Universidad de Alcalá

Departamento de Automática

Arquitectura de Computadores 4º Curso – I. de Telecomunicación

Práctica 5
Memoria Virtual y Algoritmos de Reemplazo de Páginas

1 Objetivos

- Entender el funcionamiento básico de la Memoria Virtual y la MMU.
- Implementación de algunos algoritmos sencillos de reemplazo de páginas.

2 Gestión de memoria virtual

2.1 Introducción

Cuando hablamos de memoria virtual nos referimos a una serie de mecanismos habilitados por el sistema operativo para simular, de forma transparente para el usuario, la existencia de mayor cantidad de memoria física que la realmente instalada en realidad.

A grandes rasgos esto se consigue empleando un dispositivo de almacenamiento secundario (típicamente un disco duro) y una serie de mecanismos hardware y software que permitan mantener en memoria sólo los fragmentos de memoria que se están usando en un momento dado, almacenando en el disco el resto y realizando la carga y el almacenamiento de los mismos según es necesario en cada momento.

Veremos a continuación cómo se consigue este propósito.

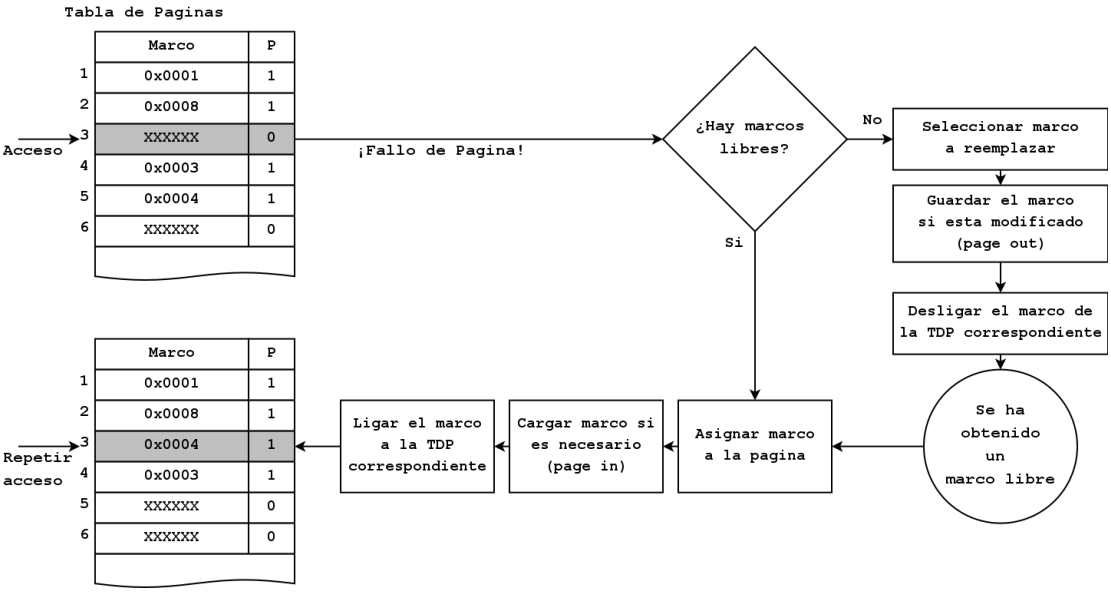
2.2 Implementación de la memoria virtual

La implementación de la memoria virtual normalmente se apoya en esquemas de

gestión de memoria como la segmentación o la paginación, y por lo tanto requiere la existencia de ciertas características hardware. Así, el sistema cargará y almacenará desde/hacia disco los segmentos o páginas según sea necesario, y será la propia CPU a través de la unidad de gestión de memoria (MMU) la que detecte cuándo un segmento o página está o no cargado, desencadenando los mecanismos software necesarios para resolver la situación.

Cuando se intenta acceder a una página o segmento y éste no está cargado en la memoria, se debe detectar la situación y actuar en consecuencia. Para detectar esta situación la MMU cuenta en las tablas de páginas (TDP) o tablas de segmentos (TDS) con un bit asociado a cada página o segmento denominado *bit de presencia* (P). En adelante nos centraremos en sistemas basados en paginación por ser más sencillos y ser actualmente los más utilizados.

Cuando se realiza un acceso a memoria y el bit de presencia está a 0, se genera una excepción de *fallo de página*. El manejador de esta excepción será el encargado de determinar el origen de la situación anómala y solucionarla si es posible. En el caso que nos ocupa (una página no está presente pero es correcta) la situación se debe simplemente al funcionamiento del sistema de memoria virtual, y por lo tanto habrá que buscar un marco de página libre, cargarlo con los datos necesarios desde el disco, actualizar la TDP y relanzar el acceso a memoria, más o menos como describe el siguiente diagrama.



En la figura se describen brevemente las acciones que se desencadenan tras un fallo de página provocado por la ausencia de una página y se ilustra con un ejemplo, representado por las TDP a la izquierda.

En este ejemplo se puede observar cómo un acceso sobre la página 3 provoca un fallo de página, y tras determinar que no hay marcos libres en el sistema se elige un marco víctima a ser desalojado mediante un *algoritmo de reemplazo*. El marco elegido aparentemente es el 0x0004, por lo que se guarda su contenido en el disco si es necesario y se desliga de la página 5 del proceso. A continuación se carga del disco si es necesario con los datos correspondientes a la página 3 y se liga el marco a dicha página. Una vez hecho esto se puede reiniciar el acceso a memoria, que ya no generará ninguna excepción. El estado inicial y final de la TDP se muestra en la figura.

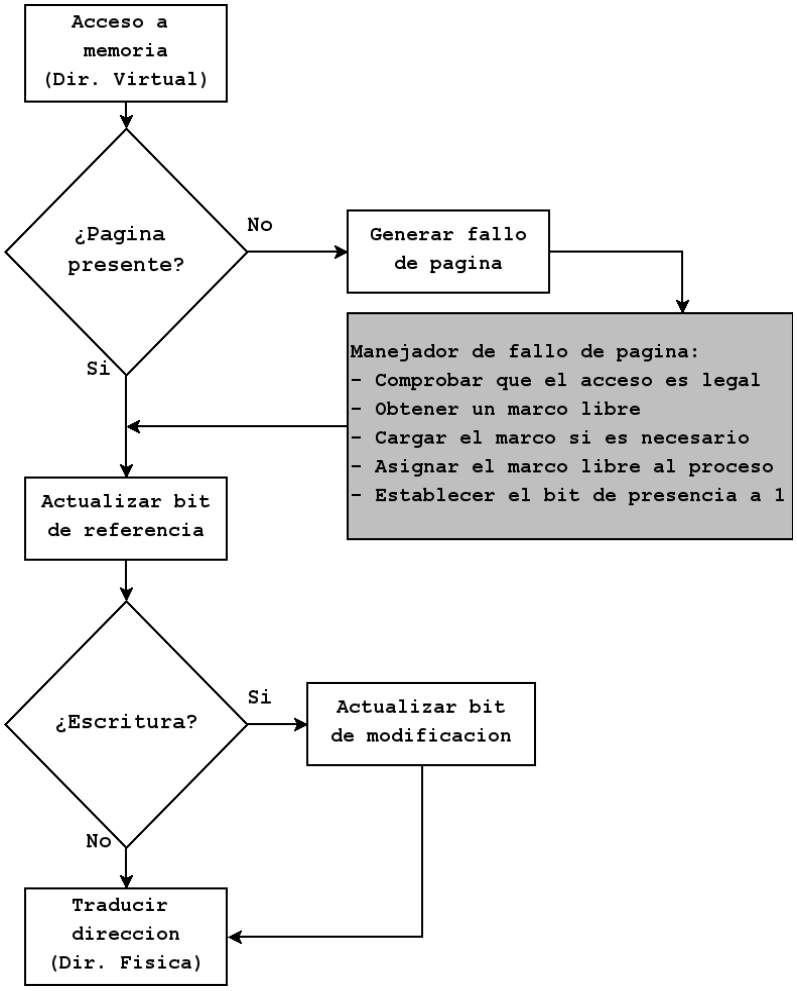
Lógicamente existirán múltiples posibilidades a la hora de implementar el sistema,

tal como se explica en teoría. Existen diferentes políticas para cada aspecto que se contempla en el anterior ejemplo y por lo tanto, múltiples decisiones de diseño. Son sistemas complejos que dependen de muchos parámetros y que al mismo tiempo condicionan en gran medida las prestaciones de un sistema.

En esta práctica nos centraremos en los algoritmos de reemplazo de páginas y en el funcionamiento básico de la MMU.

2.3 La MMU y la memoria virtual

El funcionamiento de la MMU es sencillo en lo que respecta al funcionamiento básico de la memoria virtual. Principalmente se limita a comprobar el bit de presencia y generar la excepción de fallo de página en caso de que éste esté a 0; cuando éste bit se encuentra a 1 (o bien el manejador de fallo de página resuelve la incidencia), la MMU realiza la traducción de dirección virtual a física y actualiza los bits de referencia y modificación.



La figura muestra las tareas que realiza esta MMU simplificada, indicando en gris las tareas que debería realizar el manejador de excepción de fallo de página. Como se puede observar, la mayor parte de la responsabilidad recae en dicho manejador, y el hardware se limita a tareas de detección, actualización de bits de referencia y modificación y traducción de direcciones.

Naturalmente, la MMU realiza otras tareas importantes en un sistema real, como es comprobar el nivel de privilegio a la hora de acceder a la página, comprobar que el acceso está permitido (p.e. sólo permitir una escritura si la página está marcada como escritura), etc. Pero para el estudio que nos ocupa, podemos obviar estos aspectos.

2.4 Algoritmos de reemplazo de páginas

Una de las situaciones más frecuentes que se pueden dar en un sistema con memoria virtual es que sea necesario conseguir un marco donde proyectar una página y no haya ninguno disponible en el sistema. En estos casos es necesario elegir un marco de los que ya están siendo utilizados (marco víctima) y liberarlo para poderlo usar. El criterio que se sigue para la elección del marco puede ser de lo más dispar, y forma parte de lo que se denomina *política de reemplazo de páginas*. En ella se consideran aspectos como:

- Si el marco víctima se debe elegir de entre los marcos asignados al proceso que causa el fallo de página (reemplazo local) o de entre todos los asignados en el sistema (reemplazo global).
- El algoritmo de reemplazo (FIFO, FIFO con segunda oportunidad, LRU, aleatorio, etc.)
- Si se emplea algún mecanismo de prepaginación, mantenimiento de un mínimo de marcos libres, escritura en segundo plano de páginas modificadas, etc.

Gran parte de estas tareas se realizan cuando se produce un fallo de página y el manejador debe obtener un marco libre.

Se sugiere al alumno que haga uso de los apuntes de teoría para obtener detalles sobre todos estos aspectos y especialmente algoritmos de reemplazo.

3 Simulación del subsistema de gestión de memoria

3.1 Introducción y estructura del sistema de pruebas

Con objeto de familiarizar al alumno con el funcionamiento de los mecanismos implicados en la implementación de la memoria virtual, se ha creado una aplicación que simula dichos mecanismos en un caso muy concreto y simplificado.

Esta aplicación simula los accesos a memoria en un **sistema paginado** con **páginas de 256 bytes**, con una cierta cantidad de RAM física que se puede especificar como parámetro en la línea de órdenes y un **espacio de direccionamiento virtual de 4096 bytes**. Los accesos a memoria que se deben simular provienen de un archivo de traza que el programa es capaz de leer y en el que se indica en un formato de texto sencillo la dirección virtual accedida y si el acceso se produce para leer o escribir.

El programa se invoca de la siguiente forma.

```
user@host:~/pracarqui5$ ./prac5 3 traza1.txt
```

donde el primer parámetro es el número de marcos de página físicos del sistema y el segundo el fichero de traza, que tiene el siguiente formato:

```
0x0701 read
0x0001 read
0x0101 read
0x0201 write
0x0001 read
0x0301 read
[...]
```

Para cada acceso a memoria virtual, la aplicación simula el funcionamiento de una MMU accediendo a la tabla de páginas y traduciendo la dirección virtual por la física, actualizando los bits de referencia y modificación. No se tienen en cuenta más aspectos que estos en la MMU.

En caso de que al realizar la traducción la página en cuestión no se encuentre presente, el simulador de MMU genera una excepción de fallo de página. El manejador de excepción está simulado llamando a una función, y se encarga de obtener un marco de página libre (invocando el algoritmo de reemplazo si no quedan marcos libres en el sistema) y de actualizar la tabla de páginas apropiadamente para que la aplicación pueda continuar realizando la traducción de direcciones.

Para simplificar el ejemplo consideraremos que los fallos de página siempre son provocados por acceso a páginas correctas pero no presentes en el sistema, que sólo existe un proceso, que todos los fallos de página provocan la carga de una página de disco (*page in*) y que la política de reemplazo es global.

Adicionalmente, al finalizar la traza el programa imprime por pantalla el estado final de la tabla de páginas y el estado interno del subsistema que gestiona el reemplazo de marcos de página. Por ejemplo, para un sistema con política de reemplazo FIFO, la invocación y la salida final es la que sigue para una traza igual a la expuesta en teoría para ilustrar el algoritmo FIFO:

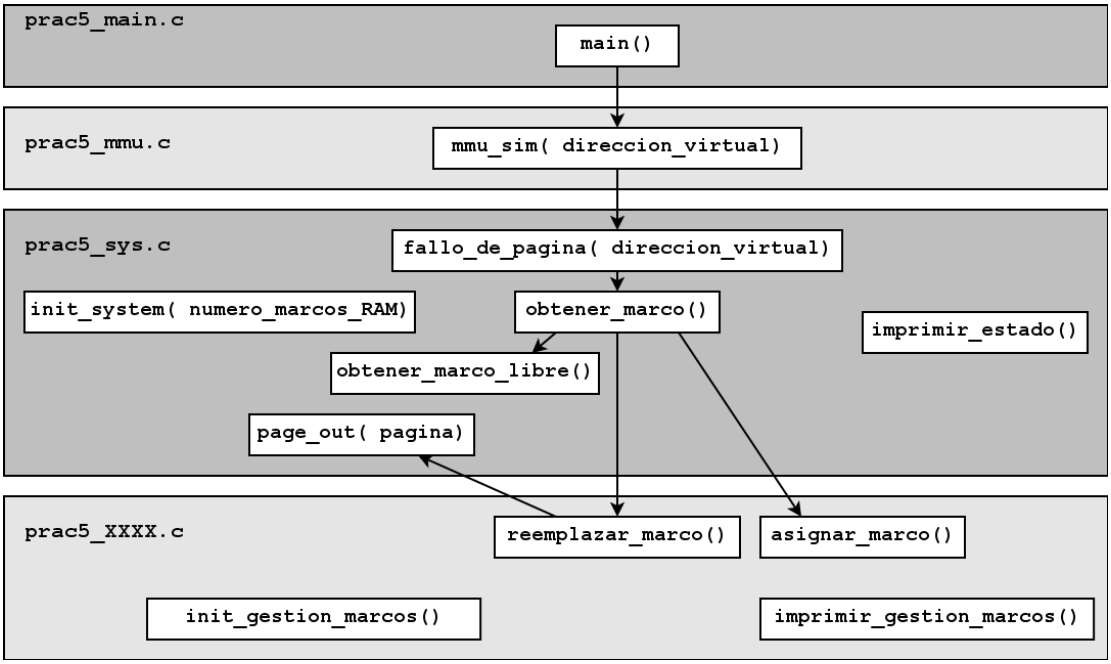
```
user@host:~/pracarqui5$ ./prac5_fifo 3 traza1.txt
        Dir. Virtual        Dir. Física
                0701                0001
                0001                0101
                0101                0201
                0201                0001
                0001                0101
                0301                0101
                0001                0201
                0401                0001
                0201                0101
                0301                0201
                0001                0001
                0301                0201
                0201                0101
                0101                0101
                0201                0201
```

```
               0001                0001
Estadisticas del sistema y estado de la tabla de páginas
------------------------------------------------------
Han ocurrido 12 fallos de página.
Ha sido necesario salvar a disco 0 páginas.

Contenido de la TDP:
    Pagina  Presente     Marco     Referencia     Modificado
         0         1         0              1              0
         1         1         1              1              0
         2         1         2              1              0
         3         0       N/A            N/A            N/A
         4         0       N/A            N/A            N/A
         5         0       N/A            N/A            N/A
         6         0       N/A            N/A            N/A
         7         0       N/A            N/A            N/A
         8         0       N/A            N/A            N/A
         9         0       N/A            N/A            N/A
        10         0       N/A            N/A            N/A
        11         0       N/A            N/A            N/A
        12         0       N/A            N/A            N/A
        13         0       N/A            N/A            N/A
        14         0       N/A            N/A            N/A
        15         0       N/A            N/A            N/A

Informe de gestión de marcos mediante algoritmo FIFO
Lista de marcos en uso:
-----------------------

Pos. en el FIFO     Num. Marco    Pagina     Referencia     Modificado
              1              0         0              1              0
              2              1         1              1              0
              3              2         2              1              0
- FIN DEL INFORME -
```

Dada la relativa complejidad, el sistema se entrega al alumno parcialmente implementado, y la tarea a realizar será completar dicho sistema y probar su correcto funcionamiento con las trazas que se proveen.

Para aclarar el funcionamiento global del sistema, las funciones implementadas y la distribución en los diferentes archivos fuente se proporciona el siguiente esquema de la aplicación con las funciones principales que el alumno debe conocer. En color más oscuro se encuentran los archivos fuente cuyas funciones se proporcionan, y en color más claro aquello que el alumno deberá implementar. Se indica también la relación entre las diferentes funciones del sistema de gestión de memoria para

aclarar el funcionamiento. Las funciones de inicialización e impresión serán invocadas desde el programa principal, pero no se indica así para mayor claridad.



Como puede observar, será necesario implementar básicamente dos cosas: La MMU simulada y las funciones que implementan la política de reemplazo, cuyo cometido se aclara a continuación. Dado que se pide al alumno más de una política de reemplazo, el fichero fuente relacionado se indica con `prac5_XXXX.c`. En realidad se deberá generar un archivo fuente para cada política de reemplazo solicitada (p.e. `prac5_fifo.c` y `prac5_fifo2op.c`) y en consecuencia se generarán ejecutables diferentes según se enlace el programa con unas u otras funciones (p.e. `prac5_fifo` y `prac5_fifo2op`).

3.2 Detalles de implementación: Estructuras de datos

A pesar de su aparente complejidad, el sistema anterior es bastante simple y sólo requiere la comprensión de un par de estructuras de datos: La tabla de páginas y la tabla de descriptores de marco de página. La primera se explica en clase de teoría, y a continuación se expone la utilidad de la segunda.

La tabla de páginas (TDP) es una estructura con la que el alumno ya debe estar familiarizado. La TDP es empleada por la MMU para realizar la traducción de direcciones y el formato de las entradas está predefinido en cada arquitectura. En nuestro caso, las entradas de la TDP tienen el siguiente formato:

```
struct tdp_entry {
        int marco;
        char presencia;
        char referencia;
        char modificado;
};
```

Los campos de cada entrada son autoexplicativos. Simplemente reseñar que los flags de presencia, referencia y modificación están modelados con un campo de tipo `char` en lugar de bits individuales para simplificar su uso. La TDP en sí se crea de forma dinámica automáticamente, y el alumno debe accederla a través de la siguiente declaración definida en `prac5_sys.h`.

```
extern struct   tdp_entry * tdp;
```

El alumno puede asumir que este puntero ya se encuentra apuntando a un array de estructuras que conforman la TDP; no debe realizar reserva de memoria dinámica. De esta forma, el acceso y modificación de la TDP es tan simple como en el siguiente ejemplo:

```
tdp[ num_pagina].referencia = 1;
```

La tabla de descriptores de marco de página es una estructura de datos del sistema operativo, y de ella la MMU no tiene conocimiento alguno. Es una tabla en la que cada entrada (descriptor) contiene información útil para el sistema operativo sobre cada marco de página del sistema, de manera similar a cómo la TDP tiene información sobre las páginas. En nuestro caso, el descriptor va a indicar simplemente la página proyectada sobre el marco correspondiente al índice de la entrada en cuestión, y va a contener también un campo útil para formar listas enlazadas según la necesidad; por ejemplo este campo podría servir para formar una lista de marcos libres en el sistema, o para implementar algoritmos de reemplazo como FIFO. El descriptor y la tabla de descriptores están declarados de la siguiente forma:

```
struct mdp_desc {
      int pagina;
      struct mdp_desc * next;
};


extern struct   mdp_desc * mdp_table;
```

El alumno necesitará usar dicha tabla para implementar los algoritmos de reemplazo solicitados, y al igual que antes debe suponer que dicha tabla ya está apuntada por el puntero `mdp_table`, y que contiene tantos descriptores como marcos haya en el sistema.

3.3 Implementación de la política de reemplazo

Para implementar la política de reemplazo normalmente hace falta una estructura de datos (por ejemplo, una lista) donde tener constancia de qué marcos pertenecen al proceso. Esta estructura es construida y gestionada por mediante unas funciones, típicamente una para añadir marcos a dicha estructura y otra para descartarlos.

Por ejemplo para implementar una política FIFO lo natural será mantener los marcos que pertenecen al proceso (o mejor dicho, sus descriptores) en una lista enlazada, y para construirla será necesario al menos un puntero al primer elemento del FIFO y a continuación emplear el campo `next` de los descriptores para construir la lista. También podríamos mejorar la implementación utilizando elementos opcionales en este esquema, como un puntero al último elemento; o bien implementar el sistema de otra forma, como queramos. En última instancia, dependerá totalmente de quien implementa el sistema.

Por otra parte la política de reemplazo se aplicará casi siempre de una forma básicamente parecida. En el caso que nos ocupa existen básicamente dos funciones: `reemplazar_marco()` y `asignar_marco()`. La primera descarta un marco del subsistema según la política de reemplazo que se está implementando, y la segunda añade un marco al subsistema. Observe que si bien lo que hace cada función dependerá de la política de reemplazo concreta, el efecto en ambos casos es siempre el mismo; añadir o quitar un marco de la estructura de datos que internamente esté empleando el software para llevar a cabo la implementación de dicha política. Por ejemplo, en el caso de FIFO, `reemplazar_marco()` extraería el primer marco del FIFO, y `asignar_marco()` añadiría al final del FIFO un nuevo marco, especificado mediante un parámetro.

Para otras políticas de reemplazo puede ser necesario utilizar estructuras de datos más complejas o algoritmos más complicados para las anteriores funciones de acceso; pero todo ello queda encapsulado y transparente para el resto del sistema, que sólo emplea estas dos funciones.

3.4 Detalles de implementación: Funciones

Se relacionan a continuación las funciones que componen el sistema y funcionamiento aproximado de cada una de ellas, como guía para el alumno en la realización de la práctica. Puede encontrar más información en los archivos de cabecera proporcionados.

- **`main()`**

  El programa principal recibe como argumentos el número de páginas de memoria RAM que tendrá disponible el sistema y el nombre del archivo de traza a emplear para las pruebas.

  Tras llamar a las funciones de inicialización `init_system()` e `init_gestion_marcos()`, comienza a generar accesos a memoria según el archivo de traza, realizando sucesivas llamadas a `mmu_sim()` e imprimiendo la traducción entre direcciones virtuales y físicas. Una vez finalizada la traza, llama a `imprimir_estado()` e `imprimir_gestion_marcos()` para poder observar las estadísticas de la ejecución de la traza y el estado final de las tablas.

- **`mmu_sim( dir_virtual, operación)`** (Función a implementar)

  Esta función simula el comportamiento de la MMU. Obtiene el número de página y desplazamiento a partir de la dirección virtual, obtiene el marco de página a partir de la TDP y el número de página, y devuelve la dirección física traducida. Además actualizará los flags de referencia y modificación según la operación sea de lectura (`'r'`) o escritura (`'w'`), y en caso de que el bit de presencia esté a 0 generará una llamada a `fallo_de_pagina()` antes de realizar ni la traducción ni la modificación de los flags.

- **`fallo_de_pagina( dir_virtual)`**

  Esta función es invocada cuando la MMU va a realizar una traducción y la página no está presente. Para simplificar supondremos que todo fallo de página implica obtener un marco de página libre mediante llamada a `obtener_marco()` y cargar el contenido adecuado. A continuación, la función actualiza la entrada de la TDP correspondiente y el descriptor de marco de página asociado antes de devolver el control a la MMU.

- **obtener_marco()**

  Esta función consigue un marco libre y lo asigna al proceso, para que el manejador de fallo de página pueda a continuación cargar su contenido desde el disco y actualizar las tablas de páginas y de descriptores de marco.

  Inicialmente esta función llama a la función del sistema `obtener_marco_libre()`, por si el sistema tuviera aún marcos disponibles. Si no es así, es necesario descartar un marco ya asignado, para lo cual llama a `reemplazar_marco()`, que invoca al algoritmo de reemplazo para liberar un marco (el marco deja de pertenecer al proceso). En cualquiera de los dos casos, a continuación llama a `asignar_marco()`, para que el marco obtenido anteriormente sea asignado al proceso y por lo tanto, administrado por el subsistema de gestión de marcos usados.

  Es necesario observar que en nuestro caso concreto, si no hay marcos libres el proceso perderá un marco que a continuación se le va a asignar de nuevo; pero se deben programar las funciones `reemplazar_marco()` y `asignar_marco()` sin pensar en que esto va a suceder siempre así.

- **obtener_marco_libre()**

  Esta función obtiene un marco nunca usado anteriormente, de la lista de marcos libres del sistema. Inicialmente dicha lista contendrá todos los marcos, pero una vez se hayan hecho tantas solicitudes como marcos hay, la función devolverá siempre error ya que no se pueden asignar mas f marcos.

  La gestión de marcos libres la realiza íntegramente el sistema, y el alumno no tiene que hacer nada a este respecto.

- **reemplazar_marco()** (Función a implementar)

  Esta función es una de las que implementa la política de reemplazo del sistema. Debe seleccionar y liberar uno de los marcos asignados empleando como criterio el algoritmo concreto (p.e. *FIFO* o *FIFO con segunda oportunidad*). El marco seleccionado será eliminado de dicho subsistema y sus contenidos deben ser almacenados en disco si la página ha sido modificada, mediante una llamada a la función `page_out`().

  La función devuelve el número del marco que ha quedado liberado tras el reemplazo, y debe actualizar las tablas de la manera que sea adecuada.

- **asignar_marco(marco)** (Función a implementar)

  Esta segunda función añadirá el marco en cuestión al subsistema de gestión de marcos, según la política de reemplazo concreta. Por ejemplo en el caso de FIFO, añadirá el marco al final del FIFO.

- **page_out(página)**

  Esta función, en un sistema real, guardaría el contenido de un marco de página en disco. En nuestro sistema sirve simplemente para mantener una estadística de cuántos marcos se habrían de llevar a disco para una determinada traza y algoritmo de reemplazo.

- **init_system(num_marcos)**

  Esta función crea e inicializa la TDP y la tabla de descriptores de marco de página. Además, crea la lista de marcos libres del sistema. Es invocada una

única vez desde la función `main()`.

- **`init_gestion_marcos()`** (Función a implementar)

  Esta función es llamada una vez desde `main()` para inicializar el algoritmo de reemplazo. Puede ser usada por el alumno para inicializar listas, dar valores iniciales a variables o para nada; lo que estime oportuno.

- **`imprimir_estado()`**

  Esta función se invoca tras la ejecución de la traza y muestra por pantalla el estado final de la TDP, cuántos fallos de página y cuántas escrituras en disco se hubieran producido. Puede ver un ejemplo de la salida que produce en un apartado anterior del enunciado de la práctica.

- **`imprimir_gestion_marcos()`** (Función a implementar)

  Esta función imprime el estado actual del subsistema de gestión de marcos usados que emplea el algoritmo de reemplazo (por ejemplo, los contenidos del FIFO en el caso de dicho algoritmo de reemplazo). Es invocada por main() al menos una vez al final de la ejecución de la traza, pero el alumno puede invocarla desde donde estime oportuno por ejemplo para realizar depuración. El formato de salida debe ajustarse estrictamente al del ejemplo ilustrado en un apartado anterior.

4 Tareas a realizar

4.1 Introducción

Esta práctica va a servir para ayudar en la comprensión de los mecanismos involucrados en la paginación y los sistemas con memoria virtual.

Para ello se ha construido un sistema que simula el funcionamiento de los mecanismos básicos de gestión de memoria, tanto software como hardware, de un computador y un sistema operativo muy sencillos.

Se proveerá al alumno de un software al que le faltan por implementar algunas funciones y de unas trazas de acceso a memoria para realizar pruebas que permitan verificar la correcta implementación de las mismas.

4.2 Desarrollo y realización de la práctica

La práctica consistirá en la implementación de:

- La función **`mmu_sim()`**, que simula el comportamiento de una MMU sencilla. El código residirá en el archivo `mmu_sim.c`.
- Los **algoritmos de reemplazo FIFO y FIFO con segunda oportunidad**. El código de cada uno de ellos residirá respectivamente en los archivos `prac5_fifo.c` y `prac5_fifo2op.c`, y los prototipos de función se ajustarán al archivo de cabecera proporcionado `prac5_reemplazo.h`.

El resto del código ya implementado se proporciona en forma de una librería `libprac5.a` y un archivo objeto para el programa principal, `prac5_main.o`. Se proporcionan además todos los archivos de cabecera necesarios y dos trazas de

prueba.

Se creará un `Makefile` adecuado para compilar los archivos fuente y que sea capaz de generar dos ejecutables, uno por cada algoritmo de reemplazo, llamados **`prac5_fifo`** y **`prac5_fifo2op`**.

Finalmente, se probarán ambos programas con las trazas proporcionadas, `traza1.txt` y `traza2,txt`. La primera es una traza sencilla, igual a la empleada en el ejemplo de clase de teoría para ilustrar los algoritmos de reemplazo, y que permitirá al alumno comprobar el buen funcionamiento del sistema ya que se deben obtener los mismos resultados que en el ejemplo de teoría.

La segunda traza es un poco más compleja y permite evaluar el comportamiento de los algoritmos. Los resultados que se deben obtener para la ejecución con 4 páginas de memoria física son:

- FIFO: 18 fallos de página, 5 páginas escritas en disco.
- FIFO con segunda oportunidad: 15 fallos de página, 3 páginas escritas en disco.

Debe recordar que para enlazar la biblioteca `libprac5.a` será necesario emplear el parámetro **`-lprac5`** y probablemente **`-L.`** para incluir el directorio de trabajo en la ruta de búsqueda de bibliotecas. **Es obligatoria la realización de un Makefile que construya automáticamente ambos ejecutables**.

Resumen de archivos de la práctica:

– `Makefile`	Archivo a realizar por el alumno.
– `prac5_main.o`	Objeto que contiene la función main().
– `prac5_sys.h`	Archivo de cabecera del resto de funciones proporcionadas.
– `libprac5.a`	Biblioteca con las funciones proporcionadas, salvo main().
– `prac5_mmu.h`	Archivo de cabecera con el prototipo de las funciones del archivo fuente prac5_mmu.c que el alumno debe implementar.
– `prac5_reemplazo.h`	Archivo de cabecera con el prototipo de las funciones que el alumno debe implementar tanto en prac5_fifo.c como en prac5_fifo2op.c.
– `prac5_mmu.c`	Archivo a realizar por el alumno. Incluirá el código necesario para implementar la MMU simulada.
– `prac5_fifo.c`	Archivo a realizar por el alumno. Incluirá el código necesario para implementar el algoritmo de reemplazo FIFO.
– `prac5_fifo2op.c`	Archivo a realizar por el alumno. Incluirá el código necesario para implementar el algoritmo de reemplazo FIFO con segunda oportunidad.
– `trazaN.txt`	Trazas de memoria de prueba.

5 Cuestiones

- Pruebe el comportamiento de los algoritmos para diferentes cantidades de memoria RAM.
- Plantée cuales serían las necesidades para implementar otros algoritmos, como LRU, reloj global, etc.
- OPCIONAL: Implemente algún otro de los algoritmos de reemplazo explicados en clase y explique los detalles de la implementación durante la defensa de la práctica. Se valorará positivamente la realización de este apartado.