CIS 415 Operating Systems

Project 2 Report Collection

Submitted to:
Prof. Allen Malony

Author:
Jamie Whiting
jwhiting
951972557

Report

Introduction

In this project, I developed a Master Control Program (MCP) designed to manage and schedule multiple processes with varying resource needs. The primary goal of this project was to implement a round-robin scheduler that allocates CPU time to processes while tracking resource usage, such as memory, CPU time, and voluntary context switches, through data available in the /proc filesystem.

Background

The main algorithm used in this project was a round robin scheduling procedure. Round-robin scheduling involves assigning each process a fixed time slice, during which it has exclusive access to the CPU. Once a process's time slice expires, it is suspended, and the next process in line is scheduled. This method ensures fairness by giving each process an equal opportunity to use the CPU, which is particularly useful in a multi process environment. However, round-robin scheduling does not adapt to the unique demands of different processes, like those that are I/O bound vs CPU bound.

Implementation

The implementation of the MCP was broken down into several parts, each corresponding to a phase of the final project. The initial implementation focused on creating a basic program to read an input file and properly make fork() and exec() calls to run processes, then call wait() to wait for processes to terminate, after which exit() is called. Next the goal was to implement signals with the process' so that each child process must wait for a signal before executing. This gave the MCP control of the execution process and scheduling situation. The next part of the implementation was the juicy section, implementing the round robin scheduling algorithm. I found this implementation to be quite intuitive and fun to work with. The final part of this project was to add the functionality to gather data on the processes and analyze it through a table. At first, I wished to implement data collection and analyzation of memory used, CPU usage, and io bytes read. I found the first two relatively simple to capture and present. I struggled vividly to implement the bytes read as I kept finding the issue that my io bytes read was being captured but the value was 0, after many different attempts to change input and find my solution I decided to dig back into /proc and find a different piece of information that interested me to look at. I landed on voluntary context switches and implemented that value into my table. As the scheduler goes through the whole program I find it very interesting to see the number of context switches increase each time around and visualize the relationship between context switches and CPU usage.

Performance Results and Discussion

Through intensive testing of my final implementation of the MCP I found some common trends through the analyzation of my results. The three metrics I focused on were CPU usage, memory usage, and voluntary context switches. I found the memory usage to be quite consistent for each process, typically with values around 900kb. I believe this to mean that the MCP is efficiently allocating memory for each process without any excessive use. When using valgrind with each output, as included through my log#.txt files, there are no memory leaks or errors through the processes and execution. The CPU tick time and context switches increase incrementally and show how the scheduler suspends and resumes each process. It is interesting to look at the differences in the context switches when you change the time slice of the round robin process to be larger, reducing the context switch amount. Through each of my parts I got 0 errors from valgrind, this is all visible in my log documents.

Conclusion

Through working on this project, I learned a lot about working with the scheduling processes and making correct system calls. I found it intriguing working with the alarm calls and signaling the processes to stop continue or exit.