Course Title: CSC564 Operating Systems

Team: Abdul-Kamil Fusheini

Project Title: Discrete Event Simulation of First-Come-First-Served (FCFS), Shortest-Job-Next (SJN), and Round-Robin (RR) Scheduling Algorithms using known dataset.

Date: December 7, 2023.

1. Project Overview

The project aims to perform discrete event simulation and measure the performances of three fundamental process scheduling algorithms: first-come-first-served (FCFS), shortest-job-next (SJN), and round-robin (RR), using a known process dataset. The approach would be to design and implement a simulation kernel and a simulation application for the scenario. This report comprehensively presents the findings and is accompanied by a video presentation.

2. Project Objectives

i. Design and implement an event-driven process scheduling simulator to simulate FCFS, SJN, and RR scheduling algorithms.
ii. Use the given test cases to evaluate the performance of these algorithms in terms of Average Turnaround Time, Average Process Wait Time, Average Queue Length, and Maximum Queue Length.
iii. Measure the effects of varying the time quantum and context switching time on the given set of data in the case of RR.
iv. Create a well-structured project report.
v. Produce a video presentation to share the project's key findings.

3. Project Scope

The scope of this project is defined by the project objectives and the following limitations and assumptions. The findings and observations must therefore be viewed within this scope.

i. There is a single processor in the system.
ii. Processes are independent, single-threaded, non-periodic, and have no deadlines.
iii. Processes involve only compute-bound phases and do not block on input/output operations.
iv. The CPU scheduler selects, dispatches, and executes processes. In a real system, the CPU scheduler only selects the next process to run from the ready queue. The dispatcher dispatches processes and the CPU executes the processes.
v. There is no preemption of the running process in the case of SJN when a process with a shorter burst arrives in the ready queue.
vi. The programming language chosen is C++.

4. Introduction

CPU scheduling otherwise known as short-term scheduling is a crucial aspect of modern operating systems and refers to the process by which the operating system selects the next process to execute on the CPU. Effective CPU scheduling is essential for optimizing system performance, ensuring fairness, and providing a responsive user experience in multitasking environments. Different scheduling algorithms have different strengths and weaknesses, and the choice of algorithm depends on the specific requirements and characteristics of the system. There are several fundamental algorithms that are designed to achieve this selection in computer systems, and they exhibit different characteristics even with the same set of processes.

The aim of this project was to implement a simulator and perform a discrete event-driven performance analysis of the FCFS, SJN, and RR scheduling algorithms using a known dataset. In this report, the results of the simulation in terms of key performance indicators such as average turnaround time, average process wait time, average queue length, and maximum queue length are presented and analyzed. In the case of RR, variations of the time quantum and context switch time were introduced leading to twenty-four (24) iterations of the RR algorithm. Overall, the three (3) scheduling algorithms were simulated in twenty-six (26) iterations and a detailed analysis of the results in terms of the performance metrics is presented in the following sections.

5. Background

First-come-first-served (FCFS), shortest-job-next (SJN) and round-robin (RR) scheduling algorithms have distinct characteristics and trade-offs in terms of efficiency, fairness, and overhead. The choice of which algorithm to use depends on specific system requirements, workload characteristics, and desired performance metrics.

5.1. Process

A process is a program in execution. It consists of the program code, data, and the execution context (registers, program counter, etc.). In a multitasking environment, multiple processes become eligible to run on the CPU and are put in a container or data structure called a ready queue from where they compete for CPU time. This creates the need for a scheduler that selects processes and determines the order in which the processes are dispatched to run on the CPU.

The arrival time of a process refers to the first time it becomes eligible to run and is in the ready queue; the burst time refers to the total time a process executes until it blocks or completes if it is not interrupted; the wait time is the time from its arrival until it is first scheduled to run on the CPU; the completion time is the time the process finishes execution and leaves the ready queue (and CPU); and the turnaround time is the time difference between the arrival time and the process completion time. The process of saving the context of one process from the CPU to the main memory and loading the context of the next process to run from the main memory to the CPU is called a context switch and the time taken is the context switch time. When a process is scheduled, it may be specified by the scheduler to run for a specific time called the time quantum at which time it is interrupted by the scheduler if it does not voluntarily release the CPU.

5.2. Ready Queue

A queue is a container or data structure used to keep data objects in a computer system and generally operates on a first-in-first-out basis in one direction. In particular, the ready queue is a special queue integral to the CPU scheduling mechanism of the operating system and keeps the processes that are eligible to run on the CPU. Thus, the CPU scheduler orders and selects processes from this container. There are several variations in the practical implementation of the ready queue depending on the desired operational principle. In this project, a data structure in C++ called deque was used to implement the ready queues. A deque is a double-ended queue that allows efficient insertion and deletion at both ends, unlike a queue. It is implemented as a sequence of blocks, providing fast random access to elements and efficient insertion/removal at both ends. It also allows dynamic resizing giving it an advantage over arrays. The deque container type allowed us to iterate and sort the ready queue objects to implement the SJN algorithm (and the other two algorithms). This was not possible with a queue container type and dynamic resizing was not also possible with an array type.

5.3. Other Queues

In an operating system, there are many queues - other than the ready queue - that are used for different purposes. Some of them are the job list and device wait queues. In this simulation, two other queues were used, the *jobList* and the *processes* queues both of which were implemented with a vector container type. The C++ vector container combines the benefits of a dynamically resizable array with a convenient set of member functions but does not operate on a first-in-first-out basis like queues or deques. To load the processes from the input file and send them to an invoked scheduler, I instantiated and used two containers of type std::vector which gives similar high-performance benefits as deque with additional flexibility to insert objects at any index.

5.4. First Come First Serve (FCFS)

FCFS is one of the simplest scheduling algorithms. It operates based on the principle of serving processes in the order they arrive in the ready queue. The first process to arrive is the first to be served by the CPU. Theoretically, FCFS is easy to implement, and is expected to yield longer average wait times, especially if processes with longer burst times arrive earlier. It suffers from the "convoy effect," where shorter processes get delayed due to longer processes executing first.

5.5. Shortest Job Next (SJN)

SJN selects the process with the shortest burst time for execution and the non-preemptive SJN is considered in the scope of this work. It aims to minimize wait times and turnaround times for processes with shorter burst times and therefore reduces the queue length. However, it requires knowledge of the burst times of all processes in advance, which might not be feasible in practical scenarios. While SJN can reduce wait times for smaller jobs, longer processes might suffer from starvation. In a skewed dataset where the processes arrive in ascending order of burst times, the SJN gives a similar performance as the FCFS scheduler. In the dataset that was used, this was not the case and SJN improved some of the measured indicators over FCFS. The total simulation duration for a dataset would always be the same for SJN and FCFS and this is supported by the results of this simulation.

5.6. Round Robin (RR):

RR is a preemptive scheduling algorithm that allocates CPU time to processes in fixed time slices known as time quanta. Each process receives a small unit of CPU time (time quantum) and, if not completed, is moved to the end of the ready queue. This algorithm ensures fairness among processes and prevents starvation. The key difference between RR and FCFS is that RR reduces the time quantum from infinity to a smaller value to give shorter wait times to processes. However, with a smaller time quantum, RR might incur higher overhead due to frequent context switches leading to a longer duration for the simulation.

6. Code Functionality

In this project, C++ programming language was used as it is renowned for its object-oriented paradigm and robust manipulation of objects. C++ has extensive libraries and allows for the encapsulation of data as well as methods within objects, fostering clean, modular, and reusable code. The functionality of my code is detailed below.

6.1. Libraries Used

i. ***iostream***: Input-output stream handling.
ii. ***fstream***: Input-output file handling.
iii. ***vecto***r: Dynamic array-like data structure.
iv. ***deque***: Double-ended queue data structure.
v. ***algorithm***: Provides various algorithms to operate on sequences like sorting.

6.2. Process Struct

I modeled a process using the C++ struct "Process" which is a class in C++. A "Process" object has attributes such as an integer process ID that uniquely identifies a process, a boolean status variable that indicates if the process has been scheduled before and float variables that capture the arrival time, burst time, and wait time of a process. It also includes two constructors to allow for two different ways of object instantiation. The struct “Process” is passed as the class type to the vector and deque class templates to create the *"readyQueue", "processes", and "jobList*" queues.

6.3. Global Variables

i. ***processes***: a vector container to store the processes to be used in the simulation.
ii. ***jobList***: a vector container used to store new jobs when they arrive (created).
iii. ***readyQueue***: A queue container to hold processes that are ready for CPU scheduling.
iv. Various counter variables to keep track of time, queue lengths, and scheduling statistics.

6.4. Utility Functions

i. ***createProcesses():*** Reads process information from an input file ("input_processes.txt"), creates "Process" objects and populates the "processes" vector.
ii. ***resetStats():*** Resets various statistics and time variables used during scheduling.
iii. ***printStats():*** Prints scheduling statistics to an output stream.
iv. ***arrivalEvent():*** Enqueues processes from "jobList" to the "readyQueue" based on arrival times.
v. "***operator<()"*** function overloading: Defines comparison for the "Process" struct to facilitate sorting.

6.5. Scheduling Algorithms

i. ***fcfsScheduler():*** Implements the First Come First Serve (FCFS) scheduling algorithm.
ii. ***sjnScheduler():*** Implements the Shortest Job Next (SJN) scheduling algorithm.
iii. ***rrScheduler():*** Implements the round-robin (RR) scheduling algorithm.

6.6. *main()* Function

i. The main() function initializes the time quanta and context switch times.
ii. It then calls the "***createProcesses()"*** function to read process information from the input file.
iii. Runs the RR scheduler for various combinations of time quantum and context switch times.
iv. Outputs statistics for each combination of RR scheduler parameters and reset counters using utility function before the next iteration.
v. Runs FCFS scheduler, outputs statistics, and resets counters using utility functions.
vi. Runs SJN scheduler, outputs statistics, and resets counters using utility functions.
vii. Closes the output file.

7. Overall Flow

i. Processes are read from the input file ("input_processes.txt") and stored in the "processes" vector.
ii. Scheduling algorithms are run, and the queue statistics including total simulation time, average turnaround time, average wait time, average queue length, and maximum queue length are calculated. The queue statistics are saved in a file named "output.txt" created in the directory where the script invoked.

8. Results and Analysis

Scheduling Algorithm	**Time Quantum (ms)**	**Context Switch Time (ms)**	**Total Simulation Time (s)**	**Average Turnaround Time (s)**	**Average Process Wait Time (s)**	**Average Queue Length**	**Maximum Queue Length**
First-Come-First-Served			795.57	61.2136	28.7841	1.45455	2
Shortest-Job-Next			795.57	**57.5282**	25.0986	1.31818	2
Round Robin	50	0	795.451	80.7338	0.0696314	2.48967	5
Round Robin	50	5	857.994	127.923	0.139664	3.59716	7
Round Robin	50	10	921.145	173.229	0.181487	4.47049	8
Round Robin	50	15	984.307	216.008	0.251506	5.14971	9
Round Robin	50	20	1048.18	259.992	0.309961	5.75758	10
Round Robin	50	25	1111.93	**304.157**	0.363675	6.28807	11
Round Robin	100	0	795.525	80.7579	0.155372	2.48908	5
Round Robin	100	5	826.6	104.023	0.21517	3.06033	6
Round Robin	100	10	858.073	127.936	0.271253	3.59658	7
Round Robin	100	15	889.67	151.292	0.312901	4.07111	7
Round Robin	100	20	921.193	173.179	0.362889	4.4685	8
Round Robin	100	25	952.722	194.56	0.440442	4.82167	8
Round Robin	250	0	795.57	80.7263	0.362711	2.48691	4
Round Robin	250	5	807.897	89.9681	0.428477	2.71972	5
Round Robin	250	10	820.409	99.3222	0.447495	2.94799	5
Round Robin	250	15	833.011	108.759	0.554826	3.16894	6
Round Robin	250	20	845.624	118.31	0.627354	3.38499	6

Round Robin	250	25	858.242	127.92	0.679635	3.59476	7
Round Robin	500	0	795.57	**80.7263**	0.714984	2.48714	4
Round Robin	500	5	801.736	85.2707	0.812221	2.6025	5
Round Robin	500	10	807.959	89.9814	0.853339	2.72064	5
Round Robin	500	15	814.23	94.6816	0.916889	2.83739	5
Round Robin	500	20	820.516	99.3569	0.967058	2.94788	5
Round Robin	500	25	826.855	103.976	1.11046	3.05698	6

Table 1: Simulation Results

8.1. Total Simulation Time (s)

Total simulation time refers to the total time it takes to start and complete the execution of all processes in a set. In terms of total simulation time, the three algorithms yielded the same results of 795.57 seconds when the context switch time is constant (zero). The same trend is expected when the context switch time is varied. This evidence supports the fact that the CPU scheduler (, dispatcher and CPU) should not shorten or lengthen the execution times of processes when it schedules them. All processes should be able to start and run to completion according to their program code and data. It was noted that slight floating point precision errors were introduced for small time quanta of 50 and 100 milliseconds.

When the context switch time was varied in the case of RR, the total simulation time was proportional to the context switch time. Again, the same trend is expected to hold when the context switch time is varied for FCFS or SJN. This means that context switch time is an overhead and lengthens the time it takes to complete the same set of processes thereby reducing the throughput as reflected in the increasing average turnaround time in the RR iterations for increasing context switch time.

8.2. Average Turnaround Time (s)

Turnaround time refers to the time difference between a process's arrival time and its completion time. The average turnaround time of a CPU scheduler therefore reveals its throughput performance. In the simulation, the SJN algorithm gave the best average turnaround time of **57.5282** seconds showing an improvement of 6% over FCFS and a minimum of 28% over RR iteration of the same context switch time of zero. The SJN achieves this gain in throughput performance by giving priority to jobs with shorter bursts and by executing each scheduled job to completion which ensures that context switching is done a maximum of 1 time to run a process. For these reasons, SJN is expected to yield better average turnaround times when context switch time is varied for all three algorithms. The FCFS and RR both use the FIFO priority schemes but the FCFS improved average turnaround time by at least 24% over RR even for the same context switch time of zero. This is because the FCFS executes each scheduled job to completion giving smaller turnaround times to earlier scheduled jobs compared to the case of RR for the same jobs.

Further, the average turnaround time is proportional to context switch time and inversely proportional to time quantum indicating that RR strategy is worst for average turnaround time and throughput. For example, for a time quantum of 50 instead of 500 milliseconds and a context switch time of 25 instead of 0 milliseconds, the average turnaround time was degraded from 80.7263 seconds to 304.157 seconds. This degradation is over 275%.

8.3. Average Process Wait Time (s)

The wait time is the time from a process's arrival until it is first scheduled to run on the CPU. The response time of a user program therefore cannot be smaller than the process wait time of its process. Average process wait time reveals how a CPU scheduler performs in terms of fairness and response time of processes. The simulation shows that the average process wait time is proportional to context switch time and proportional to time quantum in the range that time quantum is not greater than the maximum process burst time. By decreasing the time quantum from tens of seconds (in FCFS and SJN) to tens of milliseconds (in RR), the average process wait time was reduced from tens of seconds to tenths of seconds showing that RR with the smallest time quantum and smallest context is the best strategy for average process wait time and fastest response times of processes. FCFS represents the worst RR for any process dataset. SJN improves average process wait time over FCFS and worst RR by prioritizing processes with shorter burst times.

8.4. Queue Length

The queue length counter was the number of processes in the ready queue recorded each time just before the CPU scheduler selects a process. The average and the maximum of these numbers in the same simulation iteration yielded the average queue length and the maximum queue length respectively. The average indicates how long processes stay in the ready queue. The maximum queue length indicates the size of the ready queue data structure required to implement each scheduling strategy. Extreme growth of the ready queue can lead to more overhead in swapping processes in and out of main memory. The average queue length can be used along with the average turnaround time to assess the throughput performance of a CPU scheduler. The average queue length and the maximum queue length were observed to be proportional to the context switch time and inversely proportional to the time quantum.

The results showed that SJN is the best scheduler for the queue statistics considered in this simulation. FCFS is on par in terms of maximum queue length. RR was the worst scheduler overall.

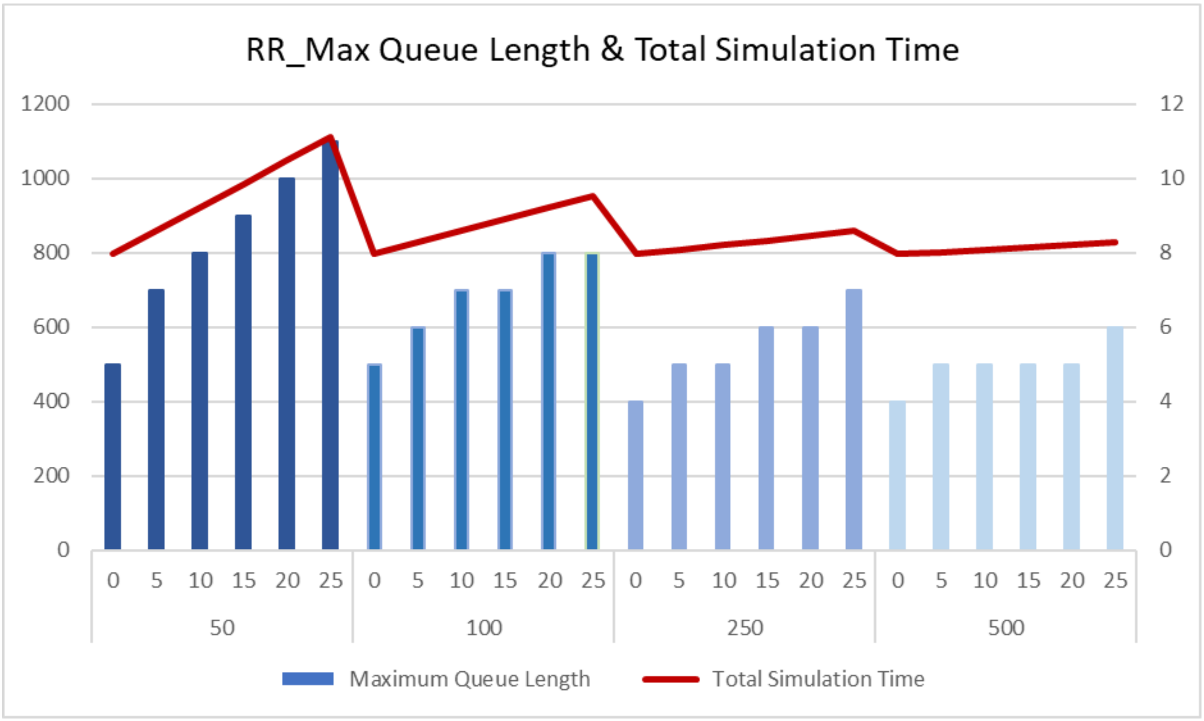

Figure 1: Comparative analysis of RR based on queue length and simulation time

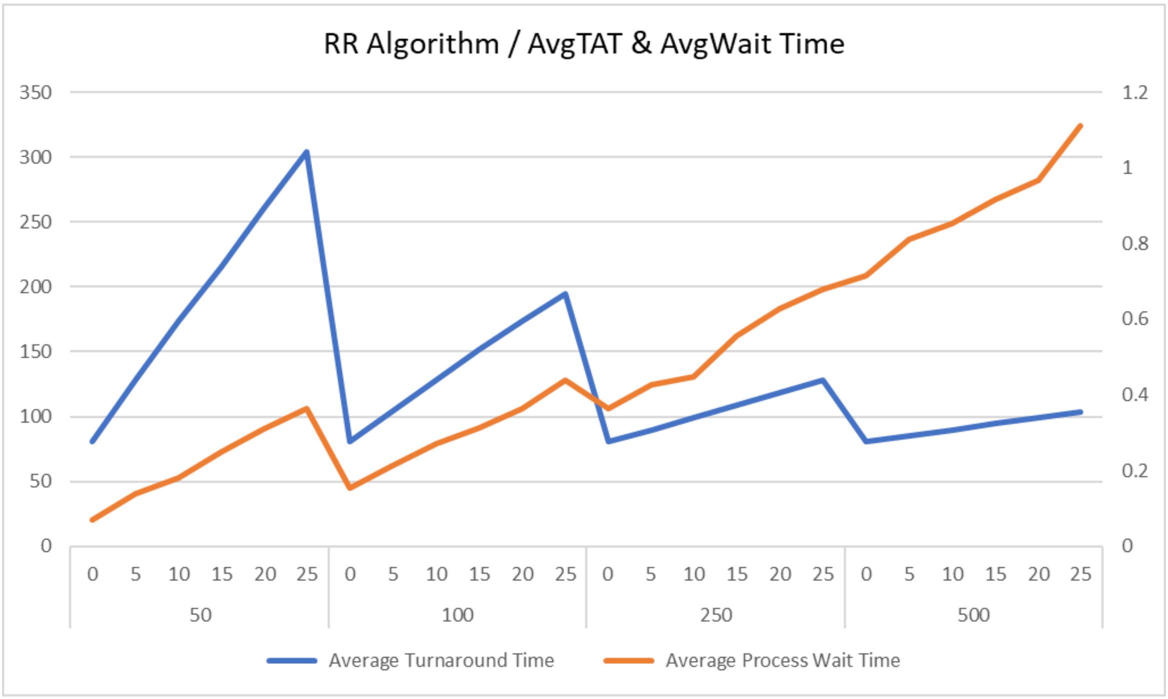

Figure 2: RR algorithm analysis based on Turnaround Time and Wait Time

Scheduling Algorithms	**Average Turnaround Time**	**Average Process Wait Time**
Round Robin	3172.79	11.55
First-Come-First-Served	61.21	28.78
Shortest-Job-Next	57.53	25.10

Table 2: Table comparing average wait times and turnaround times for RR, SJN, and FCFS scheduling algorithms

Comparison:

FCFS and SJN tend to have lower maximum queue lengths compared to RR in the scenarios outlined.
RR's maximum queue length and total simulation time tend to vary significantly based on the time quantum and context switch time, demonstrating how different settings impact the queuing behavior and waiting times of processes.
FCFS and SJN algorithms demonstrate relatively consistent total simulation times across various scenarios, as they execute processes in a more deterministic order without preemption or time slicing.
The total simulation time for FCFS and SJN remains relatively stable compared to RR, as they execute processes based on their arrival order or burst times without frequent context switching.

Conclusion:

Through simulation and analysis, valuable insights into the strengths, weaknesses, and practical implications of the different scheduling algorithms (FCFS, SJN, and RR) were realized.
The simulation results show that the SJN algorithm is more optimal than FCFS and RR. RR performs better than SJN on average wait time which is ideal for interactive processes. However, the requirement of prior knowledge of process burst times to implement SJN poses challenges in real-world scenarios where complete information might not be available upfront and the algorithm may be impractical.
On the other hand, the FCFS produces a balanced performance and is easy to implement in real systems where complete information on process burst times is unavailable.

In conclusion, this project and report serve as a foundation for understanding and comparing different CPU scheduling algorithms, emphasizing their significance in optimizing resource utilization and improving system performance.