OPERATING SYSTEMS (COMP 310 / ECSE 427) – Midterm – February 10, 2017

Name: **Student ID:**

Part I –Some multiple-choice questions can have **more than one correct** answer. Circle *all and only correc*t answers. (Answer **all 16 questions – 5 points each**)

1. 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?

   protection
   relocation

2. In UNIX, a process is totally isolated from another process. Inter-process communication is the component of the OS that provides a mechanism for a process to communicate with another process. What are the two basic approaches for realizing an inter-process communication mechanism?

   Shared memory
   message passing

3. Which of the following are true regarding I/O interrupts from a particular device controller?
   a. An I/O interrupt can occur at the completion of an I/O activity.
   b. An I/O interrupt can occur at the beginning of an I/O activity.
   c. An I/O interrupt cannot occur when a CPU is busy.
   d. An I/O interrupt can only occur when the device controller is idle.

4. What could be an advantage of batch processing over time sharing?

   little overhead on context switching

5. What could be an advantage of time sharing over batch processing?

   ability to handle interactive jobs

6. Consider an application that took 100 seconds in 4 cores and 60 seconds in 8 cores. What would be its runtime in a single core machine? Assume that the overhead of parallelization is zero.

   $100 = S + P/4 \Rightarrow 40 = P/8 \Rightarrow P = 320$
   $60 = S + P/8 \quad S = 20$

   Single core $S + P = 340$ seconds

7. When would a **`fork()`** system call fail?

When OS has reached the max processes limit.

8. State Amdhal's law in words (speedup equation is not the law).

Performance improvement gained from an enhancement is limited by the fraction of the problem to which enhancement can be applied.

9. Consider two different realizations of a web browser: (a) uses a multi-process architecture and (b) uses a multi-threaded architecture. The multi-threaded architecture uses kernel-level threads. In either case, a process or thread is dedicated for a particular activity like screen rendering or network processing. Now consider the following scenarios.
   a. Switching from Process A to Process B in the multi-processed web browser
   b. Switching from Thread X to Thread Y in the multi-threaded web browser

   What did not happen in a X to Y switch that happened in an A to B switch? That is, a process switch involves at least one important function that a thread switch does not do. Identify this function.

A to B switch involved an address space switch.

10. When would an **`exec()`** system call or its variation that you might have used in the shell program fail?

When the executable file is not present or permission, or type is invalid.

11. Which of the following is true regarding the privileged and non-privileged execution in a computer system (as indicated by the mode bit)?
   a. Memory is divided into privileged and non-privileged regions.
   b. Data structures maintained by the kernel are loaded into non-privileged memory region.
   c. Instructions are divided into privileged and non-privileged classes.
   d. CPU executing in user mode receives a privilege violation exception if it tries to access a privileged memory region.

12. What is a race condition?

In a concurrent computation, the outcome is determined by the exact order in which the computing operations took place.

13. Briefly describe the five important steps of system call processing.

(i) invoke system call
(ii) switch mode; check arguments
(iii) branch to sys call processing routine via syscall table
(iv) complete execution of routine; switch mode
(v) return from system call

14. State the four conditions that should hold for a solution proposed for implementing a critical section.

- No two processes in the CS at the same time
- No assumption regarding CPU speed or number
- No process outside the CS prevent another from getting into
- No process wait forever to get into CS.

15. We write a program that has two kernel level threads. Each thread is running a very simple function: a long running loop that increments a global shared counter. The counter is initialized to 0 before the threads are created. Suppose the loop runs for 10000 iterations, we expect the final result to be 20000. We run the program and get results that are always less than 20000. We realize that there is a race condition. Why the result is always less than 20000 and never more than 20000?

global counter update: read counter, incr. counter, update. Due to race conditions, some incr. get lost. So the final value is less than the expected value.

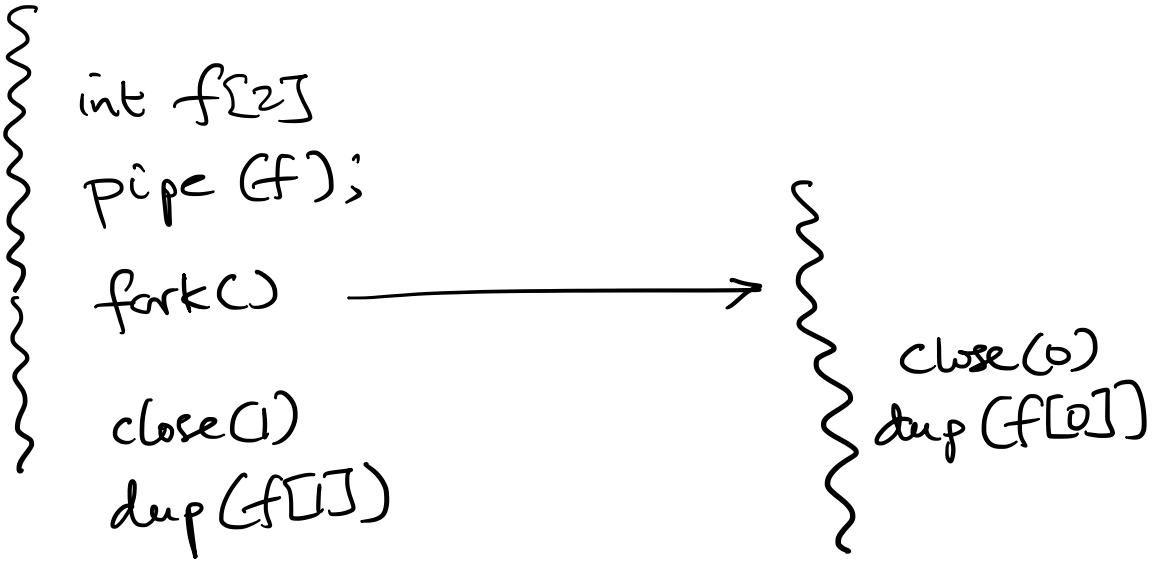
16. You write a program with multi-threading to solve a particular compute-intensive problem. You are using kernel-level threads. When you run the program on a given data set the runtime was 100 seconds and the program was using only one thread. Now, you run the same program on a data set that take double the compute time (i.e., the compute workload has doubled). The program uses two threads. Assume 20% of the program execution is a critical section. You are using a mechanism based on busy waiting to realize the critical section. What is the total runtime? Assume that there is no parallelization overhead.

If one core: $100 + 100 + 20 = 220$ seconds

If two cores: 120 seconds

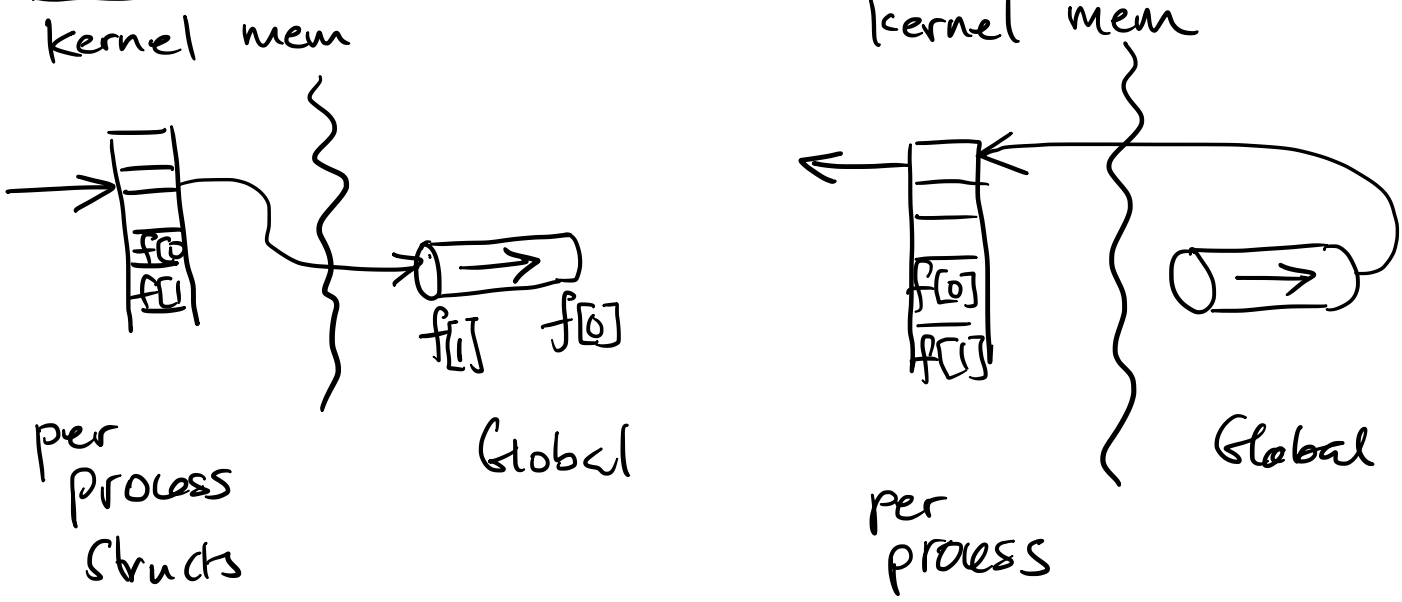
Part II – Long Form Question (**provide the shortest possible answer**)

1. **(20 points)** You want to implement piping between two processes such that the standard output of one process is the standard input of the other process. Show pseudo code (simplified) that illustrates the major operations performed in setting up the pipe redirection. Show all the kernel level data structures and their modifications as performed by the system calls used in the pseudo code. Very briefly explain how the piping is working in your pseudo code. Keep your discussion as brief as possible. Diagrams may be used for illustration.



Any write to standard out will go into the input of the pipe

The standard in will read from the pipe



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 the general idea – no need for pseudo code) the tee command and how it would work with the pipe redirection.

tee:

```
while (1) {
    read input → q
    write q to stdout
    write q to file (if specified)
}
```