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

Fall 2018 Ion Stoica

Second Midterm Exam

October 29, 2018

CS162 Operating Systems

Your name	
SID	
CS162 login (e.g., s162)	
TA Name	
Discussion section time	

This is a closed book and two 2-sided handwritten notes 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
P1	**12**	
P2	**16**	
P3	**18**	
P4	**19**	
P5	**17**	
P6	**18**	
Total	**100**	

True/False and Why? (12 points)

MARK THE CHECKBOX NEXT TO YOUR ANSWER. For each question: 1 point for true/false correct, 1 point for explanation. An explanation **cannot** exceed 2 sentences.

a) When comparing physical and virtual addresses, the number of offset bits in both addresses must always be the same.

▢ **TRUE** ▢ **FALSE**

Why?

This allows us to translate from virtual to physical addresses and vice versa.

b) When repeatedly looping through an amount of data larger than the cache size, LRU will yield less page faults than FIFO.

▢ **TRUE** ▢ **FALSE**

Why?

Given this access pattern, LRU will essentially become FIFO.

c) Having a page table fit in a single memory page (frame) reduces external fragmentation.

▢ **TRUE** ▢ **FALSE**

Why?

With paging there is no external fragmentation, as all pages have the same size.

d) A multi-level page table is a better solution for a sparse address space than a single level page table approach.

▢ **TRUE** ▢ **FALSE**

Why?

With a single level page table you need to keep entries for every gaps in the virtual address space.

e) Deadlock will not occur if resources are allowed to be shared up to three times. That is, three threads can acquire the resource before the fourth one tries and blocks.

▢ **TRUE** ▢ **FALSE**

Why?

Although threads are allowed to share a resources up to three times, consider the counter example of 6 threads and 2 semaphores with an initial value of 3. 3 of the threads, down the first semaphore and the other 3 threads down the second semaphore. Then they try to down the other semaphore and deadlock.

f) Banker's algorithm guarantees that all threads eventually receive their requested resources.

▢ **TRUE** ▢ **FALSE**

Why?

Consider a program that receives its resources but has an infinite while loop. Banker's algorithm is a deadlock avoidance/resource allocation algo to prevent deadlock under the assumption that all threads will eventually terminate and release their resources after receiving their resources.

Mentions of starvation were equivalent to the above scenario and were awarded points as well. However, any solutions related to unsafe states were not considered to be valid solutions because the banker's algorithm does guarantee that your system is in a safe state if you were to run at all time steps throughout the execution of your system.

P2. Demand Paging (16 points)

Recall that in on-demand paging, a page replacement algorithm is used to manage system resources. Suppose that a newly-created process has 4 page frames allocated to it, and then generates the page references indicated below.

A B C D B A C E A B C D A E A C B D

(a) (4 points) How many page faults would occur with FIFO page replacement? Put your answer under **Total**. Additionally, **place an 'X'** in each box that corresponds to a page fault.

A	B	C	D	B	A	C	E	A	B	C	D	A	E	A	C	B	D	Total
x	x	x	x				x	x	x	x	x		x	x		x		12

(b) (4 points) How many page faults would occur with LRU page replacement? Put your answer ungder **Total**. Additionally, **place an 'X'** in each box that corresponds to a page fault.

A	B	C	D	B	A	C	E	A	B	C	D	A	E	A	C	B	D	Total
x	x	x	x				x				x		x			x	x	9

(c) (4 points) How many page faults would occur with clock replacement? Put your answer under **Total**. Additionally, **place an 'X'** in each box that corresponds to a page fault.

A	B	C	D	B	A	C	E	A	B	C	D	A	E	A	C	B	D	Total
x	x	x	x				x	x	x	x	x		x			x	x	12

(d) (4 points) Suppose we have a new form of eviction that evicts in order of frequency such that pages accessed less frequently than others are evicted first, where the frequency is measured

by the number of accesses since the page was paged in. This method of page replacement is called Least Frequently Used (LFU). How many page faults would occur with LFU page replacement? Put your answer under **Total**.
(You may assume ties are broken with FIFO order)
Additionally, **place an 'X'** in each box that corresponds to a page fault.

A	B	C	D	B	A	C	E	A	B	C	D	A	E	A	C	B	D	Total
x	x	x	x				x				x		x				x	8

P3. The Kitchen Sync (18 points)

We are given the following implementations for a lock in Pintos, with two threads, A and B, that each run `acquire()` then `release()`. For each implementation, answer the following questions:

- Does the implementation enforce mutual exclusion (Can we ensure one thread starts `release` before the other thread finishes `acquire`)?
- If not, give an interleaving that breaks mutual exclusion. You only need to fill in as many line executions as it takes to break mutual exclusion.
- Is it possible for either A or B to be permanently block?
- If yes, which threads can be blocked and on which lines they can be stuck on?

Please make the following assumptions:

- `wait_list` has been initialized already
- `move_thread` is an atomic operation that moves a thread from its current list to the specified list
- Line numbers are specified to the left of each line
- Thread A finishes line 2 before any context switching occurs
- Being "stuck" on a line means that line has started or finished executing, but the next line has not started executing yet
- `thread_block ()` will still work if called with interrupts enabled
- The scheduler skips threads with status THREAD_BLOCKED

(Question continues on next page)

(a) (6 points)

```
bool held = false;
struct list wait_list;
```

```
1. acquire() {
2.   if (held) {
3.     move_thread (thread_current (),
                    wait_list);
4.     thread_block ();
5.   } else {
6.     held = true;
7.   }
8.}
```

```
9.  release() {
10.  if (!list_empty (wait_list)) {
11.    thread_unblock (list_front
           (wait_list));
12.    move_thread (list_front
           (wait_list), ready_list);
13.  } else {
14.    held = false;
15.  }
16. }
```

Does this enforce mutual exclusion? If not, please indicate an order of line execution that proves this. If needed, we have filled out the first line of execution for you.

Yes [ ] **No** [X]

Step	1	2	3	4	5	6	7
Running Thread	A	B	A	B			
Line	2	2	6	6			

Is it possible for a thread to be blocked forever? If so, which thread(s) and on what lines(s)?

Yes [X] **No** [ ]

Thread Blocked	B	B		
Line Blocked On	3	4		

Thread A can never block since we specify that it executes line (2) before context switching, so it can never enter the `true` branch of the `if` block in `acquire`. If thread B context switches after executing line 3 and after thread A calls `release`, it will never be scheduled again since it will not be on the ready list. It's also possible that thread B will move itself to the wait list, be moved back, then call thread_block. In this case it is also blocked forever.

(b) (6 points)

```
int guard = 0;
bool held = false;
struct list wait_list;
```

```
1. acquire() {
2.   while (test&set (&guard));
3.   if (held) {
4.     move_thread (thread_current (),
                    wait_list);
5.     guard = 0;
6.     thread_block ();
7.   } else {
8.     held = true;
9.     guard = 0;
10. }
11.}
```

```
12.release() {
13.   while (test&set (&guard));
14.   if (!list_empty (wait_list)) {
15.     thread_unblock (list_front
            (wait_list));
16.     move_thread (list_front
            (wait_list), ready_list);
17.   } else {
18.     held = false;
19.   }
20.   guard = 0;
21. }
```

Does this enforce mutual exclusion? If not, please indicate an order of line execution that proves this. If needed, we have filled out the first line of execution for you.

Yes [X] **No** [ ]

Step	1	2	3	4	5	6	7
Running Thread	A						
Line	2						

Is it possible for a thread to be blocked forever?

Yes [X] **No** [ ]

Thread Blocked	B	B		
Line Blocked On	5	6		

Same reasoning as above.

(c) (6 points)

```
bool held = false;
struct list wait_list;
```

```
1. acquire() {
2.   intr_disable ();
3.   if (held) {
4.     move_thread (thread_current (),
                    wait_list);
5.     thread_block ();
6.   } else {
7.     held = true;
8.   }
9.   intr_enable ();
10.}
```

```
11.release() {
12.   intr_disable ();
13.   if (!list_empty (wait_list)) {
14.     thread_unblock (list_front
            (wait_list));
15.     move_thread (list_front
            (wait_list), ready_list);
16.   } else {
17.     held = false;
18.   }
19.   intr_enable ();
20.}
```

Does this enforce mutual exclusion? If not, please indicate an order of line execution that proves this. If needed, we have filled out the first line of execution for you.

Yes [X] **No** [ ]

Step	1	2	3	4	5	6	7
Running Thread	A						
Line	2						

Is it possible for a thread to be blocked forever? If so, which thread(s) and on what lines(s)?

Yes [ ] **No** [X]

Thread Blocked				
Line Blocked On				

P4. Deadlock, Deadlock, Deadlock (19 points)

Suppose we have the following total resources and threads T1, T2, T3, and T4 with current allocations and maximum required allocations.

Total Resources		
A	B	C
5	12	7

	Current Allocation			**Maximum Allocation**		
	A	B	C	A	B	C
T1	1	5	1	3	10	3
T2	0	2	1	2	4	2
T3	2	1	0	2	3	3
T4	1	2	2	2	8	5

(a) (5 points) Is the system in a safe state? If so, provide a sequence of resource allocations that would allow all threads to terminate. If not, explain why in two sentences or less.

Yes, the system is in a safe state. We see that T3 is able to request up to its max allocation of both B and C and finish and release its resources. From there, we have 2 of A, 3 of B, and 3 of C. We can use this to run T2 and finish it, which results in 2 of A, 5 of B, and 4 of C being free. We then run T1 and then T4. So the sequence is T3, T2, T1, T4 request needed resources.

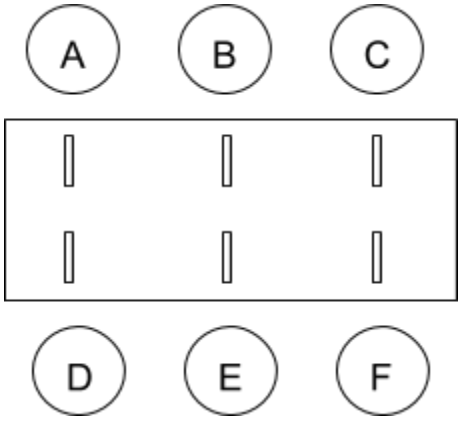
(b) (4 points) Oh no! We allow threads to acquire resources as they request them and as a result we have gotten ourselves into the deadlocked state, as shown below. We can force a SINGLE thread to release all of its current resources. Choose a SINGLE thread to forcefully release all of its resources in order to ensure that all programs finish or if this is not possible state "No thread". Write the answer in the box. In both cases, provide a short explanation outside of the box (no more than two sentences).

	Current			Maximum		
	A	B	C	A	B	C
T1	2	7	2	3	10	3
T2	0	3	1	2	4	2
T3	1	0	1	2	3	3
T4	1	2	3	2	8	5

Thread:

"No thread" because since we stated that threads acquire resources as they request them. For example, it is possible that the thread we just force released now runs without being context switched and acquires the same set of resources resulting in deadlock (since the problem states that this is a deadlocked state). This shows that it is important to use algorithms and checks such as Banker's algorithm to deny or grant requests.

Now let's consider another problem also dealing with deadlock.



Suppose philosophers A, B, C, D, E, and F are sitting at a rectangular table.

- Philosophers need "n" chopsticks to eat and once done eating will place its chopsticks back to where they belong. Then they try to eat again.
- Each philosopher has one chopstick in front of them.
- Philosophers can use chopsticks that are in front of themselves, their neighbors, or the philosopher across from them. For example in the provided seating, philosopher B can grab A, B, C, or E's chopstick if it is free.
- Philosophers have a preferred set of chopsticks they use. They will try to grab their first preferred chopstick and wait if it is taken and so on.

(c) (5 points) Consider the problem where philosophers need THREE chopsticks to eat (n=3). Provide a preference of three chopsticks for each philosopher in order to prevent deadlock or state that it is not possible in the table below. If the "not possible" box is checked, we will NOT grade the answers in the table.

Not possible ☐

		Philosopher					
		A	**B**	**C**	**D**	**E**	**F**
Chopstick	**1st**	A	B	B	A	B	C
	2nd	B	C	C	E	E	F
	3rd	D	E	F	D	D	E

Answers can vary. The straightforward method is to enforce an ordering (for example in the table above, philosophers should grab A, then B, then C, then F, then E, and then D if they are able to). We can recall this from the lecture on how to prevent deadlocks (enforce an ordering

on how we acquire resources). There exists deadlock if there exists a cycle of resource requests that result in cyclic waiting. By enforcing an ordering, deadlock is not possible in this case since it eliminates cyclic waiting. For example, if philosopher B preferred chopsticks B and then C and philosopher C preferred chopsticks C and then B, then that would be a deadlock.

(d) (5 points) Suppose philosophers can now reach diagonally across the table in both directions to grab a free chopstick. For example, philosopher B can try to grab D and F in addition to the previous chopsticks it could grab. Suppose that philosophers now need FOUR chopsticks to eat (n=4). Provide a preference of FOUR chopsticks for each philosopher in order to prevent deadlock or state that it is not possible in the table below.

Not possible

☐

		Philosopher					
		A	**B**	**C**	**D**	**E**	**F**
Chopstick	**1st**	A	B	B	A	A	B
	2nd	B	C	C	B	B	C
	3rd	E	F	F	E	E	F
	4th	D	E	E	D	D	E

Answers can vary (Same reasoning as above). Ordering used for this table is ABCFED.

P5. Just ‘Bout That Translation, Boss (17 points)

Suppose we have a 42 bit virtual address space, a page size of 1KB and a single level page table with a page table entry size of 4 bytes

(a) (3 points) How many bits is the virtual page number? The offset? Place your answer in the box provided

VPN: 32 bits

Offset: 10 bits

(b) (3 points) Given that we need at least 9 control bits per PTE, what is the maximum size of our physical address space? Place your answer in the box provided.

2^23*2^10 = 2^33 = 8GB

c) (8 points) Assume the following PTE format (with 9 control bits) stored in big-endian form in the following page table.

PTE:

PPN	Other (6 bits)	Read/Write	Dirty	Valid

Page Table:

Address	+0	+1	+2	+3	+4	+5	+6	+7
0x8000	6C	65	67	69	6F	6E	20	6F
0x8008	66	20	62	6F	6F	6D	43	08
0x8010	25	29	31	41	54	72	50	56
0x8018	67	6F	20	68	61	77	6B	73

Using the information above, translate each of the following virtual addresses to physical addresses. Assume that the page table pointer is at 0x8000. Place your final answer in the box provided. If you encounter and error, write **ERROR** in the box.

- 0x00000000B03

0x0CC40C703

- 0x00000001F59

0x0C2EED759

- 0x0000000051B

0x0DEDC411B

- 0x000000014B2

invalid

d) (3 points) Now suppose we want to transform our single level page table into a multi-leveled page table. Assuming that every page table is required to fit into a single page, how many total levels of page tables do we need to address the entire virtual address space? Place your answer in the box provided.

2^8 * 2^8 * 2^8 * (2^8* 2^10) = 2^42

P6. ML Homework (18 Points)

As part of a machine learning homework, Natalie wrote a dense matrix-vector multiplication kernel (y=Ax). Natalie's computer has a single-core processor with 1 level of L1 cache. The cache is 4-way associative, with 8-byte cache blocks, an LRU eviction policy, and an overall cache size is 512 bytes. Write operations allocate the relevant block into the cache.

Assumptions:

- The variable i,j are stored in registers and do not consume memory/cache space.
- We define "hit rate" as the percentage of memory accesses (both read accesses and write accesses) that have a cache hit.
- A is saved as a 2d array starting at the address 0x10
- X is saved as an array starting at address 0x500000010
- Y is saved as an array starting at address 0x1000000010
- Z is saved as an array starting at address 0x1500000010
- The matrix A is stored as a row-major matrix (i.e., rows are stored contiguously in memory)
- Assume each value in the vector or matrix is represented as a 1-byte "integer" (i.e. has integer values between 0 and 255).

You may leave answers in the form of a fraction

Natalie's implementation of her dense matrix-vector multiplication kernel is:

```
for (int j=0; j < N; j++) {
     for (int i=0; i < M; i++) {
         y[i] += A[i][j] * x[i];
         z[i] = i;
     }
}
```

Natalie started by testing her code on a small matrix.
Assume
N = 8
M = 8

(a) (5 points) Compute the hit-rate of the L1 cache at the end of the program. (you can leave you answer in the form of a fraction). Place your answer in the box provided.

There are overall 5*8*8 memory accesses (8*8 matrix, 5 access in each iteration of the loop: 1 to x, 1 to A, 1 to read y, 1 to write y, 1 to write z)
The entire matrix and vector fit in the cache, and we have 4 way set-associative, so we will have a miss only every 8 accesses for each of the data-structures.
I.e. 1 miss for x, 1 miss for y, 1 miss for z and 8 misses for A. The write operations to y will never miss, since they were all brought into the cache during the read operation of y.
Hence the miss rate will be (1 + 1 + 1 + 8) / (5*8*8), and the hit rate will be:
1 - (1 + 1 + 1 + 8) / (5*8*8) = 309/320 = **96.6%**

Natalie was happy with her results, so she ran her code on a bigger matrix she got from her friends in the machine learning lab.

The dimensions of the new matrix are:
N = 128
M = 128

(b) (8 points) For each of the arrays in the problem, state whether its cache hit rate will increase, remain the same, or decrease compared to the rate of the previous part. What miss types (compulsory, capacity, conflict) do each of the array experience?

Array	Increase/Decrease/Same	Miss Types
x	Increase	Compulsory
y	Increase	Compulsory
z	Increase	Compulsory
A	Decrease	Compulsory, Capacity, Conflict

For the grading of this problem, we gave partial credit for identifying that the first three rows (x, y, z) should be different from the last. The way points were added are as follows: The first column's x, y, z must be different from A. If the first column's x, y, z are different from A, then you get at least 0.5 points. We call this a "consistent" column. If the hit rate of x, y, and z is higher than the hit rate of A, then you get at least 1 point. Then, only if the first column was consistent, if you marked that the miss type of x, y, and z are all the same and different from A, you get awarded 1 point, up to a maximum of 2 additional points.

For x, y, and z - they still fit in the cache (each in one of the ways), so they will suffer only compulsory misses. Each of them will have 128 initial compulsory misses, and then remain in the cache for the rest of the program. There are overall (5*128*128) memory accesses, and therefore the hit rate of x and z will be 1 - (128/(128*128)) = 127/128 (which is bigger than 7/8- the hit rate for these arrays in the previous part), while the hit rate for y will be 1 - (128/(2*128*128)) = 255/256 (which is bigger than 15/16, the hit rate for this array in the previous part).
For A, we will have initial compulsory misses. The first A address that we read in is 0x10 which is 0b00010000. The second A address that we read in is 0x10 + 128 which is 0b10010000. The address split is 3 bits for offset in the cache block, and 4 bits for the index. Hence, the index is always going to be 0010. The array does not fit in the cache (since 128x128 = 16KB > 512B) so we're going to have both conflict misses and capacity misses.

(c) (5 points) In one sentence, how would you optimize Natalie's code to run faster by obtaining a better cache hit-rate for large matrices?
Switch the order of the loops, so looping over `i` in the external loop, and looping over `j` in the internal loop

P7. Potpurri (0 points)

a. Illustrate your favorite dogspotting post below

b. How are we doing so far?