CSE2431 Final Project - November Submission

**Group Members:**

Jiaqian Huang

Andrew Maloney

Alec Wilson

**Abstract**

Developing CPU scheduling algorithms can be difficult because the operating system kernel code needs to be modified and tested. Then measurements need to be made to determine how fast processes run and compare that to how fast they were predicted to. After processes start to execute in the CPU, CPU bursts of various lengths progress execution of those processes. This project aims to create a linux module to predict the length of the next CPU burst using an exponential averaging routine like those shown in Chapter 6 of *Operating System Concepts* about CPU scheduling.

**Introduction**

The operating system is able to have several processes running on a machine. The execution of those processes are the reason that CPU bursts occur. Although CPU bursts may appear to happen at random intervals for random amounts of time, CPU scheduling aims for CPU bursts to be run as efficiently as possible. With CPU scheduling, we make process execute one after another. So every process will execute for some duration, and after that process completes its execution the next one from a waiting queue will generate its own CPU burst. Therefore, it is helpful if the scheduler can predict the next CPU burst for waiting processes so that it can improve the execution order for processes.

However there were some concerns going into this project. Firstly, processes have associated priorities. In a system with preemption a currently executing process needs to stop immediately because of another process with higher priority, or because of a time-out according to the rules of the algorithm used, like in a round robin scheduling algorithm. Even menial processes are still classified according to their importance when they are created. Secondly, the schedulers job is not done when a process arrives and begins execution. The predicted time for a process to run on the CPU can be different than the actual time for this process to execute. The aim of this project is to compare the ideal length of the next burst with actual length of the next burst. This can be helpful for researchers to understand the various factors affecting the length of the next burst by monitoring the changes in burst times over time.

**Plans and Functionalities**

The preliminary plan was to calculate the length of next CPU burst with the use of exponential averaging. For this functionality, a kernel module would be implemented as a command with flags to record and/or predict CPU burst timings. A record flag could record the CPU burst timings coming from the CPU, and either stored them in a file or output them to the terminal for a specified amount of time. A prediction flag would allow the selection of different prediction methods to be applied to a log file of CPU burst information with the output going to a file or a terminal. A realtime flag could apply a selected prediction method to the current CPU burst times and could either provide that information to an updating terminal window or to a log file.

An investigation was first made on how to obtain the length of time that a process ran on the CPU, and how to log that information. It was determined that some values could be read directly from the a file located in folder named by the process’ ID within the /proc folder. This file contains status information regarding the process, and it is defined in the kernel source file as well. However, at that time it was not feasible to implement a kernel modules to retrieve that information from the process’ proc folder file. Instead the file was accessed directly, and by using the known formatting of the file, only the information required for calculations were filtered out, and each value was scanned accordingly.

Once the timing was understood a system was made to compute the exponential averaging on the process files, and some effective prediction methods were determined. Next an exponential averaging scheme was implemented into operate in real time and store the predictions and actual values to a file. All of these functions could then be combined into a readout to a terminal that showed how different prediction methods were done while predicting CPU burst times in real time.

To make teamwork more efficient and avoid duplicating work, a github repository was made to share project code with every team member (a link is provided in the references section). A Makefile command file and a process file with a kernel module were also included. The Makefile contains command code which will compile the kernel module C code in a form that can be mounted as a module.  proclog.c, the kernel modulecreates a writeable file in the /proc folder so that futher logs can be made. However, no integration was made between that and the logger.

To run this project, the file runLogger must be executed. It will ask whether the process should be logged - if not then the output shall only appear in the terminal. It will also ask for a rate at which the process will poll the proc folder, from 20 to 999 ms. The user will then be asked for a process ID for which the length of next CPU burst will be calculated. A list of process IDs can be found with help of terminal command like top, or by mounting the kernel module process.ko and running a dmesg command. After inputting the process ID, the code will access this status file in this process ID folder and create a log.txt file in the local folder to store the actual length of next 10 CPU burst, with user CPU ticks on the right and system CPU ticks on the right. The relevant information is also printed in the terminal showing the outcome with nice format so that the results can be easily read. After running for 10 bursts, the user will be asked if they want to run again. If so, they will be asked for a process ID again and the code will repeat from there. Otherwise the code will exit.

**Conclusion**

After completing this project, the goal made when starting this project was achieved: to get a prediction of the length of next CPU burst. A better understanding was made about kernel modules and the operating system itself. Actual CPU ticks were stored in the log.txt file to check out the results. In addition, there is great potential to make this project better and improve it in various ways. One idea is to get the status information about processes from a kernel module which can make the code simpler and more efficient.

**Reference:**

Github url: <https://github.com/Kalih27/CSE2431-Final-Project>

**Resources:**

[https://stackoverflow.com/questions/19181834/what-is-the-concept-of-vruntime-in-cfs#](https://stackoverflow.com/questions/19181834/what-is-the-concept-of-vruntime-in-cfs)

<https://www.cs.columbia.edu/~smb/classes/s06-4118/l13.pdf>

<https://www.linuxjournal.com/article/8110>

<https://elixir.bootlin.com/linux/v4.19/source/include/linux/sched.h>

<https://stackoverflow.com/questions/2844/how-do-you-format-an-unsigned-long-long-int-using-printf>

Mauerer, Professional Linux Kernel Architecture, Wrox Press