All questions must be done by yourself without outside assistance. Use the turnin script, located at:

http://systems.cs.uoregon.edu/apps/turnin.php

to submit your assignment. You may submit either a plain text or a PDF file. Don’t send Word or OpenOffice files. Make sure to answer the question, but remember that brevity is the soul of wit: be concise. It is more important that we understand your answer (more points!) than trying to decipher rambling discourse (less points!).

**Textbook Questions (30 pts)**

Answer the following questions from the *Operating Systems Concepts (OSC) 9/3* textbook. You only need a few sentences to answer each question.

**OSC 2.7:** What is the purpose of system programs?

**OSC 2.9:** List five services provided by an operating system and explain how each create convenience for users. In which cases would it be impossible for user-level programs to provide these services? Give a brief explanation. **OSC 2.19:** Why is the separation of mechanism and policy desirable?

**OSC 3.5:** When a process creates a new process using the `fork()` operation, which of the following states is shared between the parent process and the child process. (Addition: Describe what problems might arise if the opposite were true)

**OSC 3.9:** Describe the actions taken by a kernel to context-switch between processes.

**OSC 4.2:** What are two differences between user-level threads and kernel-level threads? Under what circumstances is one type better than the other?

**OSC 18.8:** In what circumstances is the system-call sequence `fork() exec()` most appropriate? When is `vfork()` preferable?

**Process Analysis (20 pts)**

Suppose you want to find out as much as you can about a running process, one where you do not have access to the source code. Design a methodology using the Linux `proc` filesystem to extract as much information as you can. You should be able to apply this to any of your running processes. For instance, an experiment may begin something like this:

```
% sleep 600 &
[1] 1363
...
```
The *sleep* process has been assigned 1363 as its *process ID (PID)*. You can do `p 1363` to find out some information. You can `cd /proc/1363` and run `ls`, where you will see a lot files and directories. Some of the files contain relevant information. Explain which piece of information each step of your methodology is designed to supply, and how it relates to the program, process, and operating system.

**Note:** Like all assignments in this class you are prohibited from copying any content from the Internet or sharing ideas, code, configuration, text or anything else or getting help from anyone in or outside of the class, except where noted. Consulting online sources is acceptable, but under no circumstances should anything be copied. Failure to abide by this requirement will result in sanctions ranging from zero on the assignment to dismissal from the class.

**Interrupts (20pts)**

Suppose that you are developing a new computer and OS with significant resource constraints. The hardware is only allowed to support either clock interrupts or I/O interrupts, but not both.

(a) Discuss how your system would support multiprogramming without clock interrupts, while preventing any process from monopolizing the CPU or going into an infinite loop. Your solution should support the ability to time share amongst processes by switching quickly between them. In this case, all processes are able to perform I/O.

(b) Do the same but for a system without I/O interrupts. In this case, you want to support multiprogramming as before, but also I/O for every process.

(c) Argue which you would choose with the goal of providing the best user performance for a low price. What makes your solution more economical than the alternative? Are there circumstance under which your OS performs well versus the other approach? Under what conditions would it respond poorly? Note that your design is for a uniprocessor system and that user processes and the OS must take turns with the one CPU.

There are many possible answers. Be sure that your solution is consistent with multiprogrammed uniprocessor systems (what we’ve talked about so far in class) and interrupts. Do not provide answers where the OS acts as a monitor watching over actions and intervening (e.g., an OS noticing a PC instruction run 1000 times and switching processes is a wrong answer). You are welcome to use diagrams for clarification if necessary. A good answer will be a page or less single-spaced.