

# Operating Systems Preliminary Exam September 2002

- a. Be concise, clear, and precise, and justify all answers.
- b. Organize your answers: each question should be in one booklet; any additional materials should be stapled to the booklet and marked clearly, with your number.
- c. State your assumptions clearly. Assumptions should be realistic, otherwise points will be deducted.
- d. Use of drawings is encouraged. Label all elements in the drawings. If sequences of actions take place, number the actions and/or the location of the actions. Add a legend to the figure whenever necessary.
- e. Answer as many questions as possible. You must make a **SATISFACTORY EFFORT** on the **UNIPROCESSOR SYNCHRONIZATION** question, and you must give **SUBSTANTIALLY CORRECT** answers to at least **THREE** questions to pass.
- f. Exams are closed-book and are to be done individually.

# 1 Uniprocessor Synchronization

The OS has a set of queues, each of which is protected by a lock. To enqueue or dequeue an item, a thread must hold the lock associated with the queue.

You need to implement an *atomic transfer* routine that dequeues an item from one queue and enqueues it on another. The transfer must appear to occur atomically.

Assume that the following is used to implement the atomic transfer:

```
void transfer(Queue *queue1, Queue *queue2)
{
    Item thing; /* the thing being transferred */

    queue1->lock.Acquire();
    thing = queue1->Dequeue();
    if(thing != NULL) {
        queue2->lock.Acquire();
        queue2->Enqueue(thing);
        queue2->lock.Release();
    }
    queue1->lock.Release();
}
```

You may assume that `queue1` and `queue2` never refer to the same queue. Also, assume that you have a function `Queue::Address()` which takes a queue and returns, as an unsigned integer, its address.

- a. Explain how using this implementation of `transfer()` can lead to deadlock.
- b. Write a modified version of `transfer()` that avoids deadlock and does the transfer atomically.
- c. If the transfer does not need to be atomic, how might you change your solution to achieve a higher degree of concurrency? Justify why your modification increases concurrency.

## 2 Scheduling

- a. The length of the time slice (also called the *quantum* is a parameter to round robin CPU scheduling. What is the main problem that occurs if this length is too long? What about when it is too short?
- b. Can a multilevel feedback queue scheduler lead to starvation? If yes, show a scenario; if no, explain why not.
- c. Can a multilevel feedback queue scheduler lead to deadlocks? If yes, show a scenario; if no, explain why not.
- d. Describe the advantages and disadvantages of having the short-term scheduler select threads/processes to run on a completely random basis.

### 3 Synchronization in Distributed Systems

Assume that there are three nodes in a distributed system, and each message takes 3 time units in transit.

- a. Let there be a message leaving nodes 0, 1, and 2, at time zero to request a mutual exclusion session (all nodes request the critical section at time 0, according to their local clocks). Is it necessarily true that all nodes request the critical section at exactly the same time?
- b. Let only processes 1 and 2 be requesting the critical section. In a diagram (a timeline for each processor), show when messages depart, arrive, and type of message, when the processes are using Ricart&Agrawala's algorithm.
- c. Using Lamport's clocks, what must happen between any two events? Give a brief explanation and illustrate your explanation with an example.
- d. Show in a diagram two concurrent messages, A and B. That is, two messages that neither A happens before B nor B happens before A.

## 4 RPC: Remote Procedure Calls

- a. What is stub code and why is it needed?
- b. When a client attempts to call a RPC, it must first find a server that is willing and able to satisfy the request. How is a server specified? Specifically discuss the difference between static versus dynamic binding.
- c. Discuss the following different possible failure modes for an RPC call? That is, what are the pros and cons of *at most once*, *at least once*, and *exactly once* semantics? Describe how can these be achieved (i.e., give a sketch of an implementation).
- d. Discuss what happens when orphans are created and the impact of this in the state of the system? More specifically, discuss what happens when a client crashes and what are the resulting consistency/resource issues.

## 5 File Systems and Protection

- a. What is a protection matrix (in the context of file systems)? What is it used for? What are the two axes (dimensions) of a protection matrix?
- b. What are capabilities? How do they compare with protection matrices? Why are capabilities useful for efficiency? What is their main shortcoming?
- c. In file systems, caches are typically used for speeding up the system. One scheme that can be used for caches is write-through. What are the advantages and disadvantages of using a write-through or a write-behind (delayed-write) cache? What is the compromise typically used (e.g. in Unix)? Why is this a good compromise?