TMD CPU Scheduler Simulation

Group name: Group TMD (Tan, Matt, Danny)

Group members: Tan Duong, Matthew Lebo, Xiao (Danny) Luo

Objectives:

- To better understand the differences between each of the CPU Scheduling algorithms
- To improve our C programming skills
- To save time from calculating when facing the huge number of processes
- To enhance knowledge studied in class

Motivations:

Our primary motivation in selecting this project was to better understand CPU scheduling and the different algorithms that are mainly used for it. The CPU scheduler is responsible for maintaining factors such as fairness, efficiency, response time, and throughput when switching between processes. Creating this program allowed us to become more familiar with how it handles each of those things and allowed us to easily and quickly get data from the different algorithms to analyze.

Project Overview:

- Build the programs for the types of CPU Scheduling (FCFS, SRTF, SJF, RR).
- Write all the Programs in C
- Read the input from the user's file (The burst time and arrival time of processes, as well as a time quantum and context switch overhead)
- Return output with Gantt Chart (except for SRTF) and a table with data about the average waiting time and average turnaround time of the processes given. The table also includes data on each individual process

Work Distribution:

Matthew: Wrote the round robin program, formatted the input and output of each program to match each other, wrote a program that lets you select which algorithm to use, and worked on presentations and reports.

Tan: Wrote the shortest remaining time program, wrote part of the first come first serve program, and worked on presentations and reports.

Xiao (Danny): Wrote the shortest time remaining program and wrote part of the first come first serve program.

Aside from the bulk of the code for each program, each of us did some testing, troubleshooting, and bug-fixing for all of the programs.

Project Description:

Our program takes the burst times and arrival times of a set of processes as well as a time quantum and context switch overhead time as input from a file. It then outputs a gantt chart as well as a table showing wait times, turnaround times, completion times, etc. for each process. It also includes the average wait and turnaround times across all of the processes.

We initially wanted to build a mock up CPU scheduler that could take dummy processes and switch between them, basically making a really simple version of an actual CPU scheduler. We originally chose this without having much knowledge of CPU schedulers and later realized that this would be way too large of a scope, if not essentially impossible. So instead we went with a simpler program that calculates data for four different CPU scheduling algorithms. After researching and studying for the differences and work of each, we started by implementing the FCFS algorithm (the simplest algorithm) before our progress report. Since this program worked, we continued on to the other three and then created a separate program that allows you to choose which one to run.

We used the examples in the slides and handouts that the Professor provided in class for testing because it has the solution so we could easily to check the output. We tested other cases, but most of our initial testing was with those examples.

Implementation Details:

We created the C programs of the four CPU Scheduling algorithms based on the knowledge learned from lectures and from what we have been researching since September. We compiled and ran the code on the command line, most of using Cygwin since we have Windows OS.

The ways we implemented each algorithm:

- **FCFS -** Order processes by arrival time then calculate completion times
- **SJF -** Track which processes have arrived, run the shortest one from those arrived
- **STRF -** Increment time units and run the shortest job for each unit
- **RR -** Sort the processes by arrival time and then iterate through the list and give time

slices to processes that have arrived at that time. When no processes have any remaining burst time, the results are calculated and printed out.

<u>Input and Output</u>: We used standard C functions from stdio.h in order to handle this. The programs read the input file one line at a time and store it in a string. We then used sscanf() in order to get the data from the string. This was convenient because it allowed us to use the built in formatting of scan (for example "%d, %d" in order to only get two integers separated by a comma and a space, ignoring anything else). Output was just handled by printf() statements. We made use of the option to add padding to the front of numbers for our table formatting (like %8d to guarantee the number will always take up 8 spaces). It prints the table to the command line, but it could easily be modified to print to a file by changing printf() to fprintf() and specifying an output file.

<u>Combining the programs</u>: We made a 5th program that simply prompts the user to specify what algorithm they would like to run on the input file, since all of the algorithms can accept the same format of input file. Once the user has selected the algorithm, it hands the number chosen to a switch statement with an execlp for each algorithm. It uses execlp to call the chosen algorithm and passes the input file as a command line argument.

Challenges and Problems Met:

- As mentioned above, our original vision of the project was unrealistic due to a lack of knowledge on the subject, so we had to change it half way through
- The final result doesn't feel like it entirely fits the scope of the project. It would've been nice to have more functionality
- We wanted to add a GUI, but from our research it looked like that was only really a

viable option if we had written it in an object oriented language

- Group Communication and not everyone being available at the same time
- Working in C was a challenge for some members
- Commit more often so we can collaborate more easily

Future Work:

- Translate the programs into C++ so that we could add a GUI
- Allow the program to read from different file formats (.xls, .csv, etc.) rather than a plain text file.
- Implement I/O waiting into time analysis so it would be more realistic

Repo URL (GitLab): https://gitlab.com/tandduong/tan-matthew-finalproject-cosc439/ (updated)

References:

“What Is CPU Scheduling?” *Studytonight.com*, https://www.studytonight.com/operating-system/cpu-scheduling.

“Operating System Scheduling Algorithms.” *Tutorialspoint*, https://www.tutorialspoint.com/operating_system/os_process_scheduling_algorithms.htm