On my honor, I have not given, nor received, nor witnessed any unauthorized assistance on this work.

Print name and sign: ______________________________

Question:	1	2	3	4	5	6	Total
Points:	5	4	8	6	3	4	30
Score:							

General Note: the problem set contains plenty of implementation problems with mathematical answers that ensure your understanding of the scheduling algorithms. While you should expect some of those type of questions on the quiz, you should also expect some conceptual questions. This quiz gives some examples of conceptual questions.

1. Shortest-time-to-Completion-First (STCF) is actually a pretty good scheduling algorithm.
   (a) (3 points) State how this algorithm works.

   Solution: STCF chooses the job with the least amount of time left to run and runs it. Any time a new job arrives, it preempts the currently running job and runs the job with the least time remaining. It's good for average turnaround time, but poor for response time.

   (b) (2 points) If STCF is such a good algorithm, why do we hardly ever see it implemented and in use in operating systems?

   Solution: In general, we don't know (or can't reliably predict) running time. Lacking this crucial piece of information it's virtually impossible to implement this algorithm in a real-world scenario.

2. (4 points) In a MLFQ, there is a rule which states: "After some time period S, move all the jobs in the system to the topmost queue.". What is the purpose (necessity?) of this rule?

   Solution: This rule prevents starvation. By periodically putting all jobs back on the same footing (same priority), the scheduler will run in a round-robin fashion so that all jobs receive some running time.

3. (8 points) Most hardware provides user mode (for applications) and kernel mode (for the OS). The tricky part is transitions from one to the other. We accomplish this transition using a system call. What happens during the system call?

Solution:

Jump into kernel (via trap instruction)
Raises privilege to kernel mode
Saves caller's registers

Run kernel code (as designated by the trap table)

Return from trap
Reduces privilege
Restores caller's registers

Generally, there are three major time blocks: before, during, and after the kernel code executes. When scoring these, I wouldn't be too picky about exact order within those blocks.

4. (6 points) On Unix-like operating systems, we use the `fork()`, `exec()` (and variants of `exec()` like `execvp()`) and `wait()` to manage processes. Explain (briefly!) what each of these calls do.

Solution: `fork()` is used to create a new process

`exec()` can be used to instruct the child process to run something different (some other program) instead of just a copy of the parent.

`wait()` makes sure the parent waits for the child to finish before

5. (3 points) What is the fundamental purpose of the timer interrupt?

Solution: Allows OS to retain control of the HW. By periodically firing the interrupt, it can pre-empt any long-running process which might monopolize the HW and starve other jobs.

6. (4 points) How do you calculate both response time and turnaround time for a particular scheduling algorithm? What are the tradeoffs between the two metrics?

Solution:

$T_{turnaround} = T_{completion} - T_{arrival}$: How long is the process around until it completes

$T_{response} = T_{firstrun} - T_{arrival}$: How long does a job wait before it receives any running time from the CPU

See the last paragraph on page 8 of the *CPU Scheduling* OSTEP chapter for a succint explanation of the tradeoffs.