

# CS 111 Midterm exam

Adam Christopher Cole

TOTAL POINTS

**71 / 100**

## QUESTION 1

### 1 Priority based scheduling 10 / 10

✓ - 0 pts Correct

- 5 pts Failing to discuss starvation issue
- 8 pts Why is adjusting priorities important?
- 8 pts This issue requires pre-emption, not changing priorities.
- 9 pts Not commonly an issue. There are far more important reasons.
- 10 pts Completely wrong
- 8 pts Preemption is a mechanism that makes it easier to change priorities, but it doesn't explain why you want to.
- 2 pts Incorrect description of priority inversion
- 3 pts Requires better description of starvation problem.

## QUESTION 2

### 2 Process replacement 10 / 10

✓ - 0 pts Correct

- 3 pts Only 2 distinct reasons.
- 1 pts Non-preemptive schedulers do not use time slices.
- 2 pts OS bug in scheduler is highly unlikely
- 7 pts Only one distinct reason
- 2 pts In user thread case, the completion of the thread is caught by the thread package.
- 1 pts Unless the scheduler interrupts it, a process in an infinite loop DOES run forever.
- 2 pts Overhead doesn't cause crashes.
- 7 pts I do not understand what you mean by the 2nd and 3d reasons. They sound wrong.

## QUESTION 3

### 3 MLFQ scheduling 10 / 10

✓ - 0 pts Correct

- 5 pts What information must be kept?
- 5 pts Must keep information on time slice expiration and yields
- 5 pts Not answering about lecture version of MLFQ.
- 3 pts Why does MLFQ need this information?
- 10 pts Completely wrong
- 2 pts What other information must be kept?
- 4 pts Incorrect information kept.
- 3 pts RR doesn't do that.

## QUESTION 4

### 4 Wait and determinism 10 / 10

✓ + 4 pts Without case correct

✓ + 4 pts With case correct

✓ + 2 pts What wait does

- + 0 pts All wrong
- 3 pts Not realizing parent and child does not share address space

## QUESTION 5

### 5 APIs and ABIs 3 / 10

- 0 pts Correct
- ✓ - 3 pts Not noting OS component of ABI
- 3 pts Not noting HW component of ABI
- ✓ - 4 pts Not indicating the actual relationship.
- 4 pts API is not a subset of ABI.
- 2 pts ABI is HW-specific.
- 2 pts API is not HW specific.
- 3 pts ABI is not a type of API
- 3 pts ABI is a binding of an API to an architecture.
- 7 pts What is the ABI and how does it relate to API?
- 2 pts More detail on relationship required
- 7 pts too vague

- **4 pts** ABI is not a subset of API
- **2 pts** Binding to what?

#### QUESTION 6

### 6 Page stealing algorithms **8 / 10**

- **0 pts** Correct
- **8 pts** Purpose incorrect
- **4 pts** Purpose Unclear
- ✓ - **2 pts** Metric Incorrect
- **1 pts** Metric Unclear
- **10 pts** Incorrect

#### QUESTION 7

### 7 Condition variables for producer/consumer problems **0 / 10**

- ✓ + **0 pts** All wrong
- + **2 pts** Realize more than one consumers/producers are needed.
- + **10 pts** All correct

#### QUESTION 8

### 8 Spin locks and asynchronous completion **10 / 10**

- ✓ - **0 pts** Correct
- **2 pts** Unclear
- **7 pts** No mention of resource wastage
- **10 pts** Incorrect

#### QUESTION 9

### 9 Implementing locks **0 / 10**

- **0 pts** Correct
- **5 pts** Unclear
- **5 pts** No mention of atomicity
- ✓ - **10 pts** Incorrect
- 💬 You generally cannot make the critical section atomic. The goal of compare and swap/test and set is to make the acquiring of the lock atomic. Then, you can lock a potentially long critical section and provide correctness in asynchronous scenarios.


#### QUESTION 10

### 10 Bug in lock code **10 / 10**

- ✓ + **10 pts** Correct
- + **0 pts** Incorrect
- **4 pts** Unclear why two threads will acquire the lock

**Midterm Exam**  
**CS 111, Principles of Operating Systems**  
**Summer 2019**

Name: Adam Cole

Student ID Number: 

This is a closed book, closed note test. Answer all questions.

Each question should be answered in 2-5 sentences. DO NOT simply write everything you remember about the topic of the question. Answer the question that was asked. Extraneous information not related to the answer to the question will not improve your grade and may make it difficult to determine if the pertinent part of your answer is correct. Confine your answers to the space directly below each question. Only text in this space will be graded. In particular, do not write answers on the back of the paper, since that will not be graded. No question requires a longer answer than the space provided.

1. In a priority-based scheduler, why don't we typically have a static priority for each process, set to some value when the process starts and never changed until it completes?

Because there are situations that occur during priority scheduling that could stave other processes. If a process has the lowest priority, will it ever be able to run? This is a problem, and so in priority scheduling we normally have some instances where temporarily increasing a process's priority is necessary. Another case would be if a low priority process held a lock, and another higher process was waiting on it. The low priority process must be temporarily higher priority.

2. Name three different reasons why a process running on a CPU might be replaced by another process on that same CPU.

1. The current process issues an I/O request and must be blocked while it waits for data
2. The current process has a page fault and the OS must page in memory from disk, blocking the process until so.
3. The scheduler pre-empt's the current process so a higher priority process may run.

3. What information does the MLFQ scheduler discussed in lecture have to keep per process that a round robin scheduler would not have to keep? Why must this information be kept?

In MLFQ, the scheduler must keep track of the number of times a process does and doesn't use the allotted CPU time for that queue. This is in order to dynamically shift the processes into fast queues or slow queues. Round Robin schedulers do not have to keep track of this, because Round Robin schedulers give each process a fixed CPU time, and this time never changes. Each process stays in the same queue until completion.

4. Executing a `wait()` system call in a parent process immediately after it has created a child process changes a non-deterministic situation into a deterministic one. Why is the situation non-deterministic without the `wait()` call, and why is it deterministic with it?

Without calling `wait()`, both the parent process and the child process can run concurrently - without needing a specific order. When `wait()` is called, the programmer distinguishes that order matters, making the situation deterministic. In this case, that order is for the child process to run to completion before the parent can continue. The parent process relies on the execution of the child.

5. What is the relationship between an Application Programming Interface and an Application Binary Interface?

The API is the interface between programmers and programs. Programmers utilize library functions as an interface to program functionality. On the other hand, ABI is the interface between the program and the hardware/CPU. The hardware can only work with hex/binary code, and so converting a high-level program to binary is the necessary interface to execute a program.

6. What is the purpose of a page stealing algorithm? What determines if a particular page stealing algorithm is a good one?

The goal of a page-stealing algorithm is to help the processes (running and in queue) maintain their optimal working set, which is the optimal number of page frames per process to minimize page faulting. Therefore, a successful page-stealing algorithm is one which "steals" a page frame from a process with a working set larger than its optimal one, and gives it to a process with a working set smaller than its optimal one. This is "good" opposed to stealing a page frame from any process's working set.

7. In producer-consumer code where conditions variables are being used to signal the status of buffers containing work, one should always check the condition variable in a while loop, rather than in an if statement. Why?

Checking the condition variable within an if statement is not good because it requires a different condition to check the status of the buffer. If control flow never enters the inside of the if statement, the buffer may never be checked! A while loop, however, will ensure the condition variables are checked in succession until the buffer status is done. If we never check the status of our work, we may never get the data we requested.

8. Why are spin locks often a poor solution to solving asynchronous completion problems?

Spin locks are often a poor solution to asynchronous completion problem because the threads spin in while loops continually checking the status of completion. This wastes cycles, and it can waste much, much more cycles waiting on completion than for a critical section, because critical sections are designed to be concise, but it is unknown how long a completion problem will take to finish.

9. Why can locks be correctly implemented using assembly language instructions like Compare and Swap or Test and Set?

Critical sections occur when it takes more than one machine instruction to update the state of a resource. In these situations, programmers use locks to ensure atomicity - when the critical section is run, it is run to completion as one. With Compare & Swap / Test & Set, programmers can reduce the critical section to be only 1 machine instruction. This ensures atomicity because a single machine instruction can't be interrupted or run out of order. If done correctly, hardware instructions test & set and compare & swap can successfully achieve mutual exclusion & atomicity - just as locks do.



10. The following C code is a simple implementation of a lock. It has a correctness problem. What is that problem? Describe a situation in which the problem would occur.

```
typedef struct __lock_t { int flag; } lock_t;

void init(lock_t *mutex) {
    // 0 -> lock is available, 1 -> held
    mutex->flag = 0;
}

void lock(lock_t *mutex) {
    while (mutex->flag == 1) // TEST the flag
        ; // spin-wait (do nothing)
    mutex->flag = 1; // now SET it!
}

void unlock(lock_t *mutex) {
    mutex->flag = 0;
}
```

The correctness problem in this implementation is the present race condition within `lock()`. If multiple threads are spinning, waiting for the lock to be available, then multiple threads could all access the lock if each of them exited the `while()` loop at the same time before any of them set `mutex->flag == 1`. In this case, locking the original critical section created another one.