Exam 2 Review

1. When might you choose to create a process rather than a thread?

2. Why might a thread be called a light-weight process?

3. Why might multi-threading your application allow you to make the application more responsive? Be sure to explain a situation where a single threaded application could become unresponsive and how multiple threads can solve this problem.

4. What resources are shared among two processes if one process is the parent and one process is the child (created via fork())?

5. What resources are shared among two threads in the same process?

6. How are a mutex and a semaphore different?

7. When might a spin lock be a good idea?

8. When might a spin lock be a very bad idea?

9. What are the benefits of using a Monitor?

10. When a person says, "Monitors have tool support unlike pthread mutexes!", what do they mean? Which tool(s) are they referring to and why does tool support matter?

11. Why should a critical section be as short as possible?

12. On Linux, if a process contains four threads and one thread requests I/O, are all four threads moved to the wait queue by the scheduler? Justify your answer.

13. Using a userspace thread library, if a process contains four threads and one thread requests I/O, are all four threads moved to the wait queue by the scheduler? Justify your answer.

14. We saw in class that merely using an atomic CPU instruction was not enough to build a synchronization solution that provided Bounded Waiting. Explain why this is the case. Be sure to indicate what more needs to be added to your synchronization solution to ensure Bounded Waiting.

15. What do we call an I/O bound process if the I/O comes from a human?

16. Why might a CPU scheduler want to identify processes that are likely to be I/O bound, especially those where the I/O comes from a human? What scheduling criteria is most important for these types of processes? How might the CPU scheduler better satisfy this criteria for these processes?

17. How might the workload of a web-server differ from that of a desktop computer?

18. The real-world speed up you see by adding threads does not often match your expectations. For instance, going from 1 to 2 threads does not always halve the runtime of the code. What might account for this lack of linear speed up?