Ryan Morris

1. The shortest average latency is achieved through “shortest job first” scheduling. Why don’t systems use this to schedule processes? **It is impossible to accurately predict how long a job will take before hand. So the issue doesn't lie with scheduling the shortest job first, but with determining what the shortest job is.**

2. What is the main advantages of exponential averages over arithmetic averages for computing scheduling statistics? **Exponential averages are easier to compute and are often more accurate then arithmetic averages.**

3. What is the typical distribution of lengths of time needed by processes that are scheduled? **Typical lengths of time for process time slicing range from 10-100 ms.**

4. What is the problem of starvation when it comes to process scheduling? **Starvation can occur when multilevel scheduling is used. If a system is busy, the higher priority que's will refill constantly, preventing tasks in the lowest priority que's from running at all.**

5. Why is it good that (typically) only administrative processes can upgrade their scheduling priority, but any process can downgrade their own priority. **Scheduling priorities are carefully researched and studied so that the tasks that are truly the most important to run get the highest priorities. If any task could arbitrarily upgrade its scheduling priority this would undo all the work on the side operating system developers and potentially make system performance much worse or create priority inversions. However, a process/task creator can downgrade its own priority as this will not hurt system performance.**

6. For multiple threads, give an example of a shared resource that might be a problem. **Two worker threads, one an event poll loop and another a network thread need to update the state of a gui running on a separate thread. Doing this simultaneously would screw up the state of the gui.**

7. Define “critical section” “**Critical section” is code that accesses shared variables, updates a system table, or performs I/O on a file or does anything to data structures that could simultaneously be used by another process.**

8. How do mutexes help with shared resources and critical sections? **Mutexes gives a function mutual exclusive access to shared resources and critical sections (or forces it to wait until it can get exclusive access). This guaranties that when a function enters its critical section no other bit of code is also using the same resources that that critical section accesses. Thereby preventing race conditions.**

9. What do you need to be careful about when using mutexes? **Mutexes create the potential for deadlocks and starvation. A deadlock is a situation where two or more processes are waiting indefinitely for an event that can only be caused by one of the waiting processes. Starvation occurs when a task never gets the resources it needs because it is always waiting for them to be freed.**

10. What is the idea of a readers-writer lock? **A** **readers-writer lock allows an infinite amount of readers to read from a data source as long as no writers present. As readers do not change the data there is no chance of them screwing anything up by simultaneously reading from the same source. But when something needs to write to a data source it must be given mutually exclusive access while it is writing and write as quickly as possible, since it is blocking all reads while it is writing. Reader writer locks require more overhead the semaphores and mutexes to implement but the gain of having multiple reads occurring at the same time is usually worth this overhead.**