Table of Contents

1.0 Code Compilation

Installing C language IDE

1. Download Code Blocks, VS Code or any preferred IDE of your choice.
2. Install the C language IDE using the downloaded installer along with GCC and direct the installation to your preferred path.

Downloading files

3. Download `coursework.c`

Run code

4. Run `coursework.c`

Alternative

5. Run the program using online IDE

2.0 Code Explanation

On 'run' button click, the program will prompt the user to input the total number of processes for the scheduler and assign each process a unique process ID. Then, the program will prompt the user to input the **burst time**- time required for complete execution for each process. In addition, the user is required to input the **time quantum**- fixed maximum interval time for the CPU to shift to the next process which is required for Round Robin (RR) scheduling algorithm.

Upon successful user's inputs, the program will calculate the respective **waiting time**- total amount of time spent by a process in the ready queue, and the **turnaround time**- difference between a job's submission and completion times for each process in First Come First Serve (FCFS) and Round Robin (RR) scheduler algorithms. The average waiting time and the average turnaround time for the above schedulers will then be calculated and compared to determine the efficiency of different schedulers. The program will run infinitely until the user chooses to exit.

The formula for turnaround time, waiting time, average turnaround time, and average waiting time are shown below:

Turnaround Time = Completion Time - Arrival Time

Waiting Time = Turnaround time - Burst Time

$$\textbf{Average Turnaround Time} = \frac{\textbf{Total Turnaround Time for All Processes}}{\textbf{Number of Processes}}$$

$$\textbf{Average Waiting Time} = \frac{\textbf{Total Waiting Time for All Processes}}{\textbf{Number of Processes}}$$

Assumptions:

1) The arrival time for every process is 0ms, right when the user successfully entered all required inputs. Hence, the turnaround time is equal to the completion time, which also inferred that the average turnaround time is same as the average completion time.
2) There is no overhead for every context-switch, where the process of changing between different processes is immediate and conflict-free.

2.1 Input Validation

As mentioned above, users are required to input the total number of processes, burst time for each process and time quantum. All inputs will then be parsed to respective validation functions before proceeding to the processing algorithm. If the input is invalid, the system will remind and prompt the user to enter a new input.

Three functions are implemented for input validation:

```
// functions for input validation
int checkNumber();
char checkCharacter();
bool checkTimeQuantum(int input);
```

The first function [checkNumber] determines if the input is a digit number larger than 0, as the total number of processes, burst time for each process and time quantum should not be a zero or negative number. The input will then be converted into an integer number using the floor function.

The second function [checkCharacter] determines if the input is an alphabet character when prompting user whether to continue or quit program.

The third function [checkTimeQuantum] checks whether the time quantum provided by the user is 10-100ms and returns Boolean value.

2.2 First Come First Serve (FCFS)

First Come First Serve (FCFS) is a non-preemptive scheduling algorithm which is based on the first-in-first-out (FIFO) queue. FIFO simply queue processes in the order that they arrive in the ready queue [1].

Three functions are implemented for the First Come First Serve scheduling:

```
// functions for FCFS
void findWT(int processes[], int n, int bt[], int wt[]);
void findTAT(int processes[], int n, int bt[], int wt[], int tat[]);
void printFCFS(int processes[], int n, int bt[]);
```

The first function [findWT] takes in an array of process ID, total number of processes, burst time, and an empty array for storing process waiting time. It is used to calculate the waiting time for each process using the First Come First Serve algorithm.

The second function [findTAT] takes in an array of process ID, total number of processes, burst time, an empty array for storing process waiting time, and an empty array for storing process turnaround time. It is used to calculate the turnaround time and completion time for each process using the First Come First Serve algorithm.

The third function [printFCFS] takes in an array of process ID, total number of processes, and burst time. It is used to output all the details of the First Come First Serve scheduling and calculate the average waiting time and turnaround time.

2.3 Round Robin (RR)

Round Robin (RR) is a preemptive process scheduling algorithm where each process on the ready queue is executed in turn for a fixed time quantum and the valid input for time quantum is ranged between 10ms to 100ms. If the process has a burst time that is higher than the time quantum, the time quantum expires, and the process is interrupted and added to the back of the ready queue where the next process will take over. It uses context switching to save states of pre-empted processes which leads to time overhead [2]. Note that the RR scheduling algorithm provided in this coursework does not include any overhead.

Two functions are implemented for the Round Robin scheduling:

```
// functions for RR
void algorithm_RR(int process[], int n, int burst[], int wait[], int
turnaround[], int timequantum);
void avgtime_RR(int process[], int n, int burst[], int timequantum);
```

The first function [algorithm_RR] takes in an array of process ID, total number of processes, burst time, an empty array for storing process waiting time, an empty array for storing process turnaround time, and time quantum. It is used to calculate the turnaround time and completion time for each process using the Round Robin algorithm.

The second function [avgtime_RR] takes in an array of process ID, total number of processes, burst time, and time quantum. It is used to calculate the average waiting time and average turnaround time in the Round Robin scheduler by calling the first function inside its function body. Then, it will output all the details of the Round Robin scheduling.

2.4 Multi-Level Feedback Queue (MLFQ)

Multi-level Feedback Queue (MLFQ) is a priority-based system with different queues for each priority level. It is not an independent scheduling algorithm as different queues may implement different scheduling algorithm and the processes can change their queue i.e., if a process is in queue1, then after partial execution, it can switch to queue2. Thus, if a process with a higher priority queue arrives, then the execution of the lower priority queue will stop [3].

In this coursework, the two ready queues are used by the MLFQ scheduler. RR is used to schedule the first queue, which is utilized for interactive processes. FCFS is used to schedule the second queue. The new processes enter the system through the first queue. A process is demoted to the second queue if it uses up all of its time quantum. There is no ageing and therefore no rule is implemented for process promotion.

One function is implemented for the Multi-level Feedback Queue scheduling:

```
// functions for MLFQ
void printMLFQ(int processes[], int n, int bt[], int timequantum);
```

The function [printMLFQ] takes in an array of process ID, total number of processes, burst time, and time quantum. The array of process ID enters the system through the first queue. A flag variable is used to check if any process has exhausted all of its time quantum and is demoted to the second queue. Once all processes are finished, it will output all the details of both queues.

2.5 Comparison

One function is implemented for comparison:

```
// function for comparison
void comparison();
```

All three schedulers are compared based on each scheduler's average waiting time and average turnaround time to find the best scheduler for a workload by using if-else statements. Once the comparison process is finished, it will output the three preferred schedulers—the scheduler with the shortest total average waiting time and turnaround time, the scheduler with the shortest average waiting time, and the scheduler with the shortest average turnaround time.

3.0 Conclusion

Based on the observation and discussion above we have come into a conclusion that different schedulers have its own advantages and disadvantages which are preferred for different workloads. FCFS's efficiency is dependent on a process's burst time while RR's efficiency is dependent on the time quantum. FCFS favors a smaller burst time while a RR favors a larger time quantum. For MLFQ, the execution of processes is dependent on their priority level. We have also proposed a few ways to tune MLFQ to overcome its shortage.

Moreover, we have implemented input validation in our program as a good practice to avoid user input invalid values which may cause the system to crash. We are fully aware that the program is irreversible once users entered the input in real life program which results in unwanted financial loss or other bad impacts.

Overall, we have developed a more concrete understanding on different types of scheduler programs through this assignment. We have also published our program onto an online IDE.

4.0 References

[1] "Program for FCFS CPU Scheduling | Set 1," *GeeksforGeeks*, Feb. 22, 2017. https://www.geeksforgeeks.org/program-for-fcfs-cpu-scheduling-set-1/ (accessed Dec. 25, 2022).

[2] D. Team, "Scheduling Algorithms in Operating System," *DataFlair*, Aug. 10, 2021. https://data-flair.training/blogs/scheduling-algorithms-in-operating-system/ (accessed Dec. 25, 2022).

[3] "Process Scheduling." https://people.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html (accessed Dec. 25, 2022).