Kenny Liu, Jesus Mendoza

Dr. Xiaoyu Zhang

CS 433 – Operating Systems

28 October 2023

Assignment 3 Report

Submitted Files:

- scheduler_fcfs.h
- scheduler_fcfs.cpp
- scheduler_sjf.h
- scheduler_sjf.cpp
- scheduler_priority.h
- scheduler_priority.cpp
- scheduler_rr.h
- scheduler_rr.cpp

How to Compile/Run the Program:

```
make fcfs
make sjf
make priority
make rr
```

To run, command used:

```
./fcfs schedule.txt

./sjf schedule.txt

./priority schedule.txt

./rr schedule.txt
```

Features Implemented (via. scheduler):

FCFS- First Come First Serve is a non-preemptive scheduling algorithm that executes in order based on the arrival in the ready queue. So the first process is obtain and read in the list for first come first serve.

i. Constructor initializes the count,total_turnaround, and total_wait to 0's.

ii. Destructor will pop everything in queue until empty and vector till empty.

iii. Init function uses the push function to push each pcb and keep track of the process_list into the queue with the name, priority, and cpu burst.

iv. The print_results will print out statements of the turn-around time and its waiting time.Displays also the average total_turnaround divided by count and average waiting time by total_wait divided by count.

v. Simulate will use a queue to call the first process using Q.front(). Simulate will do the calculations to calculate the total around time and waiting time to transfer back to print_results. Its all in a while loop until the queue is empty.

SJF - Short Job First will prioritize the process based on their burst time choosing the shortest job for execution.

i. From the FCFS, the SJF constructor initializes key variables, including 'count,' 'total_turnaround,' and 'total_wait,' all initially set to zero.

ii. The operation of SJF is similar to FCFS in terms of managing the queue to ensure it's empty.

iii. Unlike FCFS, SJF introduces a crucial distinction. It orders processes by their burst time, ensuring that the process with the shortest burst time is scheduled first.

iv. The sorting mechanism compares and rearranges elements within the vector, so that the process with the shortest burst time is at the front of the queue.

v. Similar to FCFS, SJF provides a detailed display of computations, showing turn-around times, their averages, waiting times, and their averages.

vi. The 'simulate()' function in SJF operates similarly to FCFS. It calculates the 'total_turnaround' and 'total_wait' times and simulates the execution of processes according to the SJF scheduling rules.

Priority- this will check each process's burst time. But it will check the highest priority of each process with the largest number. Thus, it will execute the highest priority first.

i. Same as previous, the constructor is built with the initializes the count, total_turnaround, and total_wait to 0.

ii. Destructor is the same like the previous schedulers we did by making sure the queue i empty.

iii. It will use the sort function like SJF where we assign the first and last elements of the vector to sort the highest priority and be push back.

iv. Print_results will print out the results like the previous ones lik the turnaround time, plus the average as well as the waiting time with average.

v. simulate function will do the same calculations as before.

Round Robin(rr) - uses time quantum to determine each process burst time.

i. Unlike other schedulers, RR introduces a vital element - the 'time quantum.' It's a user-defined parameter that plays a significant role in the scheduling process. Similar to other schedulers, it initializes 'count,' 'total_turnaround,' and 'total_wait,' all set to zero.

ii. Deconstruct is similar as with all schedulers, RR starts by emptying the queue of processes.

iii. RR's uniqueness lies in its treatment of the 'time quantum.' The initialize function does not require sorting processes by priority or burst time, as it's not relevant in RR's case. It follows a setup akin to FCFS

iv. In line with previous schedulers, RR prints the order of process execution by sorting the processes' IDs. It also displays turnaround times, waiting times, and their averages

v. The 'simulate()' function in RR has two distinct conditions as it iterates through the process list. The first condition checks if the current burst time is higher than the quantum, and when it's lower than the quantum. When both conditions are met, RR schedules and displays the running process and the current burst within the time quantum. Once completed, it pops both the queue and vector. In the first condition, the front element of the queue gets assigned the remaining burst time after deducting the quantum. When the second condition is met, RR continues to process all the remaining burst times. It records the total turn-around time and total waiting time throughout this phase.

Additional:

- Queue<PCB> Q stores process into a waiting queue like burst time or priority
- Vector<PCB> done serves as a container to store the processes along with their turnaround times and waiting times. Used to print out the results we collected

Choice of Data Structure: The choice of data structures, namely queues and vectors, was made to effectively implement the scheduling algorithms for this programming assignment. Queues were predominantly employed as the ready queue, serving as the data structure to manage the processes that need to be executed. This is evident in the 'init' functions of all the submitted files, where processes are pushed into the queue for simulation, helping to calculate turnaround and waiting times.

Lessons Learned/Re-Learned: This time we started the assignment ahead and did not push it to the last minute. We learned a lot as we go creating different approaches in our code. Even though at first it looked confusing and had no idea where to start. The help of in-class lectures and help of practice problems/exercises from class and HW helped a lot when performing these calculations for FCFS, SJF, rr, and priority too. It was actually nice seeing that the practice we did in class was put into a program.