Project Experience

The purpose of this project was to process a batch of jobs and simulate their scheduling using two popular scheduling algorithms: the First Come First Serve and RoundRobin (RR) algorithms respectively. I utilized C++ for this project, which was a change from using C in the previous two projects. The main reason for this was to make use of the STL Vector class, to queue up the jobs processed from the text file. Due to C being rigid with its memory management, it was less than optimal to suit this particular problem.

This problem was solved by first creating a Job class, which contained attributes for the job such as its name, arrival time, duration and remaining time. From here, I could dynamically instantiate a new job object as and when it is read from the text file allowing for more optimal program performance. To implement the two algorithms, I declared two public functions within the scope of the file and called them after parsing the file.

For the **schedule\_FCFS** function, it was rather straightforward in its implementation almost alike to a queue with a FIFO ordering scheme. Here, the task that arrives first is processed first and hence the implementation was rather trivial in the grand scheme of things.

For the **schedule\_RoundRobin** function, It takes a vector of Job objects representing tasks to be scheduled and the total time available for scheduling. The algorithm divides the available time into equal time slices, known as time quantum, set to 1 unit.

Firstly, it initializes a 2D matrix **RRmatrix** to represent the scheduling chart for each job over time. Then, it iterates through each time unit within the total time, selecting a job from the list based on its arrival time and remaining processing time. If the selected job is ready to run (arrival time is less than or equal to the current time and it still has processing time left), it is scheduled for execution for one time quantum.

The function continues this process, incrementing the current time by the time quantum until the total time is reached. For each scheduled time unit, the **RRmatrix** records the presence of the job's execution. Finally, it prints out the scheduling chart, indicating which jobs are running at each time unit with 'X', and spaces for idle time.

Top of Form

Project Difficulties & Takeaways

This project challenged my understanding of the implementation of different scheduling algorithms, as well as the consideration that goes into their integration within modern operating systems. My biggest challenge over the course of the project was in tackling the round robin scheduling algorithm as modifying the queue of jobs to be processed as and when that job gets its respective time slice was complicated and frustrating at times. The second hurdle was in trying to find a representation to visualize the schedule graph first before printing it. Unlike the FCFS algorithm where it can be printed line by line, this algorithm entailed jumping back and forth between jobs. To overcome this, I used a two-dimensional array to track job execution (with a 1) and idle (0) – later in the program code, I would iterate over the array a second time to print out the schedule accordingly. Although in the real world, this implementation would not be the most optimal as far as time complexity is concerned, it was still apt to solve the problem at hand. Thus, exploring various optimization techniques, such as adjusting the time quantum, offered insights into fine-tuning scheduling algorithms for specific system requirements.

Project Results

This program takes the following text file of jobs, parses it and runs the First-Come First-Serve (FCFS) and Round Robin (RR) algorithms.

A screenshot of a computer

Description automatically generated

(jobs.txt – Input File)

A black background with white text

Description automatically generated

**(Output with the Two Scheduling Algorithms)**