93/100

Midterm Examination
CS 111
Fall 2015

Name: Vilius Vysniauskas 704 289 956

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.

   Being tolerant of inputs means the developer takes into account as many edges cases/ situations as possible. This helps robustness because users are not limited to certain inputs/actions/methods. Being strict about outputs means the developer has clear outputs that are understandable and simple. The developer takes the many inputs and condenses them down into a much smaller number of possibilities. This helps robustness because it allows the developer to deal with the many types of input and it enables the developer to implement robustness for the user in a logical and concise form. In OS's, an example of this is a federation framework. Such frameworks are robust because they allow generality without limiting potential. This is why different printers (input) can print the same way (strict output).

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 copy-on-write is performance. Performance is increased both in time and space. The CPU has no overhead in actually performing the copy of the process and since there is no actual copying initially, no extra memory is used. If either the parent or child writes however, that process must copied. Overall, this uses computation time and memory space provided the child and parent do not write to their memory.

5

exec issue

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

Emulation is using a physical virtual machine to run another virtual machine. The main challenge in software emulation is correctness. By virtualizing a different architecture, many difficult issues, like how to use an interpreter's repertoire to run a different set of instructions, arise. But solving correctness is not enough. Once it is correct, performance is also another concern.

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?

Round Robin would have the largest overhead because it is preemptive. Because Round Robin multiplexes the processor into time splices, each process only gets a certain amount of time to accomplish its work in a singe time interval. If it does not finish in its time splice, a context switch occurs and the next process takes its turn. The other two schedulers mentioned are non-preemptive, meaning they do not interrupt the process while it is running. Once they are scheduled, they run to completion. The context switching of Round Robin is the thing that creates the large overhead.

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 first level handler saves the PS, stack memory, and necessary registers. It also selects the second level handler from a trap vector table or a similar OS structure. The second level handler actually deals with the problem. An advantage of this approach is modularity. If something does not work in the second level handler, i.e. if it does not deal with a certain situation correctly, the developer of the OS does not have to change the entire handler code. Since the first handler chooses the second level handler correctly, there is no reason to change it. The problem lies only with the second level handler and thus modularity is increased.

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 a mistake/bug/error occurs in a related piece of code, like a local procedure call, causes the code/program that called or instantiated it to also share the same fate. An example of this is if a local procedure call overwrites the stack frame of the caller, then both the caller and the callee crash. Out of the three IPC mechanisms mentioned, shared memory is most likely to have fate sharing occur. This is because the two or more processes that are communicating share the same memory space for communications. If one writes into an area it is not allowed or reads from an area it is not allowed, then both processes do not observe the correct behavior.

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 the device that controls all the other busses by telling them who to communicate with. A device other than a CPU is likely to become a bus master because of performance benefits. By delegating a different bus master, the CPU does not have to be directly involved in the I/O. It can simply tell the current bus master what it needs and the bus master can do it for the CPU. This means the CPU can go on to do other things until the bus notifies it has processed the request. The bus master does this by issuing an interrupt. The interrupt tells the CPU that the bus is finished and awaiting more instructions.

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 the problem of how to wait for something to finish without wasting time doing nothing. For example, if a process needs a resource, does it wait indefinitely for it? A spin lock is not a good solution because it does not solve the issue of wasting CPU cycles. By implementing spin locks, the CPU is forced to keep rechecking if a resource (i.e. a lock for it) is available. If it has to do this for a long time, there is no forward progress in the process it is running.

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 when one lock is associated with a single resource. This can be relaxed if multiple processes are only going to read from that resource. This works because reading a shared resource does not alter its contents. Each read, regardless of when it occurs, would produce the correct output, provided all the processes that need that resource only read from it.

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 such instructions because they are atomic. Atomic instructions cannot be interrupted because they are seen as a single instruction performing one thing.

8

why do these atomic instructions work for locks?