Universidad Politécnica de Madrid

Escuela Técnica Superior
de Ingenieros de Telecomunicación

ISEL

Reto 2: Sistema de alarma doméstica

Javier López Iniesta Díaz del Campo
Fernando García Gutiérrez
Ivan Martín Cantón

28 de Mayo de 2021

Índice

1. Introducción

El segundo reto de la asignatura de Ingeniería de Sistemas Electrónicos del itinerario de Electrónica del GITST consiste en un sistema de alarma doméstica. La alarma consta de un sensor de infrarrojos (PIR) para detectar la presencia de intrusos en el domicilio. Si la alarma está activada, cuando se detecta una nueva presencia se genera una señal de alarma. En segundo lugar, se le ha añadido un pulsador para poder desactivar la alarma se introduce la combinación correcta. Finalmente, se ha añadido un control de luces para que cuando se detecte presencia se encienda una luz durante 30 segundos.

En este reto, se ha aplicando la metodología y todos los pasos del diseño orientado a modelos vistos en clase: especificación en LTL, modelado de los comportamientos con máquinas de estados, verificación formal del modelo, implementación mediante tres formas distintas (ejecutivo cíclico, threads con prioridades mediante POSIX y Reactor) y el análisis de planificabilidad (para comprobar que se cumplen los plazos en el caso peor).

Este documento pretende justificar todas las decisiones tomadas durante la realización del reto y mostrar las maquinas de estados (*FSM*) utilizadas para cada funcionalidad previamente comentada del sistema de alarma doméstica. La Figura 1 muestra el diagrama del sistema completo implementado.

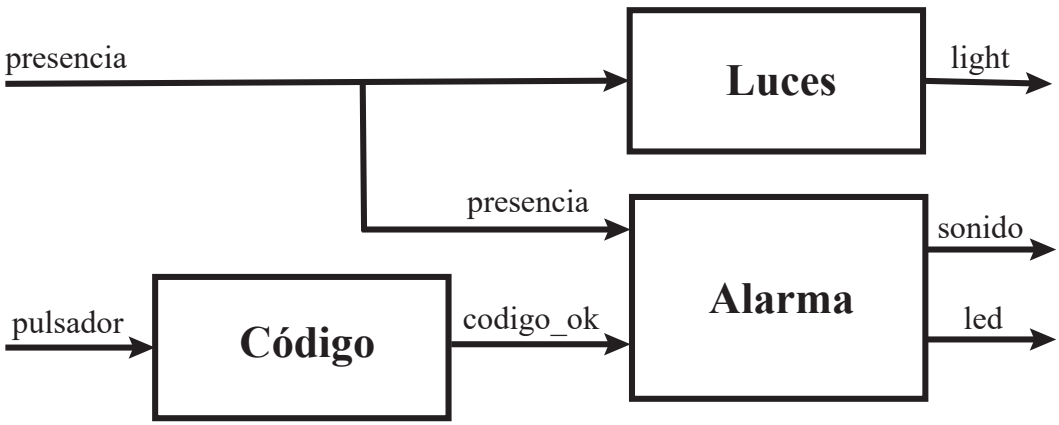

Figura 1: Sistema de alarma doméstica.

2. Especificación

En primer lugar, siguiendo la metodología del curso se han implementado las especificaciones *LTL* de los tres diferentes sistemas (alarma, código y luces). De esta forma en el directorio */spin* se incluyen los distintos ficheros Promela con estas las especificaciones, incluyendo el modelo de verificación de las máquinas de estados del sistema y el modelo del entorno:

Para ejecutar las distintas especificaciones basta con ejecutar en el directorio */spin*:

```
make
```

2.1. Alarma

(spin/alarma.pml): Se ha incluido una abstracción de la FSM del código que solo activa la señal `codigo_ok` de forma no determinista. Las especificaciones de LTL que se han incluido han sido:

- Si la alarma está activa y se detecta presencia, la alarma suena.

```
ltl alarma_activa_detecta_presencia_entonces_alarma_suena {
        [](((alarm_state==ACTIVA) && presencia && !codigo_ok) -> <>(led && sonido));
}
```

- Si la alarma no está activa, la alarma no suena.

```
ltl alarma_no_activa_alarma_no_suena {
        [](((alarm_state==INACTIVA) && presencia) -> (!led && !sonido));
}
```

- Si la alarma no está activa y se presiona el pulsador, la alarma se activa.

```
ltl alarma_no_activa_pulsador_alarma_activa {
        [](((alarm_state==INACTIVA) && codigo_ok) -> <>(alarm_state==ACTIVA));
}
```

- Si la alarma está activa y se presiona el pulsador, la alarma se desactiva.

```
ltl alarma_activa_pulsador_alarma_desactiva {
        []((((alarm_state==ACTIVA) || (alarm_state==SUENA)) && codigo_ok) ->
        <>((alarm_state==INACTIVA) && !led && !sonido));
}
```

2.2. Código

(spin/codigo.pml): Este fichero Promela contiene una abstracción de la FSM de la alarma, que solo desactiva la señal `codigo_ok` cuando se activa la alarma. En este caso, se han verificado dos especificaciones de LTL:

- En cualquier momento, si espero 10 segundos sin pulsar e introduzco el código correcto, la señal codigo_ok se activa.

```
ltl diez_sec_sin_pulsar_codigo_correcto_activa_codigo_ok_signal {
    [](((!pulsador W T10) && correcto -> <>codigo_ok)
}
```

- En cualquier momento, si espero 10 segundos sin pulsar e introduzco el código incorrecto, la señal codigo_ok no se activa

```
ltl diez_sec_sin_pulsar_codigo_incorrecto_no_activa_codigo_ok_signal {
    [](((!pulsador W T10) && !correcto -> <>!codigo_ok)
}
```

2.3. Light

(spin/light.pml): El fichero Promela *light.pml* incluye 2 especificaciones:

- Si hay presencia, la luz se enciende.

```
ltl presencia_luz_enciende {
        [](presencia -> <>light);
}
```

- Si han pasado 30 segundos desde la última vez que se detectó presencia, la luz se apaga.

```
ltl  no_presencia_luz_apaga{
    [] ((T30 && (!presencia W !light)) -> <> !light)
}
```

3. Modelado (FSM)

El modelo de las máquinas de estados (FSM) de tipo *Mealy* de todas las partes del sistema, cumpliendo todas las propiedades básicas, se muestran en las Figuras 2, 3 y 4:

3.1. Alarma

La alarma cuenta con un sensor de infrarrojos (PIR) para detectar la presencia de intrusos. Cuando la alarma esté activa y se detecte presencia se genera una señal de alarma, activando un led y un buzzer.

Entradas:

- `presencia`: sensor de infrarrojos (PIR).
- `codigo_ok`: señal que se activa si se introduce en la FSM de código la combinación correcta para activar o desactivar la alarma.

Salidas:

- `sonido`: buzzer (GPIO).
- `led`

En la Figura 2 se muestra la máquina de estados (FSM) de Mealy de la alarma.

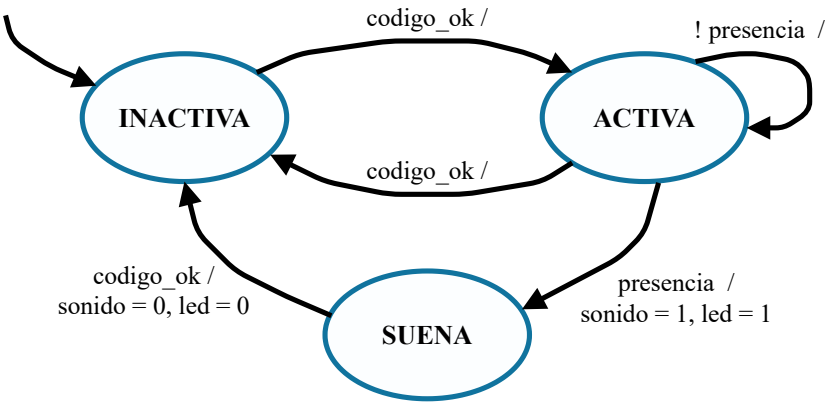

Figura 2: FSM de la alarma

3.2. Código

Además, se ha añadido otra máquina de estados para evaluar cuando se introduce el código predefinido y poder activar o desactivar la alarma. Este código se introduce por medio de un pulsador. Alguna de las características que cumple la FSM son:

- Cada nueva pulsación se incrementa el valor de la cuenta (dígito actual).
- Si se espera más de 1 segundo desde la última pulsación, avanza al siguiente dígito.
- Una espera de más de 10 segundos, vuelve al estado inicial.
- El "0" se introduce con 10 pulsaciones.
- Cuando se introducen todos los dígitos, si el código introducido es correcto, se activa la señal `codigo_ok`.

Entradas:

- `pulsador`

Salidas:

- `codigo_ok`: señal que se activa si se introduce el código correcto.

Variables:

- `count`: Es el número del dígito actual, almacena el número de veces que se presiona el pulsador. Su rango de valores es desde 0 (ninguna pulsación) hasta el 10. Para simplificar el modelo, se ha modificado el valor máximo en Spin a 4.
- `codigo_actual[3]`: Array para almacenar los 3 dígitos del código.
- `codigo_correcto[3]`: Array con la combinación correcta del código.
- `codigoCorrecto`: Variable que se activa, en caso de que todas las posiciones de codigo_actual coincidan con codigo_correcto.
- `i`: Indica el dígito que se esta modificando.

Temporizadores:

- T_1: Timer de 1 segundo para si se espera más de 1 segundo desde la última pulsación, avanza al siguiente dígito.
  - $T_1 = 0 \rightarrow$ Comenzamos el timer.
  - $T_1 = 1 \rightarrow$ Se ha cumplido el temporizador de 1 segundo.
- T_{10}: Timer de 10 segundos para si se espera más de 10 segundos volver al estado inicial.
  - $T_{10} = 0 \rightarrow$ Comenzamos el timer.
  - $T_{10} = 1 \rightarrow$ Se ha cumplido el temporizador de 10 segundos.

En la Figura 3 se muestra la máquina de estados (FSM) de Mealy del código.

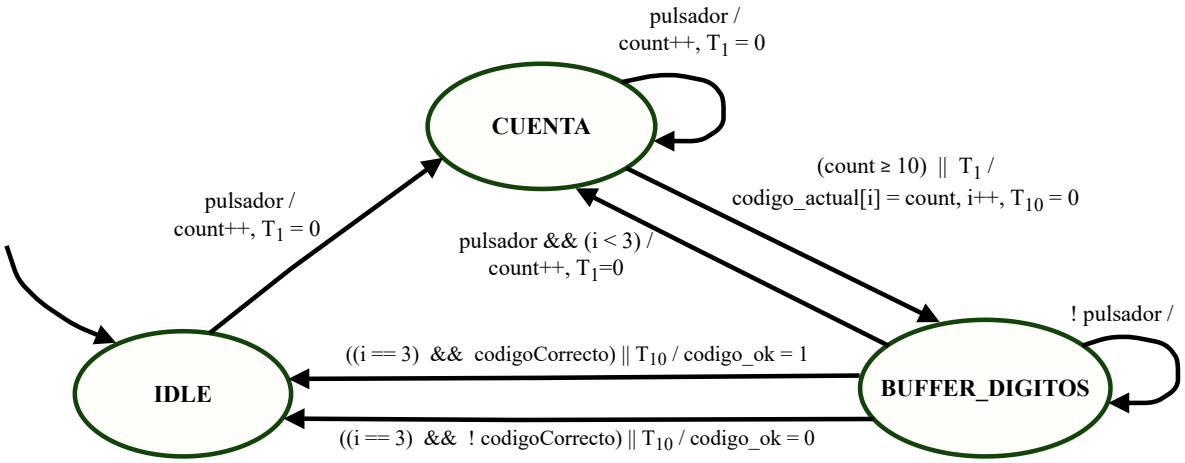

Figura 3: FSM del código

3.3. Light

Adicionalmente, se ha creado otra máquina de estados para el control automático de las luces del sistema. Cuando se detecta presencia (a través del sensor PIR que se utiliza tambien en la alarma) se enciende la luz durante 30 segundos y si pasan 30 segundos sin detectar presencia se apaga. Esto se refleja en la máquina de estados implementada, tenemos una entrada (`presencia`), una salida (`light`) y un timer (T_{30}), de funcionamiento similar a los timers de la máquina de estados anterior.

Entradas:

- `presencia`: sensor de infrarrojos (PIR).

Salidas:

- `light`: Luz (led).

Temporizador:

- T_{30}: Timer de 30 segundos para si pasan más de 30 segundos desde la última vez que se detectó presencia, la luz se apaga.
  - $T_{30} = 0 \rightarrow$ Comenzamos el timer.
  - $T_{30} = 1 \rightarrow$ Se ha cumplido el temporizador de 30 segundos.

En la Figura 4 se puede observar dicha máquina de estados de Mealy:

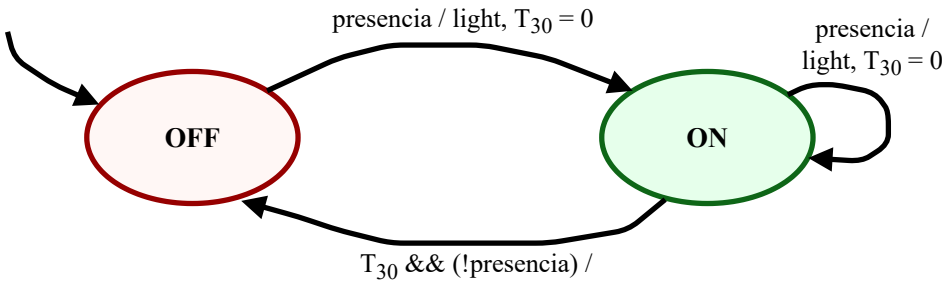

Figura 4: FSM de las luces

4. Verificación

En este punto de la metodología de desarrollo de software empotrado, se ha pasado a *Promela* las especificaciones LTL descritas en la Sección 2 y los distintos modelos de las máquinas de estados expuestos en la Sección 3.

De esta forma, se verificó mediante *Spin* los distintos ficheros *Promela*. Durante la verificación de la FSM del código se han tenido problemas en relación con el tiempo de ejecución debido al gran número de estados del sistema. La solución por la que se ha optado ha sido reducido el número de posibles dígitos de 10 a 4. De modo que se ha garantizado el contenimiento de lenguaje en todo momento.

5. Implementación

En cuanto a la implementación del software, lo hemos desarrollado en la plataforma *VirtualBox* simulando el sistema operativo *Ubuntu* correspondiente a una *Raspberry Pi*.

Debido al desarrollo de el software en la *Raspberry Pi* será necesario implementar la librería *WiringPi*. En cada uno de los ejercicios van incluidos todos los ficheros correspondientes a las maquinas de estados, al timer, al teclado, el main y por último, un Makefile en cada ejercicio. este último fichero es el que nos permitirá ejecutar el programa, para ello vamos a abrir el terminal en la carpeta del ejercicio que queramos ejecutar (ya sea ejecutivo cíclico, threads o reactor), una vez hecho esto vamos a ejecutar el comando *make* y por consiguiente el programa seleccionado se ejecutará.

6. Análisis de planificabilidad

En el análisis de planificabilidad se han considerado 4 tareas (KBD, Code, Alarma y Light). KBD y Code es la "misma" máquina de estados pero a través de KBD se puede probar a través del teclado del PC el sistema. Los plazos y los periodos se han asignado arbitrariamente, dándoles el mismo valor a los plazos y a los periodos para simplificar el modelo. Las FSM de KBD y Code son las que tienen el menor plazo, ya que se encargan de leer el pulsador y activar o desactivar la alarma a partir del código introducido. Por ello, les hemos asignado a ambas un plazo y un periodo de 50 (ms). Por otra parte, el disparo de la alarma al detectar presencia es importante que tenga lugar rápidamente: 250 (ms). Finalmente, el encendido de la luz (Light) no es muy importante que se ejecute con tanta frecuencia como las demás FSMs por ello le hemos asignado de plazo y periodo 500 (ms).

El algoritmo de planificación se ha basado siempre en el plazo (*Deadline Monotonic Scheduling*). Además, se ha utilizado el protocolo de techo de prioridad inmediato, debido a que si solo bloquea una vez y al principio minimiza los cambios de contexto. Además, asegura que el bloqueo máximo de todas las tareas es menor o igual que en el caso de usar herencia de prioridad.

Se han calculado los tiempos de ejecución en caso peor de todas las tareas, el tiempo máximo obtenido tras ejecutar 1000 veces cada una de las FSMs, mientras se interactuaban con ellas para que pasarán por todos los estados y transiciones ha sido:

- C_{KBD}: 623,05 ms.
- C_{Code}: 125,59 ms.
- C_{Alarm}: 39,63 ms.
- C_{Light}: 34,87 ms.

No obstante, se han redondeado dichos tiempos por simplificación en el análisis de planificación a $5\,ms$ y $1\,ms$. En la Tabla 1 se muestra un resumen con los tiempos máximos de ejecución, plazos y periodos de cada una de las tareas.

Tarea	**C (ms)**	**T (ms)**	**D (ms)**
KBD	5	50	50
Code	5	50	50
Alarma	1	250	250
Light	1	500	500

Tabla 1: Tiempos de ejecución, periodos y plazos de las distintas tareas.

6.1. Ejecutivo cíclico

Teniendo en cuenta los plazos y periodos de la Tabla 1, el ciclo principal se calcula a partir del hiperperiodo y este es:

$$T_M = H = m.c.m.(T_{KBD}, T_{Code}, T_{Alarma}, T_{Light}) = 500\ ms$$

En cambio, el ciclo secundario es:

$$T_S = M.C.D.(T_{KBD}, T_{Code}, T_{Alarma}, T_{Light}) = 50\ ms$$

En la Figura 5 se muestra el cronograma de las distintas tareas [1]. Se ha utilizado como algoritmo de planificación el *Deadline Monotonic Scheduling* asignando las tareas cuyo plazo es menor primero. Como se puede observar a simple vista el sistema es planificable.

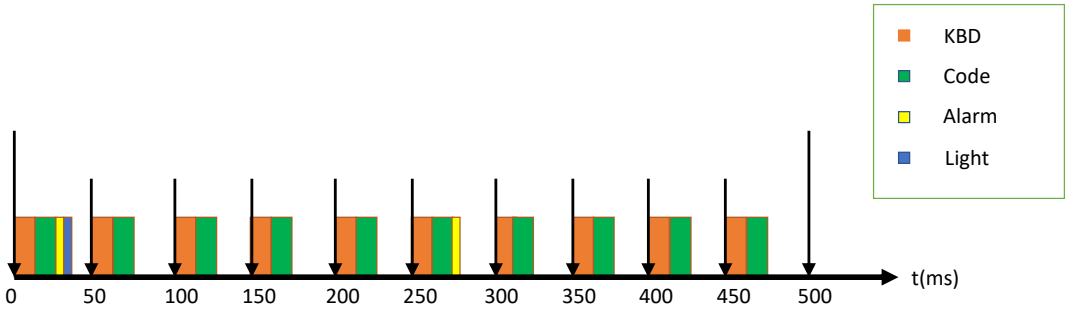

Figura 5: Análisis de planificabilidad con ejecutivo cíclico

Además, los tiempos de respuesta en caso peor de todas las tareas son inferiores a los plazos:

- $R_{KBD} = 5\ ms \leq D_{KBD} = 50\ ms$
- $R_{Code} = 10\ ms \leq D_{Code} = 50\ ms$
- $R_{Alarma} = 11\ ms \leq D_{Alarma} = 250\ ms$
- $R_{Light} = 12\ ms \leq D_{Light} = 500\ ms$

Por lo tanto, se verifica que el sistema es planificable mediante ejecutivo cíclico.

[1] Los tiempos de ejecución de la Figura 5 no están a escala, ya que son muy inferiores al hiperperiodo.

6.2. Threads con prioridades (POSIX)

En segundo lugar, se ha realizado el análisis con Threads con prioridades usando POSIX. Nuevamente, el algoritmo de planificación se ha basado en el plazo (*DMS*). Además, se ha utilizado el protocolo de herencia de prioridad debido a que la mayor parte de los sistemas operativos no implementan techo de prioridad inmediato. Con herencia de prioridad, tenemos 1 bloqueo por cada recurso compartido, y solo con recursos con techo de prioridad mayor o igual a la prioridad de la tarea.

En este sistema, solamente tenemos un recurso compartido entre las distintas máquinas de estados (r_1). Además, hemos considerado que el tiempo de acceso a dicho recurso es igual al tiempo de ejecución máximo de la tarea.

- $\mathbf{r_1}$: Variable `codigo_ok`

En la Tabla 2 se muestran además de los tiempos máximos de ejecución (C), el periodo (T) y el plazo (D), las prioridades asignadas a cada tarea (P), el bloqueo máximo (B), el recurso compartido (r_1) y el techo de prioridad (*Priority Ceiling, P.C.*) del recurso:

Tarea	**C (ms)**	**T (ms)**	**D (ms)**	**P**	**B (ms)**	**$\mathbf{r_1}$ (ms)**
KBD	5	50	50	4	5	5
Code	5	50	50	3	1	5
Alarma	1	250	250	2	0	1
Light	1	500	500	1	0	
					P.C.:	4

Tabla 2: Análisis de planificabilidad con threads con prioridades y herencia de prioridad

A continuación, se muestra el cálculo de los tiempos de respuesta:

$$R_i = C_i + B_i + I_i \tag{1}$$

De esta forma:

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j \tag{2}$$

$$B_j = \sum_{j \in lp(i),k} C_{j,k} \tag{3}$$

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j \rightarrow w = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w}{T_j} \right\rceil \cdot C_j \tag{4}$$

Para la tarea *KBD*:

$$\omega^0_{KBD} = C_{KBD} + B_{KBD} = 10\ ms \rightarrow R_{KBD} = 10\ ms \leq D_{KBD} = 50\ ms$$

Para la tarea *Code*:

$$w^0_{Code} = C_{Code} + C_{KBD} + B_{Code} = 15$$

$$w^1_{Code} = C_{Code} + B_{Code} + \left\lceil \frac{w^0_{Code}}{T_{KBD}} \right\rceil \cdot C_{KBD} = 5 + 1 + \left\lceil \frac{15}{50} \right\rceil \cdot 5 = 11$$

$$w^2_{Code} = C_{Code} + B_{Code} + \left\lceil \frac{w^1_{Code}}{T_{KBD}} \right\rceil \cdot C_{KBD} = 5 + 1 + \left\lceil \frac{11}{50} \right\rceil \cdot 5 = 11$$

Por lo tanto: $R_{Code} = 11\ ms \leq D_{Code} = 50\ ms$

Para la tarea *Alarma*:

$$w^0_{Alarma} = C_{Alarma} + C_{Code} + C_{KBD} + B_{Alarma} = 1 + 5 + 5 + 0 = 11$$

$$w^1_{Alarma} = C_{Alarma} + B_{Alarma} + \left\lceil \frac{w^0_{Alarma}}{T_{KBD}} \right\rceil \cdot C_{KBD} + \left\lceil \frac{w^0_{Alarma}}{T_{Code}} \right\rceil \cdot C_{Code}$$

$$w^1_{Alarma} = 1 + 0 + \left\lceil \frac{11}{50} \right\rceil \cdot 5 + \left\lceil \frac{11}{50} \right\rceil \cdot 5 = 11$$

Por lo tanto: : $R_{Alarma} = 11\ ms \leq D_{Alarma} = 250\ ms$

Finalmente, para la tarea *Light*:

$$w^0_{Light} = C_{Light} + C_{Alarma} + C_{Code} + C_{KBD} + B_{Light} = 1 + 1 + 5 + 5 + 0 = 12$$

$$w^1_{Light} = C_{Light} + B_{Light} + \left\lceil \frac{w^0_{Light}}{T_{KBD}} \right\rceil \cdot C_{KBD} + \left\lceil \frac{w^0_{Light}}{T_{Code}} \right\rceil \cdot C_{Code} + \left\lceil \frac{w^0_{Light}}{T_{Alarma}} \right\rceil \cdot C_{Alarma}$$

$$w^1_{Light} = 1 + 0 + \left\lceil \frac{12}{50} \right\rceil \cdot 5 + \left\lceil \frac{12}{50} \right\rceil \cdot 5 + \left\lceil \frac{12}{250} \right\rceil \cdot 1 = 12$$

Por lo tanto: : $R_{Light} = 12\ ms \leq D_{Light} = 500\ ms$

Por lo que el sistema es planificable ya que todos los tiempos de respuesta máximos (incluyendo los bloqueos) son menores a sus plazos.

6.3. Reactor

Para finalizar, se ha realizado el análisis con el reactor. Como en el caso anterior el algoritmo de planificación se ha basado en el plazo (*DMS*). Con este tipo de análisis se usa techo de prioridad inmediato, ya que así tenemos como máximo un bloqueo y al principio.

En este sistema, como en threads con prioridades, solamente tenemos un recurso compartido entre las distintas máquinas de estados (r_1) y también hemos considerado que el tiempo de acceso a dicho recurso es igual al tiempo de ejecución máximo de la tarea. En la Tabla 3 se muestran como en los casos anteriores los tiempos máximos de ejecución (C), el periodo (T) y el plazo (D), las prioridades asignadas a cada tarea (P), el bloqueo máximo (B) y el recurso compartido (r_1). Además, también sale el tiempo de cómputo del último bloque (F), las cuales tienen el mismo valor que (C).

Tarea	**C (ms)**	**T (ms)**	**D (ms)**	**P**	**B (ms)**	**r_1 (ms)**	**F (ms)**
KBD	5	50	50	4	5	5	5
Code	5	50	50	3	1	5	5
Alarma	1	250	250	2	0	1	1
Light	1	500	500	1	0		1
					P.C.:	4	

Tabla 3: Análisis de planificabilidad con reactor

A continuación, se muestra el cálculo de los tiempos de respuesta:

$$R_i = w_i + F_i \tag{5}$$

Usando la siguiente fórmula:

$$w_i^{n+1} = (C_i - F_i) + B_{max} + \sum_{j \in hp(i)} \left\lceil \frac{W_i^n}{T_j} \right\rceil C_j \tag{6}$$

Para la tarea *KBD*:

$$w_{KBD}^0 = (C_{KBD} - F_{KBD}) + B_{KBD} = (5-5) + 5 = 5ms$$

Por lo tanto: $R_{KBD} = 10\ ms \leq D_{Code} = 50\ ms$

Para la tarea *Code*:

$$w_{Code}^0 = (C_{Code} - F_{Code}) + B_{Code} + \left\lceil \frac{1}{T_1} \right\rceil \cdot C_{KBD} = (5-5) + 1 + \left\lceil \frac{1}{50} \right\rceil \cdot 5 = 6ms$$

$$w_{Code}^1 = (C_{Code} - F_{Code}) + B_{Code} + \left\lceil \frac{w_{Code}^0}{T_1} \right\rceil \cdot C_{KBD} = (5-5) + 1 + \left\lceil \frac{6}{50} \right\rceil \cdot 5 = 6ms$$

Por lo tanto: $R_{Code} = 11\ ms \leq D_{Code} = 50\ ms$

Para la tarea *Alarma*:

$$w^0_{Alarma} = (C_{Alarma} - F_{Alarma}) + B_{Alarma} + \left\lceil \frac{1}{T_1} \right\rceil \cdot C_{KBD} + \left\lceil \frac{1}{T_2} \right\rceil \cdot C_{Code}$$

$$w^0_{Alarma} = (1-1) + 0 + \left\lceil \frac{1}{50} \right\rceil \cdot 5 + \left\lceil \frac{1}{50} \right\rceil \cdot 5 = 10ms$$

$$w^1_{Alarma} = (C_{Alarma} - F_{Alarma}) + B_{Alarma} + \left\lceil \frac{w^0_{Alarma}}{T_1} \right\rceil \cdot C_{KBD} + \left\lceil \frac{w^0_{Alarma}}{T_2} \right\rceil \cdot C_{Code}$$

$$w^1_{Alarma} = (5-5) + 0 + \left\lceil \frac{10}{50} \right\rceil \cdot 5 + \left\lceil \frac{10}{50} \right\rceil \cdot 5 = 10ms$$

Por lo tanto: $R_{Alarma} = 11\ ms \leq D_{Alarma} = 250\ ms$

Finalmente, para la tarea *Light*:

$$w^0_{Light} = (C_{Light} - F_{Light}) + B_{Light} + \left\lceil \frac{1}{T_1} \right\rceil \cdot C_{KBD} + \left\lceil \frac{1}{T_2} \right\rceil \cdot C_{Code} + \left\lceil \frac{1}{T_3} \right\rceil \cdot C_{Alarma}$$

$$w^0_{Light} = (5-5) + 0 + \left\lceil \frac{1}{50} \right\rceil \cdot 5 + \left\lceil \frac{1}{50} \right\rceil \cdot 5 + \left\lceil \frac{1}{250} \right\rceil \cdot = 11ms$$

$$w^1_{Light} = (C_{Light} - F_{Light}) + B_{Light} + \left\lceil \frac{w^0_{Light}}{T_1} \right\rceil \cdot C_{KBD} + \left\lceil \frac{w^0_{Light}}{T_3} \right\rceil \cdot C_{Code} + \left\lceil \frac{w^0_{Light}}{T_2} \right\rceil \cdot C_{Alarma}$$

$$w^1_{Light} = (5-5) + 0 + \left\lceil \frac{11}{50} \right\rceil \cdot 5 + \left\lceil \frac{11}{50} \right\rceil \cdot 5 + \left\lceil \frac{11}{250} \right\rceil \cdot = 11ms$$

Por lo tanto: $R_{Light} = 12\ ms \leq D_{Light} = 500\ ms$

Por lo que el sistema es planificable ya que todos los tiempos de respuesta máximos (incluyendo los bloqueos) son menores a sus plazos.