Semaphore using FreeRTOS on LPC2148

e-Yantra Team

June 30, 2016

Contents

1 Introduction

There are a limited number of resources available to any system,Similarly any microcontroller has a limited number resources available.

As the complexity of the application Increases the number of Tasks running also Increases,more and more Tasks compete for the available Processor time or The I/O devices available.

To ensure equal availability of resources to all the Tasks Operating Systems provide a facilities through semaphores.
The Greek word sema means sign or signal, and -phore means carrier . So Semaphore = signalling.
Semaphores can be classified into

- Binary Semaphores
- Mutex
- Counting Semaphores

1.1 Binary Semaphores

Binary semaphores are used for Task synchronisation. If a process ocuppies a resource the value of Binary semaphore is 1 else 0 i.e it gives information only if the resource is available or not.

1.2 Mutex

Mutex stands for Mutual Exclusion.Any Task which requires a resource can "Block" the resource.when the Task uses the resource it can "Give" the resource.

1.3 Counting Semaphore

Counting semaphores are used to count resources and keep track of Multiple resources.

1.4 Mutex vs Binary Semaphore

- Mutexes are used for Resource Protection from other tasks//processes whereas Binary semaphores are used for task synchronistaion

- It is the responisibility of the occupying function to release the mutex,but a binary semaphore can be released even from ISR or any other functions.

- On the implementation level it is the Responibility of the Coder to ensure that the Mutex is only given by the task which takes it.

2 Requirement

1. Knowledge of C++
2. FreeRTOS source files/API
3. Keil compiler
4. Flash magic
5. FireBird V (LPC2148)

3 Binary Semaphore

3.1 Code :

```
#include<stdlib.h>
#include"FreeRTOS.h"
#include"task.h"
#include"LCD.h"
#include"semphr.h"

SemaphoreHandle_t xSemaphore;

//Look in the sample programs for Included functions variables etc

void forward(void *pvparam)
{          vTaskDelay(5); //Added so that Back Task can occupy the resource
        while(1)
        {
         if(xSemaphoreTake(xSemaphore,portMAX_DELAY)==pdTRUE)
                {          Stop();
                           Forward();
                   UART0_SendStr("Forward\n");
                vTaskDelay(5);//To avoid same Tasking Taking resources tu
                }


        }
}

void back(void *pvparam)
{
        while(1)
        {
         if(xSemaphoreTake(xSemaphore,portMAX_DELAY)==pdTRUE)
                {          Stop();
                           Back();
                  UART0_SendStr("Back\n");
                vTaskDelay(5);
                }

        }
}

void control_switcher(void *pvparam)
```

```
{
  while(1)
  {xSemaphoreGive(xSemaphore);
   UART0_SendStr("Semaphore given\n");
   vTaskDelay(1200);


   }
}


  int  main()
{
 PINSEL0 = 0x00000000;            // Reset all pins as GPIO
 PINSEL1 = 0x00000000;
 PINSEL2 = 0x00000000;
 DelaymSec(40);
        Init_Peripherals();

        UART0_SendStr("\t\tBinary Semaphore\n");
    xSemaphore=xSemaphoreCreateBinary( );

        xTaskCreate(forward,"forward", 300 ,NULL, tskIDLE_PRIORITY + 1, N
        xTaskCreate(back,"back", 300 ,NULL, tskIDLE_PRIORITY + 1, NULL);
        xTaskCreate(control_switcher,"control_switcher", 300 ,NULL, tskIDL

        vTaskStartScheduler();   //Task Scheduling

        while(1);


}
```

3.2 Explanation

- **Variable declaration**

```
SemaphoreHandle_t xSemaphore;
```

  This statement declares a variable of type "SemaphoreHandle_t"

- **Creation of the semaphore**

```
xSemaphore=xSemaphoreCreateBinary( );
```

- **Working of code**

  The forward function Waits for portMAX_DELAY i.e for maximum amount of time so that the control of Resources is available.

  Similarly the back function waits for maximum time to get access to the resources.

  As soon as execution of Tasks starts the resources are occupied by the back function(vTaskDelay restricts forward function),The control_switcher function is suspended for 1200 clock counts and Gives away the semaphore.

  As soon as the semaphore is released the forward function waiting for allocation of resources occupies them,the cycle continues with control_switcher releasing the semaphore.

- **Serial monitor Output**

4 Mutex

4.1 Code :

```
#include<stdlib.h>
#include "FreeRTOS.h"
#include "task.h"
#include "LCD.h"
#include"semphr.h"

//Refer to actual code for necessary functions and codes

SemaphoreHandle_t xSemaphore=0;//Creation of Variable for semaphore

  void forward(void *pvparam)
{
        while(1)
        {
                if(xSemaphoreTake(xSemaphore,1000) == pdTRUE )
// if available then
                {    UART0_SendStr("Forward\n");
                          Forward();
                     vTaskDelay(1200);                         // task suspended
                          Stop();
                          xSemaphoreGive( xSemaphore );
// after resource task completed, return the semaphore
                }

                else
                {         UART0_SendStr("Forward function access denied\n")

                 vTaskDelay(200);
        }
}

void back(void *pvparam)
{          while(1)
        {
                          if(xSemaphoreTake(xSemaphore,1000) == pdTRUE )
                          {
                          UART0_SendStr("Back\n");
                          Back();
                          vTaskDelay(1200);                              // perform
data tasks();
```

```
                        Stop();
                        xSemaphoreGive( xSemaphore );
// after shared data task completed, return the semaphore
        }

        else        // if available then
                { UART0_SendStr("Back Function access denied\n");

        vTaskDelay(200);                                                    /
}
}

  int main()
{
 PINSEL0 = 0x00000000;           // Reset all pins as GPIO
 PINSEL1 = 0x00000000;
 PINSEL2 = 0x00000000;
 DelaymSec(40);
        Init_Peripherals();

        UART0_SendStr("\t\tMutex\n");
        xSemaphore = xSemaphoreCreateMutex();    //Use the Handle as a MUT

        xTaskCreate(forward,"forward", 300 ,NULL, tskIDLE_PRIORITY + 1, N
        xTaskCreate(back,"back", 300 ,NULL, tskIDLE_PRIORITY + 1, NULL);

        vTaskStartScheduler();   //Task Scheduling

        while(1);
}
```

4.2 Explanation

- **Variable declaration**

```
SemaphoreHandle_t xSemaphore;
```

  This statement declares a variable of type "SemaphoreHandle_t"

- **Creation of Mutex**

```
xSemaphore = xSemaphoreCreateMutex();
```

- **Working of code**

  There are Two Tasks forward and back, when executed

  The forward function Waits for 1000 clock cycles for the resources,In case the resources are not available the Task sends a message about The lack of availability of resources. Similarly the back function waits for same amount of time for resources.

  As soon as execution of Tasks starts the resources are occupied by one of the the task and that task blocks the acess of those resources through a mutex.

  The task executes and when the execution is completed it "Gives" the Mutex and therefore the releases the resources,another waiting task then occupies those resources and blocks for a period of time it requires.

- **Serial monitor Output**

```
Mutex
Back
Forward function access denied
Forward
Back Function access denied
Back
Forward function access denied
Forward
Back Function access denied
Back
```

5 Counting Semaphore Implemented by dining Philosophers Problem

5.1 Code :

```
/*
Note: To use mutex semaphore you need to initialize  configUSE_MUTEXES to
*/

#include<stdlib.h>
#include "FreeRTOS.h"
#include "task.h"
#include "LCD.h"
#include"semphr.h"

SemaphoreHandle_t xSemaphore=0;//Creation of Variable for semaphore

int s=0;
int forks_avail[5]={0,0,0,0,0}; //The value of Variable is 0 if a fork is


void vfork( void * pvParameters )
{
        int i;
        const unsigned char* str;
        str = ( const unsigned char * ) pvParameters;


        //Assignment of forks available on the basis of name of Philosphe
        if(str[1]=='1')
        {i=0;}
        if(str[1]=='2')
        {i=1;}
        if(str[1]=='3')
        {i=2;}
        if(str[1]=='4')
        {i=3;}
        if(str[1]=='5')
        {i=4;}

        while(1)
        {
        //Waits for 1000 ticks for forks to be avaliable
        //If available checks if the fork is adjacent(Right) or not
```

```
        if(( xSemaphoreTake( xSemaphore, 1000 ) == pdTRUE )&&(forks_avail
{
                forks_avail[i]=1;


                UART0_SendStr(&str[0]);
                UART0_SendStr(":Right fork obtained\n");

        if(( xSemaphoreTake( xSemaphore, 2000 ) == pdTRUE )&&(forks_avail
        { //Waits for 2000 ticcks for Left fork to be available

                        forks_avail[(i+1)%5]=1;

                        UART0_SendStr(&str[0]);
                        UART0_SendStr(":Left fork obtained Eating :)\n");

                        vTaskDelay(2000);
                        UART0_SendStr(&str[0]);
                        UART0_SendStr(":Ate \n");

                        xSemaphoreGive(xSemaphore);
                        xSemaphoreGive(xSemaphore);
                        forks_avail[i]=0;
                        forks_avail[(i+1)%5]=0;
                    UART0_SendStr(&str[0]);
                        UART0_SendStr(":Thinking \n");
                        vTaskDelay(3000);
}

                else
                {
            UART0_SendStr(&str[0]);
                UART0_SendStr(":Returned Right fork:( \n");
                xSemaphoreGive(xSemaphore);
                forks_avail[i]=0;
                }

        }

    else
            {
            UART0_SendStr(&str[0]);
            UART0_SendStr(":Hungry\n");
            vTaskDelay(3000);
```

```c
                }
                }
}


  int main()
{
 PINSEL0 = 0x00000000;          // Reset all pins as GPIO
 PINSEL1 = 0x00000000;
 PINSEL2 = 0x00000000;
 Init_Peripherals();


        UART0_SendStr("\t\tCounting Semaphore\n");

        xSemaphore = xSemaphoreCreateCounting( 5, 5 );

         if( xSemaphore != NULL )
    {   UART0_SendStr("\tSemaphore Created\n");


xTaskCreate(vfork,"Philospher 1", 300 ,"P1", tskIDLE_PRIORITY + 1, NULL);
xTaskCreate(vfork,"Philospher 2", 300 ,"P2", tskIDLE_PRIORITY + 1, NULL);
xTaskCreate(vfork,"Philospher 3", 300 ,"P3", tskIDLE_PRIORITY + 1, NULL);
xTaskCreate(vfork,"Philospher 4", 300 ,"P4", tskIDLE_PRIORITY + 1, NULL);
xTaskCreate(vfork,"Philospher 5", 300 ,"P5", tskIDLE_PRIORITY + 1, NULL);

                vTaskStartScheduler();  //Task Scheduling
     }

        while(1)//Never reaches this Part of the main
        {UART0_SendStr("\t\tSemaphore not Created\n");  }

}
```

5.2 Explanation

- **Variable declaration**

```
SemaphoreHandle_t xSemaphore;
```

  This statement declares a variable of type "SemaphoreHandle_t"

- **Creation of Counting semaphore**

```
xSemaphore = xSemaphoreCreateCounting( 5, 5 );
```

  Here 1st parameter gives the maximum count and 2nd parameter is the initial count. If the semaphore is used for counting events 2nd parameter would be 0 and if used for resources management it would be equal to maximum or initial count.

- **Task Creation**

```
xTaskCreate(vfork," Philospher 1", 300 ,"P1",
 tskIDLE_PRIORITY + 1, NULL);
.
.
```

  Here vfork is a single Task which on variation of Parameter P1,P2...etc behaves as a different task,ecah task has its own stack and act as if they are independent.All the tasks have same priority and get equal time at the processor.

- **Working of code**

  The Tasks created are by changing the parameters of a single task.

  When each time a "Philosopher" is allocated the processor time it checks for the number of available "Forks".If the forks are available and then check for the Right fork and the philosopher "picks up the left fork" then when the "Philosopher" again gains the processor time it waits for Left fork to be available and proceeds to eat.

  when 5 "Philosophers" are allocated simulatenously the semaphore keeps track of the available forks .

- **Serial monitor output**

```
P3:Hungry
P5:Ate
P5:Thinking
P4:Left fork obtained Eating :)
P2:Right fork obtained
P4:Ate
P4:Thinking
P2:Left fork obtained Eating :)
P1:Right fork obtained
P3:Hungry
P5:Hungry
P2:Ate
P2:Thinking
P1:Left fork obtained Eating :)
P4:Right fork obtained
P1:Ate
P1:Thinking
P4:Left fork obtained Eating :)
P3:Right fork obtained
P5:Hungry
P2:Hungry
P4:Ate
P4:Thinking
P3:Left fork obtained Eating :)
P1:Right fork obtained
P3:Ate
P3:Thinking
P1:Left fork obtained Eating :)
P5:Right fork obtained
P2:Hungry
P4:Hungry
P1:Ate
P1:Thinking
```

6 References

1. http://www.rtos.be/2013/05/mutexes-and-semaphores-two-concepts-for-two-different-use-cases/
2. http://www.ocfreaks.com/cat/embedded/lpc2148-tutorials/
3. http://www.freertos.org/Inter-Task-Communication.html
4. http://tinymicros.com/
5. http://www.profdong.com/elc4438_spring2016/USINGTHEFREERTOSREALTIMEKERNEL.pdf