Homework 1

Due: April 11, 2022 **Points:** 100

Remember, you must *justify all your answers*.

Short-answer

1. (*5 points*) True or False: Time-sharing was not widespread on second-generation computers because the software was not mature enough to implement it.

   Answer: False. Time-sharing is a form of multiprogramming, and requires special hardware to provide protection for jobs when they are in memory. Second generation computers did not have this necessary hardware; when it became available, the third generation began.

2. (*5 points*) Fill in the blank: Assuming all jobs arrive at the same time, the non-preemptive job scheduling algorithm _____ is optimal.

   Answer: Shortest job first or shortest job next.

3. (*5 points*) Multiple choice: Which of the following is most true about a multi-level feedback queue?

   (a) It is intended for use when the running time of each job is known.
   (b) It adjusts the priority of a process automatically, based on the amount of time it has run so far.
   (c) It is a pre-emptive version of the shortest job next scheduling algorithm.
   (d) It is the only real-time scheduling algorithm that enables jobs to meet absolute deadlines.

   Answer: (b). The MLFQ algorithm adjusts priority based on how long each job has run, so it handles processes whose run time is unknown. It is not a pre-emptive shortest job next algorithm, as it does not always select the job with the least amount of remaining run time. There are many real-time scheduling algorithms, and the time when a job must be completed by does not figure into which queue a process is in. Finally, the longer a process has run, the lower its priority and the lower the queue it is placed in.

4. (*5 points*) Which of the following system calls does a UNIX/Linux shell use to run commands?

   (a) open(), read(), close()
   (b) open(), fork(), close()
   (c) fork(), exec(), wait()
   (d) fork(), exec(), exit()

   Answer: (c). The shell spawns a subprocess (fork()) and overlays the subprocess with the executable involved (exec()). The parent process waits for the child to complete (wait()).

Long Answer Questions

5. (*21 points*) Protecting the resident monitor is crucial to a correctly operating computer system. Providing this protection is the reason behind multiple mode operation, memory protection, and the timer. To allow maximum flexibility, however, we would also like to place minimal constraints upon the user. The following is a list of operations which are normally protected. What is the *minimal* set of instructions which must be protected?

   (a) Change to user mode.
   (b) Change to kernel mode.
   (c) Read from kernel memory.
   (d) Write into kernel memory.
   (e) Instruction fetch from kernel memory.
   (f) Turn on timer interrupt.
   (g) Turn off timer interrupt.

   Answer:

(a) Change to user mode need not bd protected, assuming this instruction does not involve loading the processor state from an instruction specified location.
(b) Change to monitor mode must be protected. If it were unprotected, a user process could change the processor state at will and gain access to monitor memory.
(c) Whether read from monitor memory needs to be protected depends on whether security is a concern. If it is not, this need not be protected. But otherwise, the program could read any confidential information in memory, such as passwords controlling the booting of the system.
(d) Write into monitor memory must be protected. If not, processes could freely alter any part of the system.
(e) Instruction fetch from monitor memory need not be protected, because instructions do not contain confidential data, and if they are executed the system will be running in user mode. Thus, execution will not affect other processes adversely.
(f) Turn on timer interrupt need not be protected unless by doing so the current value of the timer is altered. In that case, the user process shouldn't be able to set the value of the time or it could gain an unfair amount of resources by constantly resetting the time.
(g) Turn off timer interrupt must be protected. Otherwise the process could simply keep the CPU and never relinquish it to any other process, including the kernel.

6. (*39 points*) Assume you have been given the following jobs with the indicated arrival and service times:

job	arrival time	service time
A	0	5
B	1	2
C	3	7
D	5	1
E	7	4

When and in what order would these jobs run if the scheduling algorithm were a multi-level feedback queue with 2 levels, both round robin, and quanta 1 on the first level and 3 on the second? Assume that if events are scheduled to happen at the same time, new arrivals precede terminations, which precede quantum expirations.

Answer:

Here, the number in parentheses is the number of time units left for the process to execute.

	at beginning of time interval			
time	process running	level 1	level 2	comments
0–1	A(4)	*empty*	*empty*	A runs for the first quantum
1–2	B(1)	*empty*	A(4)	B arrives at time 1
2–3	A(3)	*empty*	B(1)	A's quantum is 3
3–4	A(2)	C(7)	B(1)	C arrives at time 3; it does not interrupt A's quantum
4–5	A(1)	*empty*	B(1)	A's quantum expires
5–6	C(6)	D(1)	B(1) A(1)	D arrives at time 5
6–7	D(0)	E(4)	B(1) A(1) C(6)	D completes; E arrives at time 7
7–8	E(3)	*empty*	B(1) A(1) C(6)	
8–9	B(0)	*empty*	A(1) C(6) E(3)	B completes
9–10	A(0)	*empty*	C(6) E(3)	A's quantum is 3; A completes
10–11	C(5)	*empty*	E(3)	C's quantum is 3
11–12	C(4)	*empty*	E(3)	
12–13	C(3)	*empty*	E(3)	C's quantum expires
13–14	E(2)	*empty*	C(4)	E's quantum is 3
14–15	E(1)	*empty*	C(4)	
15–16	E(0)	*empty*	C(4)	E's quantum expires; E completes
16–17	C(2)	*empty*	E(2)	C's quantum is 3
17–18	C(1)	*empty*	E(2)	
18–19	C(0)	*empty*	E(2)	C completes

Summarizing:

- A runs from 0 to 1;
- B runs from 1 to 2;
- A runs from 2 to 5;
- C runs from 5 to 6;
- D runs from 6 to 7 and terminates;
- E runs from 7 to 8;
- B runs from 8 to 9 and terminates;
- A runs from 9 to 10 and terminates;
- C runs from 10 to 13;
- E runs from 13 to 16 and terminates; and
- C runs from 16 to 19 and terminates.

7. (*20 points*) On an interactive system, multiple users may run the same program at the same time.

   (a) Under what conditions can they share the same program instructions in memory?
   (b) Can they share the data in memory? If not, can they share some part of that data?

   Answer:

   (a) As long as the code does not alter any part of itself, the code can be shared. Self-modifying code is very rare these days because code is usually loaded into read-only and/or execute-only memory.
   (b) Data that is not to be changed can be shared; this is done by placing the data into read-only memory. Any data that the code can change cannot be shared.