CSC332G Mid Term Exam Study Sheet. The exam will be in class on Wednesday, March 28. The exam will be closed book/notes/devices. Come prepared in your head! You will answer a total of 4 questions, one from each of the 4 groups. These are not the final questions, but are close. I may reword and clarify them.

Group 1

(A) The text and slides state that the semaphore, as a complex function in software, reintroduces the critical sector problem. In what specific operation on the semaphore is there the possibility of interleaving, i.e. two processes each partially executing the routine concurrently? Where do you need to reintroduce the mutex to address this problem?

(B) What are the 3 properties of any successful mutual exclusion algorithm? Demostrate that the algorithm presented in the slides (5.26) and the text for n-process bounded waiting MUTEX with test-and-set satisfies each one of them.

Group 2

(A) According to the definition of a monitor originated by H.A.R Hoare, only one process can be active in a monitor at one time. Compare the interpretation of this requirement in the Stallings text and in Silberschatz: which of them appears to allow that multiple processes can be admitted to the monitor, but that all but one of them is waiting on some condition; which of them suggests that when a process blocks on a condition, it is placed back on the entry queue, so there is only one process in the monitor at one time? What exception does Stallings seem to make to this rule, and how does he implement it?

(B) The readers and writers algorithm using semaphores which we presented in class and is in the text is referred to as solving the "first readers writers problem", and favors the readers. Explain how this is so. How would you modify the algorithm to address the "second readers writers problem" so that the writer could gain access to the shared data ASAP? Do both of these solutions allow starvation? What modification would you propose to the second solution, that favors the writer, to reduce the liability of starvation of readers.

Group 3

(A) What, if anything, do UNIX and Linux operating systems provide to recover from the situation in which a process "dies" while holding a mutex or semaphore (find what you

can, but if you cannot find anything specific, say so) ? What, *in specific*, does the POSIX API provide to deal with this situation?

(B) What are the features of the POSIX API pthread_mutex which address the problem of "spinlock contention", the system degradation which occurs when many processes are spinning while waiting for the same mutex lock?

Group 4

(C) Consider whether the monitor solution to the dining philosophers problem actually prevents two philosophers from grabbing a chopstick at the same time (i.e. calling test()). If you think the monitor prevents this, explain exactly how. If you do not think so, explain what you would have to add, and where, to avoid this problem. (Thanks to Daniel Obeng for suggesting this).

(D) It is stated in the text (exercises) that the Linux operating system does not allow a process holding a spinlock to attempt to acquire a semaphore. What is the source of this statement? What bad things could happen, which need to be avoided, if a process holding a spinlock could also try to access a semaphore (whether it gets in or not)? What is it that a process holding a spinlock cannot do, which implements the prohibition? Note that this is not the same as when a process is "spinning" waiting for the mutex.