INSTITUTO POLITÉCNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERÍA Y TECNOLOGÍAS AVANZADAS

Sistemas Operativos en Tiempo Real

Proyecto Final

Puesta en práctica de un sistema con principios de funcionamiento de un RTOS

Ángeles Pacheco Jorge Armando

Calva Sánchez Fernando Alonso

Sánchez López Axl Eduardo

CDMX, México 2017

Se entiende como sistema operativo en tiempo real, a aquel sistema que es capaz de responder bajo ciertas restricciones de tiempo, es decir, debe de ser predecible para garantizar un comportamiento correcto. Las tareas están delimitadas en tiempo de ejecución, por lo que garantiza que las respuestas sean en *tiempo real*.

Entre las múltiples características que podemos encontrar en un RTOS, tenemos que:

- Son deterministas, es decir, tienen capacidad para calcular con alta probabilidad, cuanto tiempo toma en ejecución cada tarea
- Son controlables, es decir, los procesos y/o usuarios que hacen uso del sistema tienen un amplio control sobre el sistema, pudiendo realizar cambios de prioridad en tareas, uso de memoria, tiempos de uso del procesador, etc.
- Deben ser confiables, la calidad y tiempo de ejecución no deben verse afectados con el paso del tiempo, ya que de lo contrario se tendrían consecuencias catastróficas.

La manera más sencilla de establecer tiempos y prioridades de ejecuciones para las diferentes tareas que conforman al sistema, es a través de un planificador (Schedule en inglés). Este se encarga de determinar que tarea debe atenderse actualmente, cuál con mayor o menor prioridad, y el tiempo de ejecución que se le destina a cada una de ellas.

A continuación, se presenta un código desarrollado en lenguaje C para el microcontrolador 16F627A de Microchip. El código fue desarrollado en el entorno de desarrollo CCS (PIC C) y busca cubrir algunas de las características principales de un SOTR (determinismo, multitasking, scheduler).

```
#include <16f627a.h>      //Contiene la información básica del PIC 16F627A
#FUSES INTRC_IO           //Establece el funcionamiento basado en el reloj interno
#FUSES NOMCLR             //Desactiva el uso de MasterClear para poder usarlo como I/O
#FUSES NOLVP              //Desactiva la reprogramación debida a voltajes bajos
#FUSES NOPROTECT          //Habilita que el código de programación pueda ser extraído

#use delay(clock=4000000)       //Establece la velocidad de trabajo en 4Mhz, es decir
                                //tomará 25uS ejecutar cada instrucción

int prioridad=0;          //Variable auxiliar para la prioridad de las tareas
int tempo=0;              //Variable auxiliar para la tarea 1 (Parpadeo Led)

#int_timer0               //Con esta directiva, el programa sabe a dónde ir cuando se
                          //realiza una interrupción debida al Timer 0
void timer0(){            //Se utiliza la interrupción de desbordamiento de Timer0
  tempo++;                //como Scheduler para asignación de orden de ejecución
  if (prioridad==1)       //(prioridades de las tareas), cada 65mS.
    prioridad=0;          //En otras palabras, establece a la tarea 1 como de mayor
  else                    //prioridad durante 65mS, y luego lo mismo con la tarea 2
    prioridad++;          //dando pauta a que se pueda realizar el multitasking
  set_timer0(0);          //Resetea el contador del Timer0
}

#include <lcd.c>          //incluye las directivas necesarias para el trabajo con LCD
```

```
void main(){                    //Función principal
  set_tris_a(0x00);             //Establece el puerto A del PIC como salidas
  set_tris_b(0x00);             //Establece el puerto B del PIC como salidas
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_256|RTCC_8_bit);

  /* Configura el prescaler del timer 0 para que este incremente en 1 cada 256 ciclos de reloj,
  es decir, cada 0.64uS. Ocurriendo un desbordamiento en este cuando han transcurrido
  65mS (aprox) */

  set_timer0(0);                //Inicia el contador del Timer0 en 0

  enable_interrupts(INT_TIMER0);
  enable_interrupts(GLOBAL);

  /*Con las instrucciones anteriores se configura el vector de interrupciones INTCON
  ubicado en la dirección de memoria 0x8B. Dicho registro contiene 8 bits donde se puede
  configurar lo siguiente:
  /////////INTCON/////////
  Bit.0=RBIF        Este bit es la bandera de interrupción de cambio de estado en puerto B
  Bit.1=INTF        Este bit es la bandera de interrupción externa.
  Bit.2=T0IF        Este bit es la bandera de interrupción de desborde de Timer 0.
  Bit.3=RBIE        Habilita la interrupción de cualquier cambio de estado en el puerto B
  Bit.4=INTE        Habilita interrupción de Hardware
  Bit.5=T0IE        Habilita interrupción de Timer 0
  Bit.6=PEIE        Este bit se encarga de activar las interrupciones periféricas (UART, SPI, etc)
  Bit.7=GIE         Este bit es el encargado de activar las interrupciones globales del PIC

  En base a lo anterior y por cómo está declarada la función enable_interups tenemos que
  enable_interrupts(INT_TIMER0); = asigna a GIE el valor de 1 para activar de manera global
  el uso de interrupciónes
  enable_interrupts(GLOBAL); = asigna a T0IE el valor de 1 para trabajar con interrupción
  por desbordamiento de Timer0 */

  lcd_init();           //Inicializa la LCD para poder trabajar con ella
  while(TRUE){          //Bucle infinito para ejecución de tareas
    switch (prioridad)      //Ejecuta la tarea correspondiente según lo regresado por Timer0
    {                       //en otras palabras, la prioridad asignada por el Scheduler
      case 0:               //Código de tarea 1 (Enciende/Apaga un led cada 650mS aprox)
        if (tempo==9)
        {
          output_toggle(PIN_B3);
          tempo=0;
        }
        break;
      case 1:                //Código de tarea 2 (Muestra texto en una LCD 16x2)
        lcd_putc("\fEjecutando\nTarea 2");
        break;
```

```
        default:
          break;
      }
    }
  }
```