Experiment #5
Round-Robin 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 round-robin task scheduling facilities.

Description

- Round-Robin Scheduling

  A round-robin scheduling algorithm attempts to share the CPU fairly among all ready tasks of the same priority. Without round-robin scheduling, when multiple tasks of equal priority must share the processor, a single task can usurp the processor by never blocking, thus never giving other equal priority tasks a chance to run.

Round-robin scheduling achieves fair allocation of the CPU to tasks of the same priority by an approach known as *time slicing*. Each task executes for a defined interval or *time slice*; then another task executes for an equal interval, in rotation. The allocation is fair in that no task of a priority group gets a second slice of time before the other tasks of a group are given a slice.

Round-robin scheduling can be enabled with routine *kernelTimeSlice()*, which takes a parameter for a time slice, or interval. The interval is the amount of time each task is allowed to run before relinquishing the processor to another equal priority task.

The following routine controls round-robin task scheduling:

- *kernelTimeSlice(int ticks)*: Control round-robin scheduling. The number of ticks(60 ticks equate to one second) determine the duration of the time slice.

1. Example: Round-robin Based Scheduling

In the example below, three tasks with the same priority print their task ids and task names on the console. Without round-robin scheduling, "taskOne" would usurp the processor until it was finished, and then "taskTwo" and "taskThree" would do likewise. In the event that "taskOne" was looping indefinitely, the other tasks would never get a chance to run.

To insure that the tasks get an equal share of the CPU time, a call is made to *kernelTimeSlice()*. This sets the time slice interval value to TIMESLICE. The TIMESLICE value is the time slice interval in terms of the number of clock ticks(which in the example and the M68040 is 60 ticks which is equivalent to one second). The *sysClkRateGet()* can be used to determine the number of clock ticks per second.

Having setup the time slice in the manner above, the three tasks are spawned. However, here a few implementation details that should be noted:

- Make sure that sched has a higher priority than the tasks it is spawning! Unless otherwise specified, tasks have a default priority of 100. Notice that taskOne, taskTwo, and taskThree all have priorities of 101, which makes them lower in priority than sched.

- Yow must allow enough time for the context switches to occur. Thus the reason for -> for (j=0; j < LONG_TIME; j++);

- Using printf is not ideal in the example, because it can block .This will of course cause a task transition which will upset the nice round robin picture. Instead use *logMsg()* (see vxWorks reference manual for details). The latter won't block unless the log message queue is full.

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

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

/* globals */
#define ITER1 100
#define ITER2 10
#define PRIORITY 101
#define TIMESLICE sysClkRateGet()
#define LONG_TIME 1000000

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

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

/* spawn the three tasks */
if((taskIdOne = taskSpawn("task1",PRIORITY,0x100,20000,(FUNCPTR)taskOne,0,0,0,0,0,0,0,
         0,0,0)) == ERROR)
         printf("taskSpawn taskOne failed\n");
if((taskIdTwo = taskSpawn("task2",PRIORITY,0x100,20000,(FUNCPTR)taskTwo,0,0,0,0,0,0,0,
         0,0,0)) == ERROR)
         printf("taskSpawn taskTwo failed\n");
if((taskIdThree =
taskSpawn("task3",PRIORITY,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); /* log messages */
                    for (j=0; j < LONG_TIME; j++); /* allow time for context switch */
          }
}

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); /* log messages */
                    for (j=0; j < LONG_TIME; j++); /* allow time for context switch */
          }
}

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); /* log messages */
                    for (j=0; j < LONG_TIME; j++); /* allow time for context switch */
          }
}
--------------------------------------------------------------------------------
```

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 the 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. Why must the priority of "taskOne", "taskTwo", and "taskThree" be

higher than "sched"?

Experiment 2. Write the source code necessary to vary the time of TIMESLICE(10,20, 30,40,50,60,120,180,240 and 300 clock ticks).

Experiment 3. Add a fourth task(taskFour) that has a priority of 80 that prints out the same message as the other three tasks in the example. Describe the output from running the program.

Additional Information

Refer to VxWorks User's Manual and Reference Manual.
