SVPM'S
COLLEGE OF ENGINEERING, MALEAON (BK)
DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING

Experiment No:	**Title: SEMAPHORE SIGNALLING**
Class: BE E&TC	

TITLE: Semaphore as signaling

OJECTIVE: To study Semaphore signaling.

AIM: Write a program using microcontroller OS RTOS that create two task implementing semaphore as signaling, first task is A and second task is B

SOFTWARE USED:

- SCARM(IDE)
- Flash Magic.

HARDWARE USED:

- Educational practice board for LPC2148(ARM7 kit)
- Adapter(9V DC,500mA)
- RS232
- PC

THEORY:

Semaphore is a protected variable or abstract data type rather an integer variable which constitutes the classic method for restricting access to shared resources such as shared memory in a parallel programming environment. A counting semaphore is a counter for a set of available resources, rather than a locked/unlocked flag of a single resource. It is basically a synchronizing tool. The use of semaphores normally requires hardware support to guarantee the atomicity of operations that require it. Computer machine languages typically include instructions that are designed specifically with semaphores in mind. These special instructions carry out a read-modify-write cycle to memory that is not only uninterruptible but excludes all other operations to the same location in memory by any other processors or input/output devices. The special instructions guarantee that a semaphore procedure using them can test and alter a semaphore in a single, atomic operation.

A mutex is a binary semaphore, usually including extra features like ownership or priority inversion protection. The differences between mutexes and semaphores are operating system dependent. Mutexes are meant to be used for mutual exclusion only and binary semaphores are meant to be used for event notification and mutual exclusion.

PROCEDURE:

1. Connect 9V DC Power Supply to the trainer kit.
2. Connect RS232 to evaluation board and com port of PC.
3. Open Side_arm software.
4. Create a new project. => Select manufacture- Phillips
   - ⇨ select microcontroller- LPC 2148
5. Then in editor window type main program.
6. If header files are included then add those files in ‘c files’ tab in workspace window.
7. Build and compile (Check if any errors are present and remove them)
8. Hex file will be created.
9. Now open flash magic and switch on the board
10. Step 1 => microcontroller ARM7 family, LPC2148
    - ⇨ =>Com port- Select proper com port
    - ⇨ Baud rate- 19200 (min) , 38400(max)

- ⇨ Interface- none (ISP)
- ⇨ Oscillator (MHz)- 12

11. Step 2 => Check ‘Erase all blocks used by Hex file’
12. Step 3 => Add respective Hex file using ‘Browse’ option
13. Click On ‘start’.
14. Thus the Hex file will get downloaded in the memory of the device.
15. Switch on respective switches present on the board
16. You will see the output

OUTPUT:

1. Semaphore signaling
   displays ‘A’ and ‘B’ on SPJ terminal.

CONCLUSION: This experiment demonstrates how to:

1. Configure the U-COS II.
2. Create and configure two different tasks.
3. Assign priority to the task.
4. Understand working of semaphore.

PROGRAM: **Semaphore signaling**

```
#include "includes.h"
/*********************************************************************************
*                                    main()
*
* Description : This is the standard entry point for C code.  It is assumed that your code will call
*               main() once you have performed all necessary ARM and C initialization.
* Arguments   : none
*********************************************************************************
*/
OS_STK Task1Stk[1024];
OS_STK Task2Stk[1024];
OS_EVENT *DispSem;
int  main (void)
{
   BSP_Init();                                  /* Initialize BSP functions                    */

   OSInit();                                    /*  Initialize  "uC/OS-II,  The  Real-Time  Kernel"
*/

     DispSem = OSSemCreate(30);
              OSTaskCreate(App_Task1,(void *)0,&Task1Stk[1023],6);
   OSTaskCreate(App_Task2,(void *)0,&Task2Stk[1023],7);

   OSStart();                     /* Start multitasking (i.e. give control to uC/OS-II)      */
   while(1);
}
INT16U value;
void  App_Task1 (void *p_arg);
void  App_Task1 (void *p_arg)
{
   (void)p_arg;
   while(1)
   {   /* repeate forever */
               value = OSSemAccept(DispSem);
               if(value>0)
               {
               printf("A");
       }
       OSTimeDlyHMSM(0,0,1,0);
   }
}
void  App_Task2 (void *p_arg);
void  App_Task2 (void *p_arg)
{
   (void)p_arg;
  while(1)
   {
               value = OSSemAccept(DispSem);
               if(value>0)
               {
               printf("B");
       }
       OSTimeDlyHMSM(0,0,2,0);
    }

}
```