Carleton University
Department of Systems and Computer Engineering
SYSC 4001 Operating Systems Fall 2014

Assignment 2 – Process Scheduling, Memory Management

POSTED: Tuesday, October 11th 2016.

DUE DATE: Friday, November 4th (9 PM). No late assignments are accepted.

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.

Part I – Concepts [35 marks]

Answer the following questions. **Justify your answers. Show all your work.**

a) [7.5 marks] (Kernel structure) Let us suppose we have a process in a multiprogrammed OS. The process is currently running (i.e., the CPU has been allocated to that process, and the process is in the *running* state). The CPU scheduler is using a Round Robin with Priorities algorithm (in which high priority user processes are given the CPU first, and everybody has the same quantum).

What are the possible events that can make that process **abandon** the use of the CPU? 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: there are at least *three* kinds of events to be considered, which will produce three completely different state changes and their corresponding transitions. Explain how the OS Kernel will react to this event in detail. **Consider the kind of scheduler** this system is using.

b) [2.5 marks] (Threads) What are the differences between user-level threads and kernel-level threads? Under what circumstances is one type better than the other?

c) [7.5 marks] Consider the following set of processes. Each process has a single CPU burst and does not perform any I/O.

Process	Arrival Time (sec)	Execution Time (sec)
P1	0	22
P2	9	11
P3	12	12
P4	13	11
P5	17	14

With the help of Gantt charts, compute the mean turnaround time for the following scheduling algorithms:
(i) FCFS
(ii) Round Robin (time slice of 3 sec).
(iii) Multiple queues with feedback (high priority queue: quantum = 1; mid-priority queue: quantum = 2; low priority queue: quantum = 4)

d) [5 marks] Assume that each process requests to do an I/O every 1 sec., and the duration of each of these I/O is 1 sec. Repeat part c).

e) [2.5 marks] Explain, in general, the differences in the degree to which the following scheduling algorithms discriminate in favour of long processes:

(a) FCFS
(b) RR
(c) Multilevel feedback queues

f) Memory management [10 marks]

Consider a multiprogrammed system that uses multiple partitions (of variable size) for memory management. A linked list of holes called the *free list* is maintained by the operating system to keep track of the available memory in the system. At a given point in time the free list consists of holes with sizes:

102K, 205K, 43K, 180K, 70K, 125K, 91K, and 150K

The free list is also ordered in the sequence given above: the first hole in the list is of size 102K words, which is followed by a hole of size 205K words and so on. There are a number of Jobs arriving to the system with different memory requirements; they arrive in the following order:

Job No.	Arrival Time	Memory Requirement (words)
1	t1	122K
2	t2	105K
3	t3	203K
4	t4	90K

[Given t1 < t2 < t3 < t4]

Determine which free partition will be allocated to each process for the following algorithms:
(a) First Fit (b) Best Fit (c) Worst Fit

Show all you work.

Part II – [45 marks] Design and Implementation of a Scheduler Simulator

The objective of this assignment is to build a small scheduler simulator, which could be used for performance analysis of different scheduling algorithms.

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 afterwards. The time units to be used are milliseconds.
- I/O Frequency: the processes are assumed to make an input/output with this frequency
- I/O duration: this is the duration for the I/O for each of the processes (assumed to be the same for all the I/O operations)

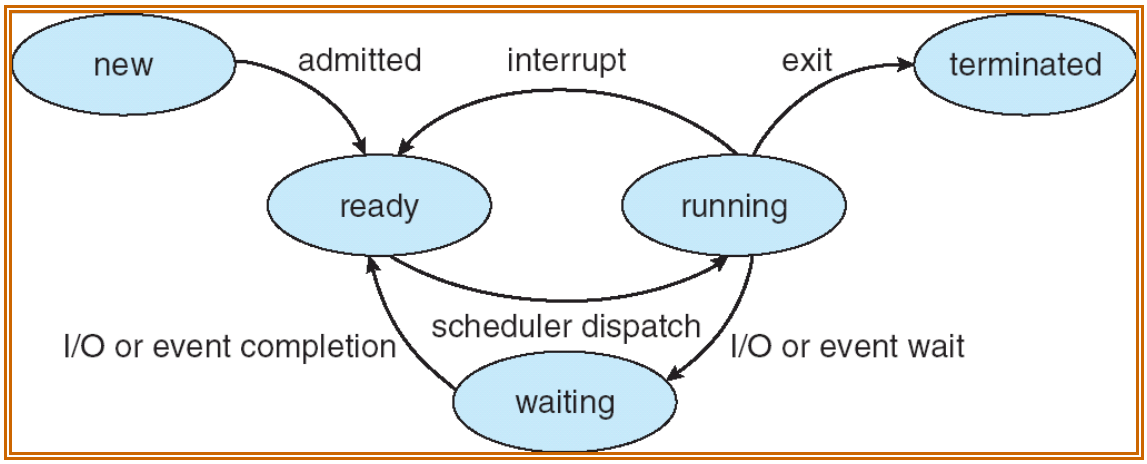
(See Part I.c) for an example).

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

ii) Simulation implementation **[30 MARKS]**

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

Your first implementation should include a FCFS scheduler only (without preemption), and should try to run the case presented in Part I. c). Your second test case should be the one presented in Part I. d). Submit your input and output files for these two scenarios.

Your second implementation should include the external Priorities (no preemption) scheduler. Modify your PCB as needed.

iii) Simulation Execution **[15 MARKS]**

Run different simulation scenarios using the different schedulers you defined. Analyze the results obtained, and write a short report (3 pages maximum – can be longer if you find interesting results and you want to elaborate) discussing the results of the simulation execution.

You can compute different metrics based on your simulation results: Throughput, Average Wait Time, Average Turnaround time, Average Response Time (i.e., the time between 2 I/O operations). Use the metrics to compare how the algorithms perform with mostly I/O bound, mostly CPU-bound processes, or processes with similar I/O and CPU bursts.

You must run at least 10 different simulation scenarios, collect simulation results, and analyze, at least:

a. **Throughput**
b. **Average turnaround time**
c. **Wait time (i.e., the time spent in the ready queue)**

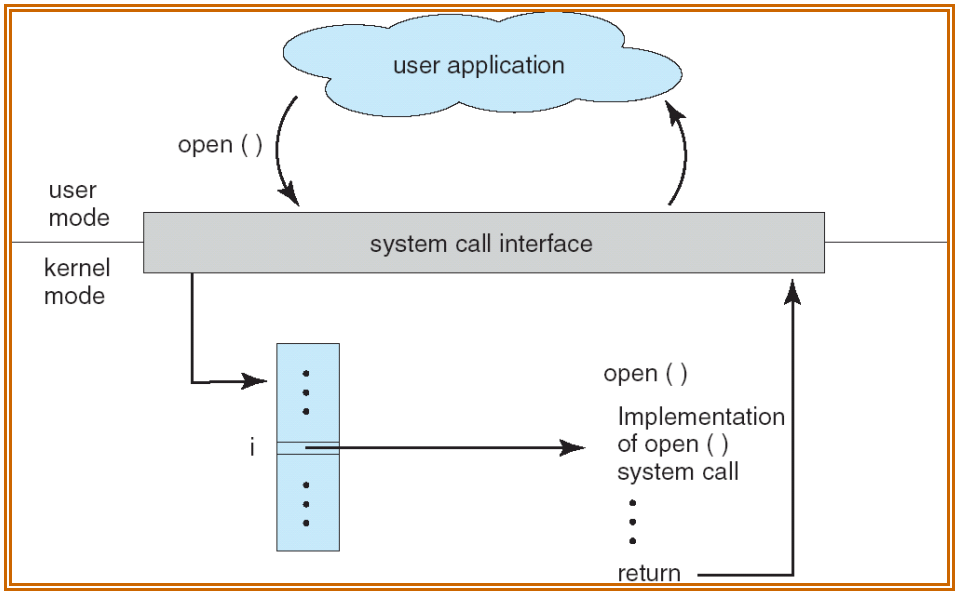
Discuss the results you obtained and compare the two algorithms based on your simulations.

The program must be written in 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.bat" and "test2.bat" that will be used to run 2 different tests automatically. The TAs will use these two basic tests, and then will modify your input files,

Part III – Initial approach to the Term Project Take Home Exam (20 bonus marks)

Defining a System Call Interface, a basic API using simulation software.

The objective of this part of the assignment is to simulate the behavior of a program running and the system call interface, based on the following figure of the textbook.



To do so, we will use a simulation tool with predefined functions.

You should use the following link to download and install the software (you will need an installation of UBUNTU Linux to run the simulation software):

http://www.sce.carleton.ca/courses/sysc-5104/lib/exe/fetch.php?media=cdboost_example_installation_templates.zip

Then, you should do the following:

Part One: reproducing what already is there

1. Install the software
2. Read the documentation and source code of the Alternating Bit Protocol (ABP) example
3. Recompile the code for that example.
4. Re-run the existing simulations and confirm that you can reproduce the original

Part Two: modifying the APB protocol and simulating the figure above.

Now that you understand how the simulator works and were able to understand the APB model, use it as a guideline to model the figure above.

You need a few components:

. A simulated "program". This is a program based on the Sender model in the APB that will:

- "Execute" (random wait in the internal transition function)
- Will issue a system call Read (in the output function)
- Will wait for an answer (in the external transition function)

. A simulated “system call interface” based on the Network model in the ABP. It will:

- Receive 2 possible requests: Read or Write (in the external transition function)
- Will transfer it to the lower layer (using the output function)
- Will passivate waiting a response from the lower layer (another input in the external transition)
- Will pass the results to the upper layer (using the output function)
- Internal states will be changed by the internal transition. Timing will be programmed in the time advance function.

. Two simulated "system calls” components (based on the Receiver model in the ABP). It will

- Be activated (external transition)
- Simulate the activity (“read” or “write”)
- Return (output function)

Test them individually and integrate them one at a time.

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.bat” and “test2.bat” that will be used to run 2 different tests automatically. The TAs will use these two basic tests, and then will modify your input files.

Discuss the results you obtained based on your simulations. Write a report and attach it to the submission.

Marking Scheme:

TOTAL: 80 marks (corresponds to 10% of the total for the course) + 20 bonus

For Part B:

You will receive a mark of Zero (0) if your code does not compile cleanly.
Correctness (including error checking): 40%
Documentation and output: 25%
Program structure: 5%
Style and readability: 5%
Simulation execution: 10%
Results analysis: 15%