CSCE 496/896-003: Real-Time Systems HW #1

Assigned: 2016-01-15
Due: 2016-01-29 in class

Homework Overview

The objectives of this homework are to demonstrate why real-time computing is important, particularly hard real-time computing. In this homework you will:

- Investigate how timing works on a non-Real-Time Operating System
- Write a simple RTS on your chosen non-RTOS
- Demonstrate how its lack of timing guarantees makes it difficult to use for real-time applications

Expected Outcomes (what I want you to learn)

- First and foremost, RTS implemented on non-RTOS are ubiquitous particularly in small embedded robotics application. This is primarily because they're cheap and easy. However, if the load is too high for the processor, or the scheduler decides to not run your task (or for any other reason), your robot may fail (particularly in a safety-critical application such as an aerial vehicle). This helps you understand why it's a risk to use a non-RTOS in these applications.
- Timing in the big OSes (Mac OS X, Windows, Linux) is usually a second-class citizen. Processes are given CPU time and resources based on the user, the kernel, or services. The reasoning is that **on average** the load will not exceed the capabilities, and if it does the only consequence is an unhappy user. This solidifies for you the nature of timing in traditional OSes.
- Up to this point you have probably learned that algorithm correctness is only based on correct output. This gives you the opportunity to demonstrate that in a RTS correctness also depends on when (in physical time) the output is computed. A correct answer at the wrong time is **incorrect** behavior for a RTS.
- You will learn how to write a simple RTS (I recommend using a cyclic executive).
- Tracing the timing in a task and understanding how it contributes to failure is very difficult. This is a first step in helping you see how timing is connected to output in a RTS and gives you experience in understanding how to examine and trace the timing in a RTS.

Deliverables (what you turn in)

- Code demonstrating your implemented simple RTS on your chosen non-RTOS (you can either print out your code or send it to me in a zip file by email)
- Evidence (e.g. plots, diagrams, graphs, pictures, etc.) supporting your hypothesis that the lack of timing guarantees in the non-RTOS you chose makes it difficult to use for real-time applications
- A brief (typed) writeup explaining what you did and how your evidence proves your hypothesis. Be sure to incorporate your evidence into the writeup.

One Possible Approach

Feel free to accomplish this assignment in any way you can come up with. I'm not restricting what you demonstrate, only that it must demonstrate how the lack of timing guarantees in a non-RTOS makes it difficult to use for real-time applications. There are many properties of RTS you could attempt to show are violated. For example, you could show how the run-time performance cannot be analytically guaranteed. You could show that resources are inadequate to meet deadlines. You could show that safety is compromised. You could show that deadlines are missed.

To get you started here is one approach to accomplishing the objectives of this assignment. The objective is to show that hard-deadlines are missed. Steps:

1. Write 2 or 3 "tasks" or "processes" which do something (doesn't need to be anything interesting, print statements, compute prime numbers, or whatever). Assume at least one of the tasks a "hard" deadline task.
2. Write a cyclic executive RTS which loops over the execution of these tasks using timing statements
3. Capture the reality that deadlines are being missed
   (a) capture and collect timing information about when each task starts and finishes as well as the progression of physical time
   (b) put a high load on your machine as you collect the timing information (perhaps you run an infinite while loop in another process, or perhaps you start a video on Amazon or Netflix if you're on your personal computer)
   (c) export the collected timing data and plot it in Matlab, Python, or Excel (or whatever) against the actual progression of time
   (d) trace through the timing information and show that deadlines are missed by marking the plot

Remember this is just one idea. I encourage you to be creative. If you have access to an embedded robotics application of some kind this may be even easier since those computers can generally tolerate only a much smaller load:

1. Write a much more resource intensive control algorithm (for demonstration sake, maybe you start computing prime numbers up to some absurd threshold)
2. Execute the code on the robot and see what happens
   (a) if it's a flying vehicle, can the vehicle remain flying?
   (b) if you can capture motion information you could compare robot performance with the resource intensive control algorithm against the normal control algorithm

Grading

100 points total:

- 30 points for the code. Grading based on:
  - comments, well organized
- 40 points for the writeup. Grading based on:
  - typed format, making a convincing argument
- 30 points for convincing evidence. Grading based on:
  - ease of understanding your plots, figures, diagrams, etc.