University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Science

CS 162 I. Stoica
Spring 2011

FIRST MIDTERM EXAMINATION
Wednesday, March 9, 2011

INSTRUCTIONS—READ THEM NOW! This examination is CLOSED BOOK/CLOSED NOTES. There is no need for calculations, and so you will not require a calculator, Palm Pilot, laptop computer, or other calculation aid. Please put them away. You MAY use one 8.5" by 11" double-sided crib sheet, as densely packed with notes, formulas, and diagrams as you wish. The examination has been designed for 80 minutes/80 points (1 point = 1 minute, so pace yourself accordingly). All work should be done on the attached pages.

In general, if something is unclear, write down your assumptions as part of your answer. If your assumptions are reasonable, we will endeavor to grade the question based on them. If necessary, of course, you may raise your hand, and a TA or the instructor will come to you. Please try not to disturb the students taking the examination around you.

We will post solutions to the examination as soon as possible, and will grade the examination as soon as practical, usually within a week. Requests for regrades should be submitted IN WRITING, explaining why you believe your answer was incorrectly graded, within ONE WEEK of the return of the examination in class. We try to be fair, and do realize that mistakes can be made during the regarding process. However, we are not sympathetic to arguments of the form "I got half the problem right, why did I get a quarter of the points?"

_______________________________________ SID: _______________________________
(Signature)

_______________________________________ Discussion Section (Day/Time): ________
(Name—Please Print!)

QUESTION	POINTS ASSIGNED	POINTS OBTAINED
1	10	
2	15	
3	15	
4	20	
5	20	
6	20	
TOTAL	100	

Student Name: ______________________________ SID: ________________

Question 1. *Miscellaneous (10 points)*
For each of the following statements, indicate whether the statement is True or False, and provide a very short explanation of your selection (2 points each).

a. Several threads can share the same address space. **T** F
Rationale:
Yes, two threads in the same process will share the process' address space.

b. Changing the order of semaphores' operations in a program does not matter. T **F**
Rationale:

If a semaphore is initialized to 0 you have to call first a P(); you cannot start with V(). For another example in which you cannot change the order of semaphores see Lec5.13.

c. Paging leads to external fragmentation. **T** **F**
Rationale:
All pages have the same size, so there is no external fragmentation.

d. FIFO scheduling policy achieves lowest average response time for equal size jobs. **T** F
Rationale:
If all jobs have equal size, FIFO is equivalent with SRTF which is optimal discipline for minimizing the response time.

e. LRU exhibits the Belady anomaly. T **F**
Rationale:
LRU guarantees that the subset of pages in a cache of size X is a subset of pages hold by a cache of size X+1 (see Lec11.24).

Student Name: ______________________________________ SID: ______________________

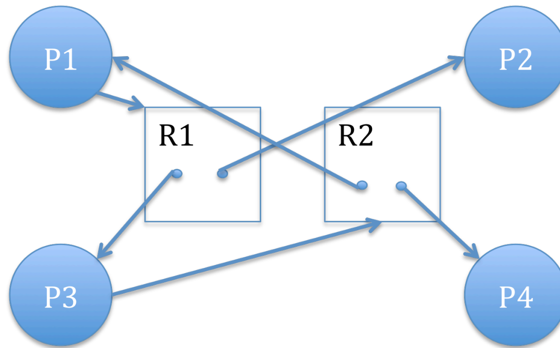
Question 2. *Deadlock (15 points)*

Consider a system with four processes P1, P2, P3, and P4, and two resources, R1, and R2, respectively. Each resource has two instances. Furthermore:

- P1 allocates an instance of R2, and requests an instance of R1;
- P2 allocates an instance of R1, and doesn't need any other resource;
- P3 allocates an instance of R1 and requires an instance of R2;
- P4 allocates an instance of R2, and doesn't need any other resource.

(5 points each question)

(a) Draw the resource allocation graph.



(b) Is there a cycle in the graph? If yes name it.

P2 and P4 are running, P1 is waiting for R1, and P2 is waiting for R2.

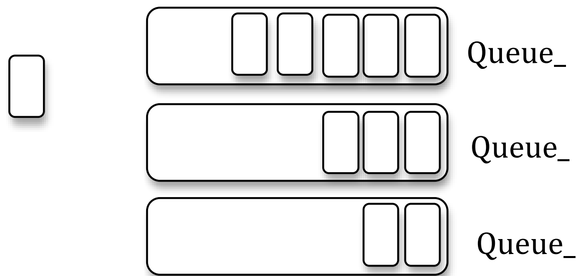
(c) Is the system in deadlock? If yes, explain why. If not, give a possible sequence of executions after which every process completes.

There is a cycle, but no deadlock.

- P2 finishes, release R1;
- P4 finishes, release R2;
- P1 acquires R1, finishes and release R1,R2;
- P3 acquires R2, finishes and release R1,R2;

Student Name: ______________________________________ SID: _____________________

Question 3. *Synchronization (15 points)*



Consider a set of queues as shown in the above figure, and the following code that moves an item from a queue (denoted "source") to another queue (denoted "destination"). Each queue can be both a source and a destination.

```
void AtomicMove (Queue *source, Queue *destination) {
        Item thing; /* thing being transferred */
        if (source == destination) {
                return; // same queue; nothing to move
        }
        source->lock.Acquire();
        destination->lock.Acquire();
        thing = source->Dequeue();
        if (thing != NULL) {
                destination->Enqueue(thing);
        }
        destination->lock.Release();
        source->lock.Release();
}
```

Assume there are *multiple* threads that call AtomicMove() concurrently. (5 points each question)

(a) Give an example involving no more than three queues illustrating a scenario in which AtomicMove() does not work correctly.

If one thread transfers from A to B, and another transfers from B to C and another from C to A, then you can get **deadlock** if they all acquire the lock on the first buffer before any of them acquire the second.

(b) Modify AtomicMove() to work correctly.

One solution to solve the problem is to impose a total order on how locks are acquired/released. The following code uses the source/destination object addresses to impose such an order, i.e., the source/destination object with a lower address acquire the lock first (the modified code is in bold):

```
void AtomicMove (Queue *source, Queue *destination) {
        Item thing; /* thing being transferred */
          if (source == destination) {
                  return; // same queue; nothing to move
           }
           if (source > destination) {
                  source->lock.Acquire();
                  destination->lock.Acquire();
          } else { // destination < source
                  destination->lock.Acquire();
                  source->lock.Acquire();
          }
        thing = source->Dequeue();
        if (thing != NULL) {
                destination->Enqueue(thing);
        }
          if (source > destination) {
                  source->lock.Release();
                  destination->lock.Release();
          } else { // destination < source
                  destination->lock.Release();
                  source->lock.Release();
          }
}
```

(c) Assume now that a queue can be either a source or a destination, but not both. Is AtomicMove() working correctly in this case? Use no more than two sentences to explain why, or why not. If not, give a simple example illustrating a scenario in which AtomicMove() (given at point (a)) does not work correctly.

The code presented at point (a) will work correctly in this case, as it cannot lead to deadlock. This is because AtomicMove() will always acquire the lock of the source, first and the lock of the destination second,

(*Next, we give a "proof"; this proof wasn't required for receiving full score.*) The fact that AtomicMove() always acquires the source lock first guarantees that you cannot end up with a cycle. Indeed, assume this is not the case, i.e., thread T1 holds the lock of queue1 and requests the lock of queue2, T2 holds the lock for queue2, and requests the lock of queue3, …., Tn holds the lock for queuen and waits for the lock of queue1. Since T1 holds the lock of queue1 but not queue2, it follows that queue1 is a source queue, while queue2 is a destination queue. Furthermore, since Tn holds the lock of queuen but not queue1, it follows that queue1 is a destination queue. But queue cannot be at the same time source and destination, which invalidates the hypothesis that the pseudocode can lead to deadlock.

Question 4. *Scheduling (20 points)*

Consider three threads that arrive at the same time and they are enqueued in the ready queue in the order T1, T2, T3.

Thread T1 runs a four-iteration loop, with each iteration taking one time unit. At the end of each iteration, T1 calls yield; as a result, T1 is placed at the end of the ready queue. Threads T2 and T3 both run a two-iteration loop, which each iteration taking three time units. At the end of first iteration, T2 synchronizes with T3, i.e., T2 cannot start the second iteration before T3 finishes the first iteration, and vice versa. While waiting, T2 (T3) is placed in the waiting queue; once T3 (T2) finishes its first iteration, T2 (T3) is placed at the end of the ready queue. Each process exits after finishing its loop.

Assume the system has one CPU. On the timeline below, show how the threads are scheduled using two scheduling policies (FCFS and Round Robin). For each unit of time, indicate the state of the thread by writing "R" if the thread is running, "A" if the thread is in the ready queue, and "W" if the thread is in the waiting queue (e.g., T2 waits for T3 to finish the first iteration, before T2 can run its second iteration).

(a) (6 points) **FCFS (No-preemption)** FCFS always selects the thread at the head of the ready queue. A thread only stops running when it calls yield or waits to synchronize with another thread. What is the average completion time?

T1	R	A	A	A	A	A	A	A	A	A	R	A	A	A	R	R
T2	A	R	R	R	W	W	W	A	A	A	A	R	R	R		
T3	A	A	A	A	R	R	R	R	R	R						

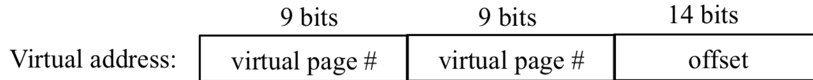
(b) (6 points) **Round Robin (time quantum = 2 units)** When a thread is preempted it is moved at the end of the ready queue. What is average completion time?

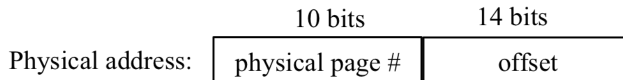
T1	R	A	A	A	A	R	A	A	A	R	A	A	A	A	R	
T2	A	R	R	A	A	A	R	W	A	A	R	R	A	A	A	R
T3	A	A	A	R	R	A	A	R	R	A	A	A	R	R		

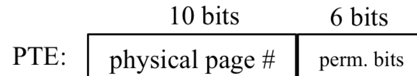
(c) (8 points) Assume there are two processors P1 and P2 in the system. The scheduler follows the policy of FCFS with no preemption. When the scheduler assigns tasks, always assign a task to P1 before assigning to P2. Instead of using "R" to mark running, use "P1" or "P2" to indicate where the task runs. What is the average completion time?

T1	P1	A	A	P1	A	A	A	P1	P1							
T2	P2	P2	P2	W	P2	P2	P2									
T3	A	P1	P1	P1	P1	P1	P1									

Question 5. *Paging (20 points)* Consider a memory architecture using two-level paging for address translation. The format of the virtual address, physical address, and PTE (page table entry) are bellow:







(4 points each question)

(a) What is the size of a page?

2^{14} bytes = 16384 bytes = 16 KB

(b) What is the size of the maximum physical memory?

2^{24} bytes = 16 MB

(c) What is the total memory needed for storing all page tables of a process that uses the entire physical memory?

There are 2^{10} = 1024 physical pages. There is one page table at the first level, and up to 2^9 = 512 page tables at the second level. Since the physical address is 3 bytes, the size of the first level page is 2^9 3 bytes = 1,536 bytes. Furthermore, the PTE is 2bytes, so the size of a second level table is 2^9 2 bytes = 1024 bytes. All in all, the page tables use 1,536 bytes + 512 * 1024 bytes = 528,384 bytes of memory.

(Notes: We gave full credit to answers assuming that the entries at the first level page are 2 bytes, as well. Indeed, the last 9 bits of an address to a page table are typically 0, and don't need to be stored.)

(d) Assume a process that is using 512KB of physical memory. What is the minimum number of page tables used by this process? What is the maximum number of page tables this process might use?

The process uses 512KB / 16 KB = 32 physical pages. Since a second level page can hold up to 512 PTEs, in the best case scenario we use only **2 page tables**: 1st level page + a 2nd level page.

In the worst case, the process may use a little bit of every physical page (e.g., 0.5 KB of each physical page), and all page tables will be populated. Thus, the process ends up using 1 + 512 = **513 page tables**.

(Note: We have also given full credit to people who assumed that the process fully uses each physical page. In this case the answer is 1 + 32 = **33 page tables**.)

(e) Assume that instead of a two-level paging we use an inverted table for address translation. How many entries are in the inverted table of a process using 512KB of physical memory?

The inverted table maintains one entry per physical page. In the worst case, the process uses all physical pages, which yields **1024 entries**. In the best case, the process fully uses each physical page, which yields **32 entries**. (Note: We gave full credit to people who only answered: **32 entries**.)

Question 6. Caches *(20 points)* A tiny system has 1-byte addresses and a 2-way associative cache with four entries. Each block in the cache holds **two** bytes. The cache controller uses the LRU policy for evicting from cache when both rows with the same "index" are full.

(a) () Use the figure below to indicate the number of bits in each field.

6 bits	1 bits	1 bits
cache tag	index	byte select

(b) Assume the following access sequence to the memory: 0xff, 0x22, 0x27, 0x24, 0x27, 0xff, 0xf0, 0x24, 0x27, 0x22. Fill in the following table with the addresses whose content is in the cache. Initially assume the cache is empty. The first entry (i.e., the one corresponding to address 0xff) is filled for you.

		0xff	0x22	0x27	0x24	0x27	0xff	0xf0	0x24	0x27	0x22
Set 0	Index: 0				0x24, 0x25				0x24, 0x25		
	Index: 0							0xf0, 0xf1			
Set 1	Index: 1	0xfe, 0xff		0x26, 0x27		0x26, 0x27				0x26, 0x27	
	Index: 1		0x22, 0x23				0xfe, 0xff				0x22, 0x23

Note: each time a byte is accessed, the entire block to which that block belongs is loaded in memory. For example when byte at address 0xff is read, both that byte and the byte at the address 0xfe are read.

(c) How many cache misses did the access sequence at point (b) cause? What is the hit rate?

7 misses, hit rate = 3/10 = 30%

(d) How many compulsory misses (i.e., misses which could never be avoided) did the access pattern at point (b) cause?

5 (0xff, 0x22, 0x27, 0x24, 0xf0)

(e) Assuming the cache access time is 10ns, and that the miss time is 100ns, what is the average access time assuming the access pattern at point (b)?

10ns * 3/10 + 100ns * 7/10 = 73ns