Lab Session - 5

Use of Semaphores

Name : Dumesh Naresh Patil
PRN : 2019BTEEN00035
Sub. : RTOS Batch : EN-1

5.3. Basics of Semaphore

1) What is semaphore ?

⇒ Semaphore is simply an integer variable that is shared between tasks. This variable is used to solve the critical section problem & to achieve task synchronization. semaphore is used to avoid the garbled behaviour of the os because of mutually shareable resources.

2) What is a shared resource ? Write examples of hardware & software shared resources.

⇒ shared resource is the resource which can be used by the multiple tasks simultaneously & mutually exclusively
Hardware : Printer, Scanner, etc.
Software : Global variables, shareable files, etc

3) What for it can be used ? List 3 uses of semaphore

⇒ (i) Controls the access to the shareable resources
(ii) signal the occurrence of an event
(iii) Allows to tasks to synchronize their activities

5.4 Creating & Using Semaphore

1) Write C statement to declare the semaphore pointer

⇒ OS_EVENT * semptr ;

2) Write C statement to initialize the shared resource.

⇒ semptr = OSSemCreate (1);
OSSemPend (semptr, 0, &err);

3) Write C statement to create semaphore. Explain the arguments & return data.

⇒ semptr = OSSemCreate (1);
Argument passed is 1
1 → Semaphore available
0 → Semaphore unavailable
This function returns pointer to Event control block the datatype of this pointer is

4) Explain the OS functions for using the semaphore (pend and post)

⇒ `OSSemPend (semptr, 0, &err);`

- semptr: pointer to the semaphore
- 0: time out period time we need to wait for resource
- &err: address to write error / success log

This function is used to acquire the resources

`OSSemPost (semptr)`

- semptr: pointer to the semaphore

This function is used to release the resources

5.5 Program

5.5.1 Program with no semaphore

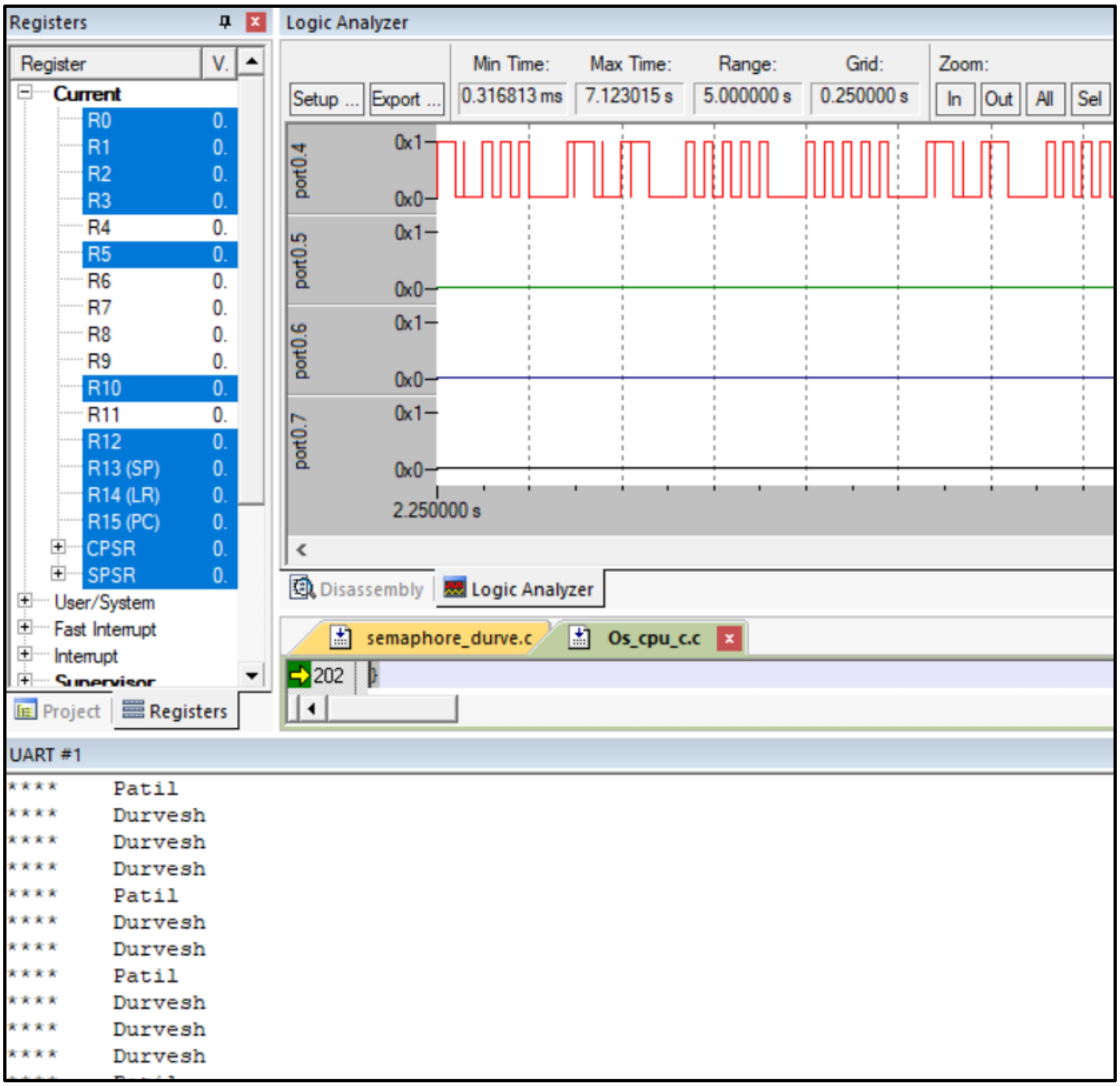
Write a C program for accessing UART0 shared resource “without” semaphore. Use ONLY two tasks.

```
#include "config.h"
#include "stdlib.h"
#include <stdio.h>

#define       TaskStkLengh       64                    //Define the Task0 stack length

OS_STK      TaskStk0 [TaskStkLengh];       //Define the Task stack
OS_STK      TaskStk1 [TaskStkLengh];       //Define the Task stack

void    Task0(void *pdata);
void    Task1(void *pdata);

char buffer[25];

int main (void)
{
        LED_init();
        UART0_Init();
        TargetInit();
        OSInit ();

        OSTaskCreate (Task0,(void *)0, &TaskStk0[TaskStkLengh - 1], 6);
        OSTaskCreate (Task1,(void *)0, &TaskStk1[TaskStkLengh - 1], 7);

        OSStart();
        return 0;

}
```

```
void Task0    (void *pdata)
{
        unsigned int i;
        pdata = pdata;                                          /* Dummy data */

        while(1)
        {
                UART0_SendData("****      Durvesh \r\n");
                for(i=0;i<5;i++)
                {
                        LED_on(0);    // All LEDs on
                        OSTimeDly(1);
                        LED_off(0);
                        OSTimeDly(1);
                }
                OSTimeDly(3);
        }
}
void Task1    (void *pdata)
{
        unsigned int i;
        pdata = pdata;                                          /* Dummy data */
        while(1)
        {
                UART0_SendData("****      Patil \r\n");
                for(i=0;i<5;i++)
                {
                        LED_on(0);    // All LEDs on
                        OSTimeDly(3);
                        LED_off(0);
                        OSTimeDly(3);
                }
                OSTimeDly(3);
        }
}
```

5.5.2 Observations

Observe the output in serial window. Take the screen shot of the serial window and paste it below.



Comments:

We get garbled output which is not the expected one.

5.5.3 Program with semaphore

Modify the C program such that, semaphore is used for accessing UART0 as shared resource.

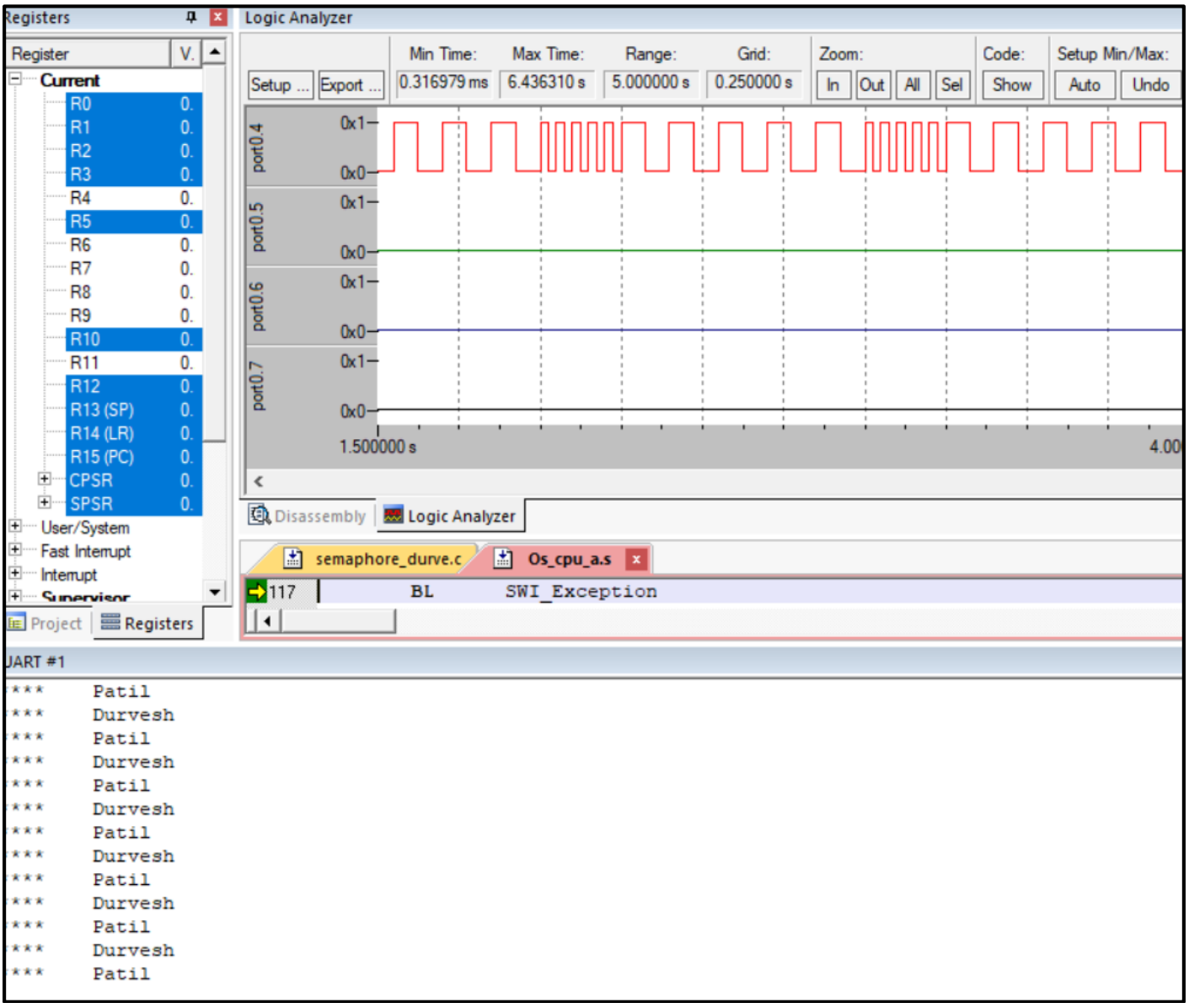
```
#include "config.h"
#include "stdlib.h"
#include <stdio.h>

#define        TaskStkLengh        64                          //Define the Task0 stack length

OS_STK       TaskStk0 [TaskStkLengh];        //Define the Task stack
OS_STK       TaskStk1 [TaskStkLengh];        //Define the Task stack

void    Task0(void *pdata);
void    Task1(void *pdata);

//pointer to semaphore

OS_EVENT* ptr_Sem_UART0;

//variable for storing error

unsigned char err;

char buffer[25];

int main (void)
{
        LED_init();
        UART0_Init();
        TargetInit();
        OSInit ();

        ptr_Sem_UART0 = OSSemCreate(1);
```

```
	OSTaskCreate (Task0,(void *)0, &TaskStk0[TaskStkLengh - 1], 6);
	OSTaskCreate (Task1,(void *)0, &TaskStk1[TaskStkLengh - 1], 7);

	OSStart();
	return 0;

}


void Task0    (void *pdata)
{
	unsigned int i;
	pdata = pdata;                                          /* Dummy data */

	while(1)
	{
		OSSemPend(ptr_Sem_UART0,0,&err);

		UART0_SendData("****      Durvesh \r\n");

		for(i=0;i<5;i++)
		{
			LED_on(0);    // All LEDs on
			OSTimeDly(1);
			LED_off(0);
			OSTimeDly(1);
		}

		OSSemPost(ptr_Sem_UART0);
		OSTimeDly(3);

	}
}
```

```
void Task1    (void *pdata)
{
        unsigned int i;
        pdata = pdata;                                          /* Dummy data */

        while(1)
        {
                OSSemPend(ptr_Sem_UART0,0,&err);
                UART0_SendData("****      Patil \r\n");

                for(i=0;i<5;i++)
                {
                        LED_on(0);    // All LEDs on
                        OSTimeDly(3);
                        LED_off(0);
                        OSTimeDly(3);
                }

                OSSemPost(ptr_Sem_UART0);
                OSTimeDly(3);

        }
}
```

5.5.4 Observations

Observe the output in serial window. Take the screen shot of the serial window and paste it below.



Comments:

We get proper output as expected using semaphore. Output is not garbled

5.6 Effect of increasing the number of tasks

Extend the program for 4 tasks with shared resource as serial port. Use a single semaphore for managing access to the shared resource among these 4 tasks.

5.6.1 Program with semaphore for four tasks

Modify the C program such that, semaphore is used for accessing UART0 (or LED) as shared resource.

```
#include "config.h"
#include "stdlib.h"
#include <stdio.h>

#define       TaskStkLengh       64                    //Define the Task0 stack length

OS_STK      TaskStk0 [TaskStkLengh];       //Define the Task stack
OS_STK      TaskStk1 [TaskStkLengh];       //Define the Task stack
OS_STK      TaskStk2 [TaskStkLengh];       //Define the Task stack
OS_STK      TaskStk3 [TaskStkLengh];       //Define the Task stack

void    Task0(void *pdata);
void    Task1(void *pdata);
void    Task2(void *pdata);
void    Task3(void *pdata);

//pointer to semaphore

OS_EVENT* ptr_Sem_UART0;

//variable for storing error

unsigned char err;
```

```
char buffer[25];

int main (void)
{
	LED_init();
	UART0_Init();

	TargetInit();
	OSInit ();

	ptr_Sem_UART0 = OSSemCreate(1);

	OSTaskCreate (Task0,(void *)0, &TaskStk0[TaskStkLengh - 1], 6);
	OSTaskCreate (Task1,(void *)0, &TaskStk1[TaskStkLengh - 1], 7);
	OSTaskCreate (Task2,(void *)0, &TaskStk2[TaskStkLengh - 1], 8);
	OSTaskCreate (Task3,(void *)0, &TaskStk3[TaskStkLengh - 1], 9);

	OSStart();
	return 0;

}


void Task0    (void *pdata)
{
	unsigned int i;
	pdata = pdata;                                          /* Dummy data */

	while(1)
	{

		// wait for semaphore to be available

		OSSemPend(ptr_Sem_UART0,0,&err);

		UART0_SendData("****      Durvesh \r\n");
```

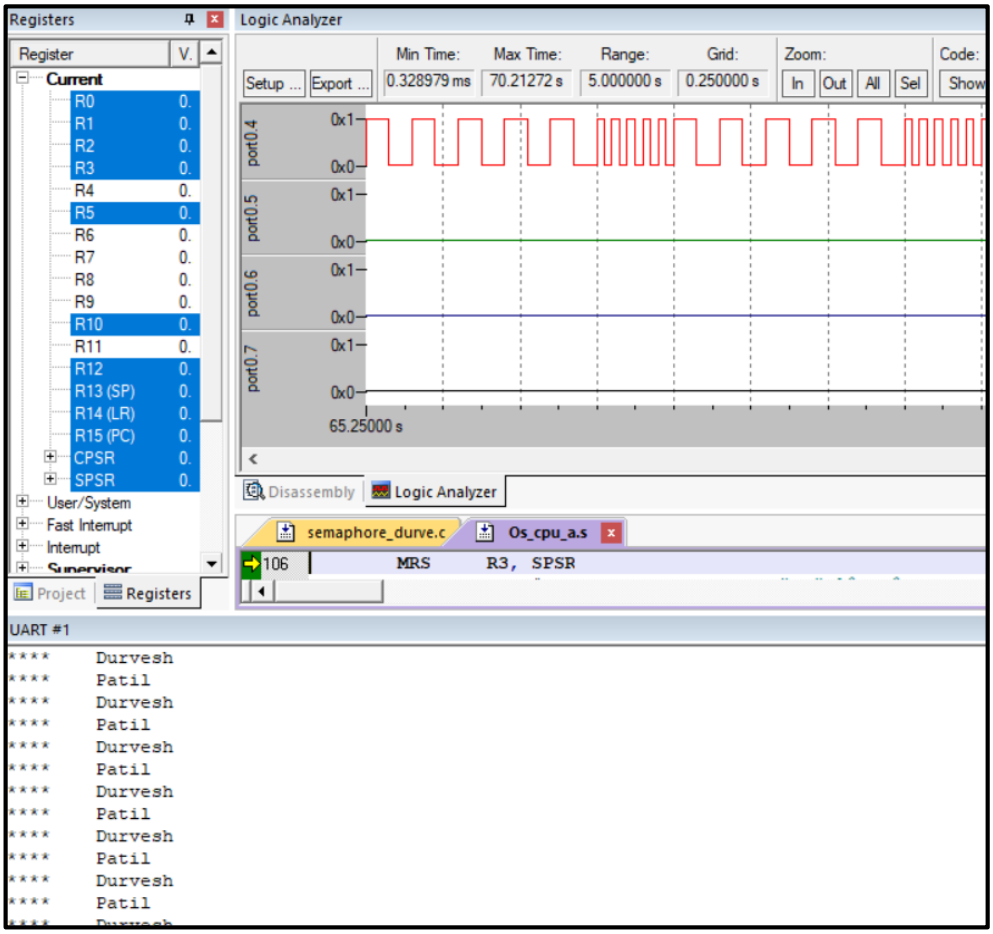
```
		for(i=0;i<5;i++)
		{
			LED_on(0);	// All LEDs on
			OSTimeDly(1);
			LED_off(0);
			OSTimeDly(1);
		}


		OSSemPost(ptr_Sem_UART0);
		OSTimeDly(3);

	}
}

void Task1	(void *pdata)
{
	unsigned int i;
	pdata = pdata;							/* Dummy data */

	while(1)
	{
		OSSemPend(ptr_Sem_UART0,0,&err);
		UART0_SendData("****	Patil \r\n");

		for(i=0;i<5;i++)
		{
			LED_on(0);	// All LEDs on
			OSTimeDly(3);
			LED_off(0);
			OSTimeDly(3);
		}

		OSSemPost(ptr_Sem_UART0);
		OSTimeDly(3);
```

```
	}
}

void Task2	(void *pdata)
{
	unsigned int i;
	pdata = pdata;

	while(1)
	{
		OSSemPend(ptr_Sem_UART0,0,&err);
		UART0_SendData("****	2019BTEEN00035 \r\n");

		for(i=0;i<5;i++)
		{
			LED_on(1);	// All LEDs on
			OSTimeDly(4);
			LED_off(1);
			OSTimeDly(4);
		}

		OSSemPost(ptr_Sem_UART0);
		OSTimeDly(3);

	}

}

void Task3	(void *pdata)
{
	unsigned int i;
	pdata = pdata;

	while(1)
	{
		OSSemPend(ptr_Sem_UART0,0,&err);
```

```
            UART0_SendData("****      B.Tech Electronics \r\n");

            for(i=0;i<5;i++)
            {
                  LED_on(1);    // All LEDs on
                  OSTimeDly(1);
                  LED_off(1);
                  OSTimeDly(1);
            }

            OSSemPost(ptr_Sem_UART0);
            OSTimeDly(3);
      }
}
```

5.6.2 Observations

Observe the output in serial window. Take the screen shot of the output below.



Comments: Only Task0 and Task1 are executing. Task2 and Task3 are not executing.

Find the reason why only two tasks work even if there are 4 tasks. Correct the program and resimulate it.

Semaphore can be acquired by only two tasks as per their priorities. So remaining tasks will not get the semaphore. To avoid this, we have to pass 3 as a parameter instead of 1 while creating the semaphore.

5.6.3 Corrected program with semaphore for four tasks

Modify the C program such that, semaphore is used for accessing UART0 (or LED) as shared resource.

```
#include "config.h"
#include "stdlib.h"
#include <stdio.h>

#define       TaskStkLengh       64                    //Define the Task0 stack length

OS_STK      TaskStk0 [TaskStkLengh];      //Define the Task stack
OS_STK      TaskStk1 [TaskStkLengh];      //Define the Task stack
OS_STK      TaskStk2 [TaskStkLengh];      //Define the Task stack
OS_STK      TaskStk3 [TaskStkLengh];      //Define the Task stack

void    Task0(void *pdata);
void    Task1(void *pdata);
void    Task2(void *pdata);
void    Task3(void *pdata);

//pointer to semaphore

OS_EVENT* ptr_Sem_UART0;

//variable for storing error
unsigned char err;
```

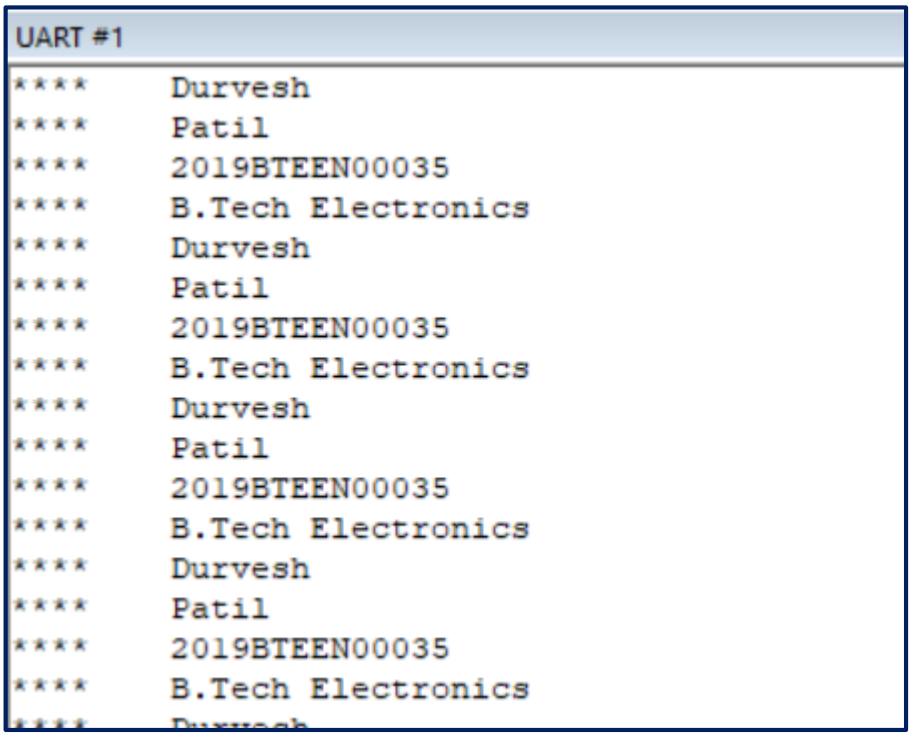
```
int main (void)
{
	LED_init();
	UART0_Init();
	TargetInit();
	OSInit ();

	ptr_Sem_UART0 = OSSemCreate(3);

	OSTaskCreate (Task0,(void *)0, &TaskStk0[TaskStkLengh - 1], 6);
	OSTaskCreate (Task1,(void *)0, &TaskStk1[TaskStkLengh - 1], 7);
	OSTaskCreate (Task2,(void *)0, &TaskStk2[TaskStkLengh - 1], 8);
	OSTaskCreate (Task3,(void *)0, &TaskStk3[TaskStkLengh - 1], 9);

	OSStart();
	return 0;

}

void Task0    (void *pdata)
{
	unsigned int i;
	pdata = pdata;                                          /* Dummy data */

	while(1)
	{
		// wait for semaphore to be available

		OSSemPend(ptr_Sem_UART0,0,&err);

		UART0_SendData("****      Durvesh \r\n");
		OSSemPost(ptr_Sem_UART0);
		OSTimeDly(3);
	}
}
```

```
void Task1    (void *pdata)
{
        unsigned int i;
        pdata = pdata;                                          /* Dummy data */

        while(1)
        {
                OSSemPend(ptr_Sem_UART0,0,&err);
                UART0_SendData("****      Patil \r\n");

                OSSemPost(ptr_Sem_UART0);
                OSTimeDly(3);

        }
}

void Task2    (void *pdata)
{
        unsigned int i;
        pdata = pdata;

        while(1)
        {
                OSSemPend(ptr_Sem_UART0,0,&err);
                UART0_SendData("****      2019BTEEN00035 \r\n");

                OSSemPost(ptr_Sem_UART0);
                OSTimeDly(3);

        }

}
```

```
void Task3    (void *pdata)
{
        unsigned int i;
        pdata = pdata;

        while(1)
        {
                OSSemPend(ptr_Sem_UART0,0,&err);
                UART0_SendData("****      B.Tech Electronics \r\n");

                OSSemPost(ptr_Sem_UART0);
                OSTimeDly(3);
        }
}
```

5.6.4 Observations

Observe the output in serial window. Take the screen shot of the output and paste it below.



Comments: All four tasks are executing properly.

5.7 Two way task synchronisation

How can the semaphore be used to synchronize activities of two tasks ? Can one semaphore be used to synchronize activities of more than 2 tasks ?

⇒ Semaphore is used in following ways :

(i) task to task synchronization
(ii) ISR to task synchronization using binary semaphore
(iii) ISR to task synchronization using counting semaphore.

No, one semaphore cannot be used for synchronization of more than 2 tasks.

5.8 Conclusion

(i) Semaphore is used for shared resources to avoid garbled output
(ii) Semaphore can be used with 2 tasks but not with 4 tasks
(iii) Pend is used to acquire the resource / semaphore & Post is used to release the resource
(iv) Types of semaphore - binary semaphore & counting semaphore.

5.9 Questions

① Write best way (according to you) for naming the variables, writing comments & need of beautifying the code etc.

⇒ According to me, variables should be given the proper names as per their use. For ex. semaphore pointer variable can be declared as ptrsem, etc. We can follow camelcase or snake-case for variable names.

Comments should be used while defining functions, calling built-in functions. We should properly indent the code. Reader should get proper understanding while reading our code.

② Suggest some good practices of managing the os based application project while you are performing the experiment.

⇒ (i) separating the ~~relating~~ related files in one folder
(ii) Tasks names should be given properly to get idea about its working.
(iii) There should be proper comments wherever necessary
(iv) Priority should be given properly to the tasks
(v) Semaphore should be used for shared resources