98/100

Midterm Examination
CS 111
Winter 2014

Name:

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. What is the Thundering Herd problem? What can be done to mitigate it?

When we have multiple threads/processes all requesting to use the same resource, the scheduler will have to put most of them to sleep, and when the first one completes the rest are woken up, one is selected to run, and the rest go to sleep again. If this is done repeatedly, there is great overhead in the waking and sleeping of all processes/threads.

To mitigate the problem, we can have the scheduler wake up just one process/thread each time. We could also have more of the resource or create fewer threads/processes in the first place. Finally, we could also try to minimize the use of the resource in question.

10

2. Processes are allowed to have stacks that grow arbitrarily, up to the limits of available memory. Threads have stacks of fixed size. Why?

Processes have their own virtual address spaces, and one process is not able to write to the memory of any other process. With threads, however, this is not the case because they can share an address space. Allowing process stacks to grow arbitrarily is fine because no other process is affected, but allowing thread stacks to grow arbitrary could cause one thread to overwrite the stack of another thread in the same address space.

10

3. One method of avoiding deadlock in memory allocation is to require applications to reserve the maximum amount of memory they might need when they start. The system guarantees that they will receive the amount required if they ask for it. Which of the four conditions required for deadlock does this approach address? Why does it invalidate that condition?

3

The condition addressed is "incremental allocation". Because we allocate all of the memory at the beginning in one shot, there is no chance that the application will find itself without enough memory while it is running. This effectively means that we don't have to do any more allocation in the program except that which we do at the very beginning. How relevant to deadlock?

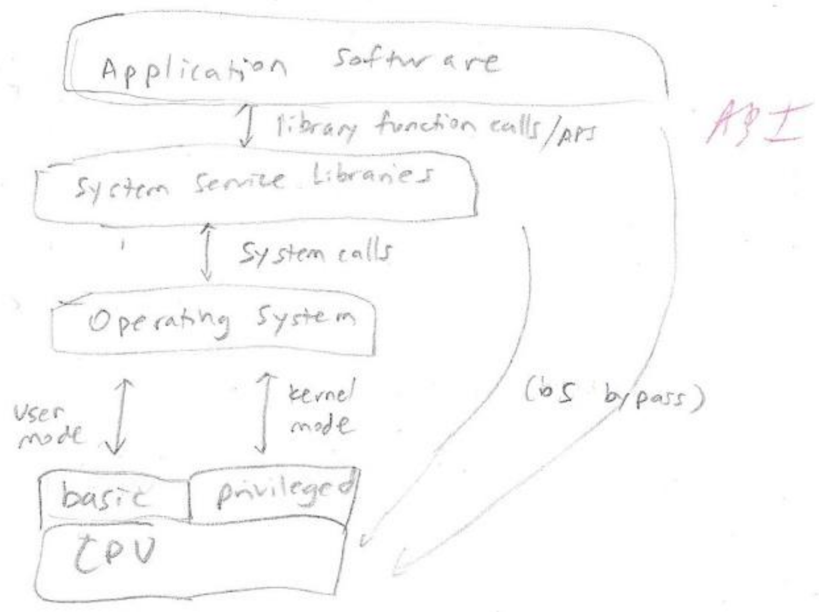
4. There are two general approaches for dealing with a writer who gets too far ahead of a reader: blocking the writer, or dropping some of the data.
(a) If we blocked a writer, and the reader never resumes, what problems might occur? How could we reasonably recover from this?
(b) If we drop some data, and then the reader catches up again, why is this a problem? How could we reasonably recover from this?

0

a) This could potentially freeze the program if the writer waits forever. We can recover from this by allowing for timeouts, at which point we assume that the reader has a problem. We can then see what the writer wants to do (retry, try another reader, skip operation, return error message, etc.).

b) If data is dropped, the reader will miss some of the writes and what it reads will be inaccurate (also violates all-or-nothing atomicity). To recover, we can have a mechanism in place where the reader tells the writer what was received, and if the writer notices data that was not successfully received, it can send it again. If the reader doesn't necessarily need all of the data (like a video streamer), it can also just ignore the dropped data and move forward.

5. Draw a diagram, illustrating the relationships between the CPU (basic and privileged instruction sets), the OS, system service libraries, and applications software. Label the major interfaces.



6. Most modern operating systems allow applications to invoke printing in a generic manner, regardless of the models of printers the system has available. What important operating system design concept does this choice illustrate? What are the advantages and disadvantages of the choice, as related to that concept?

Abstraction. This abstraction allows the application programmer to use it without needing to know the grubby details. It also makes it easier to handle changes to the hardware (printer) because it can work with any printer. However, abstraction also results in a loss of flexibility, since the programmer can't use any features unique to the current printer.

10

7. Given that we need to run some computations in parallel,
   a. Name two characteristics that would lead us to choose multiple processes
   b. Name two different characteristics that would lead us to choose multiple threads in a single process

a) Robustness (or security) - if we don't want an error / failure in one to affect the other (processes have separate domains)

Virtualization - if we want each computation to appear to have all of the hardware to itself (like in a virtual machine)

10

b) Sharing - if the computations share some data in common (threads share an address space)

Short life span - if computations are many but time spent on each is minimal (lower thread creation/destruction overhead favors threads)

8. What is the elevator algorithm? What kinds of problems would it suitably address? Does it optimize such problems for a single metric or does it provide a reasonable tradeoff between two different metrics? Explain.

In the elevator algorithm, the scheduler heads in one direction, stopping along the way to schedule tasks that are closest along that direction. Then, when it has reached the end, it changes directions and repeats the process.

This does not optimize for a single metric, but provides a trade-off between throughput and average waiting time. It doesn't have the throughput efficiency of first-come first-served, but it has better waiting time. It doesn't have the optimized waiting time of shortest job first, but it prevents starvation from occurring.

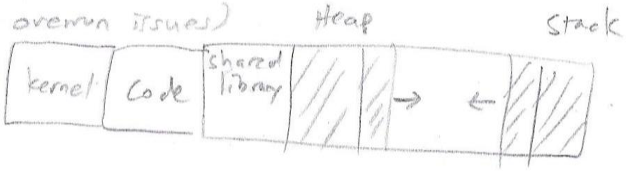
10

9. List the primary types of segments found in a process's virtual address space, and two key characteristics of each of those segments. (Not their contents, but important characteristics of them.)

Heap segment
- grows towards larger addresses
- not executeable (to avoid buffer overrun issues)

Stack segment
- grows towards lower addresses
- not executeable (to avoid buffer overrun issues)

10



Code segment
- execute only (to preventchanges)
- should not change during program execution

Shared Library segment
- execute only (to prevent changes)
- should not change during program execution

Kernel segment
- only executeable while in kernel mode
- should not change during program execution

10. Remote procedure calls have both advantages and disadvantages over ordinary procedure calls. In the realm of failures, name one such advantage and one such disadvantage.

Advantage: reduction of fate sharing - a failure in the callee will almost certainly cause a failure in the caller of an ordinary procedure call, but not in a remote procedure call because of the message-based communication.

Disadvantage: new failures - because of the message-based communication, there is now the issue of "what happens if the message is lost in transit?" and "what happens if the other side does not respond?" These must be dealt with in remote procedure calls.