




School of Computing Technologies
COSC1114 Operating Systems Principles

Assignment 2

	Assessment Type: Individual assignment; no group work. Submit online via Canvas → Assignments → Assignment 2 . Clarifications/updates may be made via announcements and relevant discussion forums.
	Due Date: Week 10, Monday 25 th September 2023, 11:59pm.
	Weighting: 100 marks that contribute 30% of the total assessment.

1. Overview

Understanding scheduling is an important skill to have in terms of understanding how operating system decides which processes use the CPU and how long they get to use them.

Your task in this assignment is to develop a **scheduling simulator** in C++.

We have provided a random data creator (`~e70949/shared/osp2023/creator`) to generate input data for your program. More details can be found at section “About the Generated Data” in page 5.

Given the CPU burst time (in milliseconds) per process, you are going to build a **scheduling simulator** in C++ for the following scheduling algorithms on a single CPU system.

- First-In First-Out (FIFO)
- Shortest Job First (SJF)
- Round Robin (RR)

At the completion of the program, your **scheduling simulator** will display the following stats:

- Average turnaround time – this is the time from when a process is enqueued (first added to the ready queue) until the process completes.
- Average waiting time – how long on average each process spends waiting in the ready queue.
- Average response time – how long on average each process spends waiting in the ready queue until the first response is produced.

To help you with this we have provided a PCB (process control block) class that contains information about an individual process and its progress. The following is a recommended design for your program:

- A launcher/driver .cpp that parses command line arguments.
- A simulator class that runs the simulation.
- A loader class to load data from the input file.
- A PCB class. You can modify it as needed.

Starter code for this assignment will be available from this URL:

<https://github.com/muaddib1971/osp2023-startup>

Once you clone/pull the repository, the code can be found in the a2 directory. Please feel free to create additional classes if needed.

2. Learning outcomes

This assessment task supports CLOs 1, 3, 4, 5 & 6.

3. Assessment details

This assessment will determine your ability to

1. Understand the concepts taught over the weeks 7 and 8 materials and some of the earlier materials.
2. Work independently in self-directed study to research the identified issues.

4. Submission

Your assignment should follow the requirement below and submit via [Canvas](#) → [Assignments](#) → [Assignment 2](#). You may resubmit the assignment if you need to, only the most recent version will be marked. Failure to follow the requirements incurs up to 10 marks penalty for this assessment.

You will submit two things:

1. Your C++ source code in [a zip file](#).

If your student ID is s1234567, then please create a zip file named s1234567_OS_A2.zip including the code you have developed and a README file, in which please specify in detail how to run your code on the provided servers.

2. Your report in [pdf format](#).

Use at least 12-point font size. If your student ID is s1234567, then please create a pdf file named s1234567_OS_A2.pdf.

It is your responsibility to correctly submit your files. Please verify that your submission is correctly submitted by downloading what you have submitted to see if your submitted file includes the correct content.

Never leave submission to the last minute – you may have difficulty uploading files.

You can submit multiple times – a new submission will override any earlier submissions. **However, if your final submission is after the due time, late penalties will apply.**

5. Academic integrity and plagiarism

The penalties for submitting code that is not your own can be severe. Do not simply copy other people's work, it is not difficult for us to detect copied work and we will pursue such cases.

6. Late submission policy

A penalty of 10% per day of the total available marks will apply for each day being late. **After 10 days, you will receive zero marks for the assignment.**

If you want to seek an extension of time for assignment submission, you must have a substantial reason for that, such as unexpected circumstances. Reasons such as, unable to cope with study load, is not substantial. Also, you must apply for an extension as soon as possible. Last minute extensions cannot be granted unless it attracts special consideration.

Please find out how to apply for special consideration online at
<https://www.rmit.edu.au/students/student-essentials/assessment-and-results/special-consideration/eligibility-and-how-to-apply>.

Any student wishing an extension must go through the official procedure for applying for extensions and must apply at least a week before the due date. Do not wait till the submission due date to apply for an extension.

7. Rubric and marking guidelines

The rubric can also be found in **Canvas → Assignments → Assignment 2.**

Requirement	Little to no Attempt (0%)	Poor (25%)	OK (50%)	Good (75%)	Excellent (100%)
1. Arguments and output including makefile. (15 marks)	No attempt.	A reasonable attempt but it does not compile.	Code compiles but the results/implementation are partially correct/incomplete.	Mostly complete implementation but there are minor problems.	Excellent job. Well done.
2. FIFO (15 marks)	No attempt.	Some attempt but did not get the code working or does not compile.	Code compiles but the results/implementation are partially correct/incomplete.	Mostly complete implementation but there are minor problems.	Excellent job. Well done.
3. SJF (20 marks)	No attempt.	Some attempt but did not get the code working or does not compile.	Code compiles but the results/implementation are partially correct/incomplete.	Mostly complete implementation but there are minor problems.	Excellent job. Well done.
4. RR (30 marks)	No attempt.	Some attempt but did not get the code working or does not compile.	Code compiles but the results/implementation are partially correct/incomplete.	Mostly complete implementation but there are minor problems.	Excellent job. Well done.
5. Report (20 marks)	No attempt.	The report is incomplete.	The report is difficult to follow in places. The results are partially correct.	The report is reasonable easy to read. The results are correct and well-described. The findings can be better interpreted.	Excellent job. Well done.

8. Assignment tasks

About the Generated Data

To generate a file called `datafile` with 10 records, you run the creator as follows:

```
~e70949/shared/osp2023/creator 10 datafile
```

```
[e52483@csitprdap01 ~]$ ~e70949/shared/osp2023/creator 10 datafile
seeding with random seed of 440101969778399
[e52483@csitprdap01 ~]$ cat datafile
1,72
2,28
3,110
4,70
5,58
6,53
7,129
8,116
9,64
10,75
```

Note: if you want to recreate the same `datafile` later, just note down the seed number output by the program and provide that as a third argument to the program later.

Each line of the `datafile` consists of two fields separated by comma:

`ProcessID,BurstTime`

where `ProcessID` is a unique ID for each process, and `BurstTime` is the running time (CPU burst) for a process (in milliseconds).

Please assume that the arrival time is 0 for all processes.

To simplify the scenario, each process has only one CPU burst and no IO operations.

Code Quality

You are expected to compile and run your code on `titan`, `jupiter` or `saturn` servers provided by the school. Please note that while code quality is not specifically marked for, it is part of what is assessed as this course teaches operating systems principles partly through the software you develop. As such lack of comments, lack of error checking, good variable naming, avoidance of magic numbers, etc., may be taken into consideration by your marker in assigning marks for the components of this assignment, although operating systems principles do take priority.

Task 1 – Arguments and Output (15 marks)

Makefile (5 marks)

Create a `makefile` to compile your programs such that if we unzip your code, we just type `'make'` in the root directory to compile your program.

`"make all"` will build 3 programs (`fifo`, `sjf`, and `rr`) at once.

If you need a refresher on these concepts, please consult the tutorialspoint tutorial available here: <https://www.tutorialspoint.com/makefile/index.htm>

Arguments (5 marks)

Your programs are expected to parse command line arguments according to the following format:

- First-In First-Out (FIFO): `./fifo datafile`
- Shortest Job First (SJF): `./sjf datafile`
- Round Robin (RR): `./rr quantum datafile`

Where the `datafile` is generated by the provided creator and the `quantum` is the amount of time (in milliseconds) that is given to each process (the time slice) in turn with the RR algorithm.

Output (5 marks)

Your programs are expected to produce the following output on the screen:

1. For each process: Process ID, burst time, turnaround time, waiting time, and response time.
2. For all processes: Average turnaround time, average waiting time, and average response time.

Task 2 – The Simulator (65 marks)

Start off copying the process control blocks (PCBs) into appropriate data structure for a ready queue which represents the processes waiting to be processed. A vector or a deque of PCBs would be a good choice.

The main loop of our simulator will be as follows:

1. ask the current scheduling algorithm for which process to schedule next;
2. load the corresponding PCB from the ready queue;
3. in the case of round robin, if a process has used up its CPU quantum, add it back to the ready queue;
4. continue this until all processes are completed.

First In First Out Scheduling (15 marks)

Write the functions that select which process to run next and load the corresponding PCB.

With this scheme, the process that requests the CPU first is allocated the CPU first. We finish the first process before moving onto the next and so on. As all processes arrive at time 0, you may follow the sequence given by the generated `datafile` for execution.

Shortest Job First Scheduling (20 marks)

Write the functions that select which process to run next and load the corresponding PCB.

This algorithm associates with each process the length of the process's CPU burst time. When the CPU is available, it is assigned to the process that has the smallest CPU burst time. If the CPU burst times of two processes are the same, FIFO scheduling is used to break the tie.

In this case, insert PCBs to the ready queue in sorted order by CPU burst time, with the shortest process as the first and the longest at the last. From there, it is the same algorithm as FIFO.

Round Robin Scheduling (30 marks)

Write the functions that select what process to run next, and load the corresponding PCB and add it back to the ready queue when the process has used up its quantum.

The round-robin algorithm is such that each process takes its turn using up its quantum until it finishes. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.

A time quantum is generally from 10 to 100 milliseconds in length. You may choose different quantum sizes within the range for your experiments.

Task 3 – Report (20 marks)

Run the provided creator and generate 10 different lots of 50 processes.

```
~e70949/shared/osp2023/creator 50 datafile
```

Collect the average turnaround time, the average waiting time, and the average response time from these runs of each algorithm and copy into a table.

Answer the following questions based on your experimental results:

1. Discuss the performance of different scheduling algorithms. For example, does a particular algorithm perform the best or the worst on a particular performance metric?
2. Discuss how quantum sizes are related to scheduling performance?
3. How could we modify one of these algorithms to favor interactive processes?

Please include sensible experimental results to support your discussion. Please ensure your report (no more than 3 pages) is neat and professionally presented.