

## Computer Science 414—Operating Systems Sample Questions from Prior Midterms

- This is an open-book, open-notes exam. However, you may not use any other resources, such as other books, the Web, etc. You are welcome to use old 414 tests to study, but may not refer to them or their solutions (even if you made your own copy of such solutions) once you start the test.
- There are several questions of equal weight. If you are stuck on a problem, don't waste time. Move on and come back to it later.
- Write your answers *legibly* in ink or dark pencil. *If your answer cannot be read or understood*, or if your answer is vague or excessively verbose, it will be marked wrong.
- Show your work for partial credit. But do not “kitchen sink” – *incorrect statements, even if irrelevant or extraneous, will be held against you!*
- In any question, make any assumptions that you need to, but *document your assumptions*.

*Time limit: Three hours*

NAME (Print legibly):

PLEDGE (Write out pledge **in full** and sign):

1. What is the difference between a heavyweight and a lightweight process? Give an example of where heavyweight processes are appropriate, and one where lightweight processes are. In each example, make clear why one or the other is more suitable.
2. Show how a Unix or NACHOS shell would implement the following command line:

```
ls -l *.c > log
```

You need to show in your answer that you understand Unix or NACHOS process management (pick one or the other), and that you understand how the shell works. As a result, you should be explicit about all aspects pertaining to process management and the shell, and avoid lengthy discussions of peripheral topics. Be explicit about which system calls are used, where relevant information is kept, process states, who interprets what information, etc. Be explicit about where different parts of the command line (-l, the wildcard, etc.) are interpreted and/or handled. But don't explain how the command line is *parsed*, or how exactly the executable file is read into memory, once it is found.

3. A *barrier* is a synchronization primitive that we discussed in class. When created or reset with `CreateBarrier(n)` or `SetBarrier(n)`, it waits for `n` processes to call `WaitAtBarrier()` before the barrier falls. Until the barrier falls, processes that call `WaitAtBarrier()` must wait; once the barrier falls, all processes waiting at this barrier are allowed to proceed unconditionally.

Show how to implement barriers using semaphores. Your solution should avoid busy-waiting. Be explicit about any initializations that you need to assume.

4. Many computers today provide software-visible power management. For example, when a program has little or no work to do, it can reduce the CPU voltage and frequency, or put it to sleep altogether. In a multiprogrammed system, should this functionality be available in user mode or only accessible via a system call? Why or why not?

5. You have been hired by the CS Department to write code to help synchronize a professor and his/her students during office hours. The professor, of course, wants to take a nap if no students are around to ask questions; if there are students who want to ask questions, they must synchronize with each other and with the professor so that (i) only one person is speaking at any one time, (ii) each student question is answered by the professor, and (iii) no student asks another question before the professor is done answering the previous one. You are to write four procedures: *AnswerStart()*, *AnswerDone()*, *QuestionStart()*, and *QuestionDone()*. The professor loops running the code: *AnswerStart()*; *give answer*; *AnswerDone()*. *AnswerStart* doesn't return until a question has been asked. Each student loops running the code: *QuestionStart()*; *ask question*; *QuestionDone()*. *QuestionStart()* does not return until it is the student's turn to ask a question. Since professors consider it rude for a student not to wait for an answer, *QuestionEnd()* should not return until the professor has finished answering the question.