Programación de Sistemas y Concurrencia

Control Bloque 1, Temas 1 - 2
Curso 2020-2021

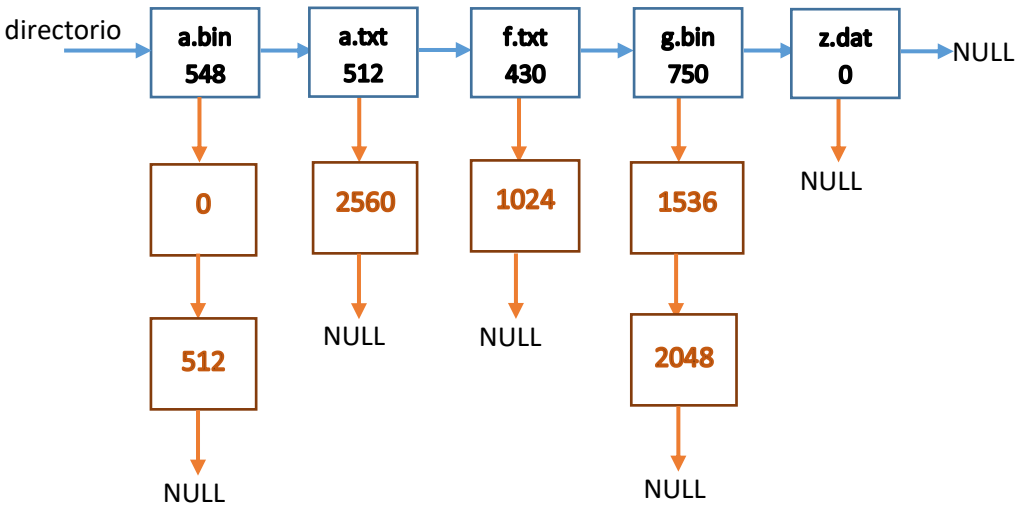
Grado en Ingeniería del Software, grupo A

APELLIDOS _____ NOMBRE _____

DNI _____ ORDENADOR _____ GRUPO/TITULACIÓN _____

Descripción del sistema

Se quiere utilizar la siguiente estructura para representar los ficheros almacenados en un directorio dentro de un sistema de ficheros. Suponemos que dentro del directorio solo podemos tener ficheros, no podemos tener otros directorios. Cada nodo de la lista enlazada principal (los nodos representados en azul en la figura) almacena el nombre del fichero (una cadena de caracteres) y el tamaño del fichero en el directorio (un número natural), además de un puntero a otra lista enlazada (nodos representados en naranja en la figura). En cada nodo de esta segunda lista enlazada se almacena la dirección de comienzo de un bloque de memoria asignado a dicho fichero. Suponemos que todos los bloques son del mismo tamaño (512 bytes). La memoria se asigna a los ficheros en bloques de 512 bytes, pero los ficheros pueden tener un tamaño variable (no tiene por qué ser múltiplo de 512 bytes). Por ejemplo, al primer nodo de la lista enlazada principal (fichero a.bin) se le han asignado dos bloques de memoria de 512 bytes (empezando en las direcciones 0 y 512) ya que su tamaño es 548 bytes. En el caso del fichero z.dat no se le han podido asignar bloques de memoria disponibles, y por tanto su tamaño de fichero es 0 bytes.



Por otro lado, tenemos otra lista enlazada con todos los bloques de memoria que se encuentran libres en un momento determinado. Por ejemplo, considerando una zona de memoria de 4Kbytes = 4096 bytes, y considerando los bloques ya utilizados por los ficheros representados en la figura anterior, la figura siguiente indicaría, de forma ordenada de menor a mayor, los bloques de memoria libres. Cada nodo almacena la dirección de comienzo de un bloque de memoria de 512 bytes.

Primera Parte – Módulo Bloques

En esta primera parte se implementa un módulo para gestionar la lista de bloques de memoria (parte naranja en las figuras). Se debe implementar una lista enlazada con los tipos y prototipos que se encuentran en el fichero "**Bloques.h**" y que se describen a continuación:

```
typedef struct Nodo *ListaBloques;
struct Nodo {
	unsigned int dirInicio; //dirección de inicio de un bloque de memoria
	ListaBloques sig;	//puntero al siguiente nodo de la lista enlazada
};
```

En el fichero **Bloques.c** deben implementarse las siguientes funciones (la descripción detallada de lo que hay que hacer en cada función la tenéis en el fichero **Bloques.h**).

```
(1.5 pto) void crear(ListaBloques *lb, unsigned int tamMemoria);

(0.5 pto) void obtenerBloque(ListaBloques *lb, ListaBloques *bloque);

(1.5 pto) void insertarBloque(ListaBloques *lb, ListaBloques bloque);

(0.5 pto) void imprimir(ListaBloques lb);

(0.5 pto) void borrar(ListaBloques *lb);
```

En el fichero **driverBloques.c**, podéis encontrar un fichero main para probar las funciones anteriores.

La salida de la ejecución de este fichero es (no se incluyen aquí los mensajes "DEBE SALIR: …" que al ejecutar la aplicación os indican cuál debería ser la salida para que podáis compararla con la vuestra):

```
 CREAMOS LA LISTA DE BLOQUES LIBRES
-----------------------------------
Bloques libres:  ( 0 512 1024 1536 2048 2560 3072 3584 )

OBTENEMOS CUATRO NODOS DE LA LISTA DE BLOQUES LIBRES
----------------------------------------------------
La direccion de comienzo del bloque obtenido es: 0
La direccion de comienzo del bloque obtenido es: 512
La direccion de comienzo del bloque obtenido es: 1024
La direccion de comienzo del bloque obtenido es: 1536
Bloques libres:  ( 2048 2560 3072 3584 )

DEVOLVEMOS LOS NODOS A LA LISTA DE BLOQUES LIBRES
-------------------------------------------------
Devolvemos el bloque con direccion de inicio 0
Bloques libres:  ( 0 2048 2560 3072 3584 )
Devolvemos el bloque con direccion de inicio 1024
Bloques libres:  ( 0 1024 2048 2560 3072 3584 )
Devolvemos el bloque con direccion de inicio 1536
Bloques libres:  ( 0 1024 1536 2048 2560 3072 3584 )
Devolvemos el bloque con direccion de inicio 512
Bloques libres:  ( 0 512 1024 1536 2048 2560 3072 3584 )
```

Segunda Parte – Módulo Directorio

En la segunda parte del ejercicio representaremos los ficheros almacenados dentro de un directorio (estructura representada en la primera figura del enunciado), que se corresponde con la siguiente definición de tipos:

```
typedef struct NodoFichero *ListaFicheros;
struct NodoFichero {
	char nombre[30];	//nombre del fichero
	unsigned int tam;	//tamaño del fichero en bytes
	ListaBloques bloques; //lista de bloques de memoria asignados (módulo anterior)
	ListaFicheros sig;	//puntero al siguiente nodo en la lista enlazada
}
```

En el fichero **Directorio.h** se encuentran definidos estos tipos y los prototipos de las funciones que hay que implementar en el fichero **Directorio.c** (la descripción detallada de lo que hay que hacer en cada función la tenéis en el fichero **Directorio.h**). Recordad que para manejar los string hay que utilizar las funciones de la librería "**string.h**". Tened en cuenta que, para implementar las funciones de **Directorio.h** habrá que llamar de forma adecuada a las funciones definidas en **Bloques.h**.

```
(2.00 pto) void nuevoFichero(ListaFicheros *lf, char *nombre, unsigned int tam, ListaBloques *lb);

(2.00 pto) void crearDesdeFicheroTexto(char * nombre, ListaFicheros *lf, ListaBloques *lb);

(0.75 pto) void imprimirDirectorio(ListaFicheros *lf);

(0.75 pto) void borrarDirectorio(ListaFicheros *lf, ListaBloques *lb);
```

En el fichero, **driverDirectorio.c** podéis encontrar una función main para probar estas funciones.

La salida por pantalla de la ejecución de este programa es:

```
CREAMOS LA LISTA DE BLOQUES LIBRES
----------------------------------
Bloques libres:  ( 0 512 1024 1536 2048 2560 3072 3584 )

CREAMOS EL DIRECTORIO CON LOS DATOS CONTENIDOS EN directorio.txt
----------------------------------------------------------------
fichero4.txt      0 ( )
fichero3.txt   2000 ( 1536 2048 2560 3072 )
fichero2.txt    750 ( 512 1024 )
fichero1.txt    512 ( 0 )

BORRAMOS LOS DATOS DEL DIRECTORIO
---------------------------------------------------------------
Directorio vacio ...
Bloques libres:  ( 0 512 1024 1536 2048 2560 3072 3584 )
```

ANEXO

Los prototipos de las funciones para manipular strings (incluidas en <string.h>) son:

char* strcpy(char *s1, char *s2): Copia los caracteres de la cadena s2 (hasta el carácter '\0', incluido) en la cadena s1. El valor devuelto es la cadena s1.

Los prototipos de las funciones para **manipulación de ficheros de texto** (incluidos en <stdio.h>) son los siguientes (se dan por conocidos los prototipos de las funciones de <stdlib.h> que necesites, como free o malloc):

FILE *fopen(const char *path, const char *mode): Abre el fichero especificado en el modo indicado ("rb"/ y "wb" para lectura/escritura binaria y "rt"/"wt" para lectura/escritura de texto). Devuelve un puntero al manejador del fichero en caso de éxito y NULL en caso de error.

int fclose(FILE *fp): Guarda el contenido del buffer y cierra el fichero especificado. Devuelve 0 en caso de éxito y -1 en caso de error.

int fscanf (FILE *stream, const char *format, ...): Lee del fichero stream los datos con el formato especificado en el parámetro format, el resto de parámetros son las variables en las que se almacenan los datos leídos en el formato correspondiente. La función devuelve el número de variables que se han leído con éxito.