ECEN 5623

Assignment #1 -- RM/DM Scheduling Feasibility Tests

Submission Format: Plain text, Rich text, PDF, or MS Word

Code Format: Turn in all code in a tar or zip file!

Upon Completion upload to learn.colorado.edu, "Lab Assignments, Lab 1".

(If there is a problem with d2learn, then E-mail to john.pratt@colorado.edu with subject line "LAB SUBMISSION 1" (subject line must be all caps exactly as it appears between quotes) and CC the TAs.)

Be sure to include all source code with your submission.

All Group Design Assignments should be uploaded to learn.colorado.edu, ECEN5623, "Project HLDR".

(If there is a problem with d2learn, then E-mail to john.pratt@colorado.edu with subject "GROUP SUBMISSION 1")

Project submission should be in an MS Word or text document with all group members clearly identified on a cover page! Only ONE submission needs to be made for all in the group.

HOMEWORK ASSIGNMENT (Individual Problem)
Homework objectives are: familiarization with Rate Monotonic and Deadline Monotonic policy and feasibility tests, understanding of sufficient vs. necessary and sufficient feasibility

Please read Deadline Monotonic Scheduling by Neil Audsley and the Real-Time Embedded Components and Systems text, Chapter 2, Processing section.

For each feasibility test, allocate sufficient memory to allow up to 20 tasks to be tested by your code. Give me evidence that you have tested each one of your feasibility tests by writing a driver to call the tests with examples from one of the papers, or your own example. The code may be automatically tested for grading, so it must conform to the naming and parameter specification given here and must be in the files requested -- Thanks!

1. Read Audsley Deadline Monotonic Theory paper.
2. Code the sufficient RM scheduling feasibility test (Liu and Layland paper p.9, Theorem 5) for following ANSI C function prototype:

```
int RM_sufficient(

int Ntasks,

int *tid,

unsigned long int *T,

unsigned long int *C,

unsigned long int *D);
```

   Where Ntasks is the number of tasks in the task set, tid is an array of unique task Ids, T is an array of the release periods, C is an array of the computation times, D is an array of the deadlines (T must equal D for each tid in RM). Finally, the function should simply return 1 if the system can be scheduled, and 0 if it can't.

3. Code the sufficient DM scheduling feasibility test (Audsley-1990, eq. 8) for following ANSI C function prototype:

```
int DM_sufficient(

int Ntasks,
```

```
int *tid,

unsigned long int *T,

unsigned long int *C,

unsigned long int *D,

double *S);
```

Where parameters are defined again as in #2 and S is an array the code populates with the scheduling feasibility of each task (i.e. the demand this task places on the CPU plus all interference with this task by tasks of higher priority).

4. Code the Completion Time Test (chap. 2 of RTECS text, Processing section) for the following ANSI C function prototype:

```
int Sched_completion(

int Ntasks,

int *tid,

unsigned long int *T,

unsigned long int *C,

unsigned long int *D);
```

Where parameters are defined again as in #2. Assume that T must equal D in all cases.

LAB ASSIGNMENT (Individual Problem)

Lab learning objectives are: basic methods for timing and tuning VxWorks task scheduling using Tornado tools.

1. Write VxWorks code which releases at least two tasks using timers and allows each task's timer to be independently configured. You may use posix_rt_timers.c as an example of how to use the POSIX timers. Each task's timing should be programmable with at least 10ms resolution. Demo your code and show the System Viewer trace that shows the tasks behave as desired.

2. Code a function to compute the Fibbonacci sequence to any number of terms. The sequence is 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ... The Fibonacci sequence or Fibonacci numbers begins with 0 and 1. The next term is then the sum of the two previous terms. (Do not be concerned if your Fibonacci number overflows an unsigned 32-bit integer, just let it overflow).

3. Now, determine how many terms in the sequence corresponds to 10 milliseconds of computation on a lab target PC (note that the answer may vary depending upon the specific target used). The easiset way to do this is to use System Viewer to measure the CPU time taken by a task calling your function with a large value - see how long that takes and then scale up or down the number as needed to achiever 10 milliseconds of computation. Repeat this to determine how many terms are required for 20 milliseconds of computation using System Viewer. [This has to be done on a lab machine, simulator will not be acceptable]

4. Use your timer implementation to create a task set with two tasks calling your Fibbonacci sequence, one with N terms for 10 milliseconds of execution and the other for 20 milliseconds. Is the system feasible if the 10 millisecond task is released every 20 milliseconds and the 20 millisecond task every 50 milliseconds? (I.e. T1=20 msec, T2=50 msec, C1=10 msec, C2=20 msec and all Di's = Ti's) - Base your answer upon the Lehockzy, Shah, and Ding Theorem.

5. Run the above system and show evidence that it either works or explain why it won't using System Viewer traces..

DESIGN ASSIGNMENT - High-Level Design (Group Problem) -- Turn in DFD (Data Flow Diagram) showing both hardware and software elements in your group project (one copy per group). Separately, turn in a paragraph describing your role in this high-level design.

Please turn individual paragraph in by uploading to d2learn Project 1.