Experiment #6
Preemptive Priority Based Task Scheduling

Introduction

Task scheduling is the assignment of starting and ending times to a set of tasks, subject to certain constraints. Constraints are typically either time constraints or resource constraints. On a time-sharing operating system, running each active process in turn for its share of time (its "timeslice"), thus creating the illusion that multiple processes are running simultaneously on a single processor.
Wind task scheduling uses a priority based preemptive scheduling algorithm as default, but it can also accommodate round-robin scheduling.

Objectives

The following are the primary objectives of this experiment:

- To demonstrate the use of VxWorks preemptive priority based task scheduling facilities.

Description

- Preemptive Priority Based Scheduling

  With a preemptive priority based scheduler, each task has a priority and the kernel insures that the CPU is allocated to the highest priority task that is ready to run. This scheduling method is *preemptive* in that if a task that has a higher priority than the current task becomes ready to run, the kernel immediately saves the current tasks's context and switches to the context of the higher priority task.

The *Wind* kernel has 256 priority levels(0-255). Priority 0 is the highest and priority 255 is the lowest. Tasks are assigned a priority when created; however, while executing, a task can change its priority using *taskPrioritySet()*.

1. Example: Preemptive Priority Based Scheduling

One of the arguments to *taskSpawn()* is the priority at which the task is to execute:
id = taskSpawn(*name, **priority**, options, stacksize, function, arg1,.. , arg10);*
By varying the priority(0-255) of the task spawned, you can affect the priority of the task. *Priority* 0 is the highest and priority 255 is the lowest.The Note the priority of a task is relative to the priorities of other tasks. In other words, the task priority number itself has no particular significance by itself.
In addition a task's priority can be changed after its spawned using the following routine:

- *taskPrioritySet(int tid, int newPriority)*: Change the priority of a task.

In the example below, there are three tasks with different priorities(HIGH,MID,LOW). The result of running the program is that the task with the highest priority, "taskThree" will run to completion first, followed by the next highest priority task, "taskTwo", and the finally the task with the lowest priority which is "taskOne."

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

/* function prototypes */
void taskOne(void);
void taskTwo(void);
void taskThree(void);

/* globals */
#define ITER1 100
#define ITER2 1
#define LONG_TIME 1000000
#define HIGH 100 /* high priority */
#define MID 101 /* medium priority */
#define LOW 102 /* low priority */
```

```
#define TIMESLICE sysClkRateGet()

void sched(void) /* function to create the two tasks */
{
int taskIdOne, taskIdTwo, taskIdThree;

if(kernelTimeSlice(0) == OK) /* turn round-robin off */
         printf("\n\n\n\n\t\t\tTIMESLICE = %d seconds\n\n\n", TIMESLICE/60);

printf("\n\n\n\n\n");
/* spawn the three tasks */
if((taskIdOne = taskSpawn("task1",LOW,0x100,20000,(FUNCPTR)taskOne,0,0,0,0,0,0,0,
         0,0,0)) == ERROR)
         printf("taskSpawn taskOne failed\n");
if((taskIdTwo = taskSpawn("task2",MID,0x100,20000,(FUNCPTR)taskTwo,0,0,0,0,0,0,0,
         0,0,0)) == ERROR)
         printf("taskSpawn taskTwo failed\n");
if((taskIdThree = taskSpawn("task3",HIGH,0x100,20000,(FUNCPTR)taskThree,0,0,0,0,0,0,0,
         0,0,0)) == ERROR)
         printf("taskSpawn taskThree failed\n");

}

void taskOne(void)
{
int i,j;
for (i=0; i < ITER1; i++)
         {
         for (j=0; j < ITER2; j++)
                  logMsg("\n",0,0,0,0,0,0);
                  for (j=0; j < LONG_TIME; j++);
         }
}

void taskTwo(void)
{
int i,j;
for (i=0; i < ITER1; i++)
         {
         for (j=0; j < ITER2; j++)
                  logMsg("\n",0,0,0,0,0,0);
                  for (j=0; j < LONG_TIME; j++);
         }
}

void taskThree(void)
{
int i,j;
for (i=0; i < ITER1; i++)
         {
         for (j=0; j < ITER2; j++)
                  logMsg("\n",0,0,0,0,0,0);
                  for (j=0; j < LONG_TIME; j++);
         }
}
--------------------------------------------------------------------------------
```

Procedures

1. Copy the source code in the example and compile it.
2. Load the object file onto the target machine.
3. Execute the following command on the WindSh terminal: "logFdSet 1". This will direct the *logMsg()* output to the virtual console.
4. Run the examples by executing the main routine("sched") of the example on WindSh terminal.
Note: Make sure you have redirected I/O, otherwise you won't see the results of the *logMsg()* commands.

Follow On Experiment

Experiment 1. Modify the program such that the order of execution is "taskOne" first, then "taskTwo", and then "taskThree."
Experiment 2. Modify the program so that "taskOne" has the highest priority and "taskOne" and "taskTwo" are running at the same priority. Is there a difference in the output when compared with the output of Experiment 1?
Experiment 3. Repeat Experiment 2 with round-robin scheduling turned on. Is there a difference in the output when compared with the output of Experiment 2?

Additional Information

Refer to VxWorks User's Manual and Reference Manual.