Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

**IIC2343 – Arquitectura de Computadores**

# Ayudantía 8 – Solución propuesta

**Profesores: Hans-Albert Löbel Díaz, Jurgen Dieter Heysen Palacios**

**Ayudante: Germán Leandro Contreras Sagredo (glcontreras@uc.cl)**

## Nota al lector

El título dice "solución propuesta" por una razón bien sencilla: Estos ejercicios pueden tener más de un desarrollo correcto. Lo que se pretende hacer aquí es mostrar un camino a la solución, sin excluir la posibilidad de rutas alternativas igual de correctas.

## Preguntas

1. a. ¿Cuáles son las desventajas de implementar multiprogramación sin memoria virtual?

   Son, principalmente, tres:

   I. Al tener cada proceso un espacio de memoria asignado, el programador tendría que conocer de antemano las direcciones físicas asociadas a cada uno de estos, de forma que no se generen inconsistencias.

   II. No existiría ningún tipo de protección: Un proceso podría acceder a espacios de memoria física que no se le asignaron.

   III. Al repartir el espacio de memoria, cada proceso tendría un tamaño de memoria fijo. Esto haría que algunos procesos tuvieran mucho espacio de sobra, mientras que a otros les hiciera falta más para poder ejecutarse.

   b. **(I3 - I/2013)** ¿Cómo debería modificarse la arquitectura del computador básico para que tenga soporte para multiprogramación?

   Se debería añadir lo siguiente:

   I. **Bit de modo:** Este bit se añade al registro `Status`, y se encarga de indicar si se está en modo usuario o en modo supervisor (*kernel*). En este último es donde se le entrega el mando al sistema operativo, el que se encarga de realizar el cambio de contexto entre procesos.

   II. **Page Table Base Register (`PTBR`):** Registro especial que se encarga de almacenar la dirección inicial de la tabla de páginas asociada a un proceso específico. Solo puede ser modificado en modo supervisor.

III. **Memory Management Unit (`MMU`):** Esta pieza de *hardware* se encarga de realizar la traducción de una dirección virtual a una dirección física. Se hace indispensable para poder obtener los datos que requiere cada proceso.

IV. **Process Control Block (`PCB`):** Corresponde a un registro especial en el que se almacena el valor de los registros del proceso en ejecución, antes de que este le otorgue el control al sistema operativo.

c. **(I3 - I/2017)** ¿Tiene sentido usar memoria virtual, si la cantidad de memoria física es igual a la cantidad de memoria direccionable? Justifique su respuesta.

Sí, sigue teniendo sentido. La memoria virtual hace que se facilite la multiprogramación al evitar las desventajas mencionadas en **1.**, ya que:

1) Cada programa tiene la impresión de estar solo en el computador, por lo que el programador no se debe preocupar de las direcciones físicas asociadas a este.
2) Como cada programa solo puede acceder a su espacio de memoria virtual y no al de otro, se añade protección.
3) Como pueden direccionar a **todo** el espacio de memoria, no hay un límite de tamaño.

Por lo tanto, sigue cumpliendo su objetivo. No hay que olvidar que la razón de implementar memoria virtual es precisamente la ejecución de diversos procesos secuencialmente, más que el poder direccionar en un espacio mayor al de la memoria física disponible.

d. **(I3 - I/2016)** Considere un computador con un espacio direccionable virtual de 32 bits, espacio de direccionamiento físico de 30 bits y páginas de 8KB.

I. Describa la composición interna de una dirección virtual.
Al tener un tamaño de página de 8KB, se tiene un total de $2^3$KB = $2^{13}$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]:

1011010010110101101011010110101110

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

II. ¿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.

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

2. a. **(I3 - II/2015)** ¿Cuál es la cantidad mínima de memoria física que debe tener un computador de 32 bits con un esquema de paginación simple?

   Debe tener suficiente espacio para:

   1) **Tabla de páginas:** Para poder tener la asociación entre una página y un marco físico.
   2) **Una página/marco físico:** Para poder hacer uso de la memoria física y *swapping* con el disco. Si no se tuviera este espacio disponible, pierde sentido el uso de la tabla de páginas ya que no habrían secciones de memoria por asignar.

   Notar que esta respuesta es independiente del número de bits.

   b. **(I3 - II/2016)** Indique bajo qué condiciones el uso de memoria virtual podría hacer más lenta la ejecución de los procesos en un computador.

   La ejecución se hace más lenta cuando se produce *swapping*. Estos se dan dos escenarios:

   I. Si no quedan marcos físicos disponibles para asignar, será necesario hacer un *swap out* para almacenar una página en el disco y un posterior *swap in* para guardar la nueva. De esta forma, queda un marco disponible para la página que lo necesita.
   II. Como consecuencia del punto anterior, si la página accedida se encuentra almacenada en el disco, será necesario hacer un *swap out* de una que se encuentre en la memoria física. Luego, se usa ese nuevo espacio disponible para realizar un *swap in* de la página que se buscaba acceder en un comienzo.

   En los dos casos se termina por acceder al disco, lo que implica un mayor tiempo de ejecución dado que implican una transferencia de datos entre ambas estructuras de memoria.

   c. **(I3 - II/2016)** Durante la mayoría de sus tiempo de ejecución, un proceso tiene el 51 % de sus páginas en el *swap file*. Otro proceso que utiliza la misma cantidad de memoria total, tiene el 49 % de sus páginas en el *swap file*, también durante la mayor parte de su tiempo de ejecución. Asumiendo que ambos se ejecutan durante la misma cantidad de tiempo, ¿cuál de los dos procesos generó más *page faults*?

   No es posible determinarlo. Por ejemplo, los accesos del primer proceso podrían ser solo a las páginas que se encuentran almacenadas en la memoria física y no en el disco, lo que no generaría ningún *page fault*. Por el contrario, el segundo proceso podría tener una cadena de accesos secuencial en lo que respecta al número de página, lo que haría que se produjeran constantemente *page faults* (se ocuparían todos los marcos físicos en un comienzo, y luego se harían *swap outs* de forma constante para los siguiente accesos). Por lo tanto, que un proceso posea más páginas en el *swap file* no determinará necesariamente que tenga una tasa mayor de *page faults*.

   d. **(I3 - I/2016)** 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. De esta forma, 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, que exista 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*).

e. **(I3 - I/2013)** La siguiente figura presenta el estado de la memoria principal de un computador con memoria virtual en un instante dado:

| 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 |
| ⋮ | ⋮ |

En base a esto, asuma la siguiente situación:

- Tamaño de cada página y de cada tabla de páginas es de 4 palabras.
- Existen dos procesos en ejecución, P1 y P2.
- Las páginas no existentes (no asociadas a marcos) se denotan con -.
- En una tabla de páginas, el bit más significativo de cada palabra indica si la página está en memoria (0) o disco (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.

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.

- **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: **11100**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: **11100**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: **11000**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: **11000**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**.

f. **(I3 - II/2011)** En un computador de 32 bits con 1 GB de RAM, se ejecuta un proceso que en cierto momento utiliza 1.5 GB de RAM. ¿Cómo es posible que ocurra esto sin que el proceso se caiga? Comente acerca del tiempo de ejecución de este proceso.

Esto se puede debido al uso de la memoria virtual. Al ser de 32 bits, el espacio direccionable del proceso corresponde a:

$$2^{32}\text{B} = 2^{2}\text{GB} = 4\text{GB}$$

Es decir, el proceso puede hacer uso de **hasta** 4 GB. Ahora, como en la RAM (nuestra memoria física) solo se dispone de 1 GB, lo que hace el proceso es realizar *swapping* constantemente para el espacio restante que utiliza. Esto evidentemente impacta la ejecución de forma negativa, debido al gasto proveniente de la transferencia de datos entre el disco y la memoria física. Por este motivo, el proceso tendrá un tiempo de ejecución mucho mayor al deseable.

g. **(I3 - II/2011)** Describa el peor caso posible, desde la óptica de la cantidad de accesos, que puede ocurrir cuando un programa intenta obtener un dato almacenado en memoria en un sistema con memoria virtual y caché.

El peor caso (*worst path*) sería la siguiente secuencia:

I. Se accede a la `TLB`, pero ninguna de sus entradas coincide con la página solicitada (`TLB` ***miss exception***).

II. El sistema operativo va a buscar la entrada a la tabla de páginas del proceso, pero la página posee la marca que indica que está en disco. Se genera un **page fault**, y será necesario realizar un *swap out* de una página que se encuentre en la memoria física, y un posterior *swap in* para almacenar la página deseada en el nuevo marco disponible. Luego, se deberá actualizar la tabla de páginas y la `TLB` con la nueva entrada.

III. Finalmente, se vuelve a repetir el acceso a la `TLB`, esta vez con un *hit*.

3. a. **(Examen - II/2014)** ¿Qué es un cambio de contexto y un `PCB`? ¿Qué relación existe entre ellos?

   El cambio de contexto corresponde al cambio del proceso que se encuentra en ejecución en la CPU, mientras que el `PCB` Corresponde a un registro especial en el que se almacena el valor de los registros del proceso en ejecución, además de otra información relevante (límites de memoria, estado, *flags*, etc.). La relación existente es que el cambio de contexto se realiza satisfactoriamente gracias al `PCB`, debido a que este último contiene toda la información necesaria que le permite al proceso retomar su ejecución desde su último estado. Es decir, permite respaldar los procesos para su posterior restauración.

   b. **(I3 - I/2012)** ¿Es posible reutilizar el contenido de la `TLB` cuando ocurre un cambio de contexto?

   No, ya que la `TLB` contendrá en sus entradas los valores de la tabla de páginas correspondientes al proceso que acaba de salir del control de la CPU. Por ello, es necesario borrar sus entradas para almacenar valores correspondientes a la tabla de páginas del nuevo proceso.

   c. **(I3 - I/2013)** Describa qué elementos de un sistema con soporte para multiprogramación (SO, CPU y Memoria) deben actualizarse al realizar un cambio de contexto.

   Los que deben actualizarse con los siguientes:

   1) `TLB`**:** Debe limpiarse por completo para almacenar la asociación entre páginas y *frames* del proceso que ejecutará posteriormente.
   2) `PTBR`**:** Debido a que se trabajará con la tabla de páginas de otro proceso (que, evidentemente, se encuentra almacenada en una dirección de memoria distinta).
   3) **Bit de modo:** Debido a que el sistema operativo es el que realiza el cambio de contexto, por lo que es necesario encontrarse en modo *kernel* para tener los permisos correspondientes.
   4) `PCB`**:** Para el proceso que sale de la ejecución, debido a que se quiere poder restaurar desde su último estado ejecutado en la CPU.

   d. **(I3 - I/2016)** Indique qué eventos generan un cambio de contexto en el esquema de *cooperative scheduling*.

   En este tipo de *scheduling*, solo hay dos formas de generar un cambio de contexto:

   I. Que un proceso entregue voluntariamente el control de la CPU al sistema operativo (***yield***).
   II. Una petición de interacción con un dispositivo I/O.

   e. **(I3 - II/2016)** ¿Por qué no es enmascarable la `IRQ0` en un computador `x86`?

   La `IRQ0` corresponde al *timer* del sistema. A través de este es que se pueden llevar a cabo los cambios de contextos entre procesos, dado que en el contexto de *pre-emptive scheduling* se hace uso del *timer* para determinar el momento en el que se debe **interrumpir** una ejecución para el cambio de contexto. Por lo tanto, no debe ser posible enmascararla ya que jugaría en contra de la dinámica del computador.

f. **(I3 - II/2015)** Explique cómo un proceso puede sufrir de inanición si se utiliza el esquema *fixed priority pre-emptive scheduling*. ¿Es posible que ocurra esto en el esquema *round-robin*?

El *fixed priority pre-emptive scheduling* se preocupa de ejecutar los procesos de mayor prioridad. De este modo, pueden haber procesos de baja prioridad que nunca son ejecutados, por el ingreso constante de otros de mayor importancia. De esta forma, los procesos menos importantes sufren de inanición. Esto no ocurre con *round robin*, ya que este tipo de *scheduling* se encarga de darle una cantidad de ciclos de ejecución constante a cada proceso en la cola, por lo que eventualmente todos tendrán su turno para ejecutar.

g. **(I3 - I/2016)** En un computador con soporte para memoria virtual, ¿el registro `PC` almacena direcciones virtuales o físicas? Justifique su respuesta considerando el efecto de los cambios de contexto y del *swapping*.

Si el `PC` solo almacenara direcciones físicas, se podrían generar problemas de consistencia. Por ejemplo, veamos el caso en el que se genere un cambio de contexto. Será necesario almacenar el registro del `PC` para poder continuar la ejecución del proceso una vez que vuelva a acceder control. Sin embargo, en el entre tanto, se puede generar un *swap out* de sus datos. De esta forma, una vez que vuelva a ser ejecutado, al hacer *swap in* para reingresar sus páginas, estas podrían quedar asociadas a un marco físico diferente al inicial. Entonces, al trabajar con direcciones físicas, el `PC` tendría una dirección que estaría asociada al mismo marco inicial, pero que contendría datos diferentes. Es por ello que debe almacenar direcciones virtuales, ya que estas siempre serán traducidas a la región correspondiente gracias a la `MMU`.

h. **(Examen - I/2018)** 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 cada página, por ejemplo en 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*.