CSCI 8530: Computer Science
Advanced Operating Systems
Spring 2020
Homework 3

Due on March 5, 2020

Please keep answers short. You may either type your answers or write them by hand and upload it to the Canvas. Note that a "good effort" on the homework includes that all answers are in your own words (short sentences). DO NOT SHARE YOUR ANSWERS WITH OTHER STUDENTS OR ATTEMPT TO OBTAIN ANSWERS FROM OTHERS.

Submission

1. The three characteristic process states (ready, running, and blocked) are extended somewhat in Xinu. How are these extended? Give names for the extended states and brief descriptions of them.
2. What's the primary rule Xinu uses to determine which process is running (or should be running)? Which piece of code (function) is primarily responsible for checking/enforcing this rule?
3. We've seen that Xinu has a particularly efficient implementation for queues and lists of processes. What are two things in the implementation that make the implementation efficient (both in memory utilization and in execution time)?
4. What is "interesting" (that is, not typical) of the semaphores provided by the Xinu operating system? If you're unsure of what's "interesting," then provide multiple things you think might be not typical of common implementations of semaphores in other operating systems. The best thing to do would be to compare Xinu semaphores with those found in Windows, Linux, Tempo, and other systems.
5. What happens to the processes that are blocked waiting on a semaphore in Xinu if a process deletes the semaphore (that is, calls `semdelete` on the semaphore)?
6. Returning from the execution of an interrupt handler occurs after the effect(s) of the interrupt has been dealt with. In particular, an interrupt could awaken one or more processes – that is, cause them to move from a blocked state to the ready state. (a) Illustrate one case where multiple processes may be awakened as the result of an interrupt. (b) Indicate how the interrupt handler prevents an awakened process from taking control of the processor before the interrupt handling has been completed.
7. Some piece of operating system code (an arbitrary function) decides it needs to guarantee it will not be interrupted during its execution. How should this be achieved? It is sufficient to illustrate how this is accomplished in Xinu. But be certain to indicate why things are done as they are (with respect to interrupt recognition).

8. Xinu has both low-level and high-level message passing facilities. Consider only the low-level message passing facilities for this question. For each message, there must be exactly one sender. (a) Could there be multiple processes that receive a low-level message? (b) Is sending a message synchronous or asynchronous? (c) Is receiving a message synchronous or asynchronous?
9. What happens to the storage for a low-level message when the intended recipient process is terminated?
10. Xinu creates a FIFO queue for each semaphore, and uses the queue to store processes that are waiting. That is, when a semaphore is sig- naled, the process that has been waiting the longest becomes ready. Imagine a modification in which the processes waiting for a semaphore are kept on a priority queue ordered by process priority (i.e., when a semaphore is signaled, the highest priority waiting process becomes ready). What is the chief disadvantage of a priority approach?