ECS150 - Operating Systems
Homework #1

**Answers at the bottom

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

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.

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.

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?

Short-Answers

1. False. Time-sharing was not widespread on second-generation computers because the *hardware*(not the software) was not mature enough to implement it.
2. Assuming all jobs arrive at the same time, the non-preemptive job scheduling algorithm Shortest Job First(SJF) is optimal.
3. b. It adjusts the priority of a process automatically, based on the amount of time it has run so far.
4. c. fork(), exec(), wait() . It will need to fork to create a child process so that the parent process does not get overwritten, the child will run exec to run the new program and the parent waits for the child to finish executing.

Long-Answers

5. Below will list whether the instructions are protected or not:
    a. Change to User Mode
        i. *Not Protected*. User is already in user mode so it would have little to no effect if they tried to change to user mode.
    b. Change to Kernel Mode
        i. *Protected*. We do not want to give the user the powers of the kernels. It should do an interrupt/trap to have the Kernel contact the Operating System.
    c. Read from Kernel Memory.
        i. *Protected*. Reading from Kernel Memory can cause a security issue.
    d. Write Into Kernel Memory.
        i. *Protected*. This can be both a security and memory protection issue. Users can write negligent or malicious code into memory.
    e. Instruction Fetch from Kernel Memory
        i. *Not Protected*. Even if a process has the ability to fetch the instruction, it is not decoding or executing the instruction. It would have little effect on the system or other processes.
    f. Turn on Timer Interrupt
        i. *Not Protected*. Timer interrupt will simply start the maximum amount of quantum allowed for the process.
    g. Turn off Timer Interrupt.
        i. *Protected*. Users can have their own process run infinitely by constantly turning off their own timers.

6. Using the information given about the MLFQ, the below will describe each job's functionality: Let T_n be defined as T = quantum time and the subscript n denotes the amount of time passed.
    a. Job Start Times: $A = T_0$, $B = T_1$, $C = T_5$, $D = T_6$, $E: T_7$
    b. Job Terminates/Completes: $D = T_7$, $B = T_9$, $A = T_{10}$, $E = T_{16}$, $C = T_{19}$

	A	B	C	D	E	F	G	H	I	J	K
1	Time (Tn)	0	1	2	3	4	5	6	7	8	9
2	Event	A arrives in Q1	B arrives in Q1		C arrives in Q1		D arrives in Q1		E arrives in Q1		
3	Process Running		A:4	B:1	A:3	A:2	A:1	C:6	D:0	E:3	B:0
4	Q1	A:5	B:2		C:7	C:7	C:7, D:1	D:1	E:4		
5	Q2			A:4	B:1	B:1	B:1	B:1, A:1	B:1, A:1, C:6	B:1, A:1, C:6	A:1, C:6, E:3
6											
7	Time	10	11	12	13	14	15	16	17	18	19
8	Event										
9	Process Running	A:0	C:5	C:4	C:3	E:2	E:1	E:0	C:2	C:1	C:0
10	Q1										
11	Q2	C:6, E:3	E:3	E:3	E:3	C:3	C:3	C:3			
12											
13											

Job	Arrival Time	Service Time
A	0	5
B	1	2
C	3	7
D	5	1
E	7	4

Start Time = Green Time - 1
End Time = Time
Job Letter : Service Time Left

    c. Using the graph above:
        - At T_0: Job A arrives and is placed into Q1. Job A begins to run
        - At T_1: Job B arrives and is placed into Q1. Job A has already used max quantum in Q1, so it is placed into Q2 and Job B begins running.
        - At T_2: Job B uses the max quantum for the queue, so Job B is placed at the back of Q2, Job A runs again
        - At T_3: Job C arrives, Job A continues running
        - At T_4: Job A continues running
        - At T_5: Job D arrives, Job A has used max quantum in Q2, Job A is placed at the back of Q2, Job C begins its run
        - At T_6: Job C has used max quantum in Q1. Job C is placed at the back of Q2. Job D begins to run
        - At T_7: Job E arrives in Q1, Job D has used max quantum but is also complete so it terminates. Job E begins to run.
        - At T_8: Job E has used max quantum in Q1. Job E is placed at the back of Q2. Job B is chosen to run.

- At T_9: Job B is completed and terminates. Job A is at the front of Q2, Job A starts to run again.
- At T_{10}: Job A is completed and terminates. Job C is at the front of Q2, Job C starts to run again.
- At T_{11} : Job C continues to run
- At T_{12}: Job C Continues to run
- At T_{13}: Job C has used max quantum in Q2. Job C gets moved to the back of Q2. Job E starts to run again.
- At T_{14}: Job E continues to run
- At T_{15}: Job E continues to run
- At T_{16}: Job E completes and terminates. Job C is set to run.
- At T_{17}: Job C continues to run
- At T_{18}: Job C continues to run
- At T_{19}: Job C completes and terminates

7.

a. Program instructions can be shared under the condition that they are supported by protection bits and that they are reentrant functions. This means that the instructions cannot hold any static or global non-constant data without synchronization with the operating system. Also the instructions may not be allowed to modify itself without synchronization. Last, it may not call on non-reentrant instructions/programs/routines. Protection bits can aid in this by adding a few bits per segment. It will indicate whether or not a program can read or write a segment, or possibly even execute a segment. This provides protection for the code to be shared.
b. Data can be shared under the circumstance that they provide mutual exclusion, bounded wait and progress. For this to be achievable, the Bernstein Conditions must be met such that:
$R(S_i) \cap W(S_j) = \emptyset \; and \; R(S_j) \cap W(S_i) = \emptyset \; and \; W(S_i) \cap W(S_j) = \emptyset$

   In addition, the processes/users that wish to interact with the data must also hold the proper permissions to do so.