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!!

<table>
<thead>
<tr>
<th>Question</th>
<th>Points assigned</th>
<th>Points obtained</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
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?

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?

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

○ TRUE  ○ FALSE

Why?

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?

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?

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

○ TRUE  ○ FALSE

Why?
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.

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

(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.

(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.
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)

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

1. acquire() {
2.   if (held) {
3.     move_thread (thread_current (),
                 wait_list);
4.   } else {
5.     held = true;
6. }
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 [ ]

<table>
<thead>
<tr>
<th>Step</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running Thread</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Line</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Yes [ ] No [ ]

<table>
<thead>
<tr>
<th>Thread Blocked</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Line Blocked On</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(b) (6 points)

```c
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 [ ] No [ ]

<table>
<thead>
<tr>
<th>Step</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running Thread</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Line</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Yes [ ] No [ ]

| Thread Blocked |   |   |   |   |   |   |   |
| Line Blocked On |   |   |   |   |   |   |   |
(c) (6 points)

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

1. acquire() {
2.  intr_disable();
3.  if (held) {
4.    move_thread (thread_current(),
        wait_list);
5.  }
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 ☐ No ☐

<table>
<thead>
<tr>
<th>Step</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running Thread</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Line</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Yes ☐ No ☐

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

<table>
<thead>
<tr>
<th>Total Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Current Allocation</th>
<th>Maximum Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>T1</td>
<td>1</td>
</tr>
<tr>
<td>T2</td>
<td>0</td>
</tr>
<tr>
<td>T3</td>
<td>2</td>
</tr>
<tr>
<td>T4</td>
<td>1</td>
</tr>
</tbody>
</table>

(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.
(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).

<table>
<thead>
<tr>
<th></th>
<th>Current</th>
<th></th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>T1</td>
<td>2</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>T2</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>T3</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>T4</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Thread:
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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chopstick</td>
<td>1st</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2nd</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3rd</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(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.

<table>
<thead>
<tr>
<th>Chopstick</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd</td>
<td></td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3rd</td>
<td></td>
<td></td>
<td>E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4th</td>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**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: ___________

Offset: ___________

(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.

__________

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

**PTE:**

<table>
<thead>
<tr>
<th>PPN</th>
<th>Other (6 bits)</th>
<th>Read/Write</th>
<th>Dirty</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Page Table:**

<table>
<thead>
<tr>
<th>Address</th>
<th>+0</th>
<th>+1</th>
<th>+2</th>
<th>+3</th>
<th>+4</th>
<th>+5</th>
<th>+6</th>
<th>+7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8000</td>
<td>6C</td>
<td>65</td>
<td>67</td>
<td>69</td>
<td>6F</td>
<td>6E</td>
<td>20</td>
<td>6F</td>
</tr>
<tr>
<td>0x8008</td>
<td>66</td>
<td>20</td>
<td>62</td>
<td>6F</td>
<td>6F</td>
<td>6D</td>
<td>43</td>
<td>08</td>
</tr>
<tr>
<td>0x8010</td>
<td>25</td>
<td>29</td>
<td>31</td>
<td>41</td>
<td>54</td>
<td>72</td>
<td>50</td>
<td>56</td>
</tr>
<tr>
<td>0x8018</td>
<td>67</td>
<td>6F</td>
<td>20</td>
<td>68</td>
<td>61</td>
<td>77</td>
<td>6B</td>
<td>73</td>
</tr>
</tbody>
</table>
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
- 0x00000001F59
- 0x0000000051B
- 0x000000014B2

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.
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:

```c
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.

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?

<table>
<thead>
<tr>
<th>Array</th>
<th>Increase/Decrease/Same</th>
<th>Miss Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(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?
P7. Potpurri (0 points)
   a. Illustrate your favorite dogspotting post below

b. How are we doing so far?

When you have 162 midterms, projects, and homeworks all within a week of each other

that wasn't very cache money of you