Carleton University
Department of Systems and Computer Engineering
SYSC 4001 Operating Systems Fall 2023

Assignment 1

THIS ASSIGNMENT MUST BE DONE BY TEAMS OF TWO STUDENTS. Only ONE student must submit the assignment, clearly stating who the members of the team are. The programming part of the Assignment should be done using the Pair Programming technique. ONE OF THE TWO STUDENTS SUBMITS THE ZIP FILE WITH THE ASSIGNMENT; A TEXT FILE WITH THE NAMES/STUDENT NUMBERS OF THE TEAM. THE SECOND STUDENT ONLY SUBMITS THE TEXT FILE.

Please submit the assignment using the electronic submission on Brightspace. The submission process will be closed at the deadline. No assignments will be accepted via email.

Part I – Concepts [45 marks]. Answer the following questions (from Chapter 1, 2 Silberschatz + lectures, marks in brackets)

a) [8 marks] Batch Operating Systems used special cards to automate processing and to identify the jobs to be done. A new job started by using a special card that contained a command like:

$JOB

and ended using another special card that contained the command

$END

After the $JOB card, there are different command cards to request specific services. For instance, the $RUN card would indicate to start executing the program running starts.

i. [4 marks] Explain why these special cards are needed, what is their function, and how do they interact with the Operating System.
ii. [4 marks] Explain what would happen if, in the middle of the program execution, we detect the card $END. What should the Operating System do in that case?

b) [6 marks] Explain how kernel and user modes are used to protect against access by other users.

c) [6 marks] Write examples of three privileged instructions, and three non-privileged instructions.

d) [4 marks] Some early computers protected the OS by placing it in ROM. Define the advantages and difficulties of such scheme.

e) [6 marks] What is the purpose of interrupts? How do they work?

f) [6 marks] What are the differences between a trap and an interrupt? How do traps work?

g) [2 marks] Explain briefly how a Time-Sharing Operating System works. How does it manage to handle multiple interactive users with a single CPU?

h) [4] (Kernel structure) Let us suppose we have a process in a multitasking OS.

i. What are the events needed to make a new process to move from the “New” to the “Ready” state?
ii. How about events to move the process from “Running” to “Terminated”?

Explain how the OS Kernel will react to these different events (in terms of the states of the system and the transitions between these states, and the routines of the kernel involved in the process).

Hint: explain at least *two* kinds of events to be considered, which will produce different state changes and their corresponding transitions. Explain how the OS Kernel will react to this event in detail. Include the OS components involved in these transitions.

k) [3] DMA is used for high-speed I/O devices to reduce overhead:
i. How does the CPU interface with the device to coordinate the transfer?
ii. How does the CPU know when the DMA operations are complete?
iii. The CPU can execute other programs while DMA is transferring data. Does this process interfere with the execution of the user programs? If so, explain how.

Part II – [55 marks] Design and Implementation of a Kernel Simulator

The objective of this assignment is to build a small simulator of an OS kernel, which could be used for performance analysis of different scheduling algorithms. This simulator will also be used in Assignment 2.

i) Input data

The simulator will receive, as an input, a list of processes to run with their trace information, as follows:

Pid	Arrival Time	Total CPU Time	I/O Frequency	I/O Duration

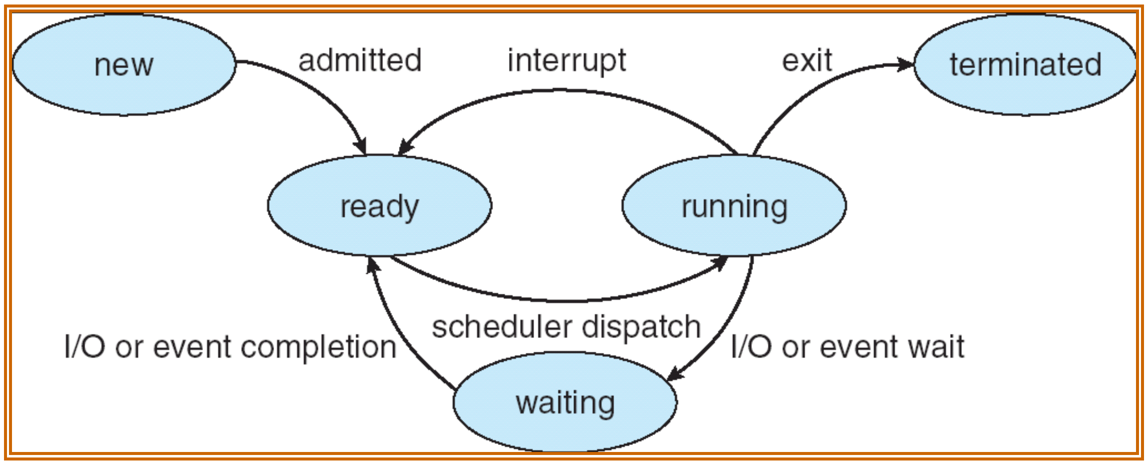
- Pid: a unique identifier for the process
- Arrival Time: the initial simulated time is 0 and the arrival time can be at 0 or any time value thereafter. The time units to be used are milliseconds.
- Total CPU Time: it is the total time the process needs to complete (does not include I/O time: only the total time needed in the CPU)
- I/O Frequency: the processes are assumed to periodically make a call to an I/O device or event wait with this frequency
- I/O duration: this is the duration the process must wait before the I/O or event completion (assumed to be the same for all the I/O operations)

The information will be stored in a file (to be submitted).

ii) Simulation implementation

Using the information on the input file, you must define a data structure in memory (array of structs, linked list) like a PCB (only include the information needed: PID, CPU, I/O information, and remaining CPU time – needed to represent preemption –).

The simulation should try to reproduce the behavior of this state diagram (from Silberschatz *et al.*):



Note: The New and Terminated states are optional (you can simply load the process information in the Ready queue when they start.

ASSUME THAT THE TRANSITIONS TAKE ZERO TIME (that is, do not worry for the time taken by the ISRs, scheduler, context switch, *etc.*). Every time the simulation changes states, the following information should be included in an output file.

ASSUME THAT YOU ALWAYS HAVE AN I/O DEVICE AVAILABLE (that is, do not worry about waiting queues at the I/O devices: whenever you request an I/O, the I/O starts immediately).

Time of transition	Pid	Old State	New State

- Time: The simulation time starts at 0, and it is measured in milliseconds.
- Pid: the id for the process that has changed state
- Old State/New State: The state of the process before and after the transition

The scheduler should get the first element of the list; do not worry about scheduling.

We will submit test cases in BrightSpace; the TAs will mark these cases first. You can include your own test cases too.

The program must be written in C/C++. You must submit an executable, the source code, all files needed to compile, test scenarios, and scripts to run them. Include, at least, two scripts named "test1" and "test2" (with extension ".bat" in Windows or .sh in Linux) that will be used to run 2 different tests automatically. The TAs will use these two to run basic tests, and then will modify your input files,

Marking Scheme:

TOTAL: 100 marks (corresponds to 10% of the total for the course)

For Part II:

Correctness (including error checking): 65%
Documentation and output: 25%
Program structure: 5%
Style and readability: 5%