Experiment #7
Priority Inversion

Introduction

Priority inversion occurs when a higher-priority task is forced to wait an indefinite period for the completion of a lower priority task. For example, **prioHigh, prioMedium,and prioLow** are task of high, medium, and low priority, respectively. **prioLow** has acquired a resource by taking its associated binary semaphore. When **prioHigh** preempts **prioLow** and contends for the resource by taking the same semaphore, it becomes blocked. If **prioHigh** would be blocked no longer than the time it normally takes **prioLow** to finish with the resource, there would be no problem, because the resource can't be preempted. However, the low priority task is vulnerable to preemption by the medium priority task, **prioMedium**, which could prevent **prioLow** from relinquishing the resource. This condition could persist, blocking **prioHigh** for an extensive period of time.

Objectives

The following are the primary objectives of this experiment:

- To demonstrate VxWorks' priority inversion avoidance mechanisms.

Description

To address the problem of priority inversion, VxWorks provides an additional option when using mutual exclusion semaphores. This option is **SEM_INVERSION_SAFE** which enables a priority inheritance algorithm. This algorithm insures that the task that owns

a resource executes at the priority of the highest priority task blocked on that resource. When execution is complete, the task relinquishes the resource and returns to its normal priority. Therefore, the inheriting task is protected from preemption by an intermediate priority task. This option must be used in conjunction with SEM_Q_PRIORITY:

semId = semMCreate(SEM_Q_PRIORITY | SEM_INVERSION_SAFE);

1. Example:

The example below illustrates a typical situation in which priority inversion takes place. Here is the what happens:

1. prioLow task locks the semaphore.
2. prioLow task gets preempted by prioMedium task which runs for a long time which results in the blocking of prioLow.
3. prioHigh task preempts prioMedium task and tries to lock the semaphore which is currently locked by prioLow.

The situation is shown in the printout from running the program:

```
-------------------------------------------------------------------------------
Low priority task locks semaphore
Medium task running
High priority task trys to lock semaphore
Medium task running
Medium task running
Medium task running
Medium task running
Medium task running
Medium task running
Medium task running
Medium task running
Medium task running
----------------------------------------Medium priority task exited
Low priority task unlocks semaphore
High priority task locks semaphore
High priority task unlocks semaphore
High priority task trys to lock semaphore
High priority task locks semaphore
High priority task unlocks semaphore
High priority task trys to lock semaphore
High priority task locks semaphore
High priority task unlocks semaphore
.....................................High priority task exited
Low priority task locks semaphore
Low priority task unlocks semaphore
Low priority task locks semaphore
Low priority task unlocks semaphore
.....................................Low priority task exited

-------------------------------------------------------------------------------
```

Since both prioLow and prioHigh are both blocked, prioMedium runs to completion(a

very long time). By the time prioHigh runs it is likely that it has missed its timing requirements.

Here is what the code looks like:

```
-------------------------------------------------------------------------------
/* includes */
#include "vxWorks.h"
#include "taskLib.h"
#include "semLib.h"

/* function prototypes */
void prioHigh(void);
void prioMedium(void);
void prioLow(void);

/* globals */
#define ITER 3
#define HIGH 102 /* high priority */
#define MEDIUM 103 /* medium priority */
#define LOW 104 /* low priority */
#define LONG_TIME 1000000
SEM_ID semMutex;


void inversion(void) /* function to create the three tasks */
{
int i, low, medium, high;
printf("\n\n...................##RUNNING##.....................\n\n\n");

/* create semaphore */
semMutex = semMCreate(SEM_Q_PRIORITY); /* priority based semaphore */

/* spawn the three tasks */
if((low = taskSpawn("task1",LOW,0x100,20000,(FUNCPTR)prioLow,0,0,0,0,0,0,0,
          0,0,0)) == ERROR)
          printf("taskSpawn prioHigh failed\n");
if((medium = taskSpawn("task2",MEDIUM,0x100,20000,(FUNCPTR)prioMedium,0,0,0,0,0,0,0,
          0,0,0)) == ERROR)
          printf("taskSpawn prioMedium failed\n");
if((high = taskSpawn("task3",HIGH,0x100,20000,(FUNCPTR)prioHigh,0,0,0,0,0,0,0,
          0,0,0)) == ERROR)
          printf("taskSpawn prioLow failed\n");
}

void prioLow(void)
{
int i,j;
for (i=0; i < ITER; i++)
          {
          semTake(semMutex,WAIT_FOREVER); /* wait indefinitely for semaphore */
          printf("Low priority task locks semaphore\n");
          for (j=0; j < LONG_TIME; j++);
          printf("Low priority task unlocks semaphore\n");
          semGive(semMutex); /* give up semaphore */
```

```
        }
printf(".....................................Low priority task exited\n");
}

void prioMedium(void)
{
int i;
taskDelay(20);/* allow time for task with the lowest priority to seize semaphore */
for (i=0; i < LONG_TIME*10; i++)
        {
        if ((i % LONG_TIME) == 0)
                printf("Medium task running\n");
        }
printf("----------------------------------------Medium priority task exited\n");
}

void prioHigh(void)
{
int i,j;
taskDelay(30);/* allow time for task with the lowest priority to seize semaphore */
for (i=0; i < ITER; i++)
        {
        printf("High priority task trys to lock semaphore\n");
        semTake(semMutex,WAIT_FOREVER); /* wait indefinitely for semaphore */
        printf("High priority task locks semaphore\n");
        for (j=0; j < LONG_TIME; j++);
        printf("High priority task unlocks semaphore\n");
        semGive(semMutex); /* give up semaphore */
        }
printf(".....................................High priority task exited\n");
}
--------------------------------------------------------------------------------
```

Procedures

1. Copy the source code in the example and compile it.
2. Load the object file onto the target machine.
3. Run the examples by executing the main routine("inversion") of the example on WindSh terminal.

Note: Make sure you have redirected I/O, otherwise you won't see the results of the *printf()* commands.

Follow On Experiment

Experiment 1. When the program lab7.c is executed, the priority inversion does not take place, since the constant LONG_TIME is not large enough. Explain why the priority inversion does not take place and modify LONG_TIME so that the priority inversion takes place. When the priority inversion takes place, proceed to Experiment 2.

Experiment 2. Modify the program so that the problem with priority inversion is eliminated and the printout from the program looks like the following:

```
Low priority task locks semaphore
Medium task running
High priority task trys to lock semaphore
Low priority task unlocks semaphore
High priority task locks semaphore
High priority task unlocks semaphore
High priority task trys to lock semaphore
High priority task locks semaphore
High priority task unlocks semaphore
High priority task trys to lock semaphore
High priority task locks semaphore
High priority task unlocks semaphore
.....................................High priority task exited
Medium task running
Medium task running
Medium task running
Medium task running
Medium task running
Medium task running
Medium task running
Medium task running
Medium task running
----------------------------------------Medium priority task exited
Low priority task locks semaphore
Low priority task unlocks semaphore
Low priority task locks semaphore
Low priority task unlocks semaphore
.....................................Low priority task exited
```

Additional Information

Refer to VxWorks User's Manual and Reference Manual.
