Sistemas Operativos

Práctica 3

Gestión de Memoria

Equipo "Sistemas Operativos DISCA/DSIC"

Universidad Politécnica de Valencia

INTRODUCCIÓN

El objetivo de esta práctica consiste en construir un programa que simule el funcionamiento de un sistema informático en el cual se utiliza la técnica de memoria virtual bajo paginación por demanda. La simulación consistirá en obtener la serie de direcciones físicas correspondientes a una serie de direcciones lógicas emitidas por un conjunto de procesos. El simulador incorporará las funciones propias de la unidad de gestión de memoria así como las funciones del sistema operativo relacionadas con la administración de memoria. Se obtendrán las características del sistema y una serie de referencias lógicas a partir de varios ficheros de entrada. A partir de esta serie de referencias lógicas, se aplicará el mecanismo de traducción de direcciones propio de la paginación por demanda, teniendo en cuenta cuestiones como el tratamiento de fallos de página y la aplicación de algoritmos de reemplazo.

Para la realización de esta práctica se proporciona un fichero de código (**`Gestor_de_memoria.java`**) en lenguaje JAVA, en el que faltan por rellenar ciertas partes, a implementar por el alumno. En los siguientes apartados se plantean los requerimientos del programa, así como las decisiones que se han tomado para llevar a cabo el código que se propone en la práctica.

REQUERIMIENTOS

En este apartado se describe la forma en que se presentan los datos de entrada al programa simulador y la forma en que quedarán los datos de salida.

Datos de entrada

Los datos de entrada se obtienen de cuatro ficheros:

SISTEMA	Descripción del sistema
TAMPROC	Tamaño de los procesos
DIRLOG	Serie de direcciones de entrada (direcciones lógicas)
PROC	Serie de identificadores de proceso

Estos ficheros son ficheros de texto, de forma que su contenido puede ser creado con cualquier editor de textos o a partir de la ejecución de un *script*. Esto facilita la ejecución de pruebas con diferentes instancias de datos.

A continuación se expone el formato que presenta cada uno de los ficheros.

FICHERO	FORMATO	DESCRIPCION	EJEMPLO
SISTEMA	p m e A	Este fichero contendrá cuatro líneas. La primera contendrá un valor **p** tal que $p = \log_2$ Tamaño_de_página La segunda contendrá un valor **m** tal que $m = \log_2$ Número_de_marcos La tercera contendrá un valor **e** tal que $e = \log_2$ Tamaño_del_espacio_lógico La cuarta contendrá un valor numérico **A** que indicará el algoritmo de reemplazo a utilizar: 0 para el algoritmo LRU global, 1 para el algoritmo LRU local, 2 para el algoritmo LFU global, … En el ejemplo, el sistema descrito utiliza páginas de 2^{10} = 1024 bytes, dispone de 2^2 = 4 marcos, el espacio lógico para cada proceso es de 2^{16} = 65536 bytes y el algoritmo de reemplazo a utilizar es el LFU global.	10 2 16 2

FICHERO	FORMATO	DESCRIPCION	EJEMPLO
TAMPROC	T_0 T_1 T_2 ... ... T_{n-1}	Este fichero contendrá **n** líneas, tantas como procesos intervengan en la simulación. Cada línea contendrá un valor T_i que corresponderá al tamaño en bytes del proceso **i**. Los procesos se nombrarán mediante los números **0** a **n-1**. En el ejemplo, en la simulación intervendrán cuatro procesos: 0, 1, 2 y 3. El tamaño del proceso 0 será 5000, el del 1 será 8000, el del 2 será 60000 y el del 3 será 2000.	5000 8000 60000 2000

FICHERO	FORMATO	DESCRIPCION	EJEMPLO
DIRLOG	D_0 D_1 D_2 ... ... D_{k-1}	Este fichero contendrá **k** líneas, tantas como referencias vayan a ser generadas en la simulación. Cada valor D_i será una referencia al espacio lógico de un proceso, siendo este proceso el valor P_i en la misma línea del fichero PROC descrito a continuación. En el ejemplo, se generarán un total de 12 referencias a memoria, siendo la primera dirección la número 5000 dentro del espacio lógico del proceso 0 (ver el ejemplo de la tabla siguiente).	5000 2048 2148 0 7999 50000 100 2248 1024 0 4096 1024

FICHERO	FORMATO	DESCRIPCION	EJEMPLO
PROC	P_0 P_1 P_2 ... ... P_{k-1}	Este fichero contendrá, al igual que el fichero DIRLOG, **k** líneas, tantas como referencias vayan a ser generadas en la simulación. Cada valor P_i corresponde al identificador del proceso que emite la i-ésima dirección. La dirección correspondiente será el valor D_i del fichero DIRLOG descrito anteriormente. En el ejemplo, se generarán un total de 12 referencias a memoria, siendo la última dirección la número 1024 dentro del espacio lógico del proceso 2 (ver el ejemplo de la tabla anterior).	0 1 1 1 1 2 1 1 1 2 2 2

Datos de salida

Como resultado de la simulación, se obtendrán dos ficheros de salida:

DIRFIS	Serie de direcciones físicas calculadas por el simulador
FALLOS	Indicación de los fallos de página producidos

En un primer fichero se tendrán las direcciones físicas que correspondan a las direcciones lógicas emitidas. Este fichero se denominará DIRFIS y su formato es el siguiente:

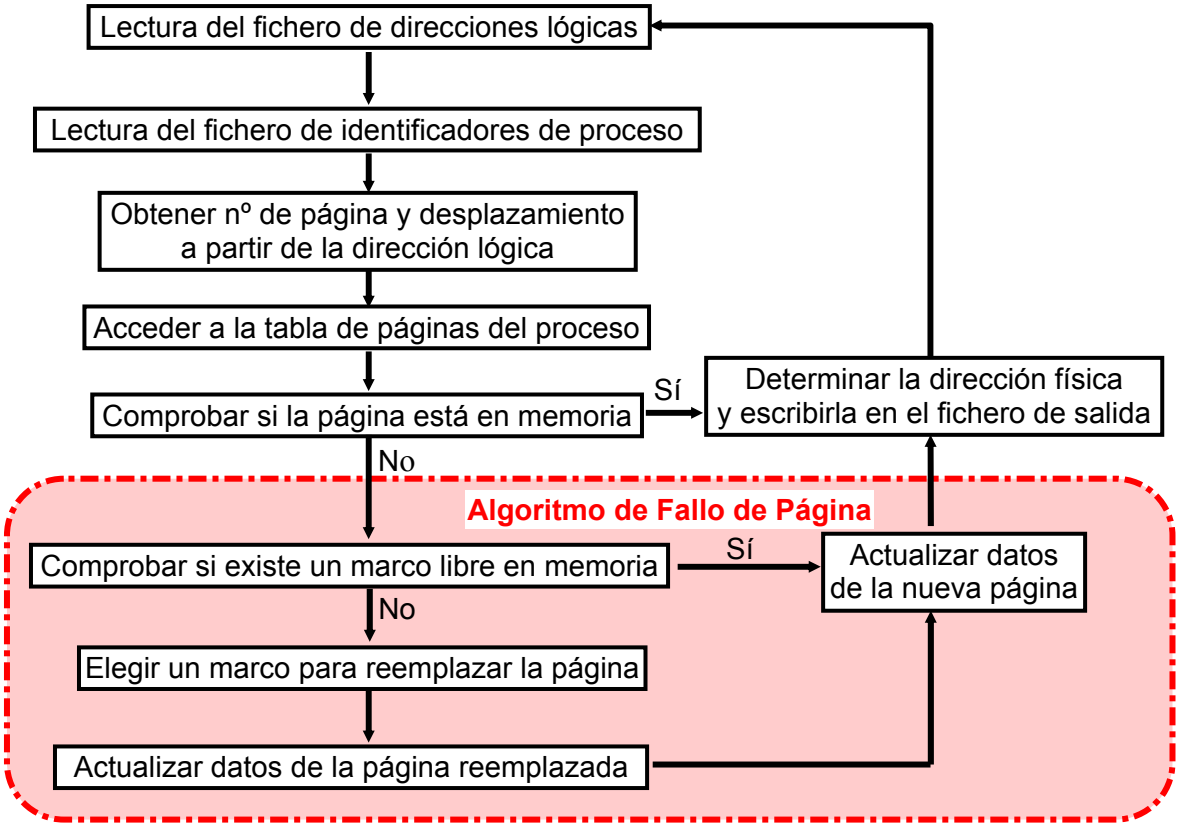
FICHERO	FORMATO	DESCRIPCION	EJEMPLO
DIRFIS	F_0 F_1 F_2 ... ... F_{k-1}	Este fichero contendrá, al igual que los ficheros DIRLOG y PROC, **k** líneas, tantas como referencias hayan sido generadas en la simulación. Cada valor F_i representará la dirección física correspondiente a la dirección lógica D_i emitida por el proceso P_i. Si una dirección es errónea, el valor que se almacenará será la palabra "ERROR". En el ejemplo se muestra el contenido del fichero DIRFIS una vez finalizada la simulación del modelo descrito en los ficheros de entrada anteriores.	904 1024 1124 2048 3903 848 2148 1224 0 0 0 0

En el segundo fichero de salida se marcarán los accesos que han producido fallo de página. El fichero se denominará FALLOS y su formato es:

FICHERO	FORMATO	DESCRIPCION	EJEMPLO
FALLOS	V_0 V_1 V_2 ... ... V_{k-1}	Este fichero contendrá, al igual que los ficheros DIRLOG y PROC, **k** líneas, tantas como referencias hayan sido generadas en la simulación. Cada valor V_i será bien una "F" indicando que en la referencia i-ésima se ha producido un fallo de página, bien "ERROR" si se ha producido una referencia errónea, o bien una cadena vacía "" indicando que no se ha producido fallo de página ni error. En el ejemplo se muestra el contenido del fichero FALLOS una vez finalizada la simulación del modelo descrito en los ficheros de entrada anteriores.	F F F F F F F F F

ACERCA DE LA IMPLEMENTACIÓN PROPUESTA

El siguiente esquema representa la parte central del algoritmo propuesto para el programa de simulación. Se corresponde con el proceso necesario para tratar una referencia a memoria. Por tanto, todo este proceso se repetirá tantas veces como referencias haya en los ficheros de entrada.



Las principales **estructuras de datos** que se utilizan en la implementación son:

- **Tabla de páginas**: Para cada proceso se guarda una tabla de páginas con información referente a todas sus páginas. Se implementa como un vector de objetos **`Clase_pagina`**. Para cada página se guarda una indicación de si es válida o no y, en el caso de que sea válida, el número de marco en memoria donde está la página.

```
// Clase para las paginas de procesos
static class Clase_pagina {
    int bit_validez, marco;
    ...
}
```

- **Tabla de procesos**: Los procesos se representan mediante un vector de objetos **`Clase_proceso`**. Para cada proceso se guarda información sobre su tamaño, su número de páginas y la tabla de páginas con información para todas sus páginas.

```
// Clase para la informacion de procesos
static class Clase_proceso {
    Clase_pagina[] Tabla_paginas;
    ...
    int talla() { ... }
    int Ultima_pagina() { ... }
    Clase_proceso(int v_talla) { ... }
}
static Clase_proceso[] Tabla_procesos;
static int Ultimo_proceso;
```

- **Tabla de marcos**: La memoria se representa como un vector de objetos **`Clase_marco`**. Para cada marco se guarda información sobre el proceso que lo está ocupando, el número de la página de este proceso que hay en el marco y el contador, útil para implementar los algoritmos de reemplazo.

```
// Clase para la informacion de marcos
static class Clase_marco {
    int pid, pagina, contador;
    ...
}

// Implementacion de la memoria
static Clase_marco[] Memoria;
static int Ultimo_marco;
```

Además de estas estructuras de datos, en el programa se utilizan estas funciones:

- Función auxiliar para elevar un número a otro:

```
static int Elevar_a( int a, int b );
```

- Función auxiliar para mostrar por pantalla el contenido de la memoria:

```
static void Escribe_memoria();
```

- Función auxiliar para mostrar por pantalla los datos de la tabla de procesos:

```
static void Escribe_Tabla_procesos();
```

- Función que encuentra un marco de memoria víctima según una política LFU global (a implementar):

```
static int Aplica_LFU_GLOBAL();
```

- Función principal, donde está implementado el algoritmo central del programa de simulación. Tan sólo faltan algunos fragmentos que deben desarrollarse en la práctica.

MÉTODO DE TRABAJO

La forma de proceder consiste en ir modificando el código fuente del programa utilizando un editor de textos cualquiera (emacs, kedit, vi, ...) y compilarlo desde la línea de comandos con la orden siguiente:

```
javac Gestor_de_memoria.java
```

El programa ya compilado se ejecuta mediante el comando:

```
java Gestor_de_memoria
```

Se deberá completar el programa para que realice la simulación completa, según el algoritmo de paginación especificado en los ejercicios correspondientes, que se presentan más adelante.

UTILIZACIÓN DE LOS TESTS AUTOMÁTICOS

Para la verificación del programa desarrollado, se suministran un *shell script* llamado **prueba** y un conjunto de casos de prueba. La forma de utilizar el *script* **prueba** es la siguiente:

```
$ prueba
Uso: prueba nombre_programa [numeros_de_casos]
```

El *script* **prueba** recibe como argumentos el nombre de la clase principal de JAVA desarrollada con el programa y los números de los casos a probar. Si no se especifica ningún número de caso, se probarán todos.

El *script* ejecutará el programa desarrollado para cada uno de los casos especificados, comparando al final el contenido de los ficheros DIRFIS y FALLOS generados, con la solución que se encuentra en la carpeta de cada caso.

La salida del *script* consiste en una lista donde se indica para cada caso probado si los ficheros DIRFIS y FALLOS se corresponden con lo esperado (**Correcto**) o no (**ERROR**).

A continuación se muestra un ejemplo de la invocación del *script* **prueba** para comprobar el caso 1 (suponiendo que la clase principal del programa se llama **Gestor_de_memoria**):

```
$ prueba Gestor_de_memoria 1

-------------------- Probando caso1

Fichero DIRFIS: Correcto
Fichero FALLOS: Correcto
```

Al final de la ejecución, se mantienen en el directorio los ficheros DIRFIS y FALLOS del último caso probado. Si algún caso no funciona bien, conviene repetir la ejecución del *script* sólo con ese caso, para que se queden sus ficheros de salida. Entonces puede compararse su contenido con el correspondiente a los ficheros solución de la carpeta de ese caso (bien sea viéndolos con **cat** o directamente con el comando **diff**). Si, por ejemplo, se quiere comparar el fichero DIRFIS generado por el programa y el de la solución para el caso 1, podría utilizarse el comando:

```
$ diff DIRFIS caso1/
```

Con esto puede verse dónde se producen diferencias. Entonces convendrá ejecutar el programa, prestando atención a su salida estándar para observar cómo se comporta en las inmediaciones del acceso a memoria que ha sido erróneo.

Con este *script* y los casos resueltos se pretende facilitar la tarea de depurar el código fuente, ya que habrá que ir modificándolo hasta comprobar que todos los casos de prueba funcionan bien.

EJERCICIOS

1. Complétese el código en JAVA que se proporciona (puede verse en las siguientes páginas) para realizar un simulador de un *Gestor de Memoria Virtual basado en Paginación bajo Demanda*. Deberá comprobarse con los ejemplos suministrados que el código funciona correctamente. Los lugares donde falta completar código están marcados con comentarios de la forma **`xx@@@`**, siendo **`xx`** un número. A continuación se resume lo que hay que hacer en cada uno de ellos:

   1. **`Inicialización de memoria y procesado de dir. lógica`**

      `01@@@`: Inicializar el campo contador al inicializar la memoria

      `02@@@`: Calcular número de página y desplazamiento a partir de dirección lógica

      `03@@@`: Condición para ver si la página a la que se hace referencia está dentro del rango válido de páginas para ese proceso

   2. **`Actualización datos de página y cálculo dirección física`**

      `04@@@`: Actualizar los datos en memoria del marco utilizado

      `05@@@`: Registrar el marco en la tabla de páginas del proceso

      `06@@@`: Calcular la dirección física

   3. **`Búsqueda de algún marco libre en memoria`**

      `07@@@`: Buscar un marco libre en memoria

      `08@@@`: Condición para ver si se ha encontrado un marco libre en memoria

   4. **`Comprobación fallo de página`**

      `09@@@`: Condición para ver si la página deseada no está en memoria

   5. **`Procesado fallo de página`**

      `10@@@:` Buscar el marco con menor valor del contador y devolver ese marco (algoritmo de reemplazo LFU global)

      `11@@@`: Si ese marco estaba ocupado, "desocuparlo"

      `12@@@`: Actualizar el valor del contador para el marco usado, poniéndolo a cero si se acaba de traer a memoria o incrementándolo si ya estaba en memoria (esto es así para el algoritmo de reemplazo LFU global)

Nota: Se aconseja compilar y ejecutar antes de empezar este ejercicio y también tras la implementación de cada uno de estos 5 puntos, observando que lo que el programa muestra por pantalla se corresponde con lo implementado hasta el momento.

2. ¿Cuáles son las modificaciones mínimas necesarias para utilizar un **algoritmo de reemplazo FIFO** en lugar del algoritmo implementado? Sugerencia: pensar la forma

de realizar un algoritmo FIFO a partir de un algoritmo LFU global implementado mediante contadores.

3. ¿Cuáles son las modificaciones mínimas necesarias para utilizar un **algoritmo de reemplazo LRU global** en lugar del algoritmo implementado? Sugerencia: pensar la forma de realizar un algoritmo LRU global a partir de un algoritmo LFU global implementado mediante contadores.

CÓDIGO PROPUESTO PARA LA REALIZACIÓN DE LA PRÁCTICA

El código del programa a completar es el siguiente:

```
//
// Gestor_de_memoria
//

import nsIO.*;

class Gestor_de_memoria {
    static final int
        MAX_PROCESOS = 8,     // Maximo numero de procesos
        NADA         = -1,
        LRU_GLOBAL   = 0,
        LRU_LOCAL    = 1,
        LFU_GLOBAL   = 2,
        INVALIDA     = 0,
        VALIDA       = 1;

    // Clase para las paginas de procesos
    static class Clase_pagina {
        int bit_validez, marco;
        Clase_pagina() {
            bit_validez = INVALIDA;
            marco = NADA;
        }
    }

    // Clase para la informacion de procesos
    static class Clase_proceso {
        private int m_talla;
        Clase_pagina[] Tabla_paginas;
        private int m_ultima_pagina;

        int talla() {
            return m_talla;
        }
        void talla(int talla) {
            int i;
            m_talla = talla;
            m_ultima_pagina = ( talla + Tamanyo_pagina - 1 )/Tamanyo_pagina - 1;
            Tabla_paginas = new Clase_pagina[ m_ultima_pagina + 1 ];
            for ( i=0 ; i<=m_ultima_pagina ; i++ )
                Tabla_paginas[i] = new Clase_pagina();
        }
        int Ultima_pagina() {
            return m_ultima_pagina;
        }
        Clase_proceso(int v_talla) {
            talla(v_talla);
        }
    }
```

```
// Clase para la informacion de marcos
static class Clase_marco {
    int pid, pagina, contador;

    Clase_marco() {
        // Hay que inicializar la memoria
        pid = NADA;
        pagina = NADA;
        // 01@@@ Inicializar el campo contador
    }
}

// Implementacion de la tabla de procesos
static Clase_proceso[] Tabla_procesos;
static int Ultimo_proceso;

// Implementacion de la memoria
static Clase_marco[] Memoria;
static int Ultimo_marco;

// Datos iniciales calculados
static int Tamanyo_pagina;

// FUNCIONES
// Para elevar un numero a otro
static int Elevar_a( int a, int b ) {
    int i, resultado;
    if ( a != 2 ) {
        resultado = 1;
        for ( i = 1 ; i <= b ; i++ ) resultado *= a;
    } else
        resultado = 1 << b ;
    return resultado;
}

// Procedimiento auxiliar para imprimir la memoria
static void Escribe_memoria() {
     int i;
     output pantalla = new output();
     format f3 = new format(3);
     pantalla.writeln( "Contenido de la MEMORIA:" );
     pantalla.writeln( "   POSICION    PID    PAGINA    CONTADOR" );
     pantalla.writeln( "   ========    ===    ======    ========" );
     for ( i=0 ; i<=Ultimo_marco ; i++)
         pantalla.writeln( "      " + f3.fmt(i) + "       " +
          f3.fmt(Memoria[i].pid)+"       "+f3.fmt(Memoria[i].pagina)+
          "         " + f3.fmt(Memoria[i].contador) );
} // Escribe_memoria

// Procedimiento auxiliar para imprimir la tabla de procesos
static void Escribe_Tabla_procesos() {
    int i, j;
    output pantalla = new output();
    format f4 = new format(4), f6 = new format(6);

    pantalla.writeln( "Contenido de la TABLA de PROCESOS: " );
    for ( i=0 ; i<=Ultimo_proceso ; i++ ) {
        pantalla.writeln( "  Proceso " + i );
        pantalla.writeln( "   Talla           = " +
          f6.fmt(Tabla_procesos[i].talla()) );
        pantalla.writeln( "   Numero de paginas = " +
         f4.fmt(Tabla_procesos[i].Ultima_pagina()+1));
        pantalla.writeln( "   Tabla de paginas: " );
        for ( j=0 ; j<=Tabla_procesos[i].Ultima_pagina() ; j++ )
            if ( Tabla_procesos[i].Tabla_paginas[j].bit_validez == VALIDA )
                pantalla.writeln( j + " " +
                  Tabla_procesos[i].Tabla_paginas[j].marco );
```

```
        pantalla.writeln();
    }
} // Escribe_Tabla_procesos

// Funcion que encuentra una victima segun una politica LFU global
static int Aplica_LFU_GLOBAL() {

    // 10@@@ Buscar el marco con menor valor del campo contador

    return 0; // 10@@@ Devolver el marco encontrado
} // Aplica_LFU_GLOBAL

// EL PROGRAMA PRINCIPAL
public static void main( String args[] ) {
    // Ficheros involucrados en E/S
    input Fich_sistema, Fich_tamproc, Fich_dirlog, Fich_proc;
    output Fich_fallos, Fich_dirfis;

    // Tabla de procesos
    Tabla_procesos = new Clase_proceso[MAX_PROCESOS];

    // Datos iniciales del fichero sistema
    int p, m, e, n_politica;

    // Datos iniciales calculados
    int Tamanyo_e_logico, politica;

    // Contador de direcciones generadas
    int Contador_refs;

    // Variables para leer la dir. logica y calcular la dir. fisica
    int tamanyo, n_proceso, dir_logica, n_pagina, desplazamiento, n_marco,
        dir_fisica, pag_victima;

    // Variables auxiliares
    int i;
    output pantalla = new output();
    format f2 = new format(2),
           f3 = new format(3),
           f4 = new format(4),
           f6 = new format(6);

    // Primero, abrir los ficheros
    Fich_sistema = new input("SISTEMA");
    Fich_tamproc = new input("TAMPROC");
    Fich_dirlog  = new input("DIRLOG");
    Fich_proc    = new input("PROC");
    Fich_dirfis = new output("DIRFIS");
    Fich_fallos = new output("FALLOS");

    // Ahora, analizarlos
    // Primero, el de sistema:
    p = Fich_sistema.readint();
    m = Fich_sistema.readint();
    e = Fich_sistema.readint();
    n_politica = Fich_sistema.readint();
    Fich_sistema.close();

    // Obtener los parametros del sistema
    Tamanyo_pagina = Elevar_a( 2, p );
    Tamanyo_e_logico = Elevar_a( 2, e );
    Ultimo_marco = Elevar_a( 2, m ) - 1;
    if ( n_politica != LFU_GLOBAL )
        pantalla.writeln( "Solo esta programado el LFU global" );
    politica = LFU_GLOBAL;

    pantalla.writeln( "Tamanyo pagina = " + Tamanyo_pagina );
    pantalla.writeln( "Espacio logico = " + Tamanyo_e_logico );
```

```
pantalla.writeln( "Numero marcos  = " + ( Ultimo_marco + 1 ) );
pantalla.write( "Politica       = " );
switch ( politica ) {
    case LFU_GLOBAL : pantalla.writeln( "LFU global" );
}
pantalla.writeln();

// Y ahora leer e inicializar la tabla de procesos:
Ultimo_proceso = -1;
while ( Ultimo_proceso+1<MAX_PROCESOS && Fich_tamproc.more() ) {
    Ultimo_proceso++;
    tamanyo = Fich_tamproc.readint();
    Tabla_procesos[Ultimo_proceso] = new Clase_proceso( tamanyo );
}
Fich_tamproc.close();

Escribe_Tabla_procesos();

// Vector de marcos = MEMORIA
Memoria = new Clase_marco [ Ultimo_marco + 1 ];

// Inicializar la memoria
for ( i=0 ; i<=Ultimo_marco ; i++ )
    Memoria[i] = new Clase_marco();

// El bucle principal
for ( Contador_refs = 1 ; Fich_proc.more() ; Contador_refs++ ) {
    pantalla.writeln( "Referencia numero " + f3.fmt(Contador_refs) );
    pantalla.writeln( "====================" );

    // Leer la referencia actual
    n_proceso = Fich_proc.readint();
    if ( n_proceso >= MAX_PROCESOS ) n_proceso = 0; // Por si acaso
    dir_logica = Fich_dirlog.readint();

    // Calcular la pagina y el desplazamiento
    n_pagina = 0; // 02@@@
    desplazamiento = 0; // 02@@@

    pantalla.writeln( "Proc. " + f2.fmt(n_proceso) +
      " @Logica = " + f6.fmt(dir_logica) +
      " => " + f3.fmt(n_pagina) + " " + " " + f4.fmt(desplazamiento) );

    // Pertenece esta pagina al proceso?
    if ( false ) { // 03@@@ SI SE HACE REFERENCIA A UNA PAGINA
        // QUE NO EXISTE EN EL PROCESO, ...
        Fich_fallos.writeln( "ERROR" );
        Fich_dirfis.writeln( "ERROR" );
        pantalla.writeln( "Direccion erronea.");
    } else { // Es una referencia valida para ese proceso
        // Comprobar si esa pagina esta en memoria.
        if ( true ) { // 09@@@ Si la pagina no esta en memoria
            Fich_fallos.writeln( "F" );
            pantalla.writeln( "FALLO de PAGINA: Procesando..." );

            // SI EXISTE UN MARCO LIBRE, UTILIZARLO.
            // Asignar a la variable i el valor resultante
            // de la busqueda.
            i = 0;
            // 07@@@

            if ( true ) // 08@@@  Si existe un marco libre,
                // identificado por i
                pag_victima = i;
            else { // Si no,
                // UTILIZAR UN ALG. DE REEMPLAZO DE PAGINAS
                // PARA ELEGIR UNA VICTIMA
                switch ( politica ) {
```

```
                            case LFU_GLOBAL: pag_victima = Aplica_LFU_GLOBAL();
                                     break;
                            default: pag_victima = 0;
                        }
                        pantalla.writeln(
                          "Alg. de Reemplazo --> Marco seleccionado ="
                          + f3.fmt(pag_victima) );
                    } // De si no existe un marco libre

                    // ACTUALIZAR LA TABLA DE PAGINAS Y LA TABLA DE MARCOS
                    // Una vez se tiene la pagina-victima, se ocupa con
                    // la pagina que ha producido el fallo

                    if ( Memoria[pag_victima].pagina != NADA )
                        // Si estaba ocupada, se desocupa
                        ;// 11@@@

                    // Actualizar la memoria
                    // 04@@@

                    // Y registrar el marco en la tabla de paginas del proceso
                    // 05@@@

                    // Viene bien anotarse en la variable n_marco el numero
                    // de marco en que hemos puesto la pagina
                    n_marco = pag_victima;

                    // Ahora (para el LFU global), poner a cero el contador de
                    // referencias de la pagina a la que se esta accediendo
                    // Memoria[...].contador = 12@@@;

                } // De si la pagina no esta en memoria
                else {
                    Fich_fallos.writeln();

                    // Viene bien anotarse en la variable n_marco el numero
                    // de marco en que esta la pagina
                    n_marco =
                        Tabla_procesos[n_proceso].Tabla_paginas[n_pagina].marco;

                    // Ahora (para el LFU global), incrementar el contador de
                    // referencias de la pagina a la que se esta accediendo
                    // Memoria[...].contador = 12@@@;

                }

                // Al final, calcular la direccion fisica
                dir_fisica = 0; // 06@@@

                // Y escribirla en el fichero
                pantalla.writeln(
                  ">>>>>>==========================>>> @FISICA  = "
                  + f6.fmt(dir_fisica) );
                Fich_dirfis.writeln( dir_fisica );
            } // De si es una referencia valida para el proceso
            Escribe_memoria();
            pantalla.writeln(
              "############################################################");
            pantalla.writeln();
        } // Del bucle de lectura de referencias logicas

        // Para finalizar, cerrar los ficheros
        Fich_proc.close();
        Fich_dirlog.close();
        Fich_dirfis.close();
        Fich_fallos.close();
    }
}
```