UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Diseño e implementación de un planificador de tareas para sistemas de tiempo real

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Montoliu Villamón, Jorge

Tutor/a: Balbastre Betoret, Patricia

CURSO ACADÉMICO: 2021/2022

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Diseño e implementación de un planificador de tareas para sistemas de tiempo real

TRABAJO FINAL DE

Grado en Ingeniería Electrónica Industrial y Automática

REALIZADO POR

Jorge Montoliu Villamón

TUTORIZADO POR

Patricia Balbastre Betoret

CURSO ACADÉMICO: 2020/2021

Tabla de contenido

Índice de figuras

1. Objeto del proyecto

El objeto de este proyecto de fin de grado es la implementación de una aplicación software que planifique un conjunto de tareas para un sistema de tiempo real.

El objetivo específico del proyecto es la programación de un código capaz de devolver una serie de datos que se puedan utilizar en un sistema de tiempo real industrial.

Para ello se va a realizar una aplicación con Embarcadero C++Builder que solicite una serie de datos al usuario y devuelva los resultados de forma gráfica e intuitiva.

2. Estudio de necesidades, factores a considerar y limitaciones

2.1. Introducción a los sistemas de tiempo real

Un sistema de tiempo real es «un sistema informático que interacciona con su entorno físico y responde a los estímulos del entorno dentro de un plazo de tiempo determinado y que, además, tienen que ejecutarse dentro de un intervalo de tiempo determinado» (José Luis Villarroel Salcedo, 2014)[1].

Estos sistemas se utilizan en diferentes campos que requieren de una coordinación entre software y hardware siendo muy importantes en la informática y la robótica.

Existen dos tipos de sistemas de tiempo real: los suaves o acríticos y los duros o críticos. Mientras que en los duros los tiempos de respuesta no admiten retrasos los suaves sí pueden aceptar que se sobrepasen los plazos de las tareas ocasionalmente. En este proyecto planificaremos las tareas suponiendo que el sistema va a ser duro.

Dependiendo de la forma de planificar los sistemas los podemos dividir en dos grandes grupos: estáticos y dinámicos. Los planificadores estáticos generan un plan cíclico a priori que determina si el sistema es planificable y sus resultados pueden ser representados en una tabla fácilmente. Sin embargo, son una opción poco flexible. Este es el caso de la aplicación desarrollada. Los dinámicos determinan en qué momento se activan las tareas en tiempo real permitiendo una mayor flexibilidad.

2.2. Estudio de necesidades

Los sistemas de tiempo real son muy útiles para el control de procesos industriales, en especial para los procesos más automatizados de la industria. Actualmente, muchos de los sistemas encargados del control de un proceso comerciales son concurrentes.

Un sistema concurrente es aquel que tiene la capacidad de ejecutar distintas partes de un programa de manera desordenada sin alterar el resultado final. Estas partes del programa se ejecutan en diferentes hilos de ejecución.

El orden y los instantes en los que se van a ejecutar las tareas requieren de una planificación para optimizar tiempo y recursos del sistema que controla el proceso. Adicionalmente, existen diferentes métodos de planificación que pueden tener diferentes resultados, en algunos casos incluso puede que un sistema sea planificable con un método concreto mientras que no lo es con otro método.

La aplicación que se va a realizar permite comprobar bajo qué criterios de planificación un sistema es planificable. Además, genera una serie de datos que se utilizan en los sistemas de tiempo real industriales.

La planificación de tareas de sistemas de tiempo real crítico es fundamental para poder certificar el sistema. En los sistemas altamente críticos como satélites o aviones, es necesario certificar el sistema completo bajo el estándar del sector al que se dirige para prevenir situaciones catastróficas. Algunos ejemplos son: el estándar ECSS para proyectos espaciales europeos, el DO-178B para aviónica o el IEC61508 para sistemas industriales.

Imagen 1. Logo del protocolo ECSS

Todos estos estándares obligan a que la asignación de la ejecución de las aplicaciones sea conocida a priori, es decir, antes de poner en ejecución el sistema. Para todos estos tipos de aplicaciones, y como requisito obligatorio de la certificación, es necesario disponer de aplicaciones de planificación como la que se va a desarrollar en este trabajo.

La adición de más tareas o la alteración de algún parámetro de estos sistemas exige una nueva planificación para ser certificadas de nuevo. Esto implica que la aplicación creada no pierde su utilidad una vez diseñado un proyecto. Si se quisieran implementar nuevas funcionalidades en un sistema crítico para el desarrollo de una versión superior se requeriría de una nueva planificación e incluso un cambio en el criterio de planificación de todo el sistema.

2.3. La planificación de tareas

«Un planificador (scheduler) es un algoritmo encargado de asignar los recursos del procesador a diferentes tareas concurrentes y en diferentes momentos. En este sentido, la planificación (scheduling) de un Sistema en Tiempo Real consiste en asignar tareas al procesador».[2]

Hay dos tipos de planificadores principales: los cíclicos y por prioridades. Los planificadores cíclicos son muy simples y no son concurrentes mientras que los planificadores por prioridades son dinámicos, concurrentes y permiten planificar sistemas más complejos.

En este proyecto se va a optar por una planificación por prioridades.

Las tareas de los sistemas por prioridades tienen tres parámetros principales:

- Tiempo de cómputo (wcet): Representa el número de instantes en los que una tarea se ejecuta a lo largo de un periodo. Los instantes son necesariamente números enteros.
- Plazo relativo (deadline): Representa el tiempo más grande en el que se pueden ejecutar las tareas. Si cuando se sobrepasa el instante del plazo relativo aún no se han ejecutado todos los tiempos de cómputo el sistema no será planificable.
- Periodo: Representa el instante en el que se repiten las tareas.

De las diferentes formas de planificación por prioridades destacan:

1- Plazo relativo menor: Se dará la mayor prioridad a las tareas con menor plazo relativo (deadline). Con esta política de planificación una tarea con mayor prioridad puede expulsar a otra con menor prioridad que está siendo ejecutada.

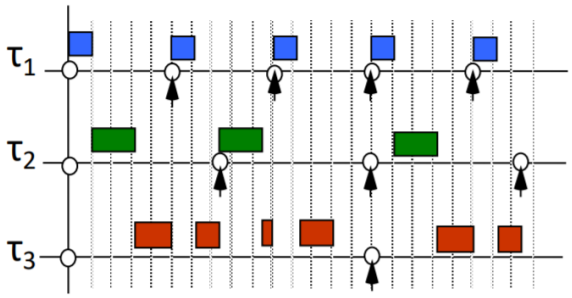


Imagen 2. Cronograma deadline menor

2- Plazo relativo más cercano: Se dará mayor prioridad a la tarea que deba realizarse antes independientemente de sus parámetros. Con este criterio de planificación no hay expulsión de otras tareas de menor prioridad.

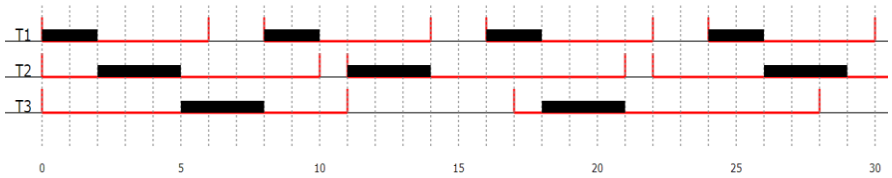


Imagen 3. Cronograma deadline más cercano

3- Tiempo de cómputo menor: Se dará mayor prioridad a la tarea que tenga el menor tiempo de cómputo (wcet). Con esta política sí puede haber expulsión de tareas menos prioritarias. La imagen 1 también podría ser un ejemplo de este tipo de política de planificación.

4- Holgura menor: Se dará mayor prioridad a la tarea con la holgura más alta. La holgura cambia en cada instante por lo que se debe estar calculando constantemente con la fórmula: $L_i = (k-1)T_i + D_i - C_i - t$
Donde L es la holgura, T el periodo de la tarea, D el plazo relativo, C el tiempo de cómputo y t el instante en que se calcula la holgura.

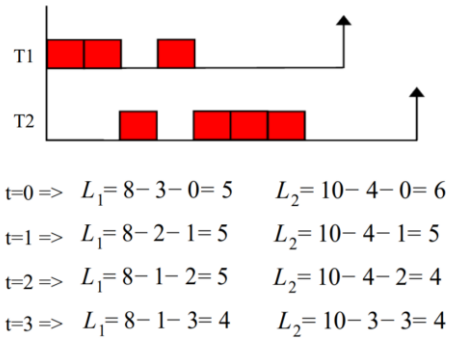


Imagen 4. Cronograma y ecuaciones simplificadas holgura menor

En un sistema de tiempo real industrial se suelen usar unos parámetros para indicar al sistema en qué instantes se empieza a ejecutar una tarea, en qué instantes deja de ejecutarse y un número identificativo que permite saber a qué tarea pertenecen estos datos.

2.4. Limitaciones

A continuación se expondrán las principales limitaciones a la hora de desarrollar el proyecto:

El principal problema son algunos errores que se generan en el programa utilizado para la realización de la aplicación cuando se utilizan valores de alrededor de 60.000 o superior para el hiperperiodo de las tareas. Esto se debe a que el programa se queda sin memoria para crear vectores del tamaño de hiperperiodo.

Adicionalmente existe una limitación con los valores. Para la aplicación se han declarado variables enteras de tipo «int». El valor máximo que puede almacenar este tipo de variables es de 2.147.483.647. De todas formas no es frecuente que el programa utilice valores tan grandes. Con valores más pequeños ya saltan los errores de memoria comentados con anterioridad.

Otra gran limitación es que a la hora de planificar las tareas desde la aplicación se supone un comportamiento ideal de las tareas. Sin embargo, en el funcionamiento real de un sistema de tiempo real existen retrasos entre el instante en el que idealmente debería comenzar una tarea y el instante en el que realmente empieza. Esta diferencia de tiempo depende de muchos factores como la velocidad del procesador que planifica o las resistencias mecánicas del hardware.

3. Planteamiento de soluciones, soluciones alternativas y justificación de la solución adoptada

3.1. Hardware

Este proyecto consiste principalmente en el desarrollo de un software por lo que el hardware es muy escaso. El único hardware que vale la pena mencionar es el ordenador portátil marca Lenovo con el que se ha realizado el proyecto.

3.2. Software

Dada la naturaleza de este proyecto, el software cobra mucha importancia y, por ende, se le deberá prestar especial atención. El sistema operativo del ordenador portátil mencionado anteriormente es Windows 10.

A continuación, se van a exponer brevemente los principales programas que se han utilizado durante el desarrollo de la aplicación.

3.2.1. Lenguaje de programación

El lenguaje de programación escogido ha sido C++, más adelante se profundizará en el porqué de esta elección. C++ es un lenguaje de programación diseñado en 1979 por Bjarne Stroustrup y está muy extendido en diversos entornos de desarrollo de informática.

3.2.2. Entorno de desarrollo

El entorno de desarrollo por el que se ha optado ha sido Embarcadero C++Builder v10.4.2. Éste es un entorno que permite el desarrollo de aplicaciones informáticas con interfaz gráfica. C++Builder puede ser programado con el lenguaje C++ y se puede descargar tanto en Windows.

Imagen 5. Logo C++Builder

3.3. Soluciones alternativas

3.3.1. Lenguajes de programación alternativos

Como alternativas al lenguaje C++ encontramos las siguientes:

1- Ada: Lenguaje de programación que apareció en 1980 que resulta muy intuitivo si se sabe inglés y que es concurrente, por lo que es muy adecuado para sistemas de tiempo real.
2- Python: Lenguaje de programación que apareció en 1991. Los códigos de Python son fácilmente entendibles por lo que puede resultar útil en proyectos con códigos muy grandes.

3.3.2. Entornos de desarrollo alternativos

Los entornos alternativos al mencionado anteriormente son:

1- Gnoga: Entorno de desarrollo para el lenguaje Ada. Muy útil par desarrollar aplicaciones con interfaz gráfica con la ventaja de estar programadas con un lenguaje concurrente pero que es muy poco intuitivo.
2- Qt Creator: Entorno de desarrollo que acepta códigos programados con C++ y Java entre otros. Es muy intuitivo para el desarrollo de aplicaciones con interfaz gráfica de usuario (GUI).

Ambos entornos se pueden ser utilizados en Windows XP y versiones superiores.

3.4. Justificación de la solución adoptada

Finalmente, se van a exponer los diferentes motivos por los que se ha optado por el lenguaje y entorno del punto 3.2 y se han desechado los del punto 3.3.

En un principio, la aplicación se iba a desarrollar con Gnoga y el lenguaje Ada por la superioridad de este lenguaje en sistemas de tiempo real con respecto a C++ por ser concurrente. Sin embargo, como se ha mencionado anteriormente, el entorno Gnoga es muy poco intuitivo. Este entorno no se ha utilizado en ninguna asignatura del grado por lo que el alumno no está familiarizado con él. Estas dos causas han hecho imposible el desarrollo de la aplicación con Gnoga.

Una vez desechada la opción de Gnoga, el segundo entorno de desarrollo que se ha contemplado es C++Builder. Ésta es la opción escogida finalmente porque el alumno está muy habituado al lenguaje C++ y ya posee experiencia con el programa. Quedan por tanto desechados los lenguajes de Ada y Python por ser incompatibles con C++Builder.

Finalmente se debe comentar que el motivo por el que no se ha utilizado el entorno QT Creator es únicamente por preferencia del desarrollador del proyecto. Este programa también acepta el lenguaje C++ pero el alumno tiene menos experiencia en su uso.

4. Descripción detallada de la solución adoptada

En este apartado se comentarán las diferentes partes que conforman la aplicación realizada. Se hará referencia tanto a la interfaz gráfica de las diferentes pantallas que la conforman como a las funciones que permiten su funcionamiento.

4.1. Interfaz gráfica

Se ha optado por la implementación de diferentes ventanas en la aplicación desarrollada para facilitar y hacer más intuitiva la interacción entre el programa y el usuario. De los tres módulos que forman el programa dos de ellos tienen interfaz gráfica.

4.1.1. Módulo Planificador

Este es el módulo que se encarga de la obtención de los parámetros de las tareas que se pretende planificar. El funcionamiento individual de esta parte del programa es el siguiente:

En un principio todos los botones y «edits» están deshabilitados a excepción del «edit» que pide el número de tareas y el botón de aceptar de su derecha. Si el número que se ha ingresado es menor a 1 o mayor que 6 o se introduce un símbolo o letra saltará un aviso y no se habilitarán los siguientes «edits». Si se introduce un valor correcto se habilitarán los «edits» de cada columna dependiendo del número de tareas introducido.

El siguiente paso es rellenar las columnas con los parámetros de las tareas. El programa no aceptará que el «wcet» sea mayor que el «deadline» y periodo y tampoco permitirá que el «deadline» sea mayor que el periodo. En caso de que en alguna tarea suceda esto no se permitirá seguir al siguiente paso al pulsar el botón de aceptar que se ha habilitado junto a los «edits» de los parámetros y aparecerá otro aviso. Si los parámetros se han introducido correctamente se habilitará el botón de «Planificar». Este botón hace que aparezca la otra ventana del programa y habilita el temporizador. El temporizador tiene una serie de funciones que requieren de un retardo con respecto la aparición de la otra ventana.

Adicionalmente existe un botón de reinicio que permite volver a utilizar el programa con diferente número de tareas y otros parámetros.

Imagen 6. Form1 en C++Builder

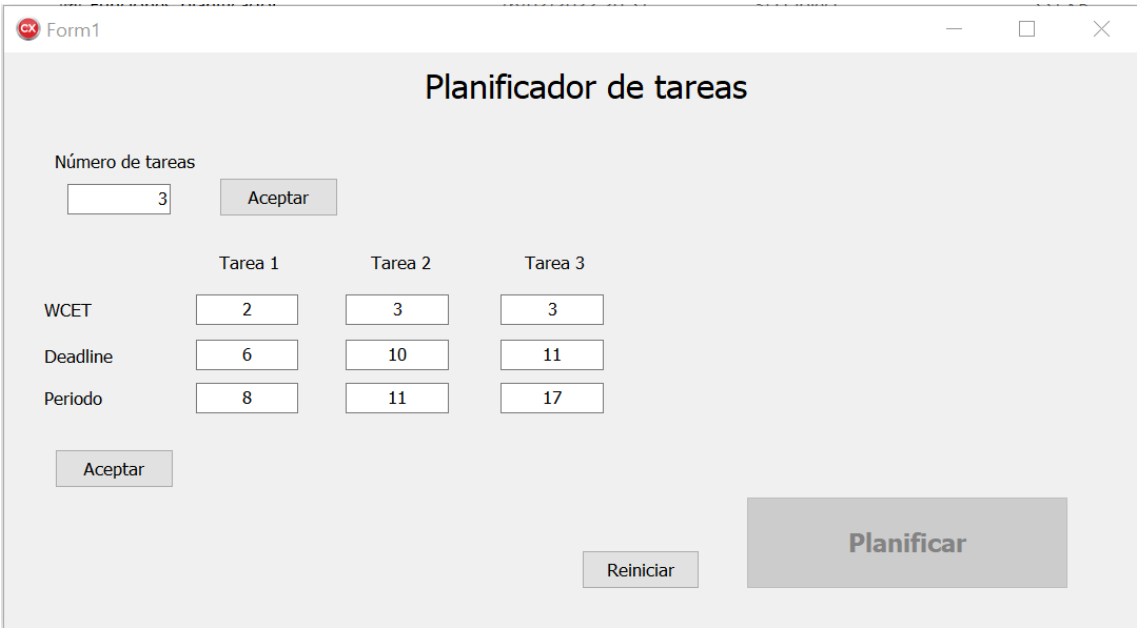


Imagen 7. Form1 en ejecución

Nombre	Tipo de elemento	Función
Form 1	TForm	Es la ventana donde se encuentran el resto de elementos.
Edit_tareas	TEdit	En él se introduce el valor del número de tareas.
Edit_w[1…6]	TEdit	En ellos se introducen los valores de los «wcet» de las tareas 1 a 6.
Edit_d[1…6]	TEdit	En ellos se introducen los valores de los «deadline» de las tareas 1 a 6.
Edit_p[1…6]	TEdit	En ellos se introducen los valores de los periodos de las tareas 1 a 6.
Button_ntareas	Tbutton	Registra el valor de «Edit_tareas» y se asegura de que sea válido.
Button_params	Tbutton	Registra los parámetros de los «edits» correspondientes y se asegura de que sean válidos.
Button_iniciar	Tbutton	Hace aparecer el Form2 y habilita el timer1.
Button_reiniciar	Tbutton	Devuelve el programa a un estado similar al inicial.
Labelx11	TLabel	En el «Form1» los «labels» sólo tienen la función de mostrar títulos y advertencias. Para optimizar en espacio, los 11 «labels» serán representados en esta única celda.
Timer1	TTimer	Realiza una serie de tareas de cara al siguiente módulo.

Tabla 1. Elementos de Form1

4.1.2. Módulo Cronogramas

En este módulo se muestra gráficamente un cronograma que será rellenado según el criterio de planificación escogido. También muestra una seri de datos que puedan ser de interés para el usuario. El funcionamiento del módulo es el siguiente:

Nada más aparecer el «Form2», el «Timer1» del módulo anterior dibuja gracias a la función «Canvas» un cronograma en negro y los límites de las tareas en rojo. El usuario deberá pulsar el botón de empezar para poder elegir el criterio de planificación que desee. Este paso es necesario porque al tratarse de un módulo diferente tiene que existir una función que envíe los datos obtenidos anteriormente. Ahora el usuario sólo tendrá

que pulsar qué política de planificación quiere y automáticamente el cronograma se rellenará con los tiempos de activación adecuados. Además, si el sistema es planificable, éste aparecerá en verde, mientras que si no lo es, aparecerá en rojo y se mostrará a su derecha el tiempo de retrasos producido durante el hiperperiodo.

Existe también una «List Box» en la que aparecerán los tiempos en los que comienzan y acaban las diferentes tareas acompañadas del «id» que les identifica. Esta tabla se borra y se vuelve a rellenar cada vez que se cambia la política de planificación.

Durante el uso de esta ventana se irán mostrando datos que pueden ser de interés como el hiperperiodo de las tareas o los parámetros que se introdujeron anteriormente.

Adicionalmente existen un temporizador que crea un retardo necesario para poder rellenar el cronograma, dos rectángulos que se hacen visible e invisibles para poder borrar el cronograma y un botón que permite cerrar la ventana.

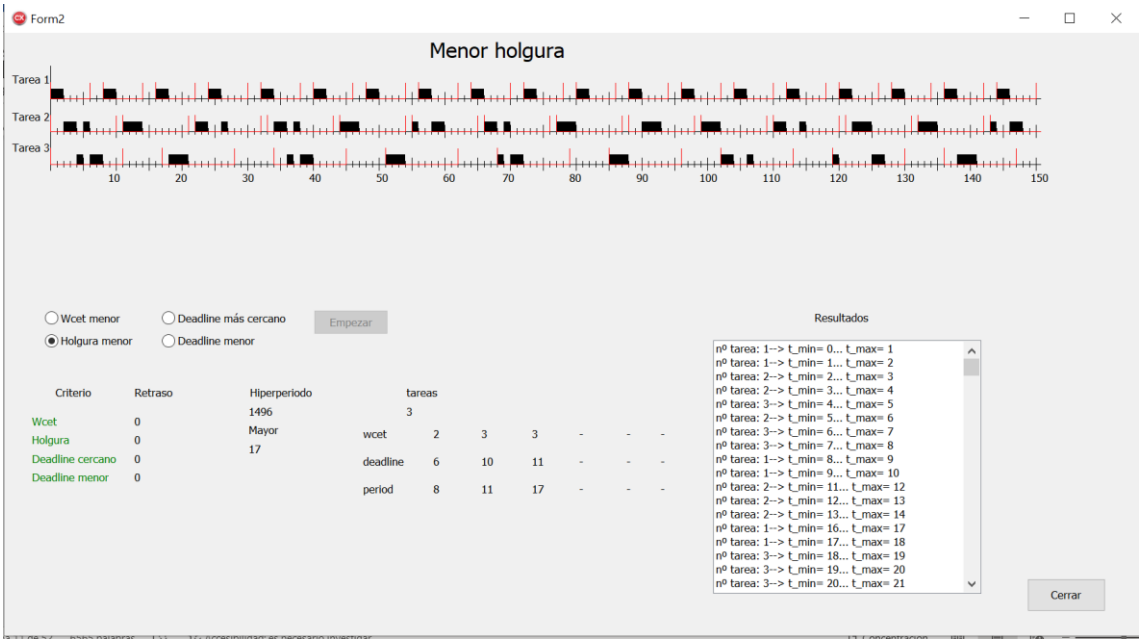


Imagen 8. Form2 en ejecución

Nombre	Tipo de elemento	Función
Form2	TForm	Es la ventana donde se encuentran el resto de elementos.
Shape_borrar	TShape	Aparece y desaparece para borrar el cronograma.
Shape_fin	TShape	Aparece y desaparece para borrar el final del cronograma para que el resultado sea más estético.
Timer1 (es diferente al de Form1)	TTimer	Genera un retardo necesario para que se pueda rellenar el cronograma.
RadioButton_dcercano	TRadioButton	Planifica las tareas por deadline cercano.
RadioButton_wcet	TRadioButton	Planifica las tareas por wcet menor.

RadioButton_dmenor	TRadioButton	Planifica las tareas por deadline menor.
RadioButton_holgura	TRadioButton	Planifica las tareas por holgura menor.
Button_empezar	TButton	Habilita todos los botones redondos y obtiene los datos del módulo anterior.
Button1	TButton	Cierra la ventana Form2.
Labelx60	TLabel	Muestran diferentes mensajes y datos de interés.
ListBox1	TListBox	Muestra una lista de todos los instantes en los que empiezan y acaban las tareas junto a su número de identificación.

Tabla 2. Elementos de Form2

4.2. Funciones

Se han implementado una serie de funciones para el correcto funcionamiento de la aplicación. Como se ha establecido con anterioridad, el lenguaje con el que se han programado estas funciones es C++. Estas funciones se pueden dividir en tres tipos: funciones básicas, de cálculo y de planificación. Esta es la parte principal del proyecto así que se le prestará especial atención. En este apartado se comentarán tanto los dos módulos comentados en el apartado anterior como el tercer módulo que conforma el programa, el módulo «Funciones_planificador».

Para optimizar el espacio en este documento, en este apartado no se mostrará el código del programa y se referirá a las funciones por su nombre. Para poder ver el código se deberá ir al apartado 6. Anexos.

4.2.1. Declaración de variables

Antes de comenzar con la exposición de las funciones se deben introducir las diferentes variables que se han sido declaradas para el desarrollo de la aplicación. En las tablas no se mostrarán las variables que se han declarado únicamente para realizar bucles, «switches» o similares. Tampoco se mencionarán la mayoría de variables que se repiten, incluso los que se repiten con nombres diferentes cuando son utilizadas en funciones diferentes.

Planificador.cpp

Nombre	Tipo	Valor inicial	Detalles
w[1…6]	int	0	Guarda los valores de los wcet.
d[1…6]	int	0	Guarda los valores de los deadline.
p[1…6]	int	0	Guarda los valores de los periodos.
ntareas	int	0	Guarda el valor del número de tareas.
Error	int	0	Cambia su valor a ‘1’ si hay errores en la introducción de datos.

Tabla 3. Variables Planificador.cpp

Cronogramas.cpp

Nombre	Tipo	Valor inicial	Detalles
ntareas	int	-	Guarda el valor del número de tareas.
Modo	int	-	Guarda un valor dependiendo de la política de planificación elegida.

wcet[6]	Int	-	Guarda los valores de los wcet. En este módulo los parámetros se guardan en vectores.
deadline[6]	Int	-	Guarda los valores de los deadline.
period[6]	int	-	Guarda los valores de los periodos.

Tabla 4. Variables Cronogramas.cpp

Funciones_planificador.cpp

Nombre	Tipo	Valor inicial	Detalles
Hiperperiodo	int	-	Guarda el valor del hiperperiodo
ntareas	int	-	Guarda el valor del número de tareas a planificar.
Wcet[6]	Int	-	Guarda los valores de los wcet. En este módulo los parámetros vuelven a guardarse en un vector.
Deadline[6]	Int	-	Guarda los valores de los deadline.
Period[6]	Int	-	Guarda los valores de los periodos.
Plan_wcet	Bool	-	Indica si el sistema es planificable con el criterio de wcet menor.
Plan_dc	Bool	-	Indica si el sistema es planificable con el criterio de deadline más cercano.
Plan_holgura	Bool	-	Indica si el sistema es planificable con el criterio de la holgura menor.
Plan_dm	bool	-	Indica si el sistema es planificable con el criterio de deadline menor.
Ret_wcet	Int	-	Recoge el valor de los retrasos que tiene el programa con el criterio del wcet menor.
Ret_dc	Int	-	Recoge el valor de los retrasos que tiene el programa con el criterio del deadline cercano.
Ret_holgura	Int	-	Recoge el valor de los retrasos que tiene el programa con el criterio de holgura menor.
Ret_dm	Int	-	Recoge el valor de los retrasos que tiene el programa con el criterio del deadline menor.
Aux	int	1580/(p*10)	Variable auxiliar para facilitar el dibujo del cronograma. La variable 'p' es el periodo.
maximo	Int	-	Guarda el máximo valor que puede alcanzar el máximo común divisor. Es una variable auxiliar.
Mayor	Int	-	Guarda el valor del periodo mayor.
aux_bug	int	-	Variable auxiliar para evitar bugs de corrupción de la memoria.

acabar	int	-	Variable que permite salir del bucle de cálculos una vez se hayan realizado todas las operaciones pertinentes.
a	int	-	Variable auxiliar.
b	int	-	Variable auxiliar.
Bloqueado[6]	int	-	Vector que permite identificar las tareas que ya han terminado de ser calculadas o que no están disponibles en un instante determinado.
orden [ntareas]	Int	-	Vector que guarda el orden en el que se ejecutan las tareas.
orden_empieza [ntareas]	Int	-	Vector que guarda los valores mínimos de los instantes en los que pueden empezar las tareas.
orden_acaba [ntareas]	int	-	Vector que guarda los valores máximos de los instantes en los que pueden acabar las tareas.
t	Int	0	Variable que guarda el valor del instante.
t_min	Int	0	Variable que guarda los valores de los instantes en los que empiezan las tareas.
t_max	int	0	Variable que guarda los valores de los instantes en los que acaban las tareas.
id	int	-	Variable que indica el número de identificación de tarea.
t_ocupado [hiperperiodo]	bool	-	Vector auxiliar que permite no repetir los instantes.
Menor	Int	999999	Variable auxiliar que permite identificar el menor de los valores para seleccionar la tarea que se va a calcular a continuacón.
Contador	Int	0	Variable auxiliar de contabilidad.
Holgura [ntareas]	int	-	Vector que guarda los valores de las holguras de las tareas en un instante determinado.
pendiente	int	-	Variable que guarda el valor de los instantes pendientes en ese periodo de la tarea.
Pendiente[ntareas]	int	-	Vector que guarda el valor de los instantes pendientes en ese periodo de la tarea. Para el cálculo según la holgura menor era necesario un vector y no una variable.
x1	Int	-	Variable que facilita el dibujo de los tiempos.
x2	Int	-	Variable que facilita el dibujo de los tiempos.
y1	Int	-	Variable que facilita el dibujo de los tiempos.

y2	Int	-	Variable que facilita el dibujo de los tiempos.

Tabla 5. Variables Funciones_planificador.cpp

4.2.2. Funciones básicas

En este grupo se recogen las funciones que habilitan y deshabilitan algunos elementos de la interfaz gráfica o realizan otras labores en el apartado visual.

Planificador.cpp

1-Función Button_ntareasClick: Esta función se ejecuta cuando se pulsa el botón «Button_ntareas». Esta función asigna a la variable «ntareas» el valor de «edit_tareas». Si el valor de «ntareas» es un número entero entre 1 y 6 se habilitarán los «edits» de los parámetros y el botón «Button_params». Si el valor no es válido aparecerá una alerta.

2-Función Button_paramsClick: Esta función se ejecuta cuando se pulsa el botón «Button_params». Esta función asigna a las variables de los parámetros el valor que ha ingresado el usuario. Adicionalmente, esta función comprueba que los valores ingresados tengan sentido y habilita «Button_iniciar». Si los valores no son válidos este botón no se habilitará y aparecerá un mensaje de alarma.

3-Función Button_iniciarClick: Esta función se ejecuta cuando se pulsa el botón «Button_iniciar». Esta función hace aparecer la ventana Cronogramas, enciende el temporizador de Form1 y llama a la función «mover_label(int)», donde se le envía el valor de «ntareas».

4-Función Button_reiniciarClick: Esta función se ejecuta cuando se pulsa el botón «Button_reiniciar». Esta función llama a la función «reiniciar()».

5-Función Timer1Timer: Esta función se ejecuta cuando en el temporizador Timer1 han transcurrido 100ms. Esta función deshabilita de nuevo el temporizador, llama a la función «pintar_cronograma(int)», a la que se le envía el valor de «ntareas» y finalmente, llama varias veces a la función «pintar_plazos(int, int, int)». Esta función será llamada tantas veces como sea el valor de «ntareas» y se le envían los valores de los «deadline», periodo y número de tarea a la que pertenecen esos parámetros.

Cronogramas.cpp

1-Función Button1Click: Esta función se ejecuta cuando se pulsa el botón «Button1» (botón cerrar). Esta función cierra la ventana de Cronogramas.

2- Función RadioButton_wcetClick: Esta función se ejecuta cuando se pulsa el botón redondo «RadioButton_wcet». Esta función borra el cronograma haciendo visible el rectángulo «Shape_borrar» y volviendo a hacerlo invisible al instante siguiente. También activa el temporizador Timer1 de Form1 para que el cronograma se vuelva a dibujar. Finalmente asigna el valor de '2' a la variable modo y activa el temporizador Timer1 de Form2.

3- Función RadioButton_holguraClick: Esta función se ejecuta cuando se pulsa el botón redondo «RadioButton_holgura». Esta función borra el cronograma haciendo visible el rectángulo «Shape_borrar» y volviendo a hacerlo invisible al instante siguiente. También activa el temporizador Timer1 de Form1 para que el cronograma se vuelva a dibujar. Finalmente asigna el valor de '3' a la variable modo y activa el temporizador Timer1 de Form2.

4- Función RadioButton_dcercanoClick: Esta función se ejecuta cuando se pulsa el botón redondo «RadioButton_dcercano». Esta función borra el cronograma haciendo visible el rectángulo «Shape_borrar» y volviendo a hacerlo invisible al instante siguiente. También activa el temporizador Timer1 de Form1 para que el cronograma se vuelva a dibujar. Finalmente asigna el valor de '4' a la variable modo y activa el temporizador Timer1 de Form2.

5-Función RadioButton_dmenorClick: Esta función se ejecuta cuando se pulsa el botón redondo «RadioButton_dmeno»r. Esta función borra el cronograma haciendo visible el rectángulo «Shape_borrar» y volviendo a hacerlo invisible al instante siguiente. También activa el temporizador Timer1 de Form1 para que el cronograma se vuelva a dibujar. Finalmente asigna el valor de '5' a la variable modo y activa el temporizador Timer1 de Form2.

6-Función Button_empezarClick: Esta función se ejecuta cuando se pulsa el botón «Button_empezar». Esta función habilita todos los botones redondos, deshabilita el botón «Button_empezar» y llama a las funciones «get_ntareas()», «get_wcet(int)», «get_deadline(int)» y «get_period(int)» y los guarda en las variables indicadas. Finalmente muestra por pantalla los valores obtenidos con los «gets».

7-Función Timer1Timer (Form2): Esta función se ejecuta cuando el temporizador Timer1 de Form2 ha realizado su intervalo de tiempo. Esta función llama a las funciones «planifica_wcet()», «planifica_holgura()», «planifica_dc()» y «planifica_dm()» dependiendo del valor de la variable «modo». Finalmente deshabilita el temporizador de Form2.

Funciones_planificador.cpp

1-Función void mover_label(int tareas): El cronograma tiene una serie de etiquetas que muestran la escala del mismo. Estas etiquetas se deben ajustar al cronograma dependiendo de la cantidad de tareas que se vayan a mostrar en el cronograma. Esta función mueve las etiquetas a su lugar indicado.

2-Función void reiniciar(void): Esta función devuelve a la mayoría de los elementos del programa a su estado inicial.

4.2.3. Funciones de cálculo

En este grupo se recogen las funciones que se encargan de los cálculos matemáticos y transmisión de información entre los diferentes módulos.

Planificador.cpp

1-Función int get_ntareas(void): Esta función devuelve el valor de «ntareas». Necesario para compartir el valor de esta variable entre módulos.

2-Función int get_wcet(int tarea): Esta función devuelve el valor del «wcet» de las diferentes tareas. Necesario para compartir el valor de estos parámetros entre módulos.

3-Función int get_deadline(int tarea): Esta función devuelve el valor del «deadline» de las diferentes tareas. Necesario para compartir el valor de estos parámetros entre módulos.

4-Función int get_period(int tarea): Esta función devuelve el valor del periodo de las diferentes tareas. Necesario para compartir el valor de estos parámetros entre módulos.

Funciones_planificador.cpp

1-Función void pintar_cronograma(int tareas): Esta función recibe el número de tareas. Dependiendo de este valor se realiza un bucle con un «switch» dentro de tal forma que cada vez que se realiza el bucle se dibuja una línea del cronograma diferente. Estas líneas se han dibujado mediante los comandos «MoveTo(x,y)» y «LineTo(x,y)» donde en el primer comando se escribe el punto cartesiano donde se desea que empiece la línea y el segundo, el punto donde acaba. También hace visibles los títulos del cronograma según el número de tareas escogido. Esta función se podría considerar como básica o como de cálculo.

```
//---------------------------------------------------------------------------
void pintar_cronograma(int tareas)
{
for(int i=1;i<tareas+1;i++){
switch (i) {
  case(1):{
  Form2->Label_t1->Visible=True;

  Form2->Canvas->MoveTo(70, 50);
```

```
        Form2->Canvas->LineTo(70, 100);
        Form2->Canvas->LineTo(1580, 100);

        for (int a = 0; a < 151; a++) {
            Form2->Canvas->MoveTo(70+(10*a), 95);
            Form2->Canvas->LineTo(70+(10*a), 105);
            if (a%5==0) {
                Form2->Canvas->MoveTo(70+(10*a), 90);
                Form2->Canvas->LineTo(70+(10*a), 110);
            }
        }
        }
        break;

        case(2):{
        Form2->Label_t2->Visible=True;

        Form2->Canvas->MoveTo(70, 100);
        Form2->Canvas->LineTo(70, 150);
        Form2->Canvas->LineTo(1580, 150);

        for (int a = 0; a < 151; a++) {
            Form2->Canvas->MoveTo(70+(10*a), 145);
            Form2->Canvas->LineTo(70+(10*a), 155);
            if (a%5==0) {
                Form2->Canvas->MoveTo(70+(10*a), 140);
                Form2->Canvas->LineTo(70+(10*a), 160);
            }
        }
        }
        break;

        case(3):{
        Form2->Label_t3->Visible=True;

        Form2->Canvas->MoveTo(70, 150);
        Form2->Canvas->LineTo(70, 200);
        Form2->Canvas->LineTo(1580, 200);

        for (int a = 0; a < 151; a++) {
            Form2->Canvas->MoveTo(70+(10*a), 195);
            Form2->Canvas->LineTo(70+(10*a), 205);
            if (a%5==0) {
                Form2->Canvas->MoveTo(70+(10*a), 190);
                Form2->Canvas->LineTo(70+(10*a), 210);
            }
        }
        }
        break;

        case(4):{
        Form2->Label_t4->Visible=True;

        Form2->Canvas->MoveTo(70, 200);
```

```
        Form2->Canvas->LineTo(70, 250);
        Form2->Canvas->LineTo(1580, 250);

        for (int a = 0; a < 151; a++) {
            Form2->Canvas->MoveTo(70+(10*a), 245);
            Form2->Canvas->LineTo(70+(10*a), 255);
            if (a%5==0) {
                Form2->Canvas->MoveTo(70+(10*a), 240);
                Form2->Canvas->LineTo(70+(10*a), 260);
            }
        }
        }
        break;

        case(5):{
        Form2->Label_t5->Visible=True;

        Form2->Canvas->MoveTo(70, 250);
        Form2->Canvas->LineTo(70, 300);
        Form2->Canvas->LineTo(1580, 300);

        for (int a = 0; a < 151; a++) {
            Form2->Canvas->MoveTo(70+(10*a), 295);
            Form2->Canvas->LineTo(70+(10*a), 305);
            if (a%5==0) {
                Form2->Canvas->MoveTo(70+(10*a), 290);
                Form2->Canvas->LineTo(70+(10*a), 310);
            }
        }
        }
        break;

        case(6):{
        Form2->Label_t6->Visible=True;

        Form2->Canvas->MoveTo(70, 300);
        Form2->Canvas->LineTo(70, 350);
        Form2->Canvas->LineTo(1580, 350);

        for (int a = 0; a < 151; a++) {
            Form2->Canvas->MoveTo(70+(10*a), 345);
            Form2->Canvas->LineTo(70+(10*a), 355);
            if (a%5==0) {
                Form2->Canvas->MoveTo(70+(10*a), 340);
                Form2->Canvas->LineTo(70+(10*a), 360);
            }
        }
        }
        break;
    }
    }

}
//---------------------------------------------------------------------------
```

Imagen 9. Función pintar_cronogramas

2-Función void pintar_plazos(int d, int p, int tareas): Esta función recibe los valores del «deadline», periodo y número de tarea cuyos plazos se deben dibujar. Mediante los mismos comandos que se han utilizado en la función «pintar_cronograma» se dibujan en rojo en el cronograma los plazos en los cuales se pueden realizar las tareas. De la misma forma que la función anterior, esta función también se puede considerar de cálculo o básica.

```
void pintar_plazos (int d, int p, int tareas)
{
   int aux=1580/(p*10);

   for (int i=0; i < aux+1; i++) {

   Form2->Canvas->Pen->Color=clRed;
   Form2->Canvas->MoveTo(70+(p*i*10), 75+(50*(tareas-1)));
   Form2->Canvas->LineTo(70+(p*i*10), 100+(50*(tareas-1)));

    if ((70+(p*i*10)+(d*10))<1580) {
     Form2->Canvas->LineTo(70+(p*i*10)+(d*10), 100+(50*(tareas-1)));
     Form2->Canvas->LineTo(70+(p*i*10)+(d*10),75+(50*(tareas-1)));
    }
    else
    {
    Form2->Canvas->LineTo(1580, 100+(50*(tareas-1)));
    i=aux;
    }

   }
  Form2->Shape_fin->Visible=True;
  Form2->Shape_fin->Visible=False;
  Form2->Canvas->Pen->Color=clBlack;
}
```

Imagen 10. Función pintar_plazos

3-Función void set_ceros(void): Esta función hace que todas las celdas de los vectores de los parámetros se igualen a ‘0’. Se ha creado esta función de manera independiente a la de reinicio para ahorrar espacio de código porque se usará para que otras funciones de cálculo no den errores.

4-Función int calcula_hiperperiodo(int n, int a, int b, int c, int d, int e, int f): Esta función recibe el número de tareas y sus valores de los periodos. Gracias a la función set_ceros esta función funcionará correctamente aunque el número de tareas no sea ‘6’. La función calcula el mínimo común múltiplo de los periodos y lo devuelve.

```
//---------------------------------------------------------------------------
int calcula_hiperperiodo (int n, int a, int b, int c, int d, int e, int f)
{
int hiperperiodo=1, aux, maximo, mayor;
//numero mayor

  if (a>=b) {mayor=a;}
  else {mayor=b;}
  if (c>=mayor) { mayor=c;}
  if (d>=mayor) { mayor=d;}
  if (e>=mayor) { mayor=e;}
  if (f>=mayor) { mayor=f;}

  Form2->Label_mayor->Caption=mayor;
//mcm
  switch (n)
  {
     case(1):
     {
       hiperperiodo=a;
     }
     break;
     case(2):
     {
       maximo=a*b;
       for (int i=1; i<=maximo; i++) {
          if (i%a==0 && i%b==0 && i>=mayor) {
             hiperperiodo=i;
             break;
          }
       }
      }
     break;
     case(3):  {
       maximo=a*b*c;
       for (int i=1; i<=maximo; i++) {
          if (i%a==0 && i%b==0 && i%c==0 && i>=mayor) {
             hiperperiodo=i;
             break;
          }
       }
     }
     break;
     case(4):  {
       maximo=a*b*c*d;
       for (int i=1; i<=maximo; i++) {
          if (i%a==0 && i%b==0 && i%c==0 && i%d==0 && i>=mayor) {
             hiperperiodo=i;
             break;
          }
       }
     }
     break;
     case(5):  {
        maximo=a*b*c*d*e;
       for (int i=1; i<=maximo; i++) {
          if (i%a==0 && i%b==0 && i%c==0 && i%d==0 && i%e==0 && i>=mayor) {
             hiperperiodo=i;
             break;
          }
       }
     }
     break;
```

```
      case(6):  {
380      maximo=a*b*c*d*e*f;
         for (int i=1; i<=maximo; i++) {
            if (i%a==0 && i%b==0 && i%c==0 && i%d==0 && i%e==0 && i%f==0 && i>=mayor)
               hiperperiodo=i;
               break;
            }
         }
      }
      break;
   }
390 Form2->Label6->Caption=hiperperiodo;
   return hiperperiodo;
}
```

Imagen 11. Función calcula_hiperperiodo

5-Función void pintar_tiempos(int id, int empieza, int acaba): Esta función genera unos rectángulos negros mediante el comando «Rectangle(x1,y1,x2,y2)». El plano cartesiano de este comando es el mismo que el usado con otras funciones, por lo que es sencillo calcular estos cuatro puntos.

```
void pintar_tiempos (int id, int empieza, int acaba)
900 {
int x1, x2, y1, y2;
 if (acaba<160) {
 Form2->Canvas->Brush->Color=clBlack;
 x1=70+(10*empieza);
 x2=70+(10*acaba);
 y1=35+(id*50);
 y2=50+(id*50);
 Form2->Canvas->Rectangle(x1,y1,x2,y2);
 }
910 }
```

Imagen 12. Función pintar_tiempos

4.2.4. Funciones de planificación

En este grupo se recogen las funciones dedicadas a la planificación. Son funciones mucho más complejas que las de grupos anteriores y son la parte principal del programa. Estas funciones se encuentran únicamente en el módulo «Funciones_planificador». Debido a la complejidad de estas funciones se comentarán únicamente las partes más importantes de las funciones.

Funciones_planificador.cpp

1-Función void planifica_dc(void): Para comenzar, esta función obtiene los parámetros de las tareas y los guarda en vectores mediante un bucle. Se crean varios vectores y se les dan valores cualesquiera para que no den errores más adelante o en caso de que se use el programa más de una vez. Se genera un bucle que calcula los instantes en los que empiezan y acaban las tareas para a continuación ordenarlo de tal manera que tengan preferencia las tareas con el plazo más cercano. Una vez conociendo el «id» de la tarea

que tiene prioridad se calculan los tiempos en los que se debe ejecutar la tarea. Los cálculos de los tiempos se realizan de la siguiente manera:

Se buscará el instante disponible más pequeño y se asignará este valor a la variable «t_min» mientras que a la variable «t_max» se le asignará el valor de «t_min+1». Una vez hecho esto se pueden presentar tres escenarios. El primero es que no queden más instantes pendientes por calcular en el plazo de la tarea, en este caso sencillamente el cálculo acabaría y se volvería al bucle de selección de prioridades. El segundo es que quedan instantes pendientes por calcular y el siguiente instante está disponible. Si se da esta situación el programa aumenta en '1' el valor de «t_max» y volvería al mismo estado en el que se plantean estas tres situaciones. Por último, se puede dar el escenario en el que queden instantes pendientes de calcular pero el instante siguiente no esté disponible. En este escenario la función da por finalizado momentáneamente el cálculo de forma similar al primer escenario con la diferencia en que buscará el siguiente instante disponible para continuar con el cálculo. Los instantes disponibles se conocen gracias al vector booleano «t_ocupado[hiperperiodo]» que cambia el estado de sus celdas a «True» cuando un instante deja de estar disponible. Los resultados de estos cálculos se podrán ver en la «ListBox» de «Form2».

Si en algún momento del cálculo una tareas no ha podido realizar en su periodo todos los instantes de su tiempo de cómputo se considerará que el sistema no es planificable y se mostrarán los retrasos que ha habido a lo largo del hiperperiodo.

```
oid planifica_dc(void){

//DECLARACIONES
int ntareas, contador=0, acabar;
int hiperperiodo, aux, aux_bug;

int menor=999999, a, b;
int bloqueado[6];
int salir=0;

int ret_dc=0, t=0, pendiente=0;
bool plan_dc=true;
int t_min=0, t_max=0, id;

//OBTENER DATOS
set_ceros();
ntareas=get_ntareas();
 for(int i=1;i<ntareas+1;i++){
   wcet[i-1]=get_wcet(i);
   deadline[i-1]=get_deadline(i);
   period[i-1]=get_period(i);
 }
 aux_bug=period[0];
 hiperperiodo=calcula_hiperperiodo(ntareas,period[0],period[1],period[2],
period[3],period[4],period[5]);
  int orden[ntareas], orden_empieza[ntareas], orden_acaba[ntareas];
  bool t_ocupado[hiperperiodo];
 //PLANIFICACION
 //deadline cercano

 //inizializar vectores y matrices
   for (int i=0; i < ntareas; i++) {
      orden[i]=0;
      orden_empieza[i]=0;
      orden_acaba[i]=0;
      bloqueado[i]=0;
   }

    for (int i=0; i <= hiperperiodo; i++) {
      t_ocupado[i]=false;
    }

    //ORDENAR
    b=0;
  while (salir==0){
   menor=999999;
   for (int i=0; i < ntareas; i++) {

     if (deadline[i]+period[i]*orden[i]<=hiperperiodo) {
       orden_acaba[i]=deadline[i]+period[i]*orden[i];
       orden_empieza[i]=orden_acaba[i]-deadline[i];
     }
     else{
       bloqueado[i]=1;

       salir=1;
       for (int i=0; i < ntareas; i++) {
        if (bloqueado[i]==0) {
         salir=0;
        }
       }

     }

     if (orden_acaba[i]<menor && bloqueado[i]==0) {
```

```
            menor=orden_acaba[i];
            a=i;
          }
        }
        orden[a]=orden[a]+1;

        //-----------------------------------------------------
         //Tiempos

         if (salir==0) {
          id=a;
          t=orden_empieza[a];
          pendiente=wcet[id];

          while (pendiente>0 && t<=orden_acaba[a])    {
           if (t!=orden_acaba[a]) {
            if (t_ocupado[t]==false) {
             t_min=t;
             t_ocupado[t]=true;
             t++;
             t_max=t;
             pendiente--;
             while (pendiente>0 && t_ocupado[t]==false) {
              t_max=t+1;
              t_ocupado[t]=true;
              pendiente--;
              t++;
             }
             //cout<<"\n id= "<<id<<"...t_min= "<<t_min<<"...t_max= "<<t_max;
             pintar_tiempos(id+1, t_min, t_max);
             Form2->ListBox1->Items->Add("n° tarea: "+ IntToStr(id+1)+ "--> t_min= "
             + IntToStr(t_min)+"... t_max= "+ IntToStr(t_max));
            }
            else{
             t++;
            }
           }
           else{
            t++;
           }
          }

          if (t>orden_acaba[a]) {
           if (pendiente>0) {
            plan_dc=false;
            ret_dc=ret_dc+pendiente;
           }

          }
         }
        }

       Form2->Label_r4->Caption=ret_dc;
      if (plan_dc==true) {

       //cout<<"\n"<<"SISTEMA PLANIFICABLE"   ;
       Form2->Label22->Font->Color=clGreen;
      }
      else{
        //cout<<"\n"<<"SISTEMA NO PLANIFICABLE...RETRASO DE "<<ret_dc   ;
        Form2->Label22->Font->Color=clRed;
      }
```

Imagen 13. Función planifica_dc

2-Función void planifica_dm(void): Esta función es casi idéntica a la anterior y funciona de la misma forma, sólo cambian el nombre de algunas variables para adecuarlas a lo que se está calculando y el bucle que selecciona las tareas según la prioridad que deben tener escoge a las tareas con el «deadline» menor en lugar del que tenga el «deadline» más cercano.

```
770 oid planifica_dm(void){

    /DECLARACIONES
    int ntareas, contador=0, acabar;
    int hiperperiodo, aux, aux_bug;

    int menor=999999, a, b;
    int bloqueado[6];
    int salir=0;

780 int ret_dm=0, t=0, pendiente=0;
    bool plan_dm=true;
    int t_min=0, t_max=0, id;

    //OBTENER DATOS
    set_ceros();
ntareas=get_ntareas();
 for(int i=1;i<ntareas+1;i++){
   wcet[i-1]=get_wcet(i);
   deadline[i-1]=get_deadline(i);
790 period[i-1]=get_period(i);
 }
 aux_bug=period[0];
 hiperperiodo=calcula_hiperperiodo(ntareas,period[0],period[1],period[2],
794 period[3],period[4],period[5]);
 bool t_ocupado[hiperperiodo];
 int orden[ntareas], orden_empieza[ntareas], orden_acaba[ntareas];
 //PLANIFICACION
 //deadline menor

800 //inizializar vectores y matrices
   for (int i=0; i < ntareas; i++) {
      orden[i]=0;
      orden_empieza[i]=0;
      orden_acaba[i]=0;
      bloqueado[i]=0;
   }

   for (int i=0; i <= hiperperiodo; i++) {
     t_ocupado[i]=false;
810 }

    //ORDENAR
```

```
 b=0;
while (salir==0){
 menor=999999;
 for (int i=0; i < ntareas; i++) {

   if (deadline[i]+period[i]*orden[i]<=hiperperiodo) {
     orden_acaba[i]=deadline[i]+period[i]*orden[i];
     orden_empieza[i]=orden_acaba[i]-deadline[i];
   }
   else{
     bloqueado[i]=1;
     salir=1;
     for (int i=0; i < ntareas; i++) {
      if (bloqueado[i]==0) {
       salir=0;
      }
     }

   }

   if (deadline[i]<menor && bloqueado[i]==0) {
     menor=deadline[i];
     a=i;
   }
 }
 orden[a]=orden[a]+1;

 //-----------------------------------------------------
  //Tiempos

  if (salir==0) {
   id=a;
   t=orden_empieza[a];
   pendiente=wcet[id];

   while (pendiente>0 && t<=orden_acaba[a])    {
    if (t!=orden_acaba[a]) {
     if (t_ocupado[t]==false) {
      t_min=t;
      t_ocupado[t]=true;
      t++;
      t_max=t;
      pendiente--;
      while (pendiente>0 && t_ocupado[t]==false) {
       t_max=t+1;
       t_ocupado[t]=true;
       pendiente--;
       t++;
      }
      //cout<<"\n id= "<<id<<"...t_min= "<<t_min<<"...t_max= "<<t_max;
      pintar_tiempos(id+1, t_min, t_max);
      Form2->ListBox1->Items->Add("n° tarea: "+ IntToStr(id+1)+ "--> t_min= "
      + IntToStr(t_min)+"... t_max= "+ IntToStr(t_max));
     }
```

```
            else{
              t++;
            }
          }
          else{
            t++;
          }
        }

        if (t>orden_acaba[a]) {
          if (pendiente>0) {
            plan_dm=false;
            ret_dm=ret_dm+pendiente;
          }

        }
      }
    }

Form2->Label_r5->Caption=ret_dm;
if (plan_dm==true) {
 //cout<<"\n"<<"SISTEMA PLANIFICABLE"   ;
 Form2->Label23->Font->Color=clGreen;
}
else{
  //cout<<"\n"<<"SISTEMA NO PLANIFICABLE...RETRASO DE "<<ret_dm    ;
  Form2->Label23->Font->Color=clRed;
```

Imagen 14. Función planifica_dm

3-Función void planifica_wcet(void): Esta función, de la misma forma que la anterior, hace lo mismo que «planifica_dc» pero modificando los nombres y el bucle de selección de prioridades para escoger las tareas con el tiempo de cómputo menor.

```
//-------------------------------------------------------------------------
void planifica_wcet(void){

//DECLARACIONES
int ntareas, contador=0, acabar;
int hiperperiodo, aux, aux_bug;

int menor=999999, a, b;
int bloqueado[6];
int salir=0;
int ret_wcet=0, t=0, pendiente=0;
bool plan_wcet=true;
int t_min=0, t_max=0, id;

//OBTENER DATOS
set_ceros();
ntareas=get_ntareas();
 for(int i=1;i<ntareas+1;i++){
   wcet[i-1]=get_wcet(i);
   deadline[i-1]=get_deadline(i);
   period[i-1]=get_period(i);
 }
 aux_bug=period[0];
 hiperperiodo=calcula_hiperperiodo(ntareas,period[0],period[1],period[2],period[3],
 period[4],period[5]);
 bool t_ocupado[hiperperiodo];
 int orden[ntareas], orden_empieza[ntareas], orden_acaba[ntareas];
 //PLANIFICACION
 //wcet menor

 //inizializar vectores y matrices
   for (int i=0; i < ntareas; i++) {
      orden[i]=0;
      orden_empieza[i]=0;
      orden_acaba[i]=0;
      bloqueado[i]=0;
   }
   for (int i=0; i <= hiperperiodo; i++) {
     t_ocupado[i]=false;
   }

   //ORDENAR
   b=0;
 while (salir==0){
  menor=999999;
  for (int i=0; i < ntareas; i++) {

    if (deadline[i]+period[i]*orden[i]<=hiperperiodo) {
      orden_acaba[i]=deadline[i]+period[i]*orden[i];
      orden_empieza[i]=orden_acaba[i]-deadline[i];
    }
    else{
      bloqueado[i]=1;

      salir=1;
      for (int i=0; i < ntareas; i++) {
       if (bloqueado[i]==0) {
        salir=0;
       }
      }

    }

    if (wcet[i]<menor && bloqueado[i]==0) {
```

```
        menor=wcet[i];
        a=i;
      }
    }
    orden[a]=orden[a]+1;

    //-----------------------------------------------------
     //Tiempos

    if (salir==0) {

     id=a;
     t=orden_empieza[a];
     pendiente=wcet[id];

     while (pendiente>0 && t<=orden_acaba[a])    {
      if (t!=orden_acaba[a]) {
       if (t_ocupado[t]==false) {
        t_min=t;
        t_ocupado[t]=true;
        t++;
        t_max=t;
        pendiente--;
        while (pendiente>0 && t_ocupado[t]==false) {
         t_max=t+1;
         t_ocupado[t]=true;
         pendiente--;
         t++;
        }
        //cout<<"\n id= "<<id<<"...t_min= "<<t_min<<"...t_max= "<<t_max;
        pintar_tiempos(id+1, t_min, t_max);
        Form2->ListBox1->Items->Add("n° tarea: "+ IntToStr(id+1)+ "--> t_min= "
        + IntToStr(t_min)+"... t_max= "+ IntToStr(t_max));
       }
       else{
        t++;
       }
      }
      else{
       t++;
      }
     }

     if (t>orden_acaba[a]) {
      if (pendiente>0) {
       plan_wcet=false;
       ret_wcet=ret_wcet+pendiente;
      }

     }
    }
   }

  Form2->Label_r2->Caption=ret_wcet;

  if (plan_wcet==true) {
   //cout<<"\n"<<"SISTEMA PLANIFICABLE"   ;
   Form2->Label20->Font->Color=clGreen;
  }
  else{
    //cout<<"\n"<<"SISTEMA NO PLANIFICABLE...RETRASO DE "<<ret_dm    ;
    Form2->Label20->Font->Color=clRed;
  }
```

Imagen 15. Función planifica_wcet

4-Función void planifica_holgura (void): Esta función es la única que funciona de forma diferente a las demás debido a que los cálculos se realizan por otro método. Ya de principio se puede observar que la variable «pendiente» se ha convertido en un vector de «ntareas» celdas y que se ha creado otro vector que guarda el valor de las holguras. Esta función está compuesta únicamente por un bucle que va de '0' a «hiperperiodo-1». A cada instante se calculará la holgura de las tareas y se seleccionará la menor de ellas. En este proceso de selección existen maneras de bloquear las tareas para que no estén disponibles si ya no tienen instantes pendientes por ser calculados o por si el instante en el que se está realizando el cálculo se encuentra entre el «deadline» y el periodo de la tarea. Como este criterio de planificación exige un cálculo por cada instante todos los «t_max» tendrán un valor de «t_min+1», siendo «t_min» el valor del instante calculado. Finalmente, de la misma forma que en las funciones anteriores, si una tarea no consigue completar todos los instantes de su tiempo de cómputo en su periodo se considerará el sistema como no planificable y se mostrará de la misma forma que con el resto de criterios de planificación.

```
oid planifica_holgura(void){

//DECLARACIONES
int ntareas, contador=0, acabar;
int hiperperiodo, aux, aux_bug;

int menor=999999, a, b;
int bloqueado[6];
int salir=0;

int ret_holgura=0, t=0;
bool plan_holgura=true;
int t_min=0, t_max=0, id;

//OBTENER DATOS
set_ceros();
ntareas=get_ntareas();
 for(int i=1;i<ntareas+1;i++){
   wcet[i-1]=get_wcet(i);
   deadline[i-1]=get_deadline(i);
   period[i-1]=get_period(i);
 }
 aux_bug=period[0];
 hiperperiodo=calcula_hiperperiodo(ntareas,period[0],period[1],period[2],
 period[3],period[4],period[5]);
 bool t_ocupado[hiperperiodo];
 int orden[ntareas], orden_empieza[ntareas], orden_acaba[ntareas];
 int pendiente[ntareas], holgura[ntareas];

//PLANIFICACION
 //holgura

 //inizializar vectores y matrices
   for (int i=0; i < ntareas; i++) {
      orden[i]=0;
      orden_empieza[i]=0;
      orden_acaba[i]=0;
      bloqueado[i]=0;
      pendiente[i]=wcet[i];
   }

   for (int i=0; i <= hiperperiodo; i++) {
```

```
    t_ocupado[i]=false;
   }

//-----------------------------------------
  while (t<=hiperperiodo-1){

   for (int i=0; i < ntareas; i++) {
     if (t>=orden_empieza[i]+period[i] && t!=0) {
      orden[i]=orden[i]+1;
     }

     if (t==orden_empieza[i]+period[i]) {
      pendiente[i]=wcet[i];
     }

   }

   menor=9999999;
   for (int i=0; i < ntareas; i++) {
    orden_acaba[i]=deadline[i]+period[i]*orden[i];
    orden_empieza[i]=orden_acaba[i]-deadline[i];
    holgura[i]=deadline[i]-pendiente[i]-t;

    if (holgura[i]<menor && pendiente[i]>0 && t<orden_acaba[i]) {
      menor=holgura[i];
      a=i;
    }
   }

   if (menor!=9999999) {
    id=a;
   }

   if (pendiente[a]>0 && t>=orden_empieza[a]) {
    t_min=t;
    t_max=t+1;
    //cout<<"\n id= "<<id<<"...t_min= "<<t_min<<"...t_max= "<<t_max;
    pintar_tiempos(id+1, t_min, t_max);
    Form2->ListBox1->Items->Add("n° tarea: "+ IntToStr(id+1)+ "--> t_min= "
    + IntToStr(t_min)+"... t_max= "+ IntToStr(t_max));

    t++;
    pendiente[a]=pendiente[a]-1;
   }
   else{
    t++;
   }

   for (int i=0; i < ntareas; i++) {
    if (t>orden_acaba[i] && pendiente[i]>0) {
       ret_holgura=ret_holgura+pendiente[i];
       plan_holgura=false;
       pendiente[i]=0;
    }
```

```
    }
  }

//-------------------------------------------------
Form2->Label_r3->Caption=ret_holgura;
if (plan_holgura==true) {
 //cout<<"\n"<<"SISTEMA PLANIFICABLE"   ;
 Form2->Label21->Font->Color=clGreen;
}
else{
  //cout<<"\n"<<"SISTEMA NO PLANIFICABLE...RETRASO DE "<<ret_dm    ;
  Form2->Label21->Font->Color=clRed;
}
```

Imagen 16. Función planifica_holgura

4.3. Obtención de resultados

Para comprobar si el sistema es planificable se puede prestar atención al color del texto de los criterios de planificación. Si el texto está verde el sistema será planificable con esa política de planificación, en caso de que sea rojo el sistema no será planificable. Además, si el sistema no es planificable aparecerá el tiempo de retrasos que tendrá el sistema durante un hiperperiodo.

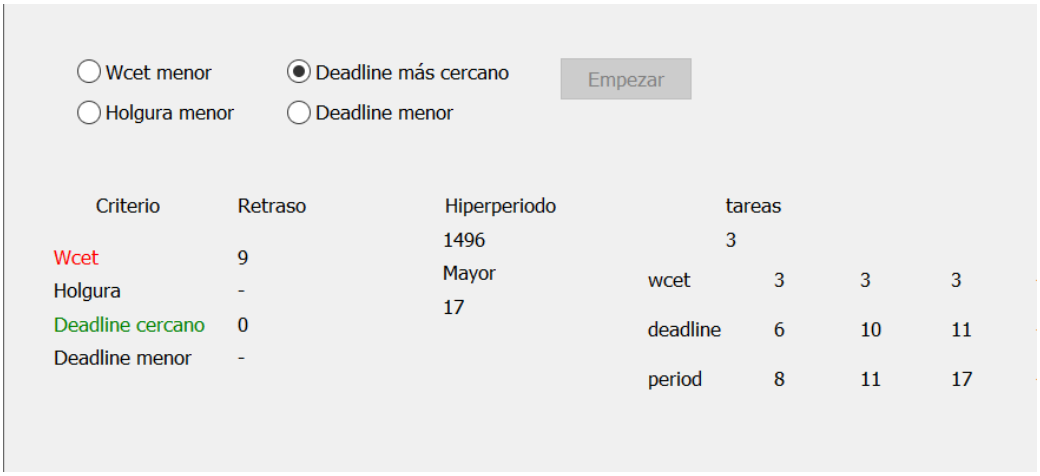


Imagen 17. Resultados de la planificación.

En la parte superior de la ventana se observará que el cronograma se ha rellenado. Los rectángulos negros que aparecen sobre los cronogramas representan los instantes en los que las tareas están activas y están escalados para que coincidan exactamente con las líneas que marcan los instantes. El cronograma sólo muestra hasta un máximo de 150 instantes y, en el caso de que una tarea no pueda realizar todos los instantes de su «wcet», los tiempos que queden fuera de plazo serán ignorados y no estarán representados.

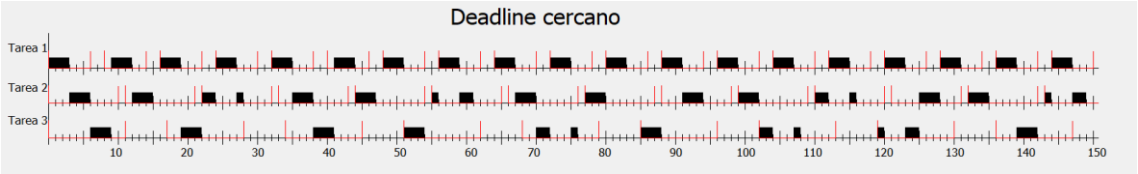


Imagen 18. Cronograma relleno.

Originalmente se pretendía generar una tabla de formato .xml donde se recogieran los tiempos en los que se activan y desactivan las tareas y su id (número de tarea). Sin embargo, finalmente se ha optado por otro sistema. En lugar de crear un documento .xml se ha optado por crear un cajón en la ventana en el que se enumerarán todos los tiempos con su id como se muestra en la siguiente imagen:

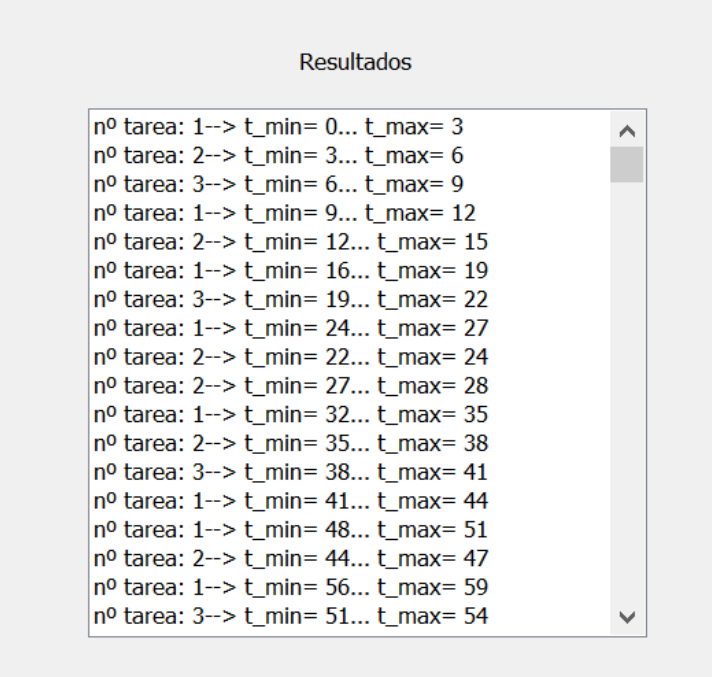


Imagen 19. Listado de tiempos

Los tiempos del listado irán apareciendo según sean calculados por lo que si utilizamos el criterio de tiempo de cómputo menor los primeros tiempos en aparecer en el listado serán siempre los del «wcet» menor, mientras que si escogemos la opción del «deadline» más cercano como es el caso del ejemplo, las tareas estarán entremezacladas.

5. Pliego de condiciones

5.1. Objeto

La presente especificación se refiere al conjunto de trabajos necesarios para el correcto uso del software «Planificador de tareas para sistemas de tiempo real».

5.2. Materiales

El dispositivo con el que se ejecute el software requiere de los siguientes requerimientos:

- Sistema Operativo Windows 98 o superior
- Mínimo de 32 Mb en RAM
- Resolución gráfica mínima de 640 x 480 pixel

5.3. Normas de ejecución

La ejecución de la aplicación se realizará preferiblemente de la siguiente forma:

1. Se desactivará el antivirus o como mínimo la función de análisis en tiempo real de éste (recomendado).
2. Se abrirá la aplicación.
3. Se ingresará el número de tareas que se desean planificar (máx. 6) y se pulsará el botón de aceptar.
4. Se ingresarán los parámetros de las tareas teniendo en cuenta que los valores de los «wcet» deben ser menores o iguales a los de los «deadline» y estos, a su vez, deben de ser menores o iguales a los periodos. A continuación, se pulsará el segundo botón de aceptar y posteriormente el de planificar.
5. Se pulsará el botón de empezar.
6. Se elegirá el criterio de planificación deseado.
7. Se obtendrán los datos deseados que devuelve el programa.
8. Se pulsarán los botones de cerrar y reiniciar.

En caso de querer ejecutar la aplicación en otro orden se deberá consultar el diagrama de flujo de la aplicación o el manual de usuario.

5.4. Pruebas y ajustes

Se recomienda antes de su uso realizar unas pruebas para comprobar el correcto funcionamiento de la aplicación. Esta prueba deberá comprobar tanto que los elementos de la aplicación se ejecutan de la forma correcta tanto los resultados que muestra, por lo que se recomienda realizar la prueba con sistemas sencillos de los que ya se conozca la planificación.

5.5. Condiciones de uso, mantenimiento y seguridad

La aplicación se ejecutará siempre preferiblemente con el antivirus desactivado y con sistema operativo Windows98 o superior. En caso de que se ejecute el programa con el antivirus activo existe la posibilidad de que éste lo detecte como un malware y elimine la aplicación del dispositivo. Para recuperar la aplicación se deberá restaurar desde la sección «elementos en cuarentena» del antivirus o volver a instalar la aplicación.

No se requiere de ninguna licencia especial para el uso de la aplicación.

6. Presupuesto

6.1. Precios unitarios

- Hora de programador 20.00€
- Amortización del ordenador portátil Lenovo por hora* 0.82€/h
- Licencia C++Builder (Versión de prueba) 0.00€

*Este precio se obtiene al dividir el coste total del ordenador ya amortizado entre el total de horas de trabajo.

- Precio del portátil 950€
- Amortización según Hacienda 26%
- Horas totales de trabajo 300h

$$\frac{950€ * 26\%}{300h} = 0.82€/h$$

6.2. Precios descompuestos

Programación del código "Planificador" mediante C++ builder				
Cód	Descripción	Nº	Precio	Total
C1	H. de programador	3	20,00€	60,00€
C2	Amort. portátil	3	0,82€	2,47€
	% Costes directos	3%	62,47€	1,87€
				64,34€

Tabla 6. Presupuesto Programación del código de “Planificador”

Programación del código "Cronograma" mediante C++ builder				
Cód	Descripción	Nº	Precio	Total
C1	H. de programador	4	20,00€	80€
C2	Amort. portátil	4	0,82€	3,29€
	% Costes directos	3%	83,29€	2,50€
				85,79€

Tabla 7. Presupuesto Programación del código de “Cronograma”

Programación del código "Funciones_planificador" mediante C++ builder				
Cód	Descripción	Nº	Precio	Total
C1	H. de programador	150	20,00€	3.000,00€
C2	Amort. portátil	150	0,82€	123,45€
	% Costes directos	3%	3.123,45€	93,70€
				3.217,15€

Tabla 8. Presupuesto Programación del código de “Funciones_lanificador”

Diseño de interfaz "Planificador" mediante C++ builder				
Cód	Descripción	Nº	Precio	Total
C1	H. de programador	1	20,00€	20,00€
C2	Amort. Portátil	1	0,82€	0,82€
	% Costes directos	3%	20,82€	0,62€
				21,45€

Tabla 9. Presupuesto diseño interfaz “Planificador”

Diseño de interfaz "Cronograma" mediante C++ builder				
Cód	Descripción	Nº	Precio	Total
C1	H. de programador	2	20,00€	40,00€
C2	Amort. Portátil	2	0,82€	1,65€
	% Costes directos	3%	41,65€	1,25€
				42,90€

Tabla 10. Presupuesto diseño interfaz “Cronogramas”

Corrección de errores				
Cód	Descripción	Nº	Precio	Total
C1	H. de programador	140	20,00€	2.800,00€
C2	Amort. portátil	140	0,82€	115,22€
	% Costes directos	3%	2,915,22€	87,47€
				3.002,68€

Tabla 11. Presupuesto corrección de errores

6.3. Mediciones

Cód	Partida	Nº
P1	Programación del código "Planificador" mediante C++ builder	1
P2	Programación del código "Cronogramas" mediante C++ builder	1
P3	Programación del código "Funciones_planificador" mediante C++ builder	1
P4	Programación del código "Planificador" mediante C++ builder	1
P5	Programación del código "Cronogramas" mediante C++ builder	1
P6	Corrección de errores	1

Tabla 12. Tabla mediciones

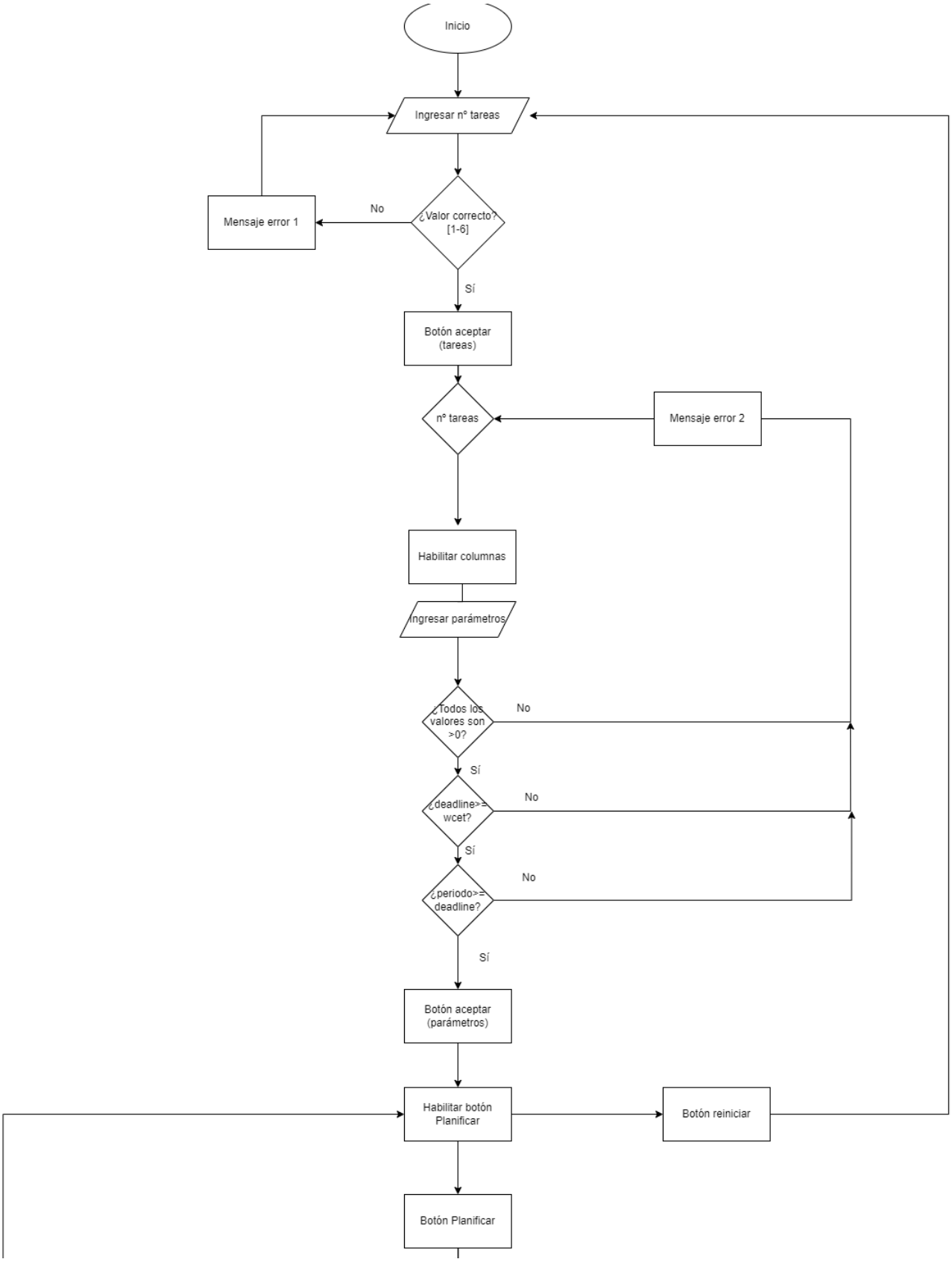
6.4. Presupuesto base

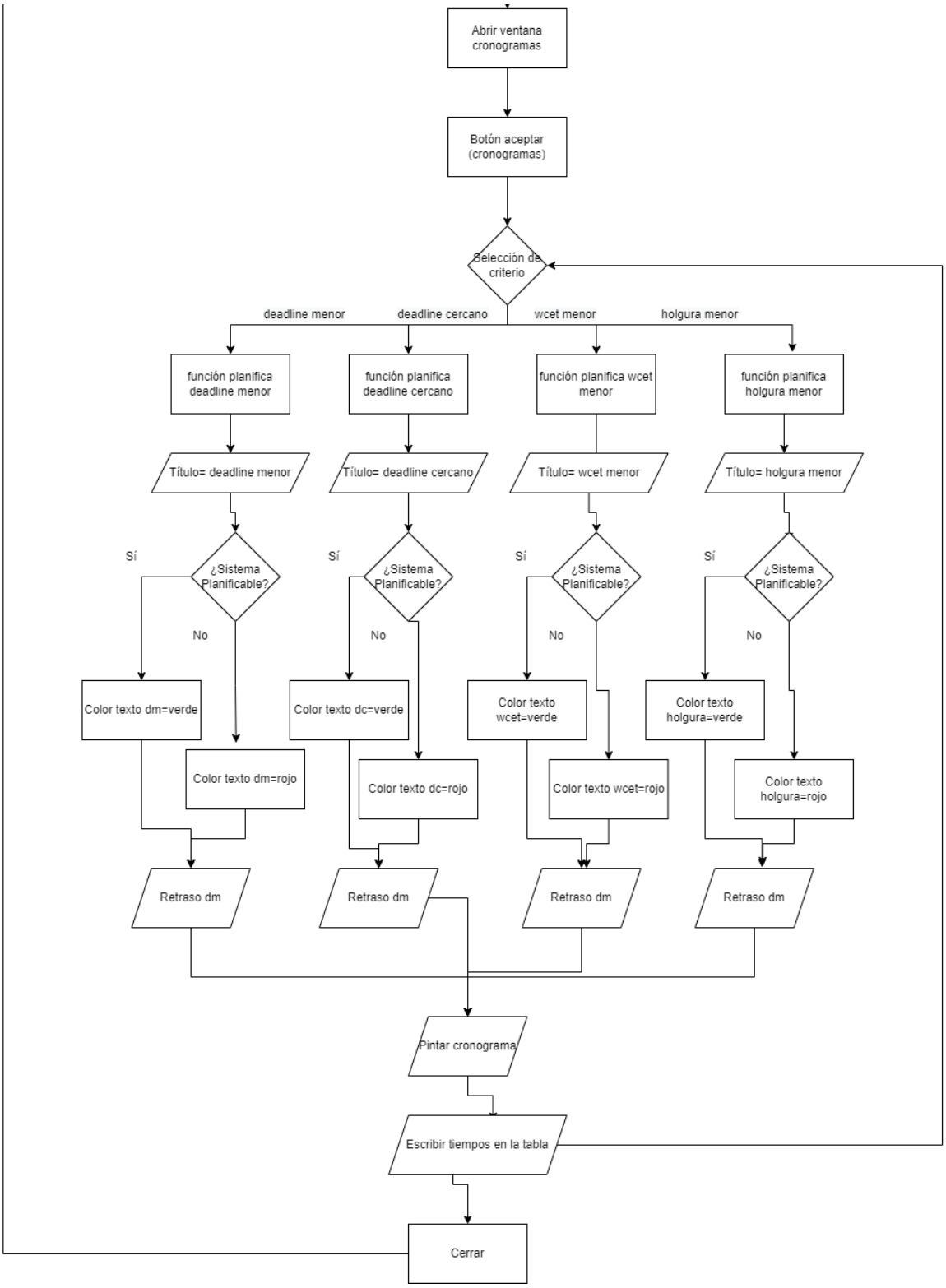
Cód	Partida	Nº	Precio	Total
P1	Programación del código "Planificador" mediante C++ builder	1	64,34€	64,34€
P2	Programación del código "Cronogramas" mediante C++ builder	1	85,79€	85,79€
P3	Programación del código "Funciones_planificador" mediante C++ builder	1	3.217,15€	3.217,15€
P4	Programación del código "Planificador" mediante C++ builder	1	21,45€	21,45€
P5	Programación del código "Cronogramas" mediante C++ builder	1	42,90€	42,90€
P6	Corrección de errores	1	3.002,68€	3.002,68€

Presupuesto base		6.434,31 €
IVA	21%	1.351,20 €
Presupuesto final		7.785,51 €

Tabla 13. Presupuesto final

7. Diagrama de flujos




Abrir ventana cronogramas
Botón aceptar (cronogramas)
Selección de criterio
deadline menor
deadline cercano
wcet menor
holgura menor
función planifica deadline menor
función planifica deadline cercano
función planifica wcet menor
función planifica holgura menor
Título= deadline menor
Título= deadline cercano
Título= wcet menor
Título= holgura menor
Sí
¿Sistema Planificable?
No
Color texto dm=verde
Color texto dm=rojo
Color texto dc=verde
Color texto dc=rojo
Color texto wcet=verde
Color texto wcet=rojo
Color texto holgura=verde
Color texto holgura=rojo
Retraso dm
Pintar cronograma
Escribir tiempos en la tabla
Cerrar

8. Conclusiones

8.1. Observaciones

Teniendo en cuenta lo descrito en el apartado «1. Objeto del proyecto», se puede concluir que el proyecto se ha realizado satisfactoriamente. La aplicación es perfectamente funcional, se han implementado correctamente las planificaciones según todos criterios y todos los resultados se muestran por pantalla de manera intuitiva.

8.2. Posibles mejoras

A pesar de la realización satisfactoria de la aplicación, existen algunas mejoras que no se han podido implementar por falta de tiempo y que han quedado fuera del proyecto final:

Lo primero es una mejor implementación de las funciones de activación y desactivación de los diferentes elementos visuales del programa como los botones y especialmente la función de reinicio. Aunque la mayoría de las situaciones indeseables se han ido solucionando durante el desarrollo de la aplicación, siguen existiendo algunas excepciones por las que se puede llevar al programa a un estado para el que no está programado. Esto se da únicamente cuando se reinician los valores después de la planificación de un sistema y se cambian los valores. En principio no existe ningún problema, pero hay algunos elementos que no se desactivan correctamente y, si se pulsan los botones de manera intencionada en un orden incorrecto, se puede hacer una mala introducción de los datos a calcular.

La segunda posible mejora es más compleja. Con las funciones que se han creado para planificar los sistemas existe una limitación de memoria. Al utilizar un vector de tamaño de hiperperiodo el programa puede no tener memoria suficiente para calcular sistemas con hiperperiodos grandes (en este caso superiores a 60.000 o 70.000). Seguramente con más tiempo y con más conocimientos podría realizar funciones capaces de calcular hiperperiodos más grandes. También se podría añadir de forma temporal alguna forma de evitar que la aplicación siga su curso normal si el hiperperiodo es demasiado grande para evitar que salga el mensaje de excepción que aparece cuando se da este caso.

8.3. Valoración personal

Para concluir con esta memoria haré una valoración personal sobre el conjunto del proyecto. Estoy muy satisfecho tanto con el trabajo que he invertido en la aplicación como en el resultado. Aunque sea una lástima no haber podido implementar todo como a mí me habría gustado inicialmente.

Durante la realización de este proyecto me he topado con una gran cantidad de obstáculos que finalmente he conseguido superar, varios errores a los que no encontraba ninguna solución y que han requerido de muchas búsquedas de información y del uso de mi ingenio.

Considero que he aprendido mucho gracias al enfrentamiento continuo que he tenido contra el programa y al hecho de tener que buscarme la vida para solucionar algunos errores que se me iban planteando. Creo que todo esto me ayudará mucho en un futuro.

9. Anexos

9.1. Manual de usuario

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

MANUAL DE USUARIO

Planificador de sistemas de tiempo real

Tabla de contenido

1. Introducción

Objetivo

Otorgar soporte a los usuarios de la aplicación «Planificador de sistemas de tiempo real» y ofrecer la información necesaria para su correcto uso.

Requerimientos

- Sistema Operativo Win 98 o superior
- Mínimo de 32 Mb en RAM
- Resolución gráfica mínima de 640 x 480

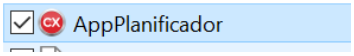
*Se recomienda desactivar el análisis en tiempo real del antivirus.

2. Opciones del Sistema

El presente manual está organizado de acuerdo al comportamiento habitual de la aplicación de la siguiente manera:

1. Ingreso al sistema
2. Ingreso de datos
3. Ventana cronogramas
4. Cambiar valores

Ingreso al sistema

Para comenzar se debe abrir la aplicación 

desde el escritorio o la carpeta de archivos. El usuario debe asegurarse de que el análisis en tiempo real del antivirus está desactivado de lo contrario la aplicación será bloqueada.

Ingreso de datos

Paso 1: En esta pantalla el usuario deberá ingresar el número de tareas que desea planificar (entre 1 y 6) y pulsar el botón «aceptar». En caso de que el valor sea incorrecto aparecerá un aviso y se deberá volver a ingresar el valor.

Paso 2: Una vez el número de tareas haya sido ingresado correctamente se habilitarán unas casillas de acuerdo con el valor ingresado. Se deberán completar con los parámetros de las tareas que deseen.

Paso 3. Una vez se hayan completado todos los huecos se deberá pulsar el botón «aceptar» que hay en la parte inferior de la pantalla. Si hubiera algún fallo en los valores de los parámetros aparecerá un aviso de error.

Paso 4. Una vez todos los datos se hayan ingresado correctamente, el usuario deberá pulsar el botón «planificar». Esta acción abrirá una nueva ventana.

Imagen 1. Ventana datos

Ventana cronogramas

Paso 1. Pulsar el botón «aceptar». Esta acción habilitará los botones de la izquierda.

Paso 2. Seleccionar el criterio de planificación deseado.

Paso 3. Lectura de datos (rectángulos azules de la imagen 5). La lectura de datos consiste en varias partes:

Criterio	Retraso
Wcet	119
Holgura	0
Deadline cercano	0
Deadline menor	119

Imagen 2. Retrasos del sistema

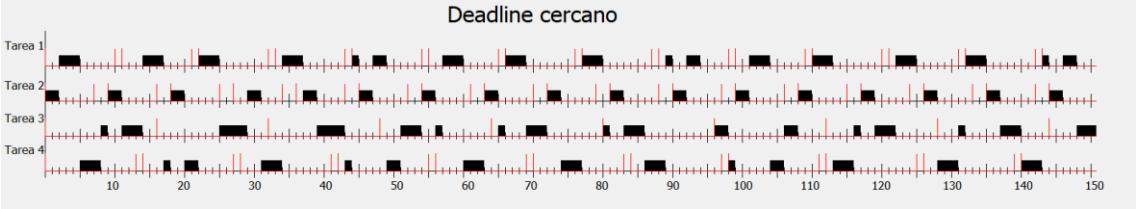


Imagen 3. Cronograma

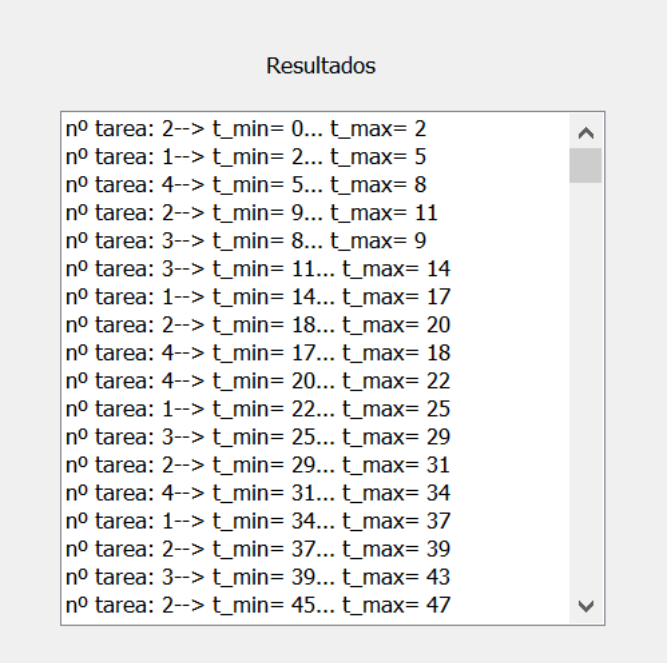


Imagen 4. Lista de tiempos

Paso 4. Cerrar ventana.

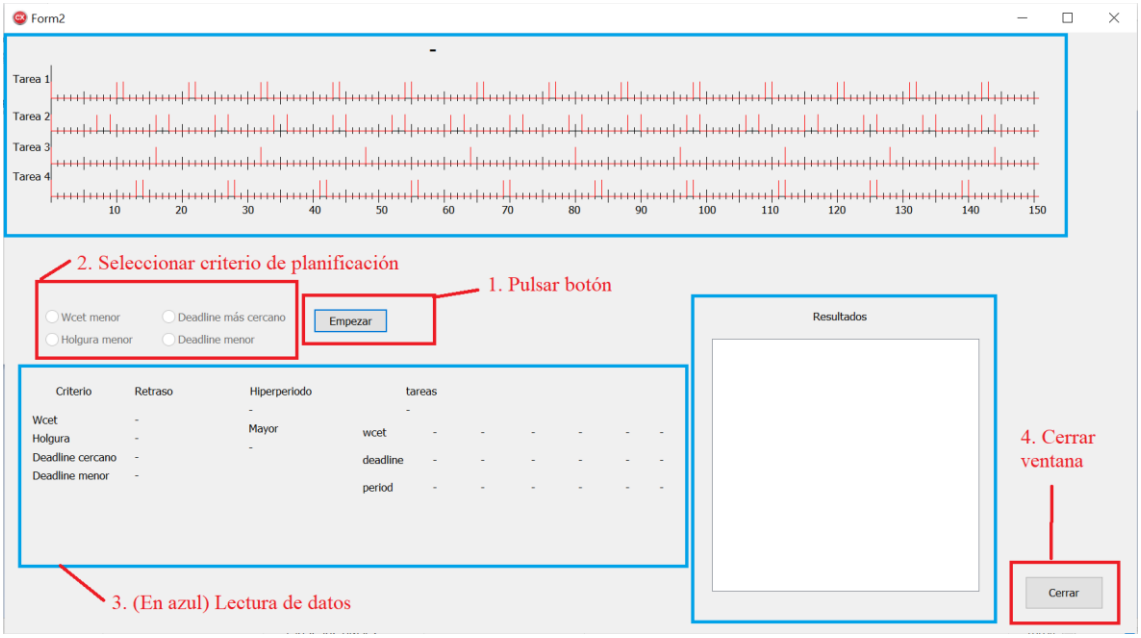


Imagen 5. Ventana cronogramas

Cambiar Valores

La aplicación ofrece la posibilidad de modificar los valores de los datos ingresados para permitir la planificación de más de un sistema sin necesidad de cerrar el programa. Esto es posible pulsando el botón «reiniciar» de la ventana de datos (ver imagen 1). Al pulsar este botón el programa vuelve a un estado igual al que el usuario se encontraba al iniciarlo por primera vez.

**Asegúrese de haber cerrado la ventana cronogramas antes de volver a ingresar los nuevos datos.

10. Bibliografía

1- Sistemas de tiempo real (José Luis Villarroel Salcedo): https://web.archive.org/web/20140222165922/http://webdiis.unizar.es/~joseluis/STR.pdf

2-El concepto de planificador para tareas en tiempo real concurrentes: http://www.comprendamos.org/alephzero/57/planificador.html

3- Tuspapapelesautonomos.es: https://tuspapelesautonomos.es/como-deducir-compra-de-ordenador-como-autonomo/#:~:text=Para%20un%20ordenador%20Hacienda%20dice,la%20declaraci%C3%B3n%20de%20la%20Renta.

4-Jobted.es: https://www.jobted.es/salario/ingeniero#:~:text=Sueldo%20del%20Ingeniero%20en%20Espa%C3%B1a&text=El%20salario%20medio%20de%20un,salario%20medio%20anual%20en%20Espa%C3%B1a.