MATH 115A/6. EXAM 1.

[redacted]

95/100

Midterm Examination
CS 111
Fall 2015

Name: [redacted]

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.

This is important because we want to suppress the propagation of errors, and this technique helps because it acts like there were never any errors in the first place if it receives an imperfect input

10

One example would be a disk device driver: it would be lenient in terms of the size of read/write requests, but it will always do the actual reads and write request in hardware of the correct size, instead of always requiring requests to be the block size.

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

The advantage of using copy-on-write is the kernel doesn't have to copy the entire process address space for the child process, it will only copy when some data is changed. What happens instead is that both processes read from the same place in memory. This is useful because processes are almost identical to forked each other; if the child process will maintain the same memory state as its parent, then it would have been a waste to copy everything over and load it into memory/cache

5

real issue

NAME

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

Emulation is using a single physical instance of a computer to run a single virtual instance. The virtual instance is generally a different interpreter (w/ different repertoire, environment reference, and instruction reference) than the physical instance.

10

The main challenge of software emulation is implementing the interface b/t the virtual and physical instance. The interface must be able to take instructions from the virtual instance and properly convert them to the physical CPU's instruction set in software. Often, this is a large obstacle and causes the virtual instance to run a lot slower. ← That's the real problem

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?

Round Robin is likely to have the most overhead. This is because this scheduling algorithm uses the most context switching and preemption. Round robin swaps out processes when their time quantum expires, whereas FCFS only context switches when a current job finishes, and Shortest Job First only context switches when a job finishes or if a shorter job arrives in the middle of the current one.

This means that Round Robin has the most preemption and causes the most overhead from the context switching which occurs at least once every time quantum. However, with a long enough time quantum, it devolves to FCFS, with the least overhead.

MATH 115A/6. EXAM I. 26 OCTOBER 2015. R. De Sapio

NAME _____

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 first level trap handler stores registers and other states of the process which trapped, and it then finds and calls the appropriate second level handler. The second level trap handler then handles the current problem, and hands control back to the first level handler when it's done. Finally, the first level handler restores the original process state and passes control to the original process.

10

As we can see, first level handler is very general, and provides a single, common point of entry into the kernel functionality, which simplifies entry, and also prevents redundant code. The second level handler is specific, and only built to handle a certain problem, having the first level call the second level reduces the risk of running the wrong code for the situation.

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 modules are not independent of the other, eg. if one crashes, so will the other. An example is in a single process, if one procedure divides by zero, then the entire process may terminate, even though the other procedures had nothing to do with the process which divided by zero.

Fate sharing is most likely with shared memory, which is dangerous because each process using the memory has the potential to overwrite/corrupt the memory of the other process. Messages have the second most likely fate sharing, but a significantly reduced amount because the processes only interact through the medium of messages, which can be checked for errors. RPC has the least fate sharing because the caller and callee are completely independant of each other except for passing arguments and the return value. RPC can be conceptually understood as having the caller and callee on different PC's, and if the callee crashes, the caller is still able to run.

MATH 115A/6.

11

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?

The bus master is the hardware device which controls who sends signals and what signals are sent and who the signals are sent to over the bus, which is a physical communication link b/t devices. Most likely the main memory device will become the main memory handler because the CPU interacts the most with memory, and this frees the CPU from having to execute the correct instructions for reading from the specific device.

Eg: A disk, which requires an algorithm to choose the order of reads and writes, and also translates a read/write request into the appropriate location on disk, and transforms the request size into the correct block size that is read in and out.

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

The asynchronous completion problem occurs when a process is waiting for another process to complete, and the two processes are running concurrently on different cores. One course of action is to spin lock, which means the process continuously polls the second, waiting for it to complete. This is bad due to wasting CPU cycles. The better course of action would be to yield, and to allow another process to run and maintain high utilization of the CPU.

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

The single acquire protocol is the case where a lock may only be acquired by one thread at a time. There are multiple cases where this is not as useful as a lock which can be acquired multiple times. One example of such a case is the use of a partitionable resource; if the resource can be used by n threads and only one thread can use it at a time, the resource has a utilization of 1/n!

10

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

This is due to before-or-after atomicity: implementing locks in the lowest layer of interpreter guarantees that the compiler will not reorder the lock instructions and also guarantees that if a process is rescheduled and is stopped from running that no possible race conditions can happen from other running processes. With compare and swap/test and set, the lock is either fully acquired or it is fully released, there is no in-between which may cause edge cases and race conditions. This is especially important because locks are the primary method of handling synchronization and preventing race conditions on the programming language scale.

10