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

CS 162
Spring 2010

I. Stoica

FIRST MIDTERM EXAMINATION
Tuesday, March 9, 2010

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 100 minutes/100 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	20	
2	20	
3	20	
4	20	
5	20	
6 (bonus question)	5	
TOTAL	100 (+5)	

Question 1. (20 points)

True or False *(12 points). Instructions: write "true" or "false" after each of the following statements (Please give one statement justification for each of your answers).*

a. (2 points) Each physical page belongs to only one process.

   False, pages can be shared between processes.

b. (2 points) Every interrupt changes the CPU from user mode to kernel mode.

   False, interrupts can happen when the processor is in kernel mode, in which case no switch to kernel mode is made.

c. (2 points) All Page Table entries will be invalid directly after a context switch between processes.

   False, since the kernel maintains a separate page table for each process, the only pages that should have been invalidated since the last time a process was running are those that were evicted (assuming the OS is using a global eviction policy) while another process was running. It could be that case that the context switch away from, and eventually back to this process does not lead to any more additional invalid page table entries.

d. (2 points) The CPU scheduling policy that minimizes average completion time can lead to starvation.

   True, Shortest Remaining Time First (SRTF) is the policy in question, and can lead to starvation (if very short jobs continuously arrive, jobs with a lot of time remaining will never be run, i.e. starve)

e. (2 points) If there is only one CPU in the computer, then at most one process can be in the "running" state at any time.

   True, assuming that the CPU is not multi-core or hyper-threaded. If you assumed that the CPU ***was*** *either of those things (and stated your assumption!) and said false, you were correct and given credit as well.*

f. (2 points) The "Shortest-Job-First" policy always results in the lowest average completion time in a batch system.

   False, batch systems still support preemption, thus Shortest Remaining Time First (SRTF) would achieve a lower average completion time. If you stated that you assumed batch system meant no preemption, you received full credit.

Multiple Choice, fill in the blank, short answer *(8 points). Instructions: Circle the correct option(s), or fill in the blanks below.*

a. (2 points) Page Tables can be stored (circle all answers that apply):

   [✓ a. In physical memory]

   [✓ b. In the L2 Cache]

   [✓ c. On the hard drive]

b. (2 points) Condition variables support the following 3 operations:

   a. __________sleep()/wait()____________

   b. _________notify()/wake()____________

   c. __notifyAll()/wakeAll()/broadcast()__

c. (2 points) Of the following items, circle those that are stored in the *thread* control block.

   [✓ a. CPU registers]

   b. page table pointer

   [✓ c. stack pointer]

   d. ready list

   e. segment table

   [✓ f. thread priority]

   [✓ g. program counter]

d. (2 points) Processes (or threads) can be in one of three states: **Running**, **Ready**, or **Waiting**. For each of the following examples, write down which state the process (or thread) is in:

   a. Spin-waiting for a variable y to become zero. ____Running_____

   b. Having just completed an I/O, waiting to get scheduled again on the CPU. ______Ready______

   c. Blocking on a condition variable for some other thread to signal it. ______Waiting_______

Question 2. *Threads and Synchronization (20 points)*

(a) (4 points) Give two reasons of why disabling interrupts can be a bad way to implement critical sections.

- *Code might have a bug, such as an infinite loop, deadlock, priority inversion, etc, which will lock up the system.*

- Code might have an error, such as a seg fault, divide-by-zero, etc, which will be ignored.

- Turning off interrupts on one processor is not enough in a multi-processor system, turning off interrupts across all processors in a multi-processor system is either impossible, difficult, or incredibly inefficient.

- If the critical section is long it will not be able to respond to events/interrupts for the rest of the system.

(b) (6 points) Assume you are given the CompareAndSwap atomic primitive defined (in C) as follows:

```
bool CompareAndSwap(int* value, int from, int to) {
  if (*value == from) {
    *value = to;
    return true;
  }
  return false;
}
```

Consider the following implementation of lock acquire using CompareAndSwap:

```
void acquire() {
  while(!CompareAndSwap(&value, UNLOCKED, LOCKED));
}
```

i) Give one reason why this implementation may not be efficient.

Spin-waiting/busy-waiting, wasting CPU resources instead of sleeping/yielding.

ii) Without preemption, is this still a valid implementation of acquire()? Why? (Use no more than two sentences for your explanation.)

No: if a thread holding a lock yields then another thread that attempts to acquire that lock will spin in an infinite loop causing a deadlock (without preemption the CPU becomes a resource you need to consider for deadlock). Note that even on a multi-processor machine, a deadlock can still occur (it depends on what the threads are doing and how many of them you have!).

(c) (8 points) Complete the following implementation of the P() and V() functions for a semaphore using condition variables using Mesa semantics.

```
void P()  {
    lock.acquire();
    while(value == 0)
      cond.sleep();
    value -= 1;
    lock.release();
}


void V()  {
    lock.acquire();
    value += 1;
    cond.wake();
    lock.release();
}
```

(d) (2 points) How could you simplify your implementation if you were given condition variables that follow Hoare semantics instead of Mesa semantics? Use no more than two sentences to explain your answer.

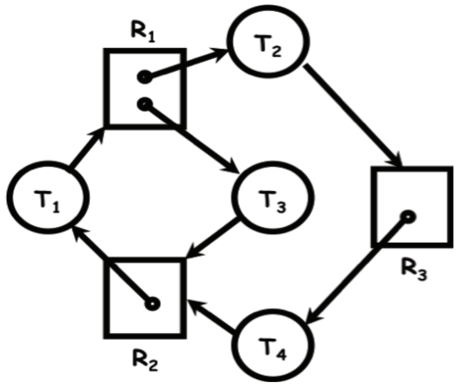
Rather than using a 'while' loop in P(), you could use an 'if' statement because Hoare semantics blocks the thread calling 'wake' and passes control immediately to the waiting thread so the invariant/condition is guaranteed to be valid.

Question 3. *Deadlock (20 points)*

(a) (4 points) Name the four conditions under which deadlock occurs.

1) Mutual exclusion
2) Hold and wait
3) No preemption
4) Circular wait

(b) Consider the resource allocation graph below consisting of four tasks (T1, T2, T3, and T4) and three resources (R1, R2, and R3). R1 has two instances while the other two resources have only one instance each. An arrow from a task to a resource denotes the fact that the task requests one instance of the resource, while an arrow from a resource to a task denotes the fact that the task holds an instance of the resource. Assume that resources cannot be preempted and tasks do not voluntary release resources before terminating.



i) (4 points) Are the tasks shown in the above resource graph deadlocked? If yes, explain how each of the four deadlock conditions is satisfied. (Use one sentence per condition.) If not, give a possible sequence in which the tasks execute.

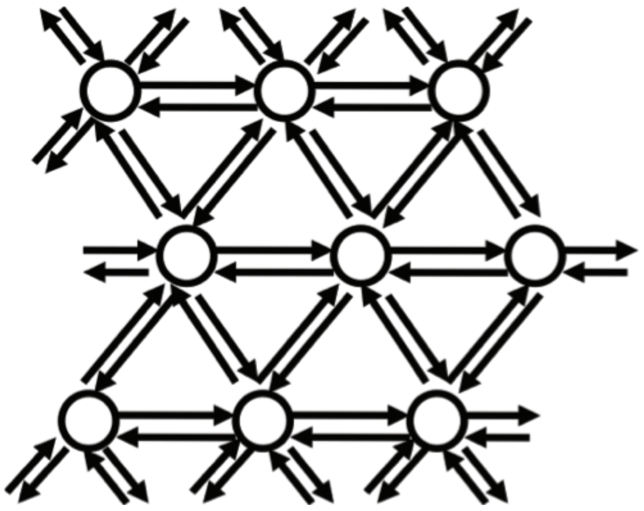
Tasks are in deadlock:
1) *Mutual exclusion: an instance can be used by only one task;*
2) *Hold & wait: a task doesn't release the resource before terminating (from problem formulation);*
3) *No preemption: a task cannot be preempted (from problem formulation);*
4) *There is a circular waiting pattern: T1→R1→T3→R2→T1; another cycle is T1→R1→T2→R3→T4→R2→T1.*

ii) (4 points) Assume you are allowed to add an extra instance of either resource R1 or R2. Are the tasks deadlocked in this case? If yes, explain how each of the four deadlock conditions is satisfied. (Use one sentence per condition.) If not, give a possible sequence in each the tasks execute.

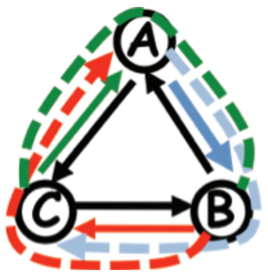
No deadlock. Assume R2 has an extra instance.

- *T3 acquires the extra instance of R2 and finishes, then release R1 and R2 instances;*
- *T1 acquires an R1 instance and finishes, then releases R1 and R2 instance;*
- *T4 acquires an R2 instance and finishes, then releases R2 and R3 instances;*
- *T2 acquires the R3 instance and finishes;*

(c) Consider the network below implementing wormhole routing between any two nodes. Each node is connected to six of its neighbors by two network links, one in each direction. Messages are routed from a source node to a destination node and can stretch through the network (i.e. consume links along the route from source to destination). Messages can cross inside nodes. Assume that no network link can service more than one message at a time, and that each message must consume a continuous set of channels (like a snake).
Hint: You can think about a message as a train that spans multiple links (possible all the way from source to destination), and of each link as a one-way rail.



i) (4 points) Show a situation (with a drawing) in which messages are deadlocked and can make no further progress. Explain how each of the four conditions of deadlock is satisfied by your example.



A sends message to C through B, holds A→B and waits for B→C;
B sends message to A through C, holds B→C and waits for C→A;
C sends message to B through A, holds C→A and waits for A→B;
Deadlock conditions:

1) *Mutual exclusion: a link can handle only one message (problem formulation);*
2) *Hold & wait: each message holds a link and waits for another above (see above example);*
3) *No preemption: message cannot be preempted (this would be equivalent to dropping message);*
4) *There is a circular waiting pattern: (see description of above example);*

ii) (4 points) Define a routing policy that avoids deadlocks in the network of (i). Explain using at most two sentences why your routing policy avoids deadlock.

Dimension reordering; first route east – west, then on diagonal northwest – southwest. Once a message starts on a diagonal it can never get back on the horizontal line (east – west) before reaching the destination.

Note: The following answers did not receive full credit:

- *"Send one packet at a time". This solution is not only inefficient, but it is also incorrect, as even a single message can deadlock itself by forming a cycle.*
- *"Preempt one message". This means to drop a message. Also, this does not guarantee that the next time the message is sent it won't get again deadlocked.*

Question 4. *CPU Scheduling (20 points)*
Your boss asks you to pick a scheduling algorithm to run the following processes:

- P1, which requires 1 hour of CPU time to complete;
- P2, which requires 2 hours of CPU time to complete;
- P3, which requires 1 min of CPU time to complete;

Assume all processes are ready at the same time.

a) (5 points) Assume you use First Come First Serve (FCFS) to schedule these processes. Which is the schedule with the largest average completion time? Which is the schedule with the smallest average completion times? Compute the average completion time in each case.

Largest completion time: P2, P1, P3

- *P2 terminates at t=2h = 120min*
- *P1 terminates at t = 3h = 180min*
- *P3 terminates at t = 3h and 1min = 181min*

Average completion time = (120min + 180min + 181min)/3 = 160min and 20sec.

Smallest completion time: P3, P1, P2

- *P2 terminates at t = 1min*
- *P1 terminates at t = 61min*
- *P3 terminates at t = 181min*

Average completion time = (1 min + 61 min + 181 min)/3 = 81min.

b) (5 points) Assume you use Round-Robin with a time quanta (slice) of 100ms. What is the completion time of each job in this case? Assume that there is no context switch overhead.

During first 3min each process runs for 1min. Process P3 finishes either after 3min, 2min 59sec and 900 ms, or 2min 59 and 800ms, depending on whether in the last round P3 is scheduled the last, the second or the first, respectively.

*For the next 118min only P1 and P2 run. P1 will finish either after 118+3 = **121** min, or after 120min and 900ms, depending on whether P1 is scheduled last or first in the last round.*

*Finally, P2 will continue to run for another 1h and finish at 121min+60min = **181**min.*

c) (5 points) Now assume that P1 arrives at t=0, P2 arrives at time t=10min, and P3 at time t=20min. What will be the completion time of each process when using the Shortest Remaining Time First (SRTF)?

P1 runs until time t=10min
At t=10min, P2 arrives, but P1 will continue running since it has a lower remaining time.
At t=20min, P3 arrives and it starts running immediately, by preempting P1 as it has the lowest running time, i.e., 1min
At t=21min, P3 will finish, and P1 will start running again.
At t=61min, P1 will finish, and P2 will start running.
At t=181min, P2 will finish.

Thus the completion times are:

- *P1: 61-0=61min;*
- *P2: 181-10=171min*
- *P3: 21-20=1min;*

d) (5 points) Assume now that P3 is performing an I/O operation every 10ms and that the I/O operation takes 40ms, but would still take 1 minute to run if it was the only process running. Assume that P1, P2, and P3 are submitted at the same time and you use Round-Robin scheduling (like in part (b)).

i) How long does it take P3 to complete if the time slice is 100ms?

P3 will run for 10ms, after which it waits for the I/O to complete. P1 and P2 will run for the next 200ms, each using a time slice. After that P3 will run again for 10ms and wait for the I/O to complete. Thus, out of 210ms, P3 runs for 10ms.

*Recall that P3 needs 1min of CPU to complete. This means that it will take P3 60sec*210/10 = 21min to complete.*

Other answers that received full credit:

- *P3 completion time: 4min and 12sec (this assumes that the total running time of P3 is 1min, out of which 12sec CPU)*
- *P1 completion time: 121min (for people who solved the problem for P1, instead of P3, due to the typo in this question).*
- *P1 completion time: 121min and 12sec (this assumes that the total running time of P3 is 1min, out of which 12sec CPU)*

ii) What time slice length would minimize the completion time of P3?

20ms. After P3 releases the CPU waiting for I/O operation to complete, P1 and P2 will use their time slices and finish exactly when P3 is ready to execute.

Other answer that receive full credit:

- *1h if P1 starts first (for people who solved the problem for P1, instead of P3, due to the typo in this question).*

Question 5. *Page Tables & TLBs (20 points)*

Orange Inc hires you to design the virtual memory system for a new cell phone with 32-bit virtual and physical addresses, in which memory is allocated in 2 KB pages. Suppose that you decide to use a single-level page table, in which you also store three metadata bits for each page: Writable, Executable and Valid.

(a) (4 points) Answer the following questions, briefly explaining your solution:

i) How long, in bits, is a virtual page number?

21 bits: a page is 2 KB = 2^{11} bytes, so an offset is 11 bits, so a page nr. is 32-11=21 bits.

ii) How long, in bits, is a physical page number?

21 bits: same as a virtual page number.

iii) How long, in bits, is an offset within a page?

11 bits, since a page is 2 KB = 2^{11} bytes.

iv) How much memory is needed to store the page table of each process?

Each page table entry contains a 21-bit physical page number plus 3 metadata bits, for a total size of 24 bits = 3 bytes. There are 2^{21} pages in each virtual address space. Therefore, the total size of the page table is $2^{21} \times 3$ bytes = 6 MB.

(b) (4 points) Your manager asks you to consider using a multi-level page table in your design. Explain one advantage and one disadvantage of multi-level page tables over single-level page tables. (Use no more than four sentences in total.)

Advantage: *When the address space is sparsely filled, a multi-level page table consumes less memory than a single-level one, because index pages at levels beyond the first level don't need to be present for parts of the address space that aren't used. This is the primary motivation for multi-level page tables. We also gave partial credit for less clear-cut but plausible answers, such as saying that multi-level page tables make it easier to share large portions of address spaces between processes.*

Disadvantage: *Multiple memory accesses are needed for a lookup in a multi-level page table, decreasing performance on TLB misses. We gave partial credit for saying that multi-level page tables are more complex but not mentioning performance as a disadvantage.*

(c) (12 points) In this question, you are asked to predict the results of a sequence of memory accesses on a machine with a single-level page table and a Translation Lookaside Buffer (TLB). Suppose that the machine has 20-bit virtual and physical addresses and 256-byte pages, and that the TLB is fully associative and holds 3 entries. The initial contents of the TLB and page table are shown on the next page. (For the page table, we show only a subset of the entries; assume that the ones not shown are not valid.)

Initial TLB:

Virtual Page #	Physical Page #	Writable?	Valid?
0x100	0x200	0	1
0x101	0x300	0	0
0x200	0x320	1	1

Page Table:

Virtual Page #	Physical Page #	Writable?	Valid?
0x100	0x200	0	1
0x101	0x100	0	1
0x200	0x320	1	1
0x201	0x321	1	0
0xFFF	0x100	1	1

For each memory access in the sequence on the *next page*, write whether the TLB is hit, whether the access succeeds, and, if so, which physical address is accessed. Also show the state of the TLB after each access, assuming that TLB entries are replaced using a Least Recently Used (LRU) policy. Assume that the TLB is not updated if an invalid page is accessed.

Instruction	TLB Hit?	Success?	Physical Address	New TLB State			
				Virtual Page #	Physical Page #	Writable?	Valid?
LOAD 0x20012	*Y*	*Y*	*0x32012*	*0x100*	*0x200*	*0*	*1*
				0x101	*0x300*	*0*	*0*
				0x200	*0x320*	*1*	*1*
STORE 0x10001	*Y*	*N*	*0x20001 (also okay to omit this)*	*0x100*	*0x200*	*0*	*1*
				0x101	*0x300*	*0*	*0*
				0x200	*0x320*	*1*	*1*
LOAD 0x10101	*N*	*Y*	*0x10001*	*0x100*	*0x200*	*0*	*1*
				0x101	***0x100***	***0***	***1***
				0x200	*0x320*	*1*	*1*
STORE 0xFFFFF	*N*	*Y*	*0x100FF*	*0x100*	*0x200*	*0*	*1*
				0x101	*0x100*	*0*	*1*
				0xFFF	***0x100***	***1***	***1***
STORE 0x20009	*N*	*Y*	*0x32009*	***0x200***	***0x320***	***1***	***1***
				0x101	*0x100*	*0*	*1*
				0xFFF	*0x100*	*1*	*1*
STORE 0x32000	*N*	*N*	*N/A (page fault reported to kernel)*	*0x200*	*0x320*	*1*	*1*
				0x101	*0x100*	*0*	*1*
				0xFFF	*0x100*	*1*	*1*

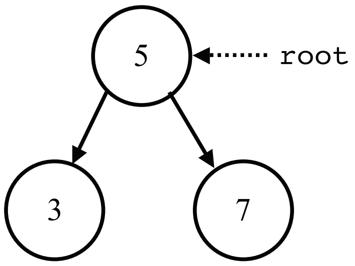
In grading this question, we gave 0.5 points for each correctly filled table cell (the TLB counting as one cell). We did not penalize students repeatedly for a mistake on one row that is propagated onto the next row in a consistent manner.

Student Name: ____ SID: ______________________

Question 6 (BONUS). Thread Interleavings (5 points)

The following Java code implements insertion into a binary tree:

```
void insert(Node node, int value) {
  if (value <= node.value) {
    if (node.left == null) {
      node.left = new Node(value);
    } else {
      insert(node.left, value);
    }
  } else {
    if (node.right == null) {
      node.right = new Node(value);
    } else {
      insert(node.right, value);
    }
  }
}
```



Unfortunately, this implementation is not thread-safe. Suppose that we start with a binary tree containing the values 3, 5, and 7, organized as shown above, and that we have a reference `root` to the root node (the one containing 5). We then launch the two threads below, each of which tries to insert two values into the binary tree. What are the possible outcomes after both threads finish? Draw each possible resulting tree in a format like the starting tree above. (You may continue on the back of this page if necessary.)

Thread 1	**Thread 2**

`insert(root, 0);`
`insert(root, 2);`
	`insert(root, 1);`
	`insert(root, 6);`

Seven trees are possible: