**CS 374 – Operating Systems**

**Study Guide for Exam I**

Since there is a wide universe of problems that I could draw on, the following should help you with your studies for this class: the exam will be approximately two thirds essay questions, mostly drawn from the question set outlined below. The remainder will be some true/false and multiple-choice questions to check your understanding of terminology and the like.

**Essay questions:**

Please give the 5-state model of process control (ie: draw the diagram); you might also be asked to explain why an arc in the diagram exists, or where deallocation of resources occurs, to give a couple of examples.

Please give a *real-world* (i.e. not computer related) example of deadlock, starvation, or a race condition. (Multiple choice variations may give you an example of one of these and ask you what it is an example of.)

Use a semaphore to provide coordinated use of a resource that has one or more equivalent units (as specified). Use a *cobegin* or *fork* construct to do the same. Translate between any of *S/P*, *fork-join*, *cobegin-coend*, *semaphores*, and *process flow graphs*. This section will likely be covered by multiple problems.

Show the trace of a *monitor* as it executes a sequence of calls.

Show the execution of *rendezvous* code over a sequence of calls.

Analyze a proposed solution to the *dining philosophers* problem and answer questions about it.

CONTINUED ON BACK…

**Indeterminate questions (more likely multiple choice):**

Chapter 1: We talked about three key design ideas for operating systems: *abstraction*, *virtualization*, and *resource management*. We talked about examples of each, such as *spooling* to printers. We talked about the *interrupt handling* process, and the difference between a *trap* and an *interrupt*. We made a distinction between *multiprogramming* and *timesharing*. Note the discussion of runtime organization, where the author talks about client-server view of OS services versus the library call/kernel call view. Know what a *kernel* is. And know the different types of hardware for parallel architectures (*multiprocessor, multicomputer, processor array*, etc…) and why *multicomputers* the dominant supercomputer by market share today.

Chapter 2: We talked about the 5-state model of processes, compared *processes* to *threads*, and discussed the importance of a *process control block*. We talked about process control graphs as a precedence ordering of processes, and their relationship to the *cobegin-coend*, *fork-join*, and *forall* constructs (the latter of which is an example of *data parallelism*). We talked about *deadlock*, *livelock*, *race conditions* and *starvation*, so be able to define/distinguish these. We showed that the views of the process synchronization problem can be viewed through the complementary lenses of locks on resources and critical sections within processes. *Semaphores* were introduced as one of the basic solutions to synchronization problems.

Chapter 3: We noted that semaphores were in some way unsatisfactory solutions to our problems. Thus, we introduced higher-level language constructs that could be constructed using semaphores and would provide necessary synchronization facilities for programmers. These include: *monitors* (Hoare and Java/Mesa), and *protected types*; the distributed facilities of *mailboxes*, *ports*, *channels*, *RPC*s and *rendevous*. We applied some of these to classic problems like the *bounded buffer*, *elevator algorithm*, *dining philosophers*, *readers/writers*, and *logical clock* problems.