UNIVERSIDAD DE MÁLAGA

Depto. de Lenguajes y Ciencias de la Computación
E.T.S.I. Informática. I.T.I
UNIVERSIDAD DE MALAGA

APELLIDOS: **NOMBRE:**
GRUPO: **ORDENADOR:**

LABORATORIO DE PROGRAMACIÓN
I.T. Informática. Sistemas
Examen de Junio. Curso 2010-2011.

Planificador de un S.O.

Un Sistema Operativo (S.O.) Multitarea permite ejecutar varios procesos/programas de forma simultánea. Para ello, se cumplen las siguientes características: (1) Cada proceso tiene asignada una prioridad, de forma que los procesos de mayor prioridad se ejecutan antes; (2) El sistema tiene una cola de procesos diferente para cada prioridad, de forma que todos los procesos en una cola tienen la misma prioridad; (3) El planificador se encarga de determinar cuál es el siguiente proceso en ser ejecutado. Para ello, desencolará un proceso de la cola de mayor prioridad. Si está vacía lo intentará con la cola de siguiente prioridad y así sucesivamente; (4) Cuando un proceso está en el procesador no se ejecuta completamente. Sólo puede estar ejecutándose de forma consecutiva durante un periodo de tiempo que llamamos *quantum*, transcurrido el cuál debe dejar paso a otro proceso, y (5) Si pasado ese *quantum* de tiempo el proceso aún no ha finalizado su ejecución volverá a encolarse en la cola correspondiente a su prioridad.

Se pide implementar un simulador del planificador de un S.O. Multitarea. Este S.O. cuenta con la posibilidad de gestionar los distintos procesos que se ejecutan en 5 niveles distintos de prioridad (nivel 1 prioridad mínima, nivel 5 prioridad máxima). Además, cada uno de estos procesos, a su vez, dispone de un *quantum* de tiempo como se ha explicado anteriormente (ver figura 1).

La arquitectura de nuestro PC permite además entrar en modo de hibernación de forma que, cuando el usuario lo indique, todos los procesos serán interrumpidos y sus datos escritos a fichero. Más adelante, cuando el PC entre de nuevo en activo, podrá retomarse la ejecución de los procesos leyendo la información del fichero.

Los datos con los que ha de contar cada proceso son: tiempo de ejecución restante (de tipo real e inicialmente igual al tiempo total de ejecución del proceso), quantum (de tipo real), prioridad (de tipo entero) e ID (identificador) de proceso (de tipo cadena de caracteres).

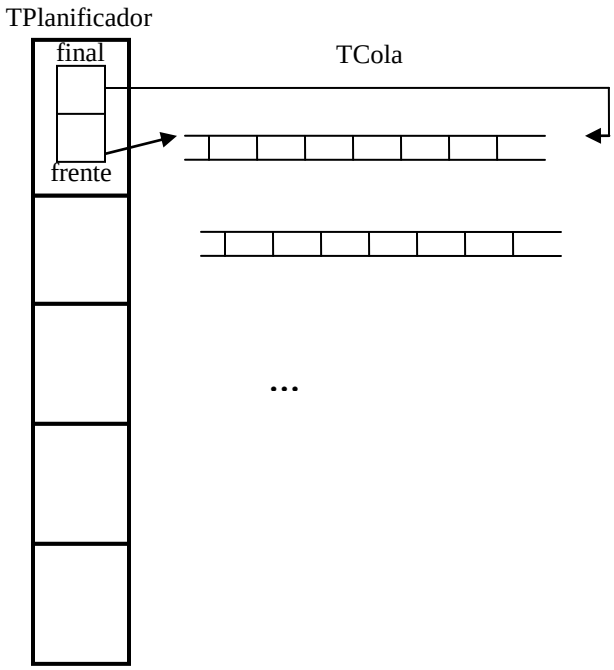

Figura 1

Implementar un menú que permita al usuario realizar las distintas opciones del planificador:

A- (1.00) Leer proceso. Permite al usuario leer por teclado todos los datos de un proceso e insertarlo en el sistema. Si la prioridad establecida por el usuario no está en el rango de la permitida por el sistema entonces se informará al usuario del error y el proceso no se insertará. Ejemplo:

```
Introduzca el ID de proceso: id1
Introduzca el tiempo total de ejecución del proceso: 12
Introduzca el quantum: 10
Introduzca la prioridad: 3
```

B- (2.00) Procesar. Permite al planificador sacar un proceso de la cola correspondiente y ejecutarlo. Ejecutar el proceso consiste en actualizar el tiempo de ejecución del proceso restando el quantum de ejecución del tiempo que le queda por finalizar. Si el proceso consume su quantum antes de finalizar su tiempo de ejecución (es decir, si tiempo-quantum >0), el proceso será encolado de nuevo en la misma cola con el tiempo de ejecución restante. Por el contrario, si finaliza, se eliminará el proceso de la cola y se imprimirá por pantalla el mensaje: "El proceso <*ID*> ha finalizado con éxito". Si no hay ningún proceso que ejecutar en el sistema se informará al usuario mediante un mensaje por pantalla.

C- (1.00) Matar proceso. Se pedirá al usuario el identificador de un proceso, que será eliminado del planificador. Si el proceso no existe se informará al usuario del error.
NOTA: Observad que existe un método `eliminarCola()` en el módulo `MCola` que recorre la cola y elimina el proceso indicado.

D- (1.00) Visualizar estado del planificador. Mostrará la situación actual del planificador.

NOTA: Observad que existe un método `visualizarCola()` en el módulo `MCola` que recorre la cola e imprime los datos por pantalla.

E- (1.75) Hibernar. Pide por teclado el nombre de un fichero y almacena en éste, en formato binario, toda la información de los procesos. Tras la ejecución de esta operación, el planificador quedará vacío. Si ya existía un fichero con el mismo nombre este se sobrescribirá.

F- (1.75) Resume. Permite a la máquina retomar el estado indicado por el fichero binario introducido por el usuario, descartando cualquier información o estado anterior de la estructura. Si el fichero no existe, nuestro programa informará convenientemente al usuario.

G- Salir. Esta opción permite abandonar el programa. Pedir confirmación antes de salir.

Para la realización de este ejercicio hay que implementar los siguientes módulos:

- `MProceso (MProceso.h, MProceso.cpp)`: Módulo de procesos. Define el tipo `TProceso`. Las funciones que ha de contener son:

```
void leerdatos(TProceso &datos);
/*lee los datos de un proceso*/
void escribirdatos(TProceso datos);
/*escribe los datos de un proceso*/
void leerdatosfichero(ifstream &f, TProceso &datos);
/*Lee los datos de un proceso de un fichero binario cuyo manejador se pasa
como parámetro*/
void escribirdatosfichero(ofstream &f, TProceso datos);
/*Escribe los datos en un fichero binario cuyo manejador se pasa como
parámetro*/
void idproc(TProceso datos, TCadena &id);
 /* Devuelve el id del proceso */
```

- `MCola(MCola.h, MCola.cpp)`: Implementación dinámica y eficiente de una cola con referencias (es decir, punteros) al frente y al final de la cola. Define los tipos `TCola` y `TErrorCola`.

```
TCola crearCola(); // Crea la Cola
void destruirCola(TCola& cola); // libera la memoria
bool colaVacia(TCola cola); // comprueba si la Cola está vacía
void meterCola(TCola &cola, TProceso proceso, TErrorCola &error);
void sacarCola(TCola &cola, TProceso &proceso, TErrorCola &error);
void visualizarCola (TCola cola);
/*Recorre la cola e imprime los datos por pantalla */
void eliminarCola (TCola &cola, TCadena id, TErrorCola &error);
/* Recorre la cola y elimina el proceso con identificador id. Si no puede
encontrarlo, lo indica utilizando el parámetro de salida "error"*/
```

- `MPlanificador(MPlanificador.h, MPlanificador.cpp)`. Implementación del módulo planificador. Define los tipos `TPlanificador` y `TErrorPlanificador`. Los procedimientos que habrán de implementarse en este módulo son:

```
TPlanificador CrearPlanificador();
/*Crea una variable de tipo TPlanificador*/
void insertarProceso(TPlanificador &p, TProceso datos, TErrorPlanificador
&e);
```

```
/*Inserta un proceso en el planificador atendiendo a la prioridad del mismo.
Si la prioridad del proceso no está en el rango permitido se informa del
error en "e"*/
void Procesar(TPlanificador &p, TErrorPlanificador &e);
/*Permite que un proceso sea ejecutado. Si no existe ningún proceso en el
planificador  se informa del error en "e"*/
void Visualizar(TPlanificador p);
/*Visualiza por pantalla el estado actual del planificador */
void Hibernar(TPlanificador &p, TCadena nom_fichero);
/* Permite que el planificador hiberne y almacene en el fichero indicado el
estado actual. El planificador se reinicializará volviendo en este caso al
estado de partida; es decir, no hay procesos que ejecutar*/
void Resume (TPlanificador &p, TCadena nom_fichero, TErrorPlanificador &e);
/* El planificador recupera el estado actual del fichero cuyo nombre se
indica por parámetro. Si el fichero no existe se informa del error en "e"*/
void DestruirPlanificador(TPlanificador &p);
/*Libera la memoria existente de la variable*/
```

Observaciones:

- Se deberá hacer un tratamiento de errores adecuado, donde los mensajes de error se muestren desde el programa principal. Los errores posibles de cada módulo serán tratados haciendo uso de tipos enumerados.
- Las opciones del menú suman 8.5 puntos. El 1.5 adicional se utilizará para valorar la correcta definición de tipos, la correcta modularización y el correcto tratamiento de errores.
- NO SE PODRÁN AÑADIR OPERACIONES EN LA PARTE DE DEFINICIÓN DE UN MÓDULO.
- NO SE PODRÁ hacer uso de detalles de implementación de la cola dentro del programa principal o del módulo MPlanificador, es decir, nada de operaciones del tipo cola.frente->sig=NULL o cola.frente =NULL.
- SE PODRÁN implementar todos los procedimientos y funciones auxiliares que se consideren necesarios dentro del programa principal o en la parte de IMPLEMENTACIÓN de los módulos.
- La implementación del módulo MCadena es opcional. Se puede usar directamente el tipo predefinido *string* de la librería *string.h*