CPSC 313 Introduction to Computer Systems
Homework 1 (July 19, 2016)
Student: Carsten Hood
UIN: 922009787

1. [1, 3] Which of the following instructions should be privileged?

(a) Set time-of-day clock.	YES
(b) Read time-of-day clock.	NO
(c) Clear memory.	NO (if for a given process, depends on the memory)
(d) Disable interrupts.	YES
(e) Change the memory map.	YES

2. [1] What is the main advantage of multiprogramming?
Efficiency – it organizes jobs so that the CPU is kept busy executing and is hence more productive.

3. [3] When a user program makes a system call to read or write a disk file, it provides indication of which file it wants, a pointer to the data buffer, and the count. Control is then transferred to the OS, which calls the appropriate driver. Suppose that the driver starts the disk and then terminates until an interrupt occurs. In the case of reading from the disk, obviously the caller will have to be blocked (because there is no data for it). What about the case of writing to the disk? Need the caller be blocking awaiting completion of the disk transfer?
Yes, because data could be overwritten; but blocking may be avoided by first loading the update to be written into a buffer.

4. [2] Why are the locations of interrupt handles generally not stored in a linked list?
Speed – traversing a linked list could quickly lead to congestion in interrupt processing; arrays are much faster.

5. [1] What are two differences between user-level threads and kernel-level threads? Under what circumstances is one type better than the other?

a. differences:
   1) Kernel-level threads are managed by the kernel; user-level threads are unknown to the kernel and are scheduled by a library.
   2) Kernel-level threads require more kernel resources and are more costly to create and transition between; it's easier to switch between user-level threads.
b. Kernel-level threads are preferable in multiprocessor systems, since they can be scheduled across multiple processors. Kernel-level threads are also generally better for high-priority system tasks like processing keyboard input, and when a process needs to bypass a system call block. User-level threads are better for lower-priority processes that perform interrelated CPU operations; they're also preferable for tasks that transition between threads frequently.

6. [1] Describe the difference of degree to which the following scheduling algorithms discriminate in favor of short processes:
(a) First-Come-First-Serve.
Least favorably of all – short processes are given no priority and must wait until prior processes have finished.

(b) Round-Robin.

Somewhat favorably – as processes are given equal CPU shots on a rotation, short processes don't have to wait for earlier long processes and will tend to be completed faster than in FCFS.

(c) Multilevel feedback queues.

Favorable – short processes will tend to be filtered into higher priority queues and hence will tend to be handled faster than in the other algorithms, which do not distinguish priorities between tasks.

7. [1] Have a look at the following program.

```
#include <stdio.h>
#include <unistd.h>
int main() {
    for(int i = 0; i < 4; i++)
        fork();
      return 0;
  }
```

How many processes are created in total?

14 processes created, or 15 counting the original process

8. [1] When a program creates a new process using fork(), which of the following states is shared between the parent process and the child process?

(a) Stack NOT SHARED

(b) Heap YES – SHARED

9. (string inversion program)

Compile with: `$   g++ -std=c++11 string_invert.cpp -o string_invert`

Run with: `$   ./string_invert my_string`

10. (digit summation program)

Compile with: `$   g++ -std=c++11 sum_of_digits.cpp -o sum_of_digits`

Run with: `$   ./sum_of_digits 12345`