11. Priority Inversion

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

Q. What is priority inversion ?

⇒ Priority inversion is an operating system scenario in which a higher priority process is preempted by a lower priority process. This implies the inversion of the priorities of the two tasks.

Q. How to avoid priority inversion ?

⇒ This is not possible to totally avoid . This problem is arising because of shared resources used .

Q. How can we minimize it's effect ?

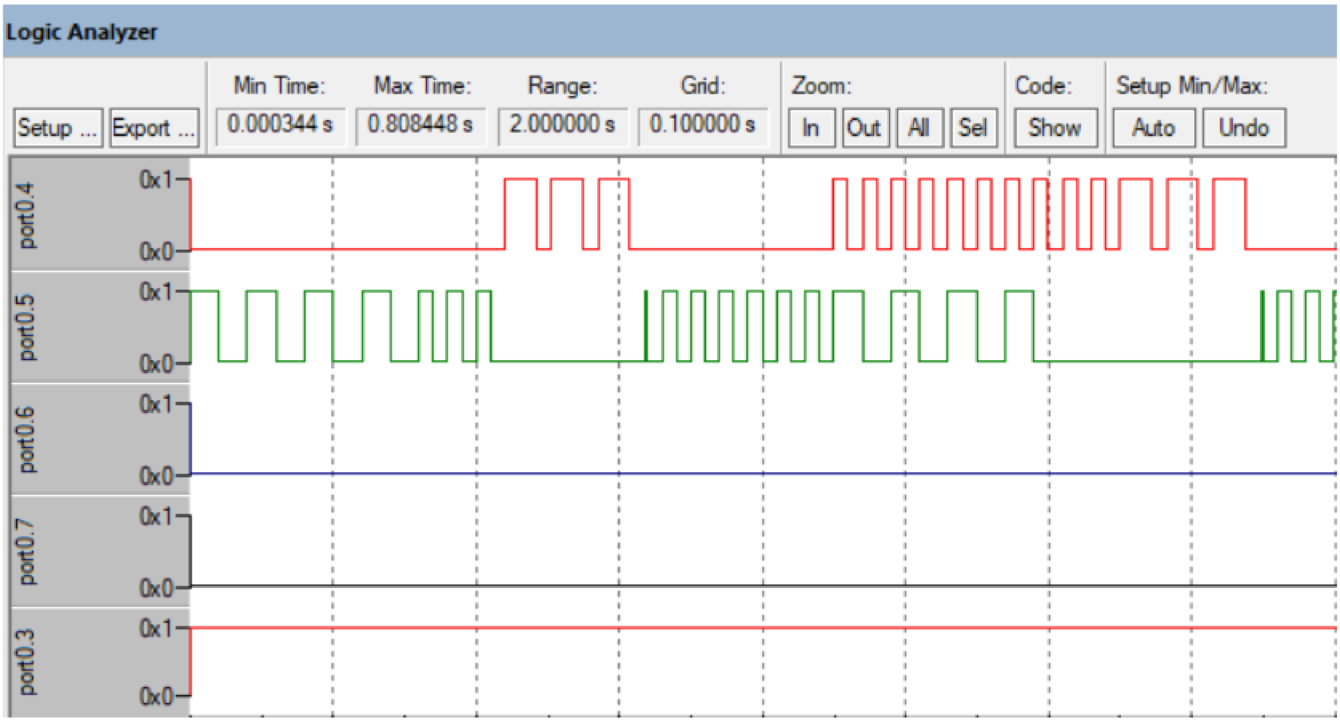
⇒ (i) Priority ceiling
(ii) Disabling interrupts
(iii) Priority inheritance
(iv) No blocking
(v) Random boosting.

Code:

```
#include "config.h"
#include "stdlib.h"

#define TaskStkLengh   64                          //Define the Task0 stack length

OS_STK          TaskStk0 [TaskStkLengh];           //Define the Task0 stack
OS_STK          TaskStk1 [TaskStkLengh];           //Define the Task0 stack

void     TaskHigh(void *pdata);                    // Task0
void     TaskLow(void *pdata);                     // Task1



// necessary for semaphore
OS_EVENT*   MySem;
unsigned char err;

char buffer[25];

/*******************************************************
         main()
*******************************************************/
int main (void)
{
         LED_init();
         TargetInit();
         OSInit ();
         MySem = OSSemCreate(1);
         OSTaskCreate (TaskHigh,(void *)0, &TaskStk0[TaskStkLengh - 1], 6);
         OSTaskCreate (TaskLow,(void *)0, &TaskStk1[TaskStkLengh - 1], 7);

         OSStart();
         return 0;

}
```

```
/***********************************************************
**                      Task0
***********************************************************/

void myDelay()
{
        unsigned int i;
        for(i=0;i<=60000;i++);
}
void TaskHigh  (void *pdata)
{
        unsigned int i;
        pdata = pdata;          /* Dummy data */
        OSTimeDly(22);
        while(1)
        {
                for(i=0;i<3;i++)
                {
                        LED_on(0);
                        myDelay();
                        myDelay();
                        LED_off(0);
                        myDelay();
                }
                // wait till semaphore is available
                OSSemPend(MySem, 0, &err);

                for(i=0;i<10;i++)
                {
                        LED_on(0);
                        OSTimeDly(1);
                        LED_off(0);
                        OSTimeDly(1);
                }
        // Semaphore released
        OSSemPost(MySem);
    }
}
void TaskLow   (void *pdata)
{
    unsigned int i;
    pdata = pdata;                  /* Dummy data */

    while (1)
    {
        for(i=0;i<4;i++)
        {
            LED_on(1);
            OSTimeDly(2);
            LED_off(1);
            OSTimeDly(2);
        }

        // wait till semaphore is available
        OSSemPend(MySem, 0, &err);

        for(i=0;i<10;i++)
        {
            LED_on(1);
            OSTimeDly(1);
            LED_off(1);
            OSTimeDly(1);
        }

        // Semaphore released
        OSSemPost(MySem);
    }
}
 //     End Of File
```

Observation:



Min Time: 0.000344 s
Max Time: 0.808448 s
Range: 2.000000 s
Grid: 0.100000 s

Comments:

Lower priority task has acquired the shared resource, because of this the highest priority task execution is delayed which is known as priority inversion.

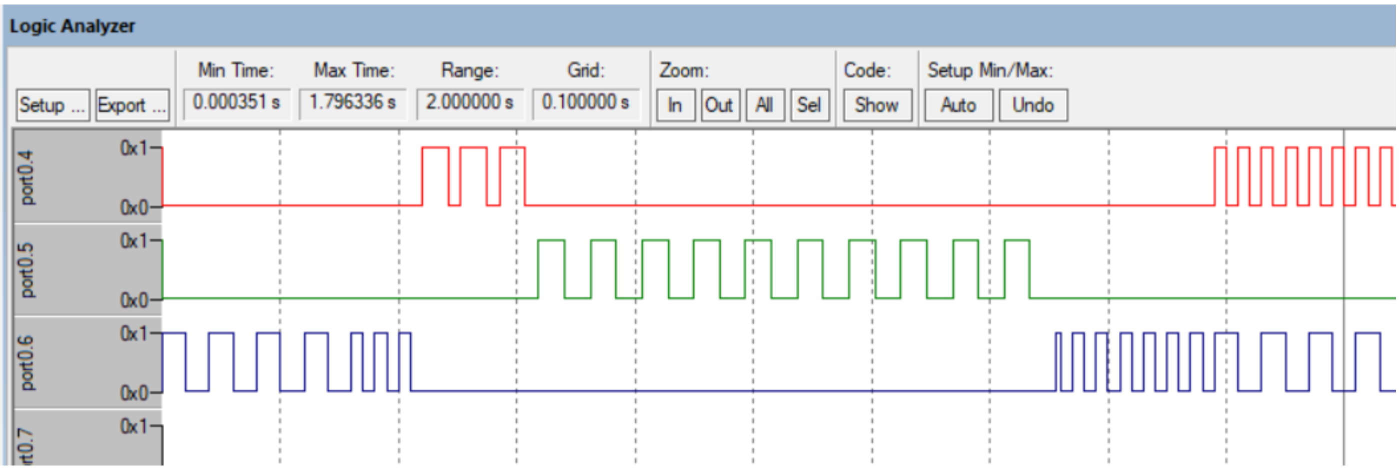
Code:

```
#include "config.h"
#include "stdlib.h"

#define TaskStkLengh   64                        //Define the Task0 stack length

OS_STK       TaskStk0 [TaskStkLengh];            //Define the Task0 stack
OS_STK       TaskStk1 [TaskStkLengh];            //Define the Task0 stack
OS_STK       TaskStk2 [TaskStkLengh];


void     TaskHigh(void *pdata);                  // Task0
void     TaskMedium(void *pdata);
void      TaskLow(void *pdata);                                  // Task1



// necessary for semaphore
OS_EVENT*   MySem;
unsigned char err;

char buffer[25];

/*********************************************************
            main()
*********************************************************/
int main (void)
{
            LED_init();
            TargetInit();
            OSInit ();
            MySem = OSSemCreate(1);
            OSTaskCreate (TaskHigh,(void *)0, &TaskStk0[TaskStkLengh-1], 6);
            OSTaskCreate (TaskMedium,(void *)0, &TaskStk1[TaskStkLengh-1], 7);
            OSTaskCreate (TaskLow,(void *)0, &TaskStk2[TaskStkLengh-1], 8);

            OSStart();
            return 0;

}
/*********************************************************
**                          Task0
*********************************************************/
void myDelay()
{
            unsigned int i;
            for(i=0;i<=60000;i++);
}
void TaskHigh  (void *pdata)
{
	unsigned int i;
	pdata = pdata;		/* Dummy data */
	OSTimeDly(22);
	while(1)
	{
		for(i=0;i<3;i++)
		{
			LED_on(0);
			myDelay();
			myDelay();
			LED_off(0);
			myDelay();
		}
		// wait till semaphore is available
		OSSemPend(MySem, 0, &err);

		for(i=0;i<10;i++)
		{
			LED_on(0);
			OSTimeDly(1);
			LED_off(0);
			OSTimeDly(1);
		}

		// Semaphore released
		OSSemPost(MySem);
	}
}

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

	OSTimeDly(28);
```

```
		while(1)
		{
				for(i=0;i<10;i++)
				{
						LED_on(1);
						myDelay();
						myDelay();
						LED_off(1);
						myDelay();
						myDelay();
				}
				OSTimeDly(30);
		}
}



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

		while (1)
		{
				for(i=0;i<4;i++)
				{
						LED_on(2);
						OSTimeDly(2);
						LED_off(2);
						OSTimeDly(2);
				}

				// wait till semaphore is available
				OSSemPend(MySem, 0, &err);
```

```
                for(i=0;i<10;i++)
                {
                        LED_on(2);
                        OSTimeDly(1);
                        LED_off(2);
                        OSTimeDly(1);
                }

                // Semaphore released
                OSSemPost(MySem);
        }
}

/*********************************************
        End Of File
*********************************************/
```

Observation:


Logic Analyzer

Min Time: 0.000351 s

Max Time: 1.796336 s

Range: 2.000000 s

Grid: 0.100000 s

Comments:

Lower priority task has acquired the shared resource, because of this the highest priority task execution is delayed which is known as priority inversion. Medium priority task is further delaying the highest priority task as it executes before the lowest priority task

2 Conclusion

(i) Priority inversion is the problem with preemptive kernel which occurs because of shared resources

(ii) It is not possible to totally avoid it

(iii) It is minimized by temporarily increasing the priority of lowest priority task.