UNIVERSIDAD DE MÁLAGA
Dpto. Lenguajes y Ciencias de la Computación

Programación de Sistemas y Concurrencia

Práctica nº1. Gestión Dinámica de Memoria

Nos han pedido realizar un programa que gestione la memoria de un dron destinado a tomar imágenes para detectar grietas y deformidades en edificios. El aparato tiene 2 tipos de memoria, la RAM normal y una memoria de alta velocidad para almacenar las fotografías que va realizando. Puesto que cada imagen puede ocupar un tamaño distinto, se desea gestionar esta última memoria del aparato de forma dinámica para optimizar su rendimiento. A dicha memoria se accede mediante sus direcciones (números naturales) `0` y `MAX=999` (constante). Para ello, se ha pensado en utilizar una estructura como la siguiente:

```
typedef struct T_Nodo* T_Manejador;
struct T_Nodo {
      unsigned inicio;
      unsigned fin;
      T_Manejador sig;
};
```

en la que cada nodo nos indica una zona de memoria continua (desde la dirección `inicio` hasta la dirección `fin`, ambas incluidas) disponible en la memoria total (ver figura ejemplo). Inicialmente existirá un único nodo con `inicio = 0` y `fin = MAX`. Esta lista estará ordenada por el valor `inicio`.

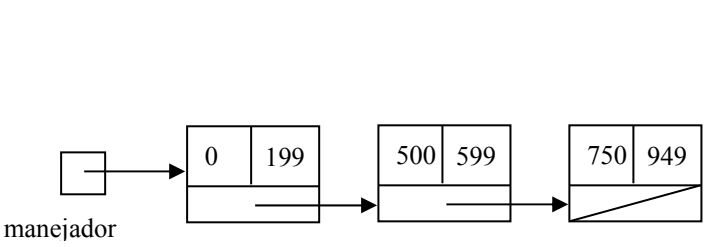
Cuando el dron hace una foto, necesita obtener cierta cantidad de memoria continua (operación `Obtener`). Para ello, se buscará un nodo con tamaño suficiente para conseguir el tamaño solicitado y la información de dicho nodo se verá modificada para reflejar que ahora hay menos memoria disponible. Si se requiere toda la memoria indicada por dicho nodo, éste desaparecerá de la lista.

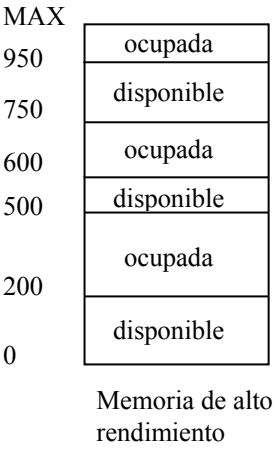
El dron no almacena todas las imágenes en su memoria, sino que las va enviando según puede mediante un módulo GSM (según la cobertura podrá enviar las fotos enteras o sólo una parte de ellas). Por tanto, cuando la foto (o una parte) se ha enviado, se tiene que liberar el trozo de memoria continua que ocupaba la foto (operación `Devolver`). Para ello, se añadirá un nuevo nodo a la lista con la información de la cantidad de memoria continua devuelta. Esta operación puede ocasionar que el trozo de memoria devuelto quede junto a uno o dos disponibles. Esto hay que tenerlo en cuenta en la lista de gestión, de forma que nunca existirán dos nodos seguidos en los que el valor del campo `fin` del primero sea uno menos que el campo `inicio` del segundo. Con esto se

gestiona lo que se denomina compactación de memoria (dos o tres nodos se transforman en uno).

Implementar las siguientes operaciones que aparecen especificadas en el fichero `GestionMemoria.h`:

```
// Crea la estructura utilizada para gestionar la memoria disponible.
void Crear(T_Manejador* Manejador);


// Destruye la estructura utilizada.
void Destruir(T_Manejador* manejador);


/* Devuelve en "dir" la dirección de memoria donde comienza el
 * trozo de memoria continua de tamaño "tam" solicitada.
 * Si la operación se pudo llevar a cabo, es decir, existe dicho
 * trozo, devolvera TRUE en "ok"; FALSE en otro caso.
 */
void Obtener(T_Manejador *manejador, unsigned tam, unsigned* dir,
unsigned* ok);


/* Muestra el estado actual de la memoria */
void Mostrar (T_Manejador manejador);


/* Devuelve el trozo de memoria continua de tamaño "tam" y que
 * comienza en "dir".
 * Se puede suponer que se trata de un trozo obtenido previamente.
 */
void Devolver(T_Manejador *manejador,unsigned tam,unsigned dir);
```





Memoria de alto rendimiento