ÇUKUROVA UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT
CEN354 Operating Systems
Assignment 2 Report

04.05.2021

Buğra Sarıkaya
2016556506
İ.Ö.

Introduction: The assignment's purpose is measuring student's coding abilities and knowledge about CPU scheduling. In assignment repository, there is an uncompleted project. It has 5 incomplete files which are required for implementing CPU scheduling. The advisor wants student to complete these files, upload them by using Git and make a video presentation.
Researching: Before developing this assignment's program, homework subject was researched. 2 video playlist were watched from internet for learning theory part of subject [1] [2]. Then several coding tutorials were examined for learning coding part of subject [3] [4]. Because assignment's reference is from the lecture book, other teachers also gave same assignment before. And because of that, there are many projects about this homework in GitHub. Some of them were examined [5] [6]. To develop a unique program, concepts which were learned from the video playlists, were sufficient. Available projects in GitHub and CPU scheduling tutorials were not used too much.
Preparation: In disk.c file, there is a function whose name is strsep() which is not in C standard. It can works in Code::Blocks with GCC compiler. Because of that, Code::Blocks with GCC compiler, was chosen for as preferred IDE to developing incomplete files. I created an empty project and added files from the repository. "main" function of "driver.c", has argc and argv parameters. I needed to run program by writing text file's name on terminal. Fortunately Code::Blocks has a feature to set program's initial arguments on IDE and I added a schedule.txt file as an argument into this Code::Blocks project. Instead of using Makefile for testing every versions of project, program was executed on IDE for testing. This simple method decreased testing time.
Functions: The schedule files have 3 principal functions. add(), pickNextTask(), schedule(). add() functions, inserts read tasks into a queue which is a singly linked list. add() function inserts tasks with its inner function of insert(). I thought, generally pushed or inserted nodes are always added tail's next but insert() function which its declaration is in list.c file, adds new nodes to head and therefore first read task is in tail node. Even though I thought this is a meaningless way, I did not change list.c file to protect assignment's original files. pickNextTask() function brings next task's pointer according to schedule algorithm. schedule() function runs tasks which are have come from pickNextTask(), by displaying tasks on console screen properly with run() function. schedule() and pickNextTask() works together to implement scheduling algorithm. Finished tasks' nodes are deleted from queue.
FCFS: schedule_fcfs.c file implements First Come First Serve algorithm. pickNextTask() function implements traversing until it reaches tail node and brings tail node's pointer. Tail node's task is first read task and FCFS algorithm works as FIFO. Always top tasks in list, are run. When a task is finished, algorithm runs next task. Because this algorithm is non-preemptive, waiting tasks do not prevent current task.
SJF: schedule_sjf.c file implements Shortest Job First algorithm. pickNextTask() brings the task which has most minimum burst time and schedule() function runs this task. This algorithm is non-preemptive. Finished task's node is deleted and then pickNextTask() brings the task which has most minimum burst time again from newly altered list. If tasks have same burst time, SJF algorithm implements FCFS. The task which has minimum task id, is run first.
Priority Scheduling: schedule_priority.c file implements priority scheduling. It works like SFJ. Difference is that pickNextTask() brings the task has maximum priority. Then schedule() function runs tasks according to their priorities. In this assignment, priority scheduling algorithm is non-preemptive. And there is not a solution for starvation.
RR: schedule_rr.c file algorithm implements Round-Robin algorithm. There is a cycle for running tasks by time-sharing. Time quantum is 10 milliseconds. Every 10 milliseconds, algorithm runs a task and then next task prevents current task. Algorithm continues to run from next task. This time-sharing schedule is determined by schedule() function. schedule() function has pickNextTask() function and it works like FCFS algorithm. In FCFS, top task is run, then it is deleted because it is finished. But in RR, top task come to schedule and it is not finished generally because of time-sharing. RR does not proceed to execute this unfinished task, it starts to finish next task according to time quantum. The program needs to remember unfinished tasks. In this file, pickNextTask() has different FCFS method. It bring the task to schedule and also remember next task by saving its pointer. Next time, the function brings saved task, then it saves next task again. Instead of saving next task, program cannot save current task because if current task is deleted, bringing current task will create segmentation fault due to using deleted pointer. In schedule() function, when a task is run, its burst time is decremented. If its burst time is 0, this means task is finished.
Priority with RR: schedule_priority_rr.c file implements priority with Round-Robin algorithm. Firstly, sort_list() function sorts queue according to priorities by Bubble Sort. Then program works as RR algorithm.
Average Times: After completing CPU scheduling algorithms, I wanted to add calculation of average times because I saw that some available projects in GitHub, shows average times. "time" variable added to program and it was initialized to 0. Every bursting, it increases. Response time, turnaround time and waiting time were calculated according to their formulas. Program saves these values into an array. Time value is assigned to array many times. "First time" value is when a task first arrives to CPU. "First time" value appears only code comments but it was expressed by "time" variable for calculations. Completion time is the last "time" value assigned to array. In this assignment, there are not arrival times. Because of this, arrival times were not used for subtracting in formulas. Arrival time value is always 0. Burst time value is from task's "burst" variable. Burst values were assigned while reading tasks but RR files has burst decrementing. Except RR files, all files uses assigned first "burst" values for calculating burst time. Instead of using assigned first "burst" values, I chose the method of incrementing burst values in every proceeding. "burst_array" variable which is an array, was added to the program and its elements were initialized to 0. Every bursting, these elements' values are increased. Because RR algorithms are preemptive and I thought, using assigned first "burst" value before its unfinished task, is not a dynamic method. Maybe CPU will stops at some point and there will be unfinished tasks. In this case, calculating unfinished tasks' burst times before they are finished, will be wrong. After calculating response times, turnaround times and waiting times, these values are divided by task count. Results shows average times. I added average time calculations extra but these provided me make comments to scheduling algorithms.
Conclusion: After examining average times on outputs, I saw RR programs have minimum average response times. Because of time-sharing with determined time quantum, tasks' first time values are minimal. With RR algorithm, tasks comes to CPU early. But their completion time is high and so RR programs' average waiting times and average turnaround times are maximum. Time-sharing method decreases response time but it affects negatively on turnaround time and waiting time by increasing them. For minimum turnaround times, SJF is best. Executing tasks which have minimum burst times firstly, decreases turnaround times and makes outputs is shown early.
YouTube Link: https://youtu.be/RhcTpCu7WbE
References:
[1] https://youtube.com/playlist?list=PLBlnK6fEyqRitWSE_AyyySWfhRgyA-rHk
[2] https://youtube.com/playlist?list=PLUUSh8du1azYHjIVKSMsaJQLqT0GSSmRk
[3] https://www.geeksforgeeks.org/operating-systems/
[4] https://www.thecrazyprogrammer.com/c-programs
[5] https://github.com/awalm/CPU-Scheduler
[6] https://github.com/ChristianAugustyn/EECS3221_MiniProject2