Computer Science Department
Stanford University
Comprehensive Examination in Software Systems
Autumn 1989

December 1, 1989

READ THIS FIRST!

1. You should write your answers for this part of the Comprehensive Examination in a BLUE BOOK. Be sure to write your MAGIC NUMBER on the cover of every blue book that you use.

2. The number of POINTS for each problem indicates how elaborate an answer is expected. For example, an essay-type question worth 6 points or less doesn't deserve an extremely detailed answer, even though a person can expound at length on just about any topic in computer science.

3. The total number of points is 60, and the exam takes 60 minutes. This "coincidence" can help you plan your time.

4. This exam is CLOSED BOOK. You may not use any notes, calculators, computers, or outside help.

5. Show your work, since PARTIAL CREDIT will be given for incomplete answers. For example, you can get credit for making a reasonable start on a problem even if the idea doesn't work out; you can also get credit for realizing that certain approaches are incorrect. On a true/false question, you might get partial credit for explaining why you think something is true when it is actually false. But no partial credit can be given if you write nothing.

Comprehensive: Software Systems (60 points) Autumn 1989

Problem 1 (15 points). Consider what happens when a page fault occurs. Briefly answer the following questions in a sentence or two.

1a. (3 points). How does the system decide which page should be brought in?

1b. (3 points). How does the system decide which page fraime should be used for the new page?

1c. (3 points). What happens to the page that used to be in that page frame?

1d. (3 points). What system tables must be updated as a result of this operation?

1e. (3 points). When does paging work? That is, when is the time cost of paging neglible compared to the execution of a program?

Problem 2 (10 points). Suppose a multi-level feedback scheduling algorithm can be parameterized by N, the number of levels, and $Q[1..N]$, an array of priorities indexed by level.

2a. (5 points). What parameter values would give the same effect as a first-come-first-serve algorithm?

2b. (5 points). What parameter values would give the same effect as a round-robin algorithm with quantum T?

Problem 3 (35 points). A certain freight train can hold a maximum of B boxes. The train travels across country once a day, stopping to drop off and pick up boxes at each of N stations along the way.

You are to write two functions, *train* and *sendbox*, that simulate the operation of the train with respect to loading and unloading boxes. The *train* function is called by a process simulating the train; *sendbox* is called by a process simulating a box that is sent on the train. You are only to write these functions; do not be concerned with the other code for the processes.

You are to use general semaphores to synchronize the train process and the box processes. The *train* function is passed a single integer parameter s that identifies a station. This function is called when the train stops at the specified station. To synchronize the unloading of boxes, it is necessary to wake up a box process and then wait until the box is taken off the train. Similarly, it is necessary to wake up box processes at the station and wait for them to be loaded on the train.

The *sendbox* function is passed two integer parameters identifying the stations where the box is sent from and to, respectively. This function should wait for the train to come to the originating station, load the box on the train, wait for the train to get to the destination station, and then unload the box. A box may not be loaded on the train if it is full (has B boxes already). If the destination of a box is in the opposite direction the train is traveling, the box should be loaded anyway and will be delivered the next day.

You can program these functions in Pascal or C. Be sure to specify initial values for all semaphores and global variables. Try to maximize concurrency while avoiding busy-waiting and the possibility of deadlock.