Written Assignment #1

Due: Check My Courses

To be completed in groups of 2-3 students. Submit a PDF file containing the answers with the names and student IDs of all the group members.

1. Explain the key differences between the operations performed in system call processing and interrupt servicing. What are the similarities between the two?
2. Instructions related to accessing I/O devices are typically privileged instructions, that is, they can be executed in kernel mode but not in user mode. Give a reason why these instructions are privileged.
3. Which of the following instructions should be allowed only in kernel mode?
    a. Disable all interrupts
    b. Read the time-of-day clock
    c. Set the time-of-day clock
    d. Change the memory map
4. What is a trap instruction? Explain its use in operating systems.
5. Multi-programming enables an operating system to run multiple programs concurrently. What are the two important functions provided by memory management that are essential for multi-programming?
6. Provide reasons why micro-kernel OS would be more reliable than a monolithic OS. Provide reasons why monolithic OS would be more reliable than a micro-kernel OS.
7. Consider a web server. Each incoming request is processed as follows: input-side processing (done on all incoming request) – 100ms, disk access – 900 ms, cache access – 100ms. The input side processing determines whether a request can be served by the cache or disk access is required. Assume that the web server has a single disk that serves the requests one after the other. The cache will hold popular requests after warming up. If the web server uses a single thread (kernel level), what are the best and worst request rates achievable by the server? Suppose we use two threads in the web server, what is the best request rate we could achieve?
8. Consider the following C code fragment. How many processes run the **`run_morecompute()`** function?

```
for (i = 0; i < 4; i++) {
      pid = fork();
      if (pid == 0)
            run_compute(i);
}

void run_compute(int i) {
      int cpid = fork();
      if (cpid == 0)
            run_morecompute();
}
```

9. Consider the following code fragment.

```
close(1);
fd = open("temp2.txt", …);
if (fork() == 0)
```

```
        printf("Message from A\n");
printf("Message from B\n");
```

What will be the contents of the **temp2.txt** file after executing this fragment? Let's say you don't want "**Message from A**" in your **temp2.txt** file. What modifications would you do to the above code fragment? What modifications could you do to the code fragment to ensure that the output of **"Message from A"** shows up on the standard output? Note: no need to create a working program. Your modification should be valid in UNIX/Linux.

10. An application took 800s to run in a single core machine. It took 290s on a four-core machine. What is the speedup you got when running the application on the four-core machine? What the expected runtimes in machines with 2, 8, 16 cores? What portion of the application is actually parallel?
11. Operating systems have a **tee** command that allows a copy of the redirected pipe contents to be logged into a file. That is, you can get a copy of the output passed by a process to the other process logged in a file. Briefly explain how you would implement (give a high level pseudo code) the tee command and how it would work with the pipe redirection.
12. When would an **exec()** system call or its variation that you might have used in the shell program fail? When would a **fork()** system call fail?