Simulated Operating Systems
Phase 3

Implement Code for Term.c

You need to study the code for terminal manager and complete the code for term.c, including completing theend of request processing actions as well as thread creation and synchronization. First, you insert code for performing necessary actions after a terminal request has completed processing. Second, in start_terminal, you need to create threads so that the managers run as separate threads. You also need to define and initialize semaphores. Finally, you add sem_wait and sem_post functions in the code to ensure proper synchronization between the threads.

Your synchronization solution should ensure that the accesses to termIO queue are mutual exclusive. You also need to properly handle the problem of waiting for an empty queue. Note that handling this synchronizationissues is not just to learn how to program for concurrent threads, but also to see how the concurrent control examples introduced in the lectures can be applied to practical problems. So, you are required to use integer semaphores only (with sem_wait and sem_post and sem_init only). You are not allowed to use more advanced synchronization functions offered in pthread library (of course it will be fun to know what other functions are available in thread libraries and you can study them). You can reference the bounded buffer problem and modifythe solution for this synchronization and mutual exclusion problem.

Correct Sync for EndIO List Accesses

In process.c, the functions for accessing the endIO list have been implemented, including function insert_endIO_list, which is called by swap.c, term.c, and clock.c, and function endIO_moveto_ready, which isinvoked locally by process.c. Similar to the case in swap.c and term.c, you need to ensure mutual exclusive accesses to the endIO list. But in addition, we prioritize these functions.

Since function endIO_moveto_ready may move multiple processes from endIO list to ready queue, whichmay be slower, we give it a lower priority than insert_endIO_list. In other words, if two threads are trying to perform endIO list insertion (insert_endIO_list) and endIO list removal (endIO_moveto_ready), then the insertion thread gets to go first, unless the removal thread has already started its access. If multiple threads aretrying to perform list insertion simultaneously, then the first comer gets the right to do its insertion first (no priority between insertion threads). You need to use integer semaphores to achieve this prioritized synchronization and mutual exclusion. Same as Phase 1, you are not allowed to use more advanced synchronization functions offered in pthread library, just semaphores and its sem_wait and sem_post and sem_init functions. You can reference the reader-writer problem introduced in the lectures and come up with aspecific solution for this case.