CSci 5103 (Fall 2014)
Assignment 2
(100 points)
Due September 30, 2014
This part of the assignment must be done individually.

Question 1 (15 points): A computer system has 1 GB of memory. The kernel takes 200 MB of the memory and the remaining memory is allocated to jobs (processes). Assume that each job requires 200 MB of memory. This system can have 4 jobs (processes) in memory, i.e. the degree of multiprogramming it can support is 4. If each job (process) spends 90% of time waiting for I/O, what is the CPU utilization? Now suppose that we add additional 1GB of memory, thus enabling degree of multiprogramming of 9. What will be the CPU utilization for this new system configuration?

Question 2 (10 points): Why a separate stack in the kernel memory space is used for handling system call functions and interrupt handlers for a process, instead of using the process stack?

Question 3 (10 points): Select the correct answer and briefly justify your answer. When a process is executing in the user-mode, what is the state of its kernel stack?
(a) The kernel stack for the process is always empty.
(b) The kernel stack for the process is always non-empty.
(c) It is not possible to assert either (a) or (b) will always hold.

Question 4 (10 points):
(A) When a UNIX process executes fork, does the child process inherit any pending signals of the parent?
(B) When a UNIX process executes fork, does the child process inherit the signal handlers of the parent process?
(C) On fork, does the child process inherit the signal mask of the parent?
(D) If a multithreaded process forks, a problem occurs if the child gets copies of all of the parent’s threads. Suppose that one of the original threads was waiting for keyboard input, now two threads are waiting for keyboard input, one in each process. Can such a problem ever arise in single-threaded processes?

Question 5 (10 points): Read from Chapter 10 Section 10.3.3 Implementation of Processes and Threads in Linux. Briefly (in less than 150 words) summarize the important issues that arise with fork by a multithreaded process.

Question 6 (25 points): Consider a system that has five jobs, with respective processing time requirements of 5, 4, 3, 2, and 1 units. Now consider a system that processes these jobs using the processor sharing discipline (e.g., the RR time-quantum is extremely small).
(i) What is the average turnaround time for the jobs in this system?
(ii) What is the average waiting time for the jobs?
(iii) What is the throughput of this system?
(iv) What is the minimum value for the turnaround time for a job in this system?
(v) What is the maximum value for the turnaround time for a job in this system?

Now consider the same system but with the scheduling discipline changed to FCFS. Answer the above six questions for this system for the worst case and the best case condition in terms of the average turnaround time for the jobs. (For this you will need to identify the order of these five jobs in the FCFS queue.)

Question 7 (10 points): Measurements of a certain system have shown that the average process runs for a time T before blocking for I/O. A process switch requires a time of S, which is effectively wasted (overhead). For round-robin scheduling with quantum Q, give a formula for CPU efficiency for each of the following cases:
a) $Q = \infty$
b) $Q > T$
c) $S < Q < T$
d) $Q = S$
e) $Q$ nearly 0

**Question 8 (10 points)** Consider a real-time system with three periodic tasks. Task A has period of 100 units and requires 20 units of processing time. Task B has period 50 units and requires 20 units of processing time. Task C has period of 75 units and requires 15 units of processing time. Suppose that these tasks are all arrive at time 0, and they are scheduled with static priority using the Rate Monotonic Scheduling (RMS) model.

(A) Is the RMS condition for guaranteeing the existence of a feasible schedule satisfied by this system?

(B) Does a feasible schedule exist when using RM based scheduling of the tasks in this system? If yes, then show a feasible schedule, otherwise show a case where a task will miss its deadline.