UNIVERSIDAD DE MÁLAGA | uma.es

Dpto. de Lenguajes y Ciencias de la Computación

Programación de Sistemas y Concurrencia

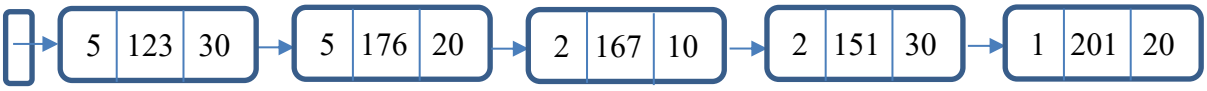
Control Parcial C
Curso 2023-2024

APELLIDOS _____ NOMBRE _____

DNI _____ ORDENADOR _____ GRUPO/TITULACIÓN _____

Bloque de C - Ejercicio Lista de Procesos

Una lista de procesos *esperando* para ser ejecutados podría representarse mediante una lista enlazada del modo siguiente:



Cada nodo representa un proceso, con su **prioridad**, su **identificador**, su **tiempo requerido de uso del procesador**, y un enlace al proceso siguiente. La prioridad debe ser siempre un valor entre 1 y 5, donde los procesos con prioridad 1 son los de menor prioridad y los procesos con prioridad 5 los de mayor prioridad. El tiempo requerido de uso del procesador deberá ser siempre mayor de 0 y múltiplo de 10. Puedes observar en el dibujo que los procesos están ordenados según su prioridad, de mayor a menor prioridad (desde los procesos con prioridad 5 hasta los procesos con prioridad 1). De cada prioridad podrá haber 0, 1 o más procesos. Observa, asimismo, que para una misma prioridad los procesos no siguen un orden determinado y se almacenarán según el orden de llegada.

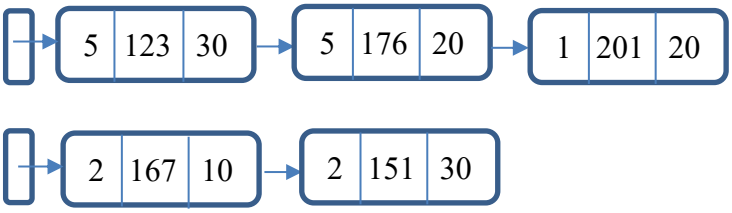
Se pide implementar el módulo Procesos (ficheros Procesos.h y Procesos.c) y utilizar el fichero driver.c proporcionado para probar las funciones implementadas. Habrá que implementar también dos funciones dentro del fichero driver.c.

Funciones a implementar en el fichero Procesos.c. Las funciones a implementar en el fichero Procesos.c se listan a continuación y **están todas descritas con más detalles** en el fichero Procesos.h

```
// (0.25 puntos). Crea una lista de procesos vacía
void crear(LProcesos *lp);

// (2 puntos). Inserta un proceso en la lista de procesos
void insertar(LProcesos *lp,unsigned prioridad, unsigned id,
              unsigned tiempo);

// (0.5 puntos). Muestra el contenido de la lista de procesos
void mostrar(LProcesos lp);

// (0.75 puntos). Guarda los procesos con una determinada prioridad
//                en un fichero de texto
void guardarFicheroPrioridad(LProcesos lp, unsigned prioridad, FILE *fich);


// (1.5 puntos). Ejecuta los procesos de la lista con varios procesadores
unsigned ejecutarMultiplesProcesadores(LProcesos *lp, unsigned tiempo,
                                       unsigned num_proc);

// (1 punto). Realiza varias ejecuciones con un solo procesador
void ejecutarMultiplesVeces(LProcesos* lp, unsigned veces, unsigned tiempo);
```

```
// (1.5 puntos). Extrae en una lista nueva todos los procesos de la
//               lista de procesos que tienen una determinada prioridad,
//               eliminándolos de la lista original
LProcesos extraerProcesos(LProcesos *lp, unsigned prioridad);
```

Por ejemplo, después de llamar a esta función con la lista enlazada mostrada arriba y un valor para el segundo parámetro de 2 (prioridad 2), la lista original y la lista devuelta quedarán de la siguiente forma.




```
// (0.5 puntos). Elimina todos los nodos y deja la lista vacía
void borrar(LProcesos *lp);
```

Funciones a implementar en el fichero driver.c. Las funciones a implementar en el fichero driver.c se listan a continuación y **están descritas en detalle** en el fichero driver.c

```
// (1 punto). Lee el contenido de la lista de procesos desde un
//            fichero binario y crea una lista de procesos.
void leerProcesosDeFicheroBinario(char * nombre, LProcesos *lp);

// (1 punto). Guarda el contenido de la lista de procesos a un fichero
//            de texto.
void guardarProcesosAFicheroTexto(char *nombre, LProcesos lp);
```

Salida del programa. Para los datos proporcionados en el fichero `driver.c` y el fichero `binario.bin` proporcionados en los recursos del examen, **la salida del programa** debe ser la siguiente:

```
Insertamos procesos y los mostramos.
[5, 123, 30] [5, 176, 20] [4, 151, 50] [4, 167, 30] [3, 162, 10] [3, 178,
20] [1, 201, 20]
Ejecutamos 4 veces con 1 procesador, siempre con tiempo de ejecucion 10.
[5, 176, 10] [4, 151, 50] [4, 167, 30] [3, 162, 10] [3, 178, 20] [1, 201,
20]
Ejecutamos con 8 procesadores y tiempo de ejecucion 10.
[4, 151, 40] [4, 167, 20] [3, 178, 10] [1, 201, 10]
2 procesadores sin utilizar.
Guardamos los procesos de la lista con prioridad 4 en un fichero de texto.
Extraemos los procesos con prioridad 4 de la lista.
Mostramos la nueva lista generada ...
[4, 151, 40] [4, 167, 20]
Mostramos la lista original modificada ...
[3, 178, 10] [1, 201, 10]
Borramos la lista y la volvemos a crear desde el fichero binario.
[5, 123, 30] [5, 176, 20] [4, 151, 50] [4, 167, 30] [3, 162, 10] [3, 178,
20] [1, 201, 20]
Guardamos la lista en un fichero de texto.
```

Anexo. Los prototipos de las funciones de lectura y escritura en ficheros de la biblioteca `<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"/ "wb" para lectura/escritura binaria y "r"/"w" 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.

```
unsigned fread(void *ptr, unsigned size, unsigned nmemb, FILE *stream);
```
Lee `nmemb` elementos de datos, cada uno de tamaño `size bytes`, desde el fichero `stream`, y los almacena en la dirección apuntada por `ptr`. Devuelve el número de elementos leídos.

```
unsigned fwrite(void *ptr, unsigned size, unsigned nmemb, FILE *stream);
```
Escribe `nmemb` elementos de datos, almacenados en la dirección apuntada por `ptr`, en el fichero `stream`. Cada uno de los datos es de tamaño `size bytes`. Devuelve el número de elementos escritos.

```
int fscanf(FILE *stream, const char *format, ...)
```
Lee desde el fichero `stream` los datos con el formato especificado en el parámetro `format`. El resto de los parámetros son las variables en las que se almacenarán los datos leídos. La función devuelve el número de valores leídos con éxito.

```
int fprintf(FILE *stream, const char *format, ...)
```
Escribe en el fichero `stream` los datos con el formato especificado en el parámetro `format`. El resto de los parámetros son las variables en las que se almacenan los datos a escribir en el formato correspondiente. La función devuelve el número de caracteres escritos con éxito.