COEN 177 Sample Midterm Answer Guide

Q1. OS Architectures

In this question, I am looking for answers that demonstrate a basic understanding of the different architectures.

a) The virtual machine architecture is focused more on the management and sharing of virtualized resources than attempting to offer a simplified or higher-level abstraction of those resources. Or another way of putting it is to highlight the focus on virtualizing the hardware, as opposed to the offering of a high-level system call interface.

b) The key point is the minimization of the amount of code running in kernel mode, which you can say directly or by indicating that the bulk of the functionality is exported and implemented as user-level processes.

Q2. OS Types

The key difference is deadlines. RT operating systems attempt to meet deadlines, but equally importantly, they provide a mechanism for specifying them (think about the last time you opened a file or launched a new process, was there any parameter you used that indicated how quickly you needed the file open task to be completed, or how much CPU time the process you launched must be given within a given real-world time period? probably not. RT operating systems need to have support for interfaces that allow you to convert that information, as well as working to meet those demands.

Q3. Scheduler Goals

Response time is a good choice. Though slight deviation is acceptable if accompanied with a strong justification for most questions, here all you are told is that the system is interactive, so the response time or responsiveness are the only criteria/metric you can feel confident about.

Q4. Scheduling Example Problem

My goal with a question like this is to give you an opportunity to demonstrate your understanding of how a particular scheduling algorithm should work. So it's important to show your work, and use a simple and clear notation for what you are doing. For example:

t:0 [B6] A16

could be used to indicate that the current time is 0 (t:0) and Job B has been assigned to the processor, and jobs B and A, respectively, require 6 and 16 time units to complete.

t:2 [B4] C14 A16

… would be the next state. Notice that C will be chosen to run before A, but that neither has been allowed to run yet because we are using SJF and B still has 4 time units to go.

t:3 [B3] D8 C14 A16

t:4 [B2] E4 D8 C14 A16

t:6 [B0] F1 E4 D8 C14 A16

(Notice that t:6 [E4] F1 D8 C14 A16 is also valid, but I will assume that the arrival of F was handled by the OS before the exit of B)

t:6 [B0/F1] E4 D8 C14 A16

t:7 [F0/E4] D8 C14 A16

t:11 [E0/D8] C14 A16

t:19 [D0/C14] A16

t:33 [A16]

t:49 [A0]

a) The order of execution in this example is BFEDCA (had we made a different assumption it would have been BEFDCA).

b) 6 jobs every 49 time units is the average completion **rate**.

c) The average is (33+0+17+8+3+0)/6, since A waited 33, B waited 0, C waited 17 (19-2), D waited 8 (11-3), E waited 3 (7-4), F waited 0 (6-6)

Q5. IPC/Synchronization Mechanisms

Initialize the variable to be used as the binary semaphore/mutex to 1. We will call this variable x in the following pseudocode:

P(x): while(!test-and-clear(x));

V(x): x = 1;

Q6. IPC/Synchronization Mechanism Usage

declare a global variable x to be used as a mutex.

mutex x;

void Swap(int &a, int &b) {

int temp;

temp=a; / ask yourself why this does **not** need to be inside the critical section */*

*/*hint: temp is a local variable, and where in memory are local variables stored?*

*/

P(x);

a=b;

b=temp;

V(x);

}

Q7. Potential deadlocks in your own code

This question can be asked in many different forms, and in any particular form has a wide spectrum of acceptable answers. The key is to realize that this is an attempt to evaluate your ability to reason about a practical application of the principles used to write deadlock-free code.

a) To answer part a you may follow one of two strategies. The first is to treat the problem as a form of one or more classical IPC problem and explain how it differs and how the known solutions to those problems could be adapted. For example, this problem is similar to the readers-writers problem, but with the added feature that there is a numeric limit on the readers (like there is a numeric limit on the waiting customers in the sleepy barber problem). At the other end of the spectrum you could answer this question by describing code that completely prevents the possibility of deadlock by breaking one of the conditions (e.g., only assigning CDs to processes in ascending numerical order of their requests, and failing to assign a CD if a process attempts to acquire a device while already possessing a higher numbered device).

b) To answer this part you must understand both the nature of deadlock and the structure and logic of your solution in relation to that. For the two examples I gave above, the latter is easy, as it was simply ensuring the requirement that there exist a cycle of dependencies would never be realized. The first was not a full answer, just a strong hint, but remember that the readers-writers solution prevented any thread

from holding a resource while waiting for another (since it wouldn't allow a writer or a reader to attempt multiple requests for the resource in different roles, in other words no reader would be a writer while reading). So it eliminates hold-and-wait, but it could also be viewed as a solution that took the medium (the data/resource being shared) and allowed it to be shared for readers, but not writers, thereby eliminating mutual exclusion when appropriate.

Q8. Virtual Memory

a) An individual page table has 2^64 byte divided by 8KB entries (one for each page). That's 2^64 / 2^13 which gives us 2^51 entries. If each entry is 8 bytes (2^3 bytes), then each table is 2^54 bytes in size. That means that a page table is 16 Petabytes in size. And we have five of them (one per process), so that's 80PB.

b) The answer to part a) multiplied by 8 (since the pages are now one 8th the size, there will be 8 times as many of them to track).

b) This is the only part that cares about the RAM (and does not care about the number of processes). The page table has 50MB divided by 8KB entries, and each entry requires 8 bytes. That is 50KB in total.

Q8. Deadlock Avoidance

(since the table says "Max" we are dealing with safe vs. unsafe states))

a) No. Not safe.

b) Yes. Yes, it's a safe state. Deadlocks cannot occur is we only allow the system to move to safe states. Deadlocks can occur in the future if we allow the system to move into an unsafe state.