/

52/100

Midterm Examination
CS 111
Fall 2015

Name: Minh Le 904478767

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.

5

This principle allows greater tolerance in terms of input so there are more handled cases of the system but at the same time, the output is restricted to allow predictability/standardization. ~~This~~

more details

~~An example are I/O devices over USB ports ...~~

An example can have different I/O peripheral devices such as mouse and keyboard over USB while the types of raw output can vary, the OS has device drivers to restrict the actual outputs to a predictable and standardized format for the system to understand and use.

Not good example

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

5

With copy-on-write optimization, the child process produced by fork ~~uses the sa~~ is data memory is mapped to the parent's data memory unless either the child or parent writes (changes) to it. This optimization means producing a new process from an existing process does not waste resources on copying memory over ~~unless~~ until necessary.

exec issue

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

Emulation is a form of virtualization in which one system replicates to have the specifications of the other. One example is emulating a 32 bit linux system with a virtual machine (though these attempt to optimize as much as possible by using as much native hardware) as a 64 bit version OS. In the computer aspect, this might be core count, memory size, or specifications (register size).

The main challenge for software emulation is speed. ~~Because one virtualizes everything~~ If one virtualize everything of the system or is trying to replicate the software overhead is much slower than using the system natively.

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?

Shortest Job first will most likely have the largest overhead mainly because it requires the scheduler to do extra to maintain the process queue. While Round Robin is just preempting a process after x number of clock cycles & F.C.F.S just bases its scheduling on request order, SJF must A) determine the length of each process request even without being able to know for sure (because it hasn't run yet!) and b) dynamically reorder the queue every time a new process joins the queue.

2

0

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.**

set up { The first level trap handler is responsible for pushing on the (saved) interrupted process's context onto a stack (includes its PC, PS, registers, etc) to be restored later on. It then goes to the trap vector table to determine the second level handler. After setting up the parameters for such it jumps to the 2nd level handler.

10

execute { Rather than setup, the 2nd level trap handler is responsible for actual execution. After running its code to service the trap, it returns any status/return value to the 1st level handler who in turn unwinds the stack back to the prev process (or + scheduler to select a new process).

The main advantage of this is modularity between set up and execution part of trap handling. This allows incremental changes in the system trap system if necessary. For example, adding a new trap would mean just extending trap table + code for 2nd level handler, 1st level is not changed at all.

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 between 2 entities (in this specific problem processes) means that the survivability of each process is not independent. This meaning means if one process has an error, it propagates to the other, being potentially disastrous not only to the buggy process but the other as well.

10

Of the IPC's, shared memory is most likely fate sharing.

- In RPC, one process (caller) "calls" a subroutine of the other "callee". Even if the callee's routine is buggy/crashed; it would not affect the caller simply be[cause] routed through virtualization of the 2 processes
- Similarly message sending is an OS controlled interaction between 2 processes (though messages can be stored in the sender's address space). The clear specification of receiver can't receive unless sent in messages still adhere to the separation of 2 processes. So an error in one does not propagate.
- ☆ Shared memory on the other hand is fate sharing. One can consider as extension of address spaces of both processes. Thus means if one is buggy and stomps at this shared memory it does affect the other process's data and thus does propagate error... This is fate sharing.

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 has control and discretion over who can use the bus to transfer data and when. Other devices, specifically, I/O device controllers can temporarily become the bus master especially during copy large amount of data to memory (like hard disks). Rather than have CPU facilitate a "trivial" execution of I/O and wasting computational time, we allow the controller (such as hard disk) to be bus master and do direct memory access and copy data in directly to memory.

10

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

The asynchronous completion problem is when dependent concurrent entities have different execution time and the order of execution plays a role in concurrency. One example is 2 process, ~~one writing to a shared buffer while the other executing concurrently. How~~ write a shared buffer. While both executes concurrently the reader to the buffer can't do so until writer finishes writing. (different execution speed). ~~One solution~~ A spin lock is not a good solution ~~simply~~ simply because the computer burns computational ~~time~~ resources (active processor) just to wait on an event (no computation done). A better solution is blocking the waiting process and ~~send~~ ~~wait~~ waking it up (via interrupt) when the event it waited for occurs. ~~I'm to ...~~

10

For the example, the reader process actively waits with a loop until writer process finishes.

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

Single acquire protocol is one resource one process/thread lock. This means one and exactly one process/thread can lock a resource. and no other process/thread can use it (mutex). ~~This can be relaxed~~ This strictness is unnecessary when talking about accessing critical sections of memory amongst multiple concurrent reading processes. Reading does not change the critical section so multiple process can use this shared resource concurrently without any ~~detrimental~~ race conditions.

10

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

Assembly instructions such as are atomic instructions meaning they are indivisible sections of execution. (cannot be interrupted /preempted in the middle). This allows for locks to be correctly implemented because we want a processes, if attempting to obtain/release a lock, to either be able to do so without preemption

10

This desire is apparent in a given example say releasing a lock requires actually releasing the lock on the resource and then saying the process does not have access to the resource anymore. If that was preempted in the middle and another process sees the "free resource", suddenly we have 2 processes on a critical resource. The same idea applies to obtaining a lock.

This means lock obtaining/release ~~[illegible]~~ needs to be indivisible so we use small, atomic instructions.