

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

Fall 2008

John Kubiawicz

Midterm I
October 15th, 2008
CS162: Operating Systems and Systems Programming

Your Name:	
SID Number:	
Discussion Section:	

General Information:

This is a **closed book** exam. You are allowed 1 page of **hand-written** notes (both sides). You have 3 hours to complete as much of the exam as possible. Make sure to read all of the questions first, 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.* On programming questions, we will be looking for performance as well as correctness, so think through your answers carefully. If there is something about the questions that you believe is open to interpretation, please ask us about it!

Problem	Possible	Score
1	24	
2	12	
3	25	
4	21	
5	18	
Total	100	

[This page left for π]

3.141592653589793238462643383279502884197169399375105820974944

Problem 1: Short Answer [24pts]

Problem 1a[2pts]: Give at least two reasons why the following implementation of a condition variable is incorrect (assume that MySemi is a semaphore initialized to 0):

```
Wait()   { MySemi.P(); }  
Signal() { MySemi.V(); }
```

Problem 1b[4pts]: What is the difference between Mesa and Hoare scheduling for monitors? Include passing of locks between signaler and signalee, scheduling of CPU resources, and impact on programmer.

Problem 1c[3pts]: The SRTF algorithm requires knowledge of the future. Why is that? Name two ways to approximate the information required to implement this algorithm.

Problem 1d[3pt]: What is priority donation? What sort of information must the OS track to allow it to perform priority donation?

<pre> Reader() { //First check self into system lock.acquire(); while ((AW + WW) > 0) { WR++; okToRead.wait(&lock); WR--; } AR++; lock.release(); // Perform actual read-only access AccessDatabase(ReadOnly); // Now, check out of system lock.acquire(); AR--; if (AR == 0 && WW > 0) okToWrite.signal(); lock.release(); } </pre>	<pre> Writer() { // First check self into system lock.acquire(); while ((AW + AR) > 0) { WW++; okToWrite.wait(&lock); WW--; } AW++; lock.release(); // Perform actual read/write access AccessDatabase(ReadWrite); // Now, check out of system lock.acquire(); AW--; if (WW > 0){ okToWrite.signal(); } else if (WR > 0) { okToRead.broadcast(); } lock.release(); } </pre>
--	--

Problem 1e[3pts]: Above, we show the Readers-Writers example given in class. It used two condition variables, one for waiting readers and one for waiting writers. Suppose that *all* of the following requests arrive in very short order (while R_1 and R_2 are still executing):

Incoming stream: $R_1 R_2 W_1 W_2 R_3 R_4 R_5 W_3 R_6 W_4 W_5 R_7 R_8 W_6 R_9$

In what order would the above code process the above requests? If you have a group of requests that are equivalent (unordered), indicate this clearly by surrounding them with braces '{ }'. You can assume that the wait queues for condition variables are FIFO in nature (i.e. `signal()` wakes up the oldest thread on the queue). Explain how you got your answer.

Problem 1f[4pts]:

Suppose that you were to redesign the code in (1e). What is the *minimum* number of condition variables that we would need in order to handle the above requests in an order that guarantees that a read always returns the results of writes that have *arrived* before it but not *after* it? (Another way to say this is that the reads and writes occur in the order in which they arrive, while still allowing groups of reads that arrive together to occur simultaneously.) Provide a two or three sentence sketch of your scheme (do not try to write code!).

Problem 1g[3pts]: What are exceptions? Name two different types of exceptions and give an example of each type:

Problem 1h[2pts]: What was the problem with the Therac-25? Your answer should involve one of the topics of the class.

Problem 2: TRUE/FALSE [12 pts]

In the following, it is important that you *EXPLAIN* your answer in **TWO SENTENCES OR LESS** (Answers longer than this may not get credit!). Also, answers without an explanation *GET NO CREDIT*.

Problem 2a[2pts]: The kernel on a multiprocessor can use the local disabling of interrupts (within one CPU) to produce critical sections between the OSs on different CPUs.

True / False

Explain:

Problem 2b[2pts]: When designing a multithreaded application, you must use synchronization primitives to make sure that the threads do not overwrite each other's registers.

True / False

Explain:

Problem 2c[2pts]: A system that provides segmentation without paging can fragment the physical address space, forcing the operating system to waste physical memory.

True / False

Explain:

Problem 2d[2pts]: A user-level library implements each system call by first executing a "transition to kernel mode" instruction. The library routine then calls an appropriate subroutine in the kernel.

True / False

Explain:

Problem 2e[2pts]: The difference between processes and threads is purely historical.

True / False

Explain:

Problem 2f[2pts]: Round robin scheduling provides a latency improvement over FCFS scheduling for interactive jobs.

True / False

Explain:

[This page intentionally left blank]

Problem 3: Atomic Synchronization Primitives [25 pts]

In class, we discussed a number of *atomic* hardware primitives that are available on modern architectures. In particular, we discussed “test and set” (TSET), SWAP, and “compare and swap” (CAS). They can be defined as follows (let “*expr*” be an expression, “&*addr*” be an address of a memory location, and “*M[addr]*” be the actual memory location at address *addr*):

Test and Set (TSET)	Atomic Swap (SWAP)	Compare and Swap (CAS)
<pre>TSET(&addr) { int result = M[addr]; M[addr] = 1; return (result); }</pre>	<pre>SWAP(&addr, expr) { int result = M[addr]; M[addr] = expr; return (result); }</pre>	<pre>CAS(&addr, expr1, expr2) { if (M[addr] == expr1) { M[addr] = expr2; return true; } else { return false; } }</pre>

Both TSET and SWAP return values (from memory), whereas CAS returns either true or false. Note that our &*addr* notation is similar to a reference in c++, and means that the &*addr* argument must be something that can be stored into (an “lvalue”). For instance, TSET could be used to implement a spin-lock acquire as follows:

```
int lock = 0; // lock is free

// Later: acquire lock
while (TSET(lock));
```

CAS is general enough as an atomic operation that it can be used to implement both TSET and SWAP. For instance, consider the following implementation of TSET with CAS:

```
TSET(&addr) {
  int temp;
  do {
    temp = M[addr];
  } while (!CAS(addr,temp,1));
  return temp;
}
```

Problem 3a[3pts]:

Show how to implement a spinlock acquire with a single while loop using CAS instead of TSET. You must only fill in the arguments to CAS below:

```
// Initialization
int lock = 0; // Lock is free

// acquire lock

while ( !CAS( , , ) );
```


Problem 3b[2pts]:

Show how SWAP can be implemented using CAS. Don't forget the return value.

```
SWAP(&addr, reg1) {
```

```
}
```

Problem 3c[3pts]:

With spinlocks, threads spin in a loop (busy waiting) until the lock is freed. In class we argued that spinlocks were a bad idea because they can waste a lot of processor cycles. The alternative is to put a waiting process to sleep while it is waiting for the lock (using a blocking lock). Contrary to what we implied in class, there are cases in which spinlocks would be more efficient than blocking locks. Give a circumstance in which this is true and explain why a spinlock is more efficient.

An object such as a queue is considered “lock-free” if multiple processes can operate on this object simultaneously without requiring the use of locks, busy-waiting, or sleeping. In this problem, we are going to construct a lock-free FIFO queue using the atomic CAS operation. This queue needs both an Enqueue and Dequeue method.

We are going to do this in a slightly different way than normally. Rather than Head and Tail pointers, we are going to have “PrevHead” and Tail pointers. PrevHead will point at the last object returned from the queue. Thus, we can find the head of the queue (for dequeuing). If we don’t have to worry about simultaneous Enqueue or Dequeue operations, the code is straightforward (*ignore the null-pointer exception for the Dequeue() operation for now*):

```
// Holding cell for an entry
class QueueEntry {
    QueueEntry next = null;
    Object stored;

    QueueEntry(Object newobject) {
        stored = newobject;
    }
}

// The actual Queue (not yet lock free!)
class Queue {
    QueueEntry prevHead = new QueueEntry(null);
    QueueEntry tail = prevHead;

    void Enqueue(Object newobject) {
        QueueEntry newEntry = new QueueEntry(newobject);
        QueueEntry oldtail = tail;
        tail = newEntry;
        oldtail.next = newEntry;
    }

    Object Dequeue() {
        QueueEntry oldprevHead = prevHead;
        QueueEntry nextEntry = oldprevHead.next;
        prevHead = nextEntry;
        return nextEntry.stored;
    }
}
```

Problem 3d[3pts]:

For this non-multithreaded code, draw the state of a queue with 2 queued items on it:

Problem 3e[3pts]:

For each of the following potential context switch points, state whether or not a context switch at that point could cause incorrect behavior of Enqueue(); Explain!

```

        void Enqueue(Object newobject) {
1 → QueueEntry newEntry = new QueueEntry(newobject);
2 → QueueEntry oldtail = tail;
3 → tail = newEntry;
        oldtail.next = newEntry;
        }

```

Point 1:

Point 2:

Point 3:

Problem 3f[4pts]:

Rewrite code for Enqueue(), using the CAS() operation, such that it will work for any number of simultaneous Enqueue and Dequeue operations. You should never need to busy wait. **Do not use locking (i.e. don't use a test-and-set lock).** The solution is tricky but can be done in a few lines. We will be grading on conciseness. Do not use more than one CAS() or more than 10 lines total (including the function declaration at the beginning). *Hint: wrap a do-while around vulnerable parts of the code identified above.*

```

void Enqueue(Object newobject) {
    QueueEntry newEntry = new QueueEntry(newobject);

    // Insert code here

}

```

Problem 3g[3pts]:

For each of the following potential context switch points, state whether or not a context switch at that point could cause incorrect behavior of Dequeue(); Explain!

```

    Object Dequeue() {
1  → QueueEntry oldprevHead = prevHead;
2  → QueueEntry nextEntry = oldprevHead.next;
3  → prevHead = nextEntry;
    return nextEntry.stored;
    }

```

Point 1:

Point 2:

Point 3:

Problem 3h[4pts]:

Rewrite code for Dequeue(), using the CAS() operation, such that it will work for any number of simultaneous Enqueue and Dequeue operations. You should never need to busy wait. **Do not use locking (i.e. don't use a test-and-set lock).** The solution can be done in a few lines. We will be grading on conciseness. Do not use more than one CAS() or more than 10 lines total (including the function declaration at the beginning). *Hint: wrap a do-while around vulnerable parts of the code identified above.*

```

    Object Dequeue() {
        // Insert code here

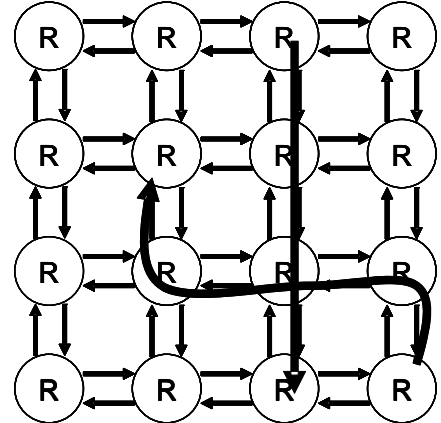
    }

```

Problem 4: Deadlock[21 pts]

Problem 4a[5pts]:

The figure at the right illustrates a 2D mesh of network routers. Each router is connected to each of its neighbors by two network links (small arrows), one in each direction. Messages are routed from a source router to a destination router and can stretch through the network (i.e. consume links along the route from source to destination). Messages can cross inside routers.



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). Messages always make progress to the destination and never wrap back on themselves. The figure shows two messages (thick arrows).

Assume that each router or link has a very small amount of buffer space and that each message can be arbitrarily long. 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 are satisfied by your example. *Hint: Links are the limited resources in this example.*

Problem 4b[3pts]:

Define a routing policy that avoids deadlocks in the network of (4a). Name one of the four conditions that is no longer possible, given your routing policy. Explain.

Problem 4c[3pts]:

Suppose that each router node contains sufficient queue space to hold complete messages (assume infinite space, if you like). Why is it impossible for deadlocks such as in (4a) to occur? Name one of the four conditions that is no longer possible, given infinite queue space in the router. Explain.

Problem 4d[4pts]:

Suppose that we have the following resources: A, B, C and threads T1, T2, T3, T4. The total number of each resource is:

Total		
A	B	C
12	9	12

Further, assume that the processes have the following maximum requirements and current allocations:

Thread ID	Current Allocation			Maximum		
	A	B	C	A	B	C
T1	2	1	3	4	9	4
T2	1	2	3	5	3	3
T3	5	4	3	6	4	3
T4	2	1	2	4	8	2

Is the system in a safe state? If “yes”, show a non-blocking sequence of thread executions. Otherwise, provide a proof that the system is unsafe. Show all steps, intermediate matrices, etc.

Problem 4e[3pts]:

Assume that we start with a system in the state of (4d). Suppose that T1 asks for 2 more copies of resource A. Can the system grant this if it wants to avoid deadlock? Explain.

Problem 4f[3pts]:

Assume that we start with a system in the state of (4d). What is the maximum number of additional copies of resources (A, B, and C) that T1 can be granted in a single request without risking deadlock? Explain.

[This page intentionally left blank]

Problem 5: Address Translation [18 pts]

Problem 5a[3pts]:

Suppose we have a 32-bit processor (with 32-bit virtual addresses) and 8 KB pages. Assume that it can address up to 2 TB (terabytes) of DRAM; $1\text{TB} = 1024\text{GB} = (1024)^2\text{MB}$. Assume that we need 4 permissions bits in each page table entry (PTE), namely Valid (V), Writable (W), Accessed (A), and Dirty (D). Show the format of a PTE, assuming that each page should be able to hold an integer number of PTEs. If you have extra bits in the PTE, you can mark them as “unused”. Explain.

Problem 5b[5pts]:

Assume that we wish to build a two-level page table for the processor from (5a) in which each piece of the page table consumes exactly a page (no more, no less). We may end up wasting space as a result. Draw and label a figure showing how a virtual address gets mapped into a physical address. Show the format of the page table (complete with access checks), the virtual address, and physical address. *Minimize pieces of the page table that consume less than a page (and thus waste space).*

Problem 5c[3pts]:

Consider a multi-level memory management scheme using the following format for virtual addresses:

Virtual seg # (2 bits)	Virtual Page # (6 bits)	Offset (12 bits)
---------------------------	----------------------------	---------------------

Virtual addresses are translated into physical addresses of the following form:

Physical Page # (8 bits)	Offset (12 bits)
-----------------------------	---------------------

Page table entries (PTE) are 16 bits in the following format, *stored in big-endian form* in memory (i.e. the MSB is first byte in memory):

Physical Page # (8 bits)	Kernel	Nocache	0	0	Dirty	Use	Writeable	Valid
-----------------------------	--------	---------	---	---	-------	-----	-----------	-------

- 1) How big is a page? Explain.

- 2) What is the maximum amount of virtual memory supported by this scheme? Explain

- 3) What is the maximum amount of physical memory supported by this scheme? Explain

Problem 5d[7pts]: Assume the memory translation scheme from (5c). Use the Segment Table and Physical Memory table given on the next page to predict what will happen with the following load/store instructions. Addresses are virtual. The return value for a load is an 8-bit data value or an error, while the return value for a store is either “ok” or an error. If there is an error, make sure to say which error. Possibilities are: “**bad segment**” (invalid segment), “**segment overflow**” (address outside range of segment), or “**access violation**” (page invalid, or attempt to write a read only page). A few answers are given:

Instruction	Result
Load [0xC1015]	0x57
Store [0x43045]	ok
Store [0xC1016]	Access violation
Load [0xD2002]	
Store [0xD2031]	

Instruction	Result
Store [0x52002]	
Load [0x04013]	
Store [0x81015]	
Store [0x03010]	
Load [0x13035]	

Segment Table (Max Segment=3)

Seg #	Page Table Base	Max Page Entries	Segment State
0	0x02030	0x20	Valid
1	0x01020	0x10	Valid
2	0x01040	0x40	Invalid
3	0x04000	0x20	Valid

Physical Memory

Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0x00000	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
0x00010	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D
....																
0x01010	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
0x01020	40	03	41	01	30	01	31	03	00	03	00	00	00	00	00	00
0x01030	00	11	22	33	44	55	66	77	88	99	AA	BB	CC	DD	EE	FF
0x01040	10	01	11	03	31	03	13	00	14	01	15	03	16	01	17	00
....																
0x02030	10	01	11	00	12	03	67	03	11	03	00	00	00	00	00	00
0x02040	02	20	03	30	04	40	05	50	01	60	03	70	08	80	09	90
0x02050	10	00	31	01	10	03	31	01	12	03	30	00	10	00	10	01
....																
0x04000	30	00	31	01	11	01	33	03	34	01	35	00	43	38	32	79
0x04010	50	28	84	19	71	69	39	93	75	10	58	20	97	49	44	59
0x04020	23	03	20	03	00	01	62	08	99	86	28	03	48	25	34	21
....																
0x10000	AA	55	AA	55	AA	55	AA	55	AA	55	AA	55	AA	55	AA	55
0x10010	A5	5A	A5	5A	A5	5A	A5	5A	A5	5A	A5	5A	A5	5A	A5	5A
....																
0x11000	00	11	22	33	44	55	66	77	88	99	AA	BB	CC	DD	EE	FF
0x11010	11	22	33	44	55	66	77	88	99	AA	BB	CC	DD	EE	FF	00
0x11020	22	33	44	55	66	77	88	99	AA	BB	CC	DD	EE	FF	00	11
....																
0x31000	01	12	23	34	45	56	67	78	89	9A	AB	BC	CD	DE	EF	00
0x31010	02	13	24	35	46	57	68	79	8A	9B	AC	BD	CE	DF	F0	01
0x31020	03	01	25	36	47	58	69	7A	8B	9C	AD	BE	CF	E0	F1	02
0x31030	04	15	26	37	48	59	70	7B	8C	9D	AE	BF	D0	E1	F2	03
....																

[This page intentionally left blank]

[This page left for scratch]