Extending the Arduino Real Time Kernel

James Boland, Andrew Johnson, Eric Zakrzewski

Introduction

The Arduino Real Time Kernel (ARTK) is a real time, priority driven, multitasking kernel built for arduino by Paul Schimpf at Eastern Washington University in 2012. This is a proof of concept with the goal to extend his original project by adding different scheduling algorithms for processes and comparing their performance.

Background

The ARTK has a priority-driven task scheduling with round-robin preemption. It operates on a fixed stack size of 256 bytes and a memory overhead of approximately 2 kilobytes. This is not including the additional housekeeping memory used for each task that is written.

The library operates similarly to using standard Arduino code, with the notable exception that a “loop()” method is not implemented. Instead, each task is defined as a void method that takes no arguments. Tasks are added to the queue with the ‘ARTK_CreateTask’ function which accepts a function pointer and priority.

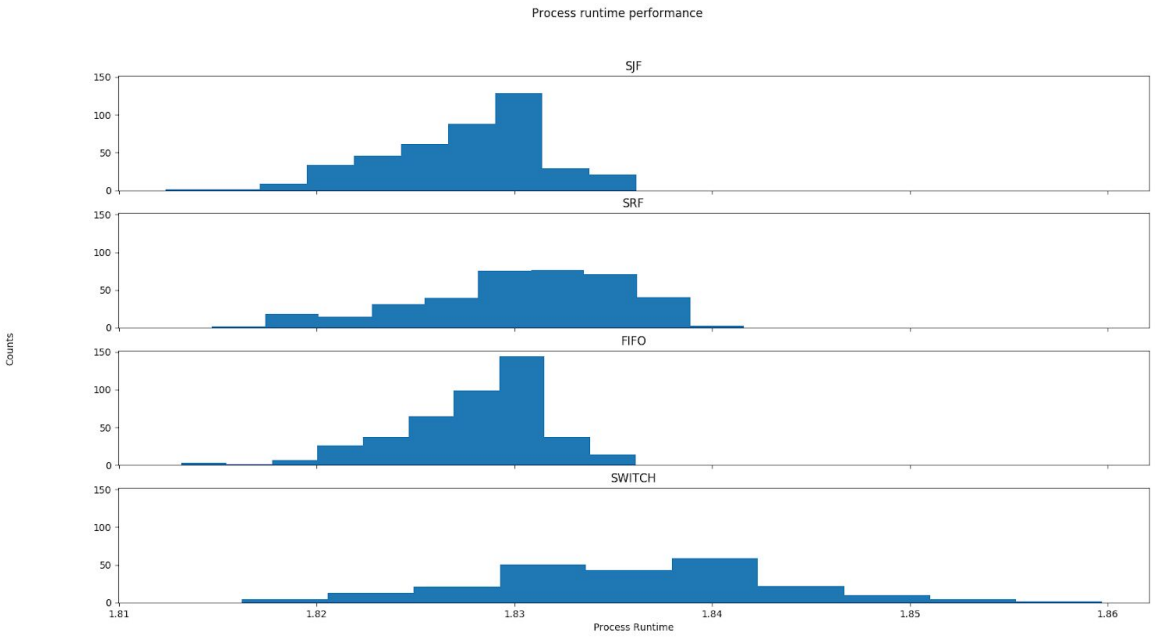
Methods

The main idea of this project is to test different stresses and scheduling efficiencies on a small processing unit with very little power so that the algorithms may be tested at a bare bones level. To do so, the Arduino UNO was chosen to make use of ARTK and also as a basis to test our modifications. The UNO board has very limited resources, even among Arduinos, as it has 8kb of main memory.

Four main scheduling algorithms were tested on the Arduino: First In First Out (FIFO), Shortest Job First (SJF), Shortest Remaining Time (SRT), and Switch Every Tick. The performance of the algorithms was calculated by measuring the amount of time needed to run two tasks. Each task counted up to a random integer from a normal distribution. To implement the various scheduling algorithms task priorities were manipulated to control when tasks start as well as calling ARTK_Yield() to swap processes at run time.

Results

For the purposes of the experiment, a tick is considered the amount of time needed to increment the counter of a single task.



From top to bottom the algorithms displayed on the graph are: Shortest Job First, Shortest Remaining First, First In First Out, and Switch every tick. Unsurprisingly, the Switch algorithm had the least consistent performance. This is indicated by the large spread on the histogram. This is used as a baseline to compare the performance of the other algorithms. SJF and FIFO had very similar performance while SRF performed slightly worse. SRF and Switch are the only two algorithms that yield at run time which would imply that the primary factor in performance is how much the context switches.

Summary

The UNO board being only able to run two tasks at a time made results hard to distinguish. Algorithms with the largest overhead performed the worst in our simulations. This can be reasoned to be a result of the limit two processes. Context switching can be quite expensive in time so the processes that switched the most had the slowest completion times. This could be why SJF performed so well because the amount of switches would be at most one due to only having two options, no other possible shorter jobs could enter the queue until

the shortest job completed. This can lead to starving, however, particularly when dealing with this few of processes. Starving processes are very hard to mitigate in this algorithm.

First in first out is likely the best performer as well as the most reasonable scheduling algorithm to implement given the circumstances. FIFO is quite fair when only two processes are queued. It also prevents starving and has the fewest context switches possible. The main drawback for the FIFO in this environment is if there was a hard real time process. FIFO does not enable preemption, so timely processes would not receive any priority.

This proof of concept provided a lot of learning opportunities. Giving justification to theory about algorithms first hand helped with understanding. Pros and cons of scheduling ideas really came too light when considering preemption and fairness. If priority preemption and task switching is needed within the Arduino, look into Interrupt Service Routines.