

- 1) *Observar el listado y describir la estructura y funcionamiento a partir de la descripción del programa.*

```
void _cdecl main(int argc, cahr **argv, cahr **envp)
```

Se establece prioridad HIGHESTPRIORITY para el proceso, en consecuencia los hilos (threads) heredarán esta prioridad (clase de prioridad + nivel de prioridad).

Mediante RtCreateThread se crean tres hilos (Task1, Task2 y Task3) en estado SUSPENDIDO, con los parámetros de seguridad y tamaño de pila por omisión.

Las prioridades de los hilos son establecidas con la función RtSetThreadPriority a un valor de HIGHPRIORITY.

Se crea un vector de tres eventos mediante RtCreateEvent. Éstos se utilizarán para sincronización de las tareas.

Asignación de las tareas a los slots mediante un vector que utilizará el planificador (Scheduler) para su trabajo.

Inicialización del planificador con el número de slots.

Se despiertan los hilos mediante la función RtResumeThread.

Creación del temporizador (Timer) mediante la función RtCreateTimer. La prioridad otorgada es la máxima RT_PRIORITY_MAX, debido a que el temporizador otorgará el periodo de planificación para gestionar las tares. La función de servicio es TimerHandler.

Activación del temporizador con una base de tiempos de 10ms mediante la función RtSetTimerRelative.

Habilitación del acceso al puerto paralelo de comunicaciones mediante la función RtEnablePortIo.

Se estable el hilo principal (proceso) en estado suspendido durante dos segundos con la función sleep.

Mediante la variable global g_endRequest=TRUE se fuerza a los hilos a su destrucción natural.

Finalmente se destruye el temporizador y se sale del proceso.

```
ULONG RTFCNDCL Taskx(void *nContext)
```

Éstas funciones se están ejecutando mientras g_endRequest sea falso.

Realizan una espera pasiva mediante la función RtWaitForSingleObject, lo hacen indeterminadamente hasta que su evento asociado esta señalizado.

Luego ejecutan el trabajo en si, que consiste en hacer oscilar su correspondiente pin del puerto paralelo.

```
Void RTFCNDCL TimerHandler (PVOID context)
```

Es la función asociada al time_out del temporizador (10ms). Se encarga de lanzar la ejecución del planificador scheduler_ini.

```
Void scheduler_isr()
```

Esta función se encarga de establecer el evento (RtSetEvent) en función del slot actual y el vector de tareas del hiperciclo. La descripción es:

Incremento del numero de slot y comprobación de su rango.

Comprobación que la tarea que se debe ejecutar no es la ceros: if (Slot_TaskIndex[slot]) ...

Realizar un set del evento correspondiente mediante RtSetEvent.

2) *Finalizar y comprobar el programa.*

Se ha podido observar el buen funcionamiento de la aplicación:

```
-Timer (0)-SCH (slot=0, task=1)-task1
-Timer (1)-SCH (slot=1, task=2)-task2
-Timer (2)-SCH (slot=2, task=0)-
-Timer (3)-SCH (slot=3, task=1)-task1
-Timer (4)-SCH (slot=4, task=3)-task3
-Timer (5)-SCH (slot=5, task=0)-
-Timer (6)-SCH (slot=6, task=1)-task1
-Timer (7)-SCH (slot=7, task=0)-
-Timer (8)-SCH (slot=8, task=0)-
-Timer (0)-SCH (slot=0, task=1)-task1
-Timer (1)-SCH (slot=1, task=2)-task2
-Timer (2)-SCH (slot=2, task=0)-
-Timer (3)-SCH (slot=3, task=1)-task1
-Timer (4)-SCH (slot=4, task=3)-task3
-Timer (5)-SCH (slot=5, task=0)-
-Timer (6)-SCH (slot=6, task=1)-task1
-Timer (7)-SCH (slot=7, task=0)-
-Timer (8)-SCH (slot=8, task=0)-
```

3) *¿Cómo se podría construir un planificador de tareas con una política “Non Preemptive Rate Monotonic”.*

Una planificación basado en un esquema de prioridad del tipo “Rate-Monotonic” consiste en establecer a las tareas de mayor frecuencia las prioridades mayores y viciersa.

En nuestro caso la tabla de ejecución de tareas ha sido asignada en tiempo de compilación, lo que quiere decir que es una planificación estática, y en consecuencia un esquema de prioridades no tiene sentido si no va acompañado de desalojo de taras.

- 4) *¿Cómo se podría construir un planificador de tareas con una política “Preemptive Rate Monotonic”.*

Para conseguir una planificación basada en prioridades y con capacidad de desalojo (“Preemptive Rate-Monotonic), es necesaria la asignación de diferentes prioridades a las distintas tareas utilizando el criterio “Rate-Monotonic”.

De esta forma el Sistema Operativo se encarga de desalojar tareas ante la espera de otra de mayor prioridad

Para implementar este esquema se establecen mayores prioridades a las tareas de mayor frecuencia mediante la función `RtSetThreadPriority`.

- 5) *Incluir comentarios finales sobre los puntos problemáticos, modificaciones realizadas, y otros aspectos de interés puntual o general.*

Hemos introducido un `RtPrintf` al principio de cada tarea y otro al final, de esta forma hemos podido observar cuando una tarea empieza y cuando finaliza.

Aumentando la carga asociada a una tarea hemos conseguido invadir el slot vecino, el resultado obtenido fue el siguiente:

```
-Timer(1)-SCH (slot=1,task=2)-(Inicio task2)
-Timer(2)-(Final task2)SCH (slot=2,task=0)-
```

El resultado nos sorprendió mucho, porque la prioridad del temporizador es mayor de la de las tareas. Debemos advertir que realizando este ensayo el Windows NT se quedó congelado (hizo crack).