University of California, Berkeley
College of Engineering
Computer Science Division – EECS

Fall 2012 Ion Stoica

Midterm Exam

October 15, 2012
CS162 Operating Systems

Your Name:	
SID AND 162 Login:	
TA Name:	
Discussion Section Time:	

General Information:
This is a **closed book and one 2-sided handwritten note** examination. You have 80 minutes to answer as many questions as possible. The number in parentheses at the beginning of each question indicates the number of points for that question. You should read **all** of the questions before starting the exam, as some of the questions are substantially more time consuming.

Write all of your answers directly on this paper. ***Make your answers as concise as possible.*** If there is something in a question that you believe is open to interpretation, then please ask us about it!

Good Luck!!

QUESTION	POINTS ASSIGNED	POINTS OBTAINED
1	24	
2	14	
3	12	
4	24	
5	12	
6	14	
TOTAL	100	

1. (24 points total) True/False and Why? **CIRCLE YOUR ANSWER**. For each question: 1 point for true/false correct, 2 point for explanation. An explanation cannot exceed 2 sentences.

   a) Each thread has its own stack.

   TRUE FALSE

   Why?

   b) Starvation implies deadlock.

   TRUE FALSE

   Why?

   c) It's generally possible to substitute a semaphore for a condition variable, because *sem.V*()*/sem.P*() have similar semantics to *cond.signal*()/*cond.wait****()***.

   TRUE FALSE

   Why?

   d) Shortest Run Time First (SRTF) is the "optimal" scheduling algorithm, but it is generally not implemented directly, due to excessive context switching overhead.

   TRUE FALSE

   Why?

e) Using a smaller page size increases the size of the page table.

TRUE FALSE

Why?

f) Moving from a single level page table to a two-level page table will always decrease the memory footprint (in aggregate) used by the page table.

TRUE FALSE

Why?

g) Unlike paging, segmentation doesn't prevent processes from accessing physical memory not allocated to them.

TRUE FALSE

Why?

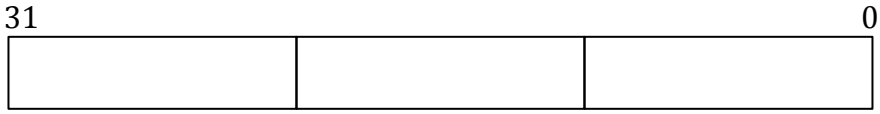
h) If you increase the size of a the page cache from 8Kb to 16Kb, and you are running a "Perfect LRU" page replacement strategy, the cache hit ratio will never get worse.

TRUE FALSE

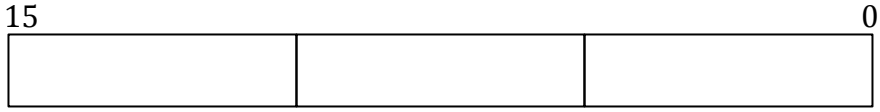
Why?

2. (14 points) **Memory hierarch:** You are responsible for designing the memory system for a byte addressable system. The virtual memory address space is 32 bits and the physical memory address space is 16 bits.

   a) (4 points) Assume the system uses a two level page table to translate a virtual address to a physical address. Show the format of the virtual address, specify the page size (pick one size if multiple sizes are feasible), and specify the length of each field in the virtual address. Make sure that each translation table fits in a page.



   b) (4 points) Assume you add to your system a 4-way set-associative **data cache** with 16 cache blocks. Each block in the cache holds 8 bytes of data. In order to address a specific byte of data, you will have to split the address into the cache tag, cache index and byte select. Which parts of the address would you associate with each component, how long will each component be (in bits) and why? (Not: Assume there are no modifiers bits in the table.)



c) (3 points) The main memory access time is 100ns, and the cache lookup time is 50ns. Assuming a cache hit rate of 90%, what is the average time to read a location from main memory? (Note: Assume the cache hit rate is the same for the data and the page translation tables.)

d) (3 points) To speed up the address translation process we introduce a TLB that has an access time of 20ns. Assuming the TLB hit rate is 95%, what is the average access time for a memory operation?

3. (12 points) **Synchronization:** A common parallel programming pattern is to perform processing in a sequence of parallel stages: all threads work independently during each stage, but they must synchronize at the end of each stage at a synchronization point called a ***barrier***. If a thread reaches the barrier before all other threads have arrived, it waits. When all threads reach the barrier, they are notified and can begin the execution on the next phase of the computation.

Example:

```
while (true) {
        Compute stuff;
        BARRIER();
        Read other threads results;
}
```

a) (4 points) The following implementation of Barrier is incomplete and has two lines missing. Fill in the missing lines so that the Barrier works according to the prior specifications.

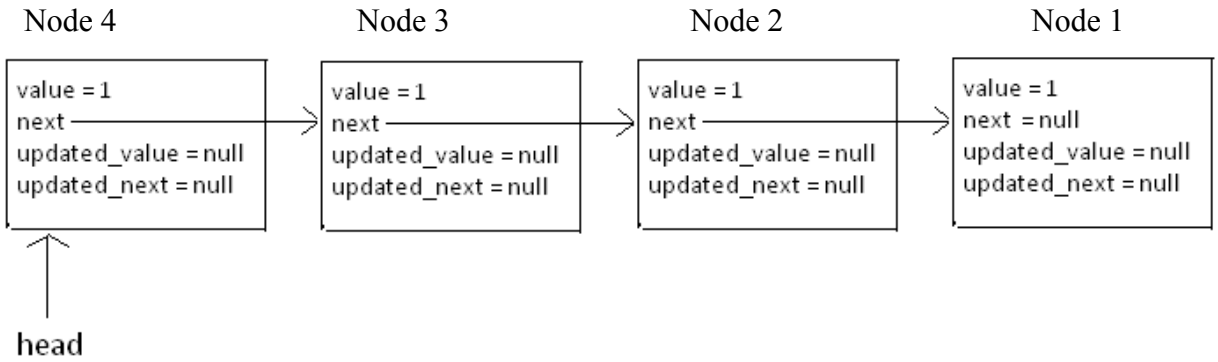
```
class Barrier() {
     int numWaiting = 0;        // Initially, no one at barrier
     int numExpected = 0;       // Initially, no one expected
     Lock L = new Lock();
     ConditionVar CV = new ConditionVar();

     void threadCreated() {
          L.acquire();
          numExpected++;
          L.release();
     }
     void enterBarrier() {
          L.acquire();
          numWaiting++;
          if (numExpected == numWaiting) { // If we are the last
               numWaiting = 0;      // Reset barrier and wake threads

               ____________________// Fill me in
          } else {    // Else, put me to sleep

               ____________________// Fill me in
          }
          L.release() ;
     }
}
```

b) (5 points) Now, let us use ***Barrier*** in a parallel algorithm. Consider the linked list below:



In our parallel algorithm, there are four threads (Thread 1, Thread 2, Thread 3, Thread 4). Each thread has its own instance variable ***node***, and all threads share the class variable ***barrier***. Initially, Thread 1's ***node*** references Node 1, Thread 2's ***node*** references Node 2, Thread 3's ***node*** references Node 3, and Thread 4's ***node*** references Node 4.

In the initialization steps, **barrier.threadCreated()** is called once for each thread created, so we have **barrier.numExpected == 4** as a starting condition.

Once all four threads are initialized, each thread calls its **run()** method. The **run()** method is identical for all threads:

```
void run() {
     boolean should_print = true;
     while (true) {
          if (node.next != null) {
               node.updated_value = node.value +
                                    node.next.value;
               node.updated_next = node.next.next;
          } else if (should_print) {
                System.out.println(node.value);
                should_print = false;
          }
          barrier.enterBarrier();
          node.value = node.updated_value;
          node.next = node.updated_next;
          barrier.enterBarrier();
     }
}
```

List all the values that are printed to stdout along with the thread that prints each value. For example, "thread 1 prints 777".

c) (3 points) In an attempt to speed-up the parallel algorithm from the previous part (2c), you notice that the line **barrier.enterBarrier()** occurs twice in **run()**'s while loop. Can one of these two calls to **barrier.enterBarrier()** be removed while guaranteeing that the output of the previous part (2c) remains unchanged? If your answer is "yes", specify whether you would remove the first or second occurrence of **barrier.enterBarrier()**.

4. (24 points total) **CPU scheduling.** Consider the following **single-threaded** processes, arrival times, and CPU processing requirements:

Process ID (PID)	Arrival Time	Processing Time
1	0	6
2	2	4
3	3	5
4	6	2

a) (12 points): For each scheduling algorithm, fill in the table with the ID of the process that is running on the CPU. Each row corresponds to a time unit.
- For time slice-based algorithms, assume one unit time slice.
- When a process arrives it is immediately eligible for scheduling, e.g., process 2 that arrives at time 2 can be scheduled during time unit 2.
- If a process is preempted, it is added at the tail of the ready queue.

Time	FIFO	RR	SJF
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			

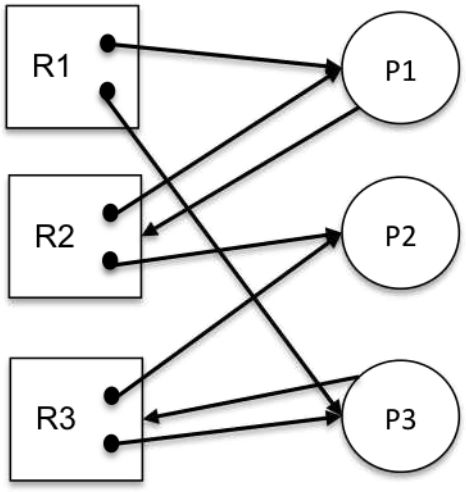
b) (6 points): Calculate the response times of individual processes for each of the scheduling algorithms. The response time is defined as the time a process takes to complete after it arrives.

	PID 1	PID 2	PID 3	PID 4
FIFO				
RR				
SJF				

c) (6 points) Consider same processes and arrival times, but assume now a processor with **two** CPUs. Assume CPU 0 is busy for the first two time units. For each scheduling algorithm, fill in the table with the ID of the process that is running on each CPU.
   - For any non-time slice-based algorithm, assume that once a process starts running on a CPU, it keeps running on the same CPU till the end.
   - If both CPUs are free, assume CPU 0 is allocated first.

Time	CPU #	FIFO	RR	SJF
0	**0**			
	1			
1	**0**			
	1			
2	**0**			
	1			
3	**0**			
	1			
4	**0**			
	1			
5	**0**			
	1			
6	**0**			
	1			
7	**0**			
	1			
8	**0**			
	1			
9	**0**			
	1			
10	**0**			
	1			

5. (12 points) **Deadlock**: Consider the following resource allocation graph:



a) (3 points) Does the above allocation graph contain a deadlock? Explain your answer using no more than ***two*** sentences.

b) (3 points) Assume now that P2 also demands resource R1. Does this allocation graph contain a deadlock? Explain your answer using no more than ***two*** sentences.

c) (3 points) Assume the allocation graph at point b), and, in addition, assume that R2 has now three instances. Does this allocation graph contain a deadlock? Explain your answer using no more than ***two*** sentences.

d) (3 points) Add to the original allocation graph an additional process P4 that demands an instance of R1. Does the allocation graph contain a deadlock? Explain your answer using no more than ***two*** sentences.

6. (14 points) **Caching**: Consider a memory consisting of four pages (frames), and consider the following reference stream of virtual pages A, B, C, D, E, C, A, B, C, D, F.

   a) (4 points) Consider the LRU page replacement algorithm. Fill in the following table showing all page faults. What is the number of page faults?

Ref Page	A	B	C	D	E	C	A	B	C	D	F
1											
2											
3											
4											

   b) (4 points) Consider now the MIN page replacement algorithm. Assume the reference stream continues with virtual pages A, C, B, F (i.e., the entire reference stream is A, B, C, D, E, C, A, B, C, D, F, *A, C, B, F*). Fill in the following table showing all page faults. How many page faults are there?

Ref Page	A	B	C	D	E	C	A	B	C	D	F
1											
2											
3											
4											

c) (3 points) Consider again the LRU replacement policy, and the original reference stream. What is the minimum memory size (in pages) such that the number of faults to be no larger than 6? Explain.

d) (3 points) Replace a **single** reference in the original reference stream (e.g., change the third reference from C to A) such that to reduce the number of page faults by **two** when using LRU. Show the resulting reference stream and the corresponding fault in the following table:

Ref Page											
1											
2											
3											
4											