/

95/100

Midterm Examination
CS 111
Fall 2015

Name: Ryan Peterman UID: 704269982

Answer all questions. All questions are equally weighted. This is a closed book, closed notes test. You may not use electronic equipment to take the test.

1. One principle of achieving good robustness in a system is to be tolerant of inputs and strict about outputs. Why? Describe an example in the context of operating systems.

It is important to be tolerant of inputs and strict on outputs because this prevents error propagation in a system. For example, in the context of the OS when handling traps, there are many different traps that a first level handler may receive, but the second level handler will receive a well defined strict output from the first level handler. Then when the second handler is finish only a strict set of outputs will be visible to the application program (i.e. condition codes)

10

2. What is the advantage of using the copy-on-write optimization when performing a fork in the Unix system?

The advantage of using the copy-on-write optimization when forking in Unix, is that the text and data segments of both processes can be shared (until written to), which is more space and time efficient. A process doesn't need a different copy of the data section until it writes to it. From there a copy will be made to write to.

5

exec issue - copy on write exec issue in Unix is great because we will exec anyway in most cases and throw away stack.

3. What is emulation? What is the main challenge in software emulation?

Emulation is when you have software implement hardware on a machine whose actual hardware is potentially very different. The main challenge is the extra overhead incurred for every instruction from the emulator being translated to operations the underlying machine can run. It is very inefficient, but has become better in recent years.

10

4. Round Robin, First Come First Serve, and Shortest Job First are three scheduling algorithms that can be used to schedule a CPU. Which one is likely to have the largest overhead? Why?

$\boxed{\text{Round Robin}}$ is most likely to have the largest overhead because it is a preemptive scheduling algorithm, which is more complex and incurs the added overhead of context switching. First come first served and shortest job first are both relatively simple algorithms that run jobs until completion based on their criteria and therefore do not waste as many cycles on context switches which happen to switch from a finished process to a new one. (OS trap/interrupt context switches happen as well, but the frequency of traps is presumably independent of scheduler)

10

5. **What is the difference between a first and a second level trap handler? Describe one advantage of using this two-level approach to handle traps.**

The difference between a first and a second level trap handler, is that a first level handler sets up for execution of kernel level code by saving registers and changing processor status word in preparation for redirection to the specific second level trap handler that deals with the actual trap $\overset{\text{implementation}}{\vee}$. An advantage of using a two level approach is that it encourages modularity in the OS code since the two roles of each are clearly defined and seperate in function, it will be easier to maintain and understand.

10

6. **What is fate sharing? Three common interprocess communications mechanisms are messages, shared memory, and remote procedure calls. For which of these is fate sharing most likely? Why?**

Fate sharing is when two processes/systems can affect each others outcomes by communicating or depending on a result from each other. For example if one program crashes then the other will crash too if they are fate sharing. $\boxed{\text{Shared memory}}$ is most likely to cause fate sharing since if a buggy program writes garbage to the shared memory and overwrites another program's memory, then this can result in the second program becoming buggy as well. RPC and message sending can at worse return a bad result, which a program can check and handle. RPC/message sending are more modular and have well defined interfaces.

10

7. **What is a bus master? Why is a device other than a CPU likely to become a bus master, and what operations will it typically use this role to perform?**

A bus master is a device that has control over the bus and can request data from the bus. Another device may likely become a bus master so that it can handle I/O operations while the CPU is doing other useful work. For example, in the case direct memory access, a device will handle copying relatively large amounts of data without the help of the CPU so that the CPU can be free to do useful work.

10

8. **What is the asynchronous completion problem? Is a spin lock a good solution for this problem? Why?**

10

The asynchronous completion problem is when multiple threads/processes finish at different times and depend on each others output at some portion of the code and must therefore wait for the slowest threads to finish. A spin lock is not a good solution in most cases because it puts the process/thread in a busy cycle doing useless work. These CPU cycles could be used much better and therefore typically, blocking is a better solution since a process thread will stop executing while it waits which frees up the CPU to do more useful work.

9. In the context of locks, what is the single acquire protocol? Describe a case in which it can be safely relaxed.

Single acquire protocal is when a resource is locked to allow only one process/thread use it at a time. A case in which this can be relaxed is if a system were to have multiples of a resource, then multiple threads/processes could use the resource at the same time. For example, if there were multiple hard drives and each process/thread wanted to write to memory (independent operations) then a program could relax the locks to allow multiple processes/threads to use the resources.

10

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

Locks can be correctly implemented using compare-and-swap or test-and-set because these instructions are atomic such that we can test a lock and set that it is being used before the OS can preempt the process and schedule a different process to run in the middle of a critical section of code. This allows for allows for a correct implentation of the lock since the critical section of code is atomic using these instructions.

10