Department of Computer Science
Qualifying Examination Solutions
Operating Systems
Fall, 2001

The following exam is open book and open notes. You may feel free to use whatever additional reference material you wish. Please note the following instructions.

- show your work whenever appropriate. There can be no partial credit unless I see how answers were obtained.
- be succinct. I will take points for facts that, while true, are not relevant to the question at hand.
- The questions are equally weighted.

1. (25 points) In the following code `x`, `y`, and `z` are all binary semaphores initialized to 1. The following code can deadlock. Explain how.

   Process A:

   ```
   wait(x);
   wait(z);
   ...some code...
   signal(x);
   signal(z);
   ```

   Process B:

   ```
   wait(y);
   wait(x);
   ...some code...
   signal(y);
   signal(x);
   ```

   Process C:

   ```
   wait(z);
   wait(y);
   ...some code...
   signal(z);
   signal(y);
   ```

   How could you modify the code to ensure mutual exclusion between the processes, but ensure no deadlock occurs?

This problem is a "gimme." Of course, it's just the Dining Philosophers Problem. So deadlock happens if they all pick up their left fork (ie they each get the first semaphore: Process A executes `wait(x)`*, Process B executes* `wait(y)`*, and Process C executes* `wait(z)`*) and then they all block trying to get the second semaphore.*

The fix is just to sort the resources so they each get their semaphores in the same order. Now Process A gets `x` *before* `z` *(which is no change), B gets* `x` *before* `y`*, and C gets* `y` *before* `z`*. This is permissible because the semantics of the programs only require that they get both semaphores before entering the protected code; if there was code between the* `wait()` *statements this wouldn't be allowable and the problem would be harder.*

2. (25 points) Consider a software distributed shared memory system with three processors implementing a strict memory consistency model. Assume memory location X's directory entry is located on processor P1, Y's is on P2, and Z's is on P3, and that the directory entries don't migrate.

   For each of the memory transactions in the following table, describe the messages which need to be sent between the processors for the transaction to occur, and the contents of the directories after the transaction is finished. You may use the following table to answer this question. In the table, "S" is the state of the variable and "P" is the set of processors with valid copies.

Transaction	Messages	X		Y		Z	
		P	S	P	S	P	S
(initial)		{P1}	RW	{P2}	RW	{P3}	RW
P1: read Y	$P1 \longrightarrow P2$ $Read\,Y$ $P2 \longrightarrow P1$ Y			{P1,P2}	R		
P1: write Y	$P1 \longrightarrow P2$ $Write\,Y$ $P2 \longrightarrow P1$ Y			{P1}	RW		
P2: write Z	$P2 \longrightarrow P3$ $Write\,Z$ $P3 \longrightarrow P2$ Z					{P2}	RW
P1: read Z	$P1 \longrightarrow P3$ $Read\,Z$ $P3 \longrightarrow P2$ $Read\,Z\,P1$ $P2 \longrightarrow P1$ Z					{P1,P2}	R
P3: write Z	$P3 \longrightarrow P1$ $Write\,Z$ $P3 \longrightarrow P2$ $Write\,Z$ $P1 \longrightarrow P3$ Z $P2 \longrightarrow P3$ Z					{P3}	RW

This one probably deserves some explanation:

(a) *In the first transaction, P1 needs to read Y. So it sends a message to Y's home (P2), requesting read-access to Y. P2 responds with Y; both P1 and P2 now have Y in read-only mode.*

(b) *Now P1 wants to write Y. Even though it already has Y, it doesn't have write-access, so it requests write access to Y. P2 responds with Y again; now only P1 has Y, and it is in read-write mode.*

(c) *P1 now wants to write Z. So it sends a message to P3 requesting write-access; P3 responds with Z. P1 now has Z in read-write mode.*

(d) *P2 wants to read Z, so it sends a message to Z's home requesting read access. P3 doesn't have a valid copy, so it asks P1 to give P2 a copy, while informing it that P2 will be reading it. P1 sends a copy to P2. Now both P1 and P2 have copies, in read-only mode.*

(e) *P3 wants to write Z, but it knows both P1 and P2 have valid copies. It sends messages to both of them requesting it; both of them respond with Z.*

3. (25 points) In modern operating systems, a process's address space is typically segmented into several pieces. Suppose a particular OS has a code segment, a data segment, a stack segment and a heap segment. Given a 32 bit architecture and assuming a page frame is 4096 bytes (2^{12} bytes), indicate which address bits would be allocated to the segment, page, and offset. If a page table entry requires 4 bytes, what is the size of a process's page table?

   It takes two bits to select a segment; 12 bits have been used for offset on the page. This leaves 32-14=18 bits for the page number. The layout looks like this:

Seg	*Page*	*Offset*
31 30	*29 12*	*11 0*

   The size is $4 \times 2^{18} = 1M$ *per page table (assuming each segment has its own page table), or 4M if there is one large page table for the entire process.*

4. (25 points) Give an example of a page reference string for which an LRU replacement strategy will select a *different* page to eject than the theoretical optimal strategy (you don't need to come up with one that has more page faults, just one that's different). What is the smallest number of page frames required to illustrate the difference?

   We can demonstrate it with just two frames and four references:

References	*0*	*1*	2	*0*
LRU	*0*	*0*	*2*	*2*
		1	*1*	*0*
Optimal	*0*	*0*	*0*	*0*
		1	*2*	*2*

We need the fourth reference so the Optimal strategy can know which page won't be used for the longest time into the future. Two frames are required to demonstrate the difference; two are clearly enough, as we just did it; one frame isn't enough as there is no choice to be made as to which page to swap out.