buscada será el marco físico a utilizar.

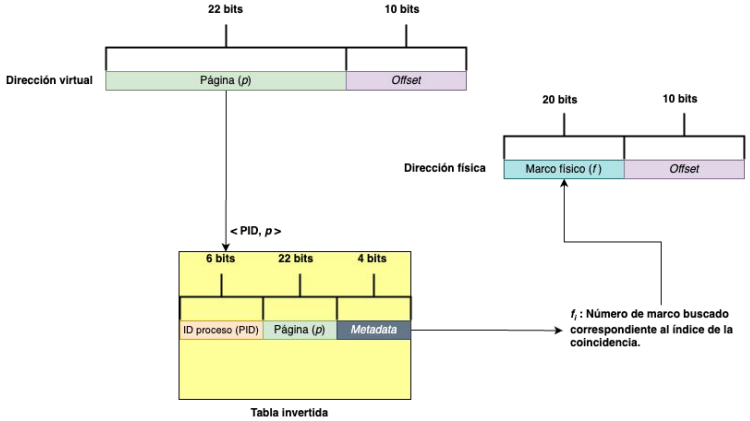


Ejemplo de paginación con tabla de páginas invertida en base al esquema anterior.

Memoria virtual - Tabla de páginas invertida

- Si asumimos un PID de 6 bits, el tamaño de tabla de página es: 2^{20} * 4B = **4MB**.

- No cabe en una página, pero es el **único** espacio de tabla utilizado para todos los procesos. Su desventaja, no obstante, es que **la traducción se debe buscar en todas las entradas de la tabla**.



Ejemplo de paginación con tabla de páginas invertida en base al esquema anterior.

Multiprogramación - Manejo de procesamiento

¿Cómo manejamos la ejecución de múltiples procesos?

- Como señalamos anteriormente, un proceso corresponde a un programa en ejecución. De esta manera, el proceso comprende tanto el **código** como los **datos en memoria** y el **estado de procesamiento en la CPU**.

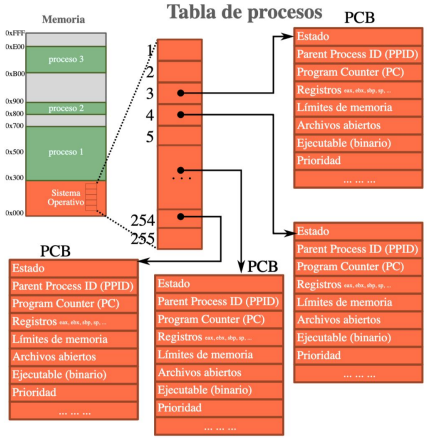
- ¿Cómo mantenemos registro de esto para cada proceso?

Multiprogramación - Manejo de procesamiento

En la sección de memoria protegida (accesible solo por el sistema operativo), se almacena una **tabla de procesos**. Cada entrada de esta tabla posee un puntero a un **Process Control Block (PCB)**.

El PCB posee la información que permite mantener registro del estado de un proceso. Entre los datos relevantes encontramos:

- Estado
- Dirección de tabla de páginas
- Valores de registros: PC, SP, etc.
- Prioridad (si corresponde)



Ejemplo de una tabla de procesos (fuente: Cristian Ruz, IIC2333).

Multiprogramación - Manejo de procesamiento

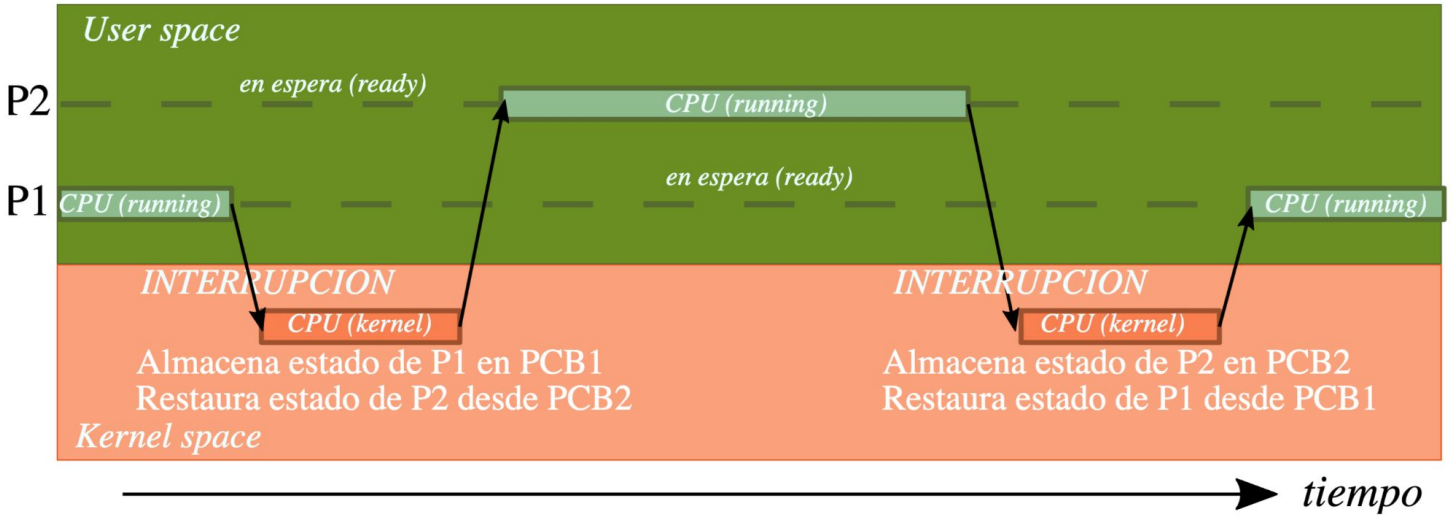
¿Cómo cambiamos la ejecución de un proceso a otro?

El sistema operativo, **que es a su vez un proceso con privilegios**, se encarga de realizar el **cambio de contexto** de la siguiente forma:

- Actualiza el bit de modo (***kernel/supervisor mode***) para tener acceso a la sección protegida de la memoria principal.
- Respalda el estado del proceso que detendrá su ejecución en su PCB correspondiente.
- Actualiza, a partir del PCB del proceso a ejecutar, los registros de la CPU para comenzar/retomar su ejecución (`PTBR`, `PC`, `SP`, etc.).

Multiprogramación - Manejo de procesamiento

¿Cómo cambiamos la ejecución de un proceso a otro?



Ejemplo de un cambio de contexto entre dos procesos P1 y P2 (fuente: Cristian Ruz, IIC2333).

Multiprogramación - Manejo de procesamiento

Trivia: Si usáramos memoria virtual en el computador básico, ¿las direcciones de los registros PC y SP serían físicas o virtuales?

R: **¡Virtuales!** En un cambio de contexto, los valores de los registros PC y SP se almacenan en el PCB. Si ocurriera *swapping*, los marcos físicos asignados a las páginas del proceso podrían cambiar y, por ende, las direcciones físicas de PC y SP ya no serían válidas. Por este motivo usan direcciones virtuales traducidas por el MMU.

La única excepción donde se almacenan direcciones físicas es **el registro PTBR**, ya que la MMU lo usa para realizar las traducciones.

Multiprogramación - Manejo de procesamiento

¿Cuándo realizamos un cambio de contexto?

El sistema operativo, mediante algoritmos de ***scheduling***, selecciona los procesos que ejecutarán en la CPU por un intervalo de tiempo determinado.

Estos algoritmos determinan el orden de ejecución y el tiempo que le toma a los procesos terminar.

Scheduling - Tipos de *scheduling*

Batch processing

Se ejecutan los procesos por *batch* y el cambio de contexto se realiza **solo cuando el proceso en ejecución termina**. Mantiene a la CPU lo más ocupada posible al reducir el *overhead* que produce un cambio de contexto, pero no otorga al usuario la sensación de "paralelismo" esperada.

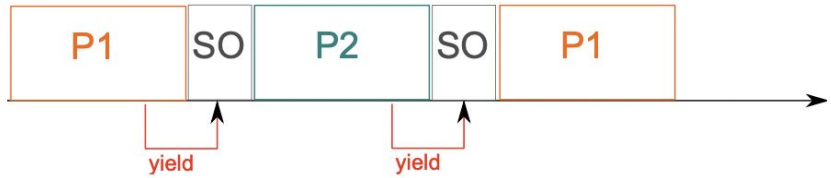
Ejemplo de *batch processing*. En este caso, la ejecución de P2 no comienza hasta que termina la ejecución de P1. En ese instante, el sistema operativo (SO) toma el control y realiza el cambio de contexto.

* En este caso no se necesita realmente almacenar el estado de los procesos en su PCB ya que, en teoría, se crean y ejecutan una única vez.

Scheduling - Tipos de *scheduling*

Non-preemptive scheduling

También conocido como ***cooperative scheduling***. Cada proceso se encarga de gatillar el cambio de contexto (*yield*), ya sea de forma voluntaria o por término de ejecución. Permite ejecución intercalada, pero no es óptimo ya que existen procesos "egoístas" que no ceden la CPU.



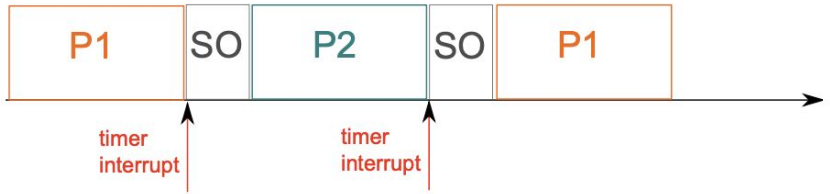
Ejemplo de *non-preemptive scheduling*. En este caso, la ejecución de P2 comienza cuando P1 cede el control. En ese instante, el SO realiza el cambio de contexto.

Algoritmo de ejemplo: First-Come, First-Served (FCFS). Los procesos se ejecutan por orden de llegada y ceden (o no) la CPU voluntariamente. Implementación simple, pero no es el óptimo en términos de tiempo promedio de término para cada proceso.

Scheduling - Tipos de *scheduling*

Preemptive scheduling

También conocido como ***interactive scheduling***. Cada proceso cuenta con un tiempo de ejecución determinado a partir de un *timer*. Cumplido el plazo, se interrumpe la ejecución y se realiza un cambio de contexto. Es el más "justo" y utilizado en la práctica.



Ejemplo de *preemptive scheduling*. En este caso, la ejecución de P2 comienza cuando P1 es interrumpido por el *timer*. En ese instante, el SO realiza el cambio de contexto.

Algoritmo de ejemplo: Round Robin (RR).
Cada proceso ejecuta una cantidad de turnos o *quantums*. La interrupción se realiza si se termina la ejecución o si se cumple la cuota de *quantums* del proceso.

Scheduling - Estados

Los procesos cuentan con **estados** que informan sobre su situación en tiempo real, considerado por los algoritmos de *scheduling*.

- ***New***: Proceso siendo creado.
- ***Running***: Proceso en ejecución.
- ***Waiting***: Proceso esperando evento (I/O, *signal*, etc.).
- ***Ready***: Proceso esperando asignación de CPU.
- ***Terminated***: Proceso cuya ejecución terminó.

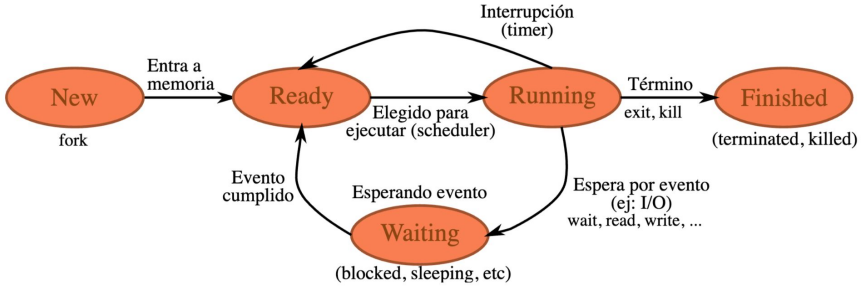


Diagrama de estados de un proceso (fuente: Cristian Ruz, IIC2333).

Scheduling - Prioridad

Adicionalmente, los procesos pueden contar con **métricas de prioridad** para que los algoritmos de *scheduling* le den turnos de CPU a los más prioritarios.

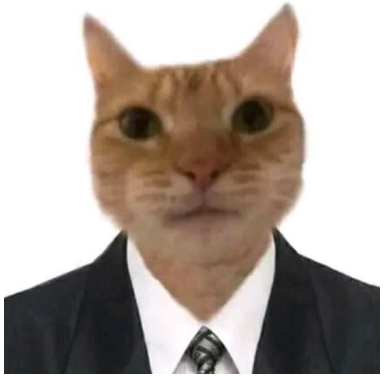
Si bien esto tiene la desventaja de que los procesos menos prioritarios sufran de **inanición** (*i.e.* que nunca terminen por procesos más prioritarios), se puede resolver con prioridades **dinámicas** (ejemplo: ***aging*** que aumenta la prioridad de procesos más viejos).

Algoritmo de ejemplo: *Multi-level feedback queue scheduling*

Ejercicios

Ahora, veremos algunos ejercicios.

Estos se basan en preguntas de tareas y pruebas de semestres anteriores, por lo que nos servirán de preparación para las evaluaciones.



Ejercicios

¿Tiene sentido usar memoria virtual, si la cantidad de memoria física es igual a la cantidad de memoria direccionable? Justifique su respuesta.

Ejercicios

Considere un computador con un espacio direccionable virtual de 32 bits, espacio de direccionamiento físico de 30 bits y páginas de 8KB.

1. Describa la composición interna de una dirección virtual.

2. ¿Cuál es el máximo número de entradas válidas que pueden existir en una tabla de páginas?

Ejercicios

En un computador con soporte para memoria virtual, ¿cómo se pueden implementar regiones de memoria protegidas y regiones de memoria compartidas?

Ejercicios

La siguiente figura presenta el estado de la memoria principal de un computador con memoria virtual en un instante dado. En base a esto, asuma:

- Tamaño de página y de cada tabla de páginas de 4 palabras.
- Existen dos procesos en ejecución, P1 y P2.
- Las páginas no existentes (sin marcos) se denotan con -.
- En una tabla de páginas, el bit más significativo de cada palabra indica si la página está en memoria o disco (0/1).
- Ambos procesos solicitan las siguientes direcciones virtuales: 0, 1, 4, 5, 8, 10, 12, 15.

Dirección	Contenido
0	6
1	15
2	3
3	−2
4	−45
5	8
6	28
7	−2
8	9
9	−3
10	8
11	12

Tabla de págs. P1

12	0
13	−12
14	24
15	–
16	0
17	−12
18	−16
19	–
20	0
21	0
22	0
23	0

Tabla de págs. P2

24	5
25	−3
26	−3
27	2
28	0
29	1
30	2
31	3
⋮	⋮

Para cada proceso, transforme las direcciones virtuales en físicas. Si la transformación fue exitosa, indique el dato obtenido. En caso contrario, indique el tipo de *page fault* generado.

Ejercicios

En un computador Von Neumann donde las instrucciones y los datos de un programa están en el mismo espacio de memoria, suponiendo que se tiene memoria virtual, ¿qué se podría hacer para que las páginas de datos nunca sean interpretadas como instrucciones por el computador?

Antes de terminar

¿Dudas?

¿Consultas?

¿Inquietudes?

¿Comentarios?

DCC

DEPARTAMENTO DE CIENCIA
DE LA COMPUTACIÓN

IIC2343

Arquitectura de Computadores

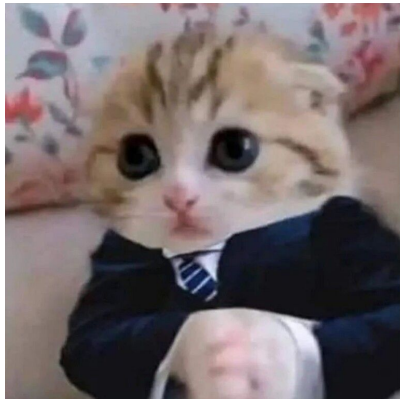
Clase 12 - Multiprogramación

Profesor: Germán Leandro Contreras Sagredo

Anexo - Resolución de ejercicios

¡Importante!

Estos ejercicios pueden tener más de un desarrollo correcto. Las respuestas a continuación no son más que soluciones que **no excluyen** otras alternativas igual de correctas.



Ejercicios - Respuesta

¿Tiene sentido usar memoria virtual, si la cantidad de memoria física es igual a la cantidad de memoria direccionable? Justifique su respuesta.

Sí. Aunque ya no se pueda utilizar más espacio de la memoria principal realizando *swapping*, se siguen teniendo las ventajas de multiprogramación: (1) El/la programador(a) no se preocupa del direccionamiento a nivel de código; (2) se añade protección al evitar accesos de memoria a otros procesos; y (3) se puede ocupar todo el espacio de direccionamiento virtual, sin cotas superiores ni inferiores en términos de direcciones.

Ejercicios - Respuesta

Considere un computador con un espacio direccionable virtual de 32 bits, espacio de direccionamiento físico de 30 bits y páginas de 8KB.

1. Describa la composición interna de una dirección virtual.

Al tener un tamaño de página de 8KB, se tiene un total de 2^3KB = 2^{13}B, es decir, se almacenan 2^{13} palabras. Para poder ubicar una palabra en la página, entonces, se necesitan 13 bits. Luego, como la dirección virtual consta de 32 bits, se tienen 32 - 13 = 19 bits para indicar el número de página (es decir, se puede tener un total de 2^{19} páginas).
Entonces, la composición interna de una dirección virtual es como sigue[1]:

1011010010110101101011010101110

Donde el segmento rojo[2] (porción más significativa) corresponde al número de página, y el naranjo (porción menos significativa) al *offset*.

[1] Número totalmente arbitrario.
[2] Si usted es daltónico, ruego me disculpe.

Ejercicios - Respuesta

Considere un computador con un espacio direccionable virtual de 32 bits, espacio de direccionamiento físico de 30 bits y páginas de 8KB.

2. ¿Cuál es el máximo número de entradas válidas que pueden existir en una tabla de páginas?

El máximo número de entradas válidas será el número máximo de marcos físicos posibles. Primero, al tener 13 bits de *offset*, se tiene un total de 30 - 13 = 17 bits para direccionar marcos, es decir, se puede tener un total de 2^{17} marcos físicos. Ahora, uno de estos marcos **debe** almacenar la tabla de páginas. Por lo que si se tiene un solo proceso, se puede tener un total de $2^{17} - 1$ entradas en tabla.

* Como los marcos que guardan las tablas de página son de la sección protegida de memoria, en la práctica son menos las entradas que pueden ser válidas al mismo tiempo.

Ejercicios - Respuesta

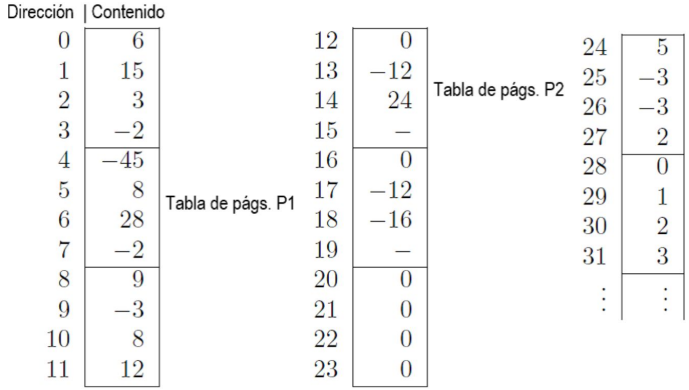
En un computador con soporte para memoria virtual, ¿cómo se pueden implementar regiones de memoria protegidas y regiones de memoria compartidas?

Para implementar regiones de memoria compartida, basta con permitir la asignación de un mismo marco a dos páginas diferentes. Así, un proceso podría acceder a los datos de otro (pues comparten el espacio). Por otra parte, para implementar regiones de memoria protegidas, basta con prohibir el uso de un marco físico para la asignación de una página (por ejemplo, con un *stack* que contenga las direcciones de los marcos físicos que pueden ser asignados). Así, las regiones de memoria protegidas podrían ser accedidas solo por el sistema operativo (modo *kernel*).

Ejercicios - Respuesta

La siguiente figura presenta el estado de la memoria principal de un computador con memoria virtual en un instante dado. En base a esto, asuma:

- Tamaño de página y de cada tabla de páginas de 4 palabras.
- Existen dos procesos en ejecución, P1 y P2.
- Las páginas no existentes (sin marcos) se denotan con -.
- En una tabla de páginas, el bit más significativo de cada palabra indica si la página está en memoria o disco (0/1).
- Ambos procesos solicitan las siguientes direcciones virtuales: 0, 1, 4, 5, 8, 10, 12, 15.



Para cada proceso, transforme las direcciones virtuales en físicas. Si la transformación fue exitosa, indique el dato obtenido. En caso contrario, indique el tipo de *page fault* generado.

*** Respuesta en la siguiente diapositiva**

Ejercicios - Respuesta

Antes de hacer el análisis, algunas cosas a notar:

I. Al tener 4 palabras por página, se necesitan 2 bits (2^2) para el *offset* dentro de una dirección.

II. Por simplicidad, se tomarán 5 bits para las direcciones virtuales (podrían ser más).

III. Notar que el segundo bloque corresponde a la tabla de páginas del proceso 1, y el séptimo bloque a la tabla de páginas del proceso 2.

IV. Desde la dirección física 28 en adelante, los datos almacenados van en orden creciente: 0,1,2,3,4,5,6,...

V. En el análisis, al hacer la traducción a dirección física, se tendrá que el segmento en negrita indicará el marco físico, y el segmento normal el *offset* dentro del mismo.

Ejercicios - Respuesta

- **Proceso 1**
  - *Acceso 0:* Dirección 00000. Corresponde a la página 0 con *offset* 0. Si revisamos la tabla de páginas, la entrada asociada es igual a -45. Por complemento a 2, sabemos que el bit más significativo de esta palabra es igual a 1, por lo que se genera un ***page fault* por tener la página en el disco**.
  - *Acceso 1:* Dirección 00001. Corresponde a la página 0 con *offset* 1. Si revisamos la tabla de páginas, la entrada asociada es igual a -45. Por complemento a 2, sabemos que el bit más significativo de esta palabra es igual a 1, por lo que se genera un ***page fault* por tener la página en el disco**.
  - *Acceso 4:* Dirección 00100. Corresponde a la página 1 con *offset* 0. Si revisamos la tabla de páginas, la entrada asociada es igual a 8. Haciendo la traducción: **1000**00 = 32. Revisando la figura, vemos que el dato almacenado es 4.
  - *Acceso 5:* Dirección 00101. Corresponde a la página 1 con *offset* 1. Si revisamos la tabla de páginas, la entrada asociada es igual a 8. Haciendo la traducción: **1000**01 = 33. Revisando la figura, vemos que el dato almacenado es 5.
  - *Acceso 8:* Dirección 01000. Corresponde a la página 2 con *offset* 0. Si revisamos la tabla de páginas, la entrada asociada es igual a 28. Haciendo la traducción: **1110**00 = 112. Si revisamos la tabla de páginas, podemos deducir el valor: Valor = Dirección - 28 = 84.
  - *Acceso 10:* Dirección 01010. Corresponde a la página 2 con *offset* 2. Si revisamos la tabla de páginas, la entrada asociada es igual a 28. Haciendo la traducción: **1110**10 = 114. Si revisamos la tabla de páginas, podemos deducir el valor: Valor = Dirección - 28 = 86.
  - *Acceso 12:* Dirección 01100. Corresponde a la página 3 con *offset* 0. Si revisamos la tabla de páginas, la entrada asociada es igual a -2. Por complemento a 2, sabemos que el bit más significativo de esta palabra es igual a 1, por lo que se genera un ***page fault* por tener la página en el disco**.
  - *Acceso 15:* Dirección 01111. Corresponde a la página 3 con *offset* 3. Si revisamos la tabla de páginas, la entrada asociada es igual a -2. Por complemento a 2, sabemos que el bit más significativo de esta palabra es igual a 1, por lo que se genera un ***page fault* por tener la página en el disco**.

- **Proceso 2**
  - *Acceso 0:* Dirección 00000. Corresponde a la página 0 con *offset* 0. Si revisamos la tabla de páginas, la entrada asociada es igual a 0. Haciendo la traducción: **0**00 = 0. Revisando la figura, vemos que el dato almacenado es 6.
  - *Acceso 1:* Dirección 00001. Corresponde a la página 0 con *offset* 1. Si revisamos la tabla de páginas, la entrada asociada es igual a 0. Haciendo la traducción: **0**01 = 1. Revisando la figura, vemos que el dato almacenado es 15.
  - *Acceso 4:* Dirección 00100. Corresponde a la página 1 con *offset* 0. Si revisamos la tabla de páginas, la entrada asociada es igual a -12. Por complemento a 2, sabemos que el bit más significativo de esta palabra es igual a 1, por lo que se genera un ***page fault* por tener la página en el disco**.
  - *Acceso 5:* Dirección 00101. Corresponde a la página 1 con *offset* 1. Si revisamos la tabla de páginas, la entrada asociada es igual a -12. Por complemento a 2, sabemos que el bit más significativo de esta palabra es igual a 1, por lo que se genera un ***page fault* por tener la página en el disco**.
  - *Acceso 8:* Dirección 01000. Corresponde a la página 2 con *offset* 0. Si revisamos la tabla de páginas, la entrada asociada es igual a 24. Haciendo la traducción: **1100**00 = 96. Si revisamos la tabla de páginas, podemos deducir el valor: Valor = Dirección - 28 = 68.
  - *Acceso 10:* Dirección 01010. Corresponde a la página 2 con *offset* 2. Si revisamos la tabla de páginas, la entrada asociada es igual a 24. Haciendo la traducción: **1100**10 = 98. Si revisamos la tabla de páginas, podemos deducir el valor: Valor = Dirección - 28 = 70.
  - *Acceso 12:* Dirección 01100. Corresponde a la página 3 con *offset* 0. Si revisamos la tabla de páginas, la entrada asociada es igual a -. Por enunciado, sabemos que se genera un ***page fault* por marco físico no asignado**.
  - *Acceso 15:* Dirección 01111. Corresponde a la página 3 con *offset* 3. Si revisamos la tabla de páginas, la entrada asociada es igual a -. Por enunciado, sabemos que se genera un ***page fault* por marco físico no asignado**.

Ejercicios - Respuesta

En un computador Von Neumann donde las instrucciones y los datos de un programa están en el mismo espacio de memoria, suponiendo que se tiene memoria virtual, ¿qué se podría hacer para que las páginas de datos nunca sean interpretadas como instrucciones por el computador?

Se puede agregar un bit **NX** a la *metadata* de una entrada de la tabla de páginas que indique si las direcciones contenidas en ella se pueden ejecutar o no. Si el bit NX está en 1, cualquier intento de ejecutar su código debiera ignorarse y/o activar un *trap*.