CS 162 Fall 2020

Operating Systems and System Programming

Midterm 2

INSTRUCTIONS

This is your exam. Complete it either at exam.cs61a.org or, if that doesn't work, by emailing course staff with your solutions before the exam deadline.

This exam is intended for the student with email address `<EMAILADDRESS>`. If this is not your email address, notify course staff immediately, as each exam is different. Do not distribute this exam PDF even after the exam ends, as some students may be taking the exam in a different time zone.

For questions with **circular bubbles**, you should select exactly *one* choice.

- ◯ You must choose either this option
- ◯ Or this one, but not both!

For questions with **square checkboxes**, you may select *multiple* choices.

- ☐ You could select this choice.
- ☐ You could select this one too!

You may start your exam now. Your exam is due at <DEADLINE> Pacific Time. Go to the next page to begin.

This is a **proctored, closed-book exam**. During the exam, you may **not** communicate with other people regarding the exam questions or answers in any capacity. If there is something in a question that you believe is open to interpretation, please use the "Clarifications" button to request a clarification. We will issue an announcement if we believe your question merits one. We will overlook minor syntax errors in grading coding questions. You do not have to add necessary `#include` statements. For coding questions, the number of blank lines you see is a suggested guideline, but is not a strict minimum or maximum. There will be no limit on the length of your answer/solution.

a)

Name

b)

Student ID

c)

Please read the following honor code: "I understand that this is a closed book exam. I hereby promise that the answers that I give on the following exam are exclusively my own. I understand that I am allowed to use two 8.5x11, double-sided, handwritten cheat-sheet of my own making, but otherwise promise not to consult other people, physical resources (e.g. textbooks), or internet sources in constructing my answers." Type your full name below to acknowledge that you've read and agreed to this statement.

1. (18.0 points) True/False

Please **EXPLAIN** your answer in **TWO SENTENCES OR LESS** (Answers longer than this may not get credit!). Also, answers without any explanation **GET NO CREDIT!**

a) (2.0 points)

1

Paging solves internal fragmentation because all pages are the same size.

◯ True

◯ False

2

Explain.

b) (2.0 points)

1

The translation of virtual to physical addresses is done by the kernel.

◯ True

◯ False

2

Explain.

c) (2.0 points)

1

Single level page tables are more efficient at representing sparse address spaces than multi-level page tables.

◯ True

◯ False

2

Explain.

d) (2.0 points)

1

Multi-level page tables are better memory-wise for sparse addresses in comparison to single level page tables.

◯ True

◯ False

2

Explain.

e) (2.0 points)

1

The number of bits in a virtual address is always the same as the number of bits in its corresponding physical address.

◯ True

◯ False

2

Explain.

f) (2.0 points)

1

On a page fault, the MMU will invalidate previous virtual to physical address mappings if needed and create new mappings in the page table based on the requested data brought from the kernel.

◯ True

◯ False

2

Explain.

g) (2.0 points)

1

For base & bound virtual memory, the two special registers BaseAddr and LimitAddr are stored in the Thread Control Block.

◯ True

◯ False

2

Explain.

h) (2.0 points)

1

Adding a TLB will always make memory lookups and accesses faster.

◯ True

◯ False

2

Explain.

i) (2.0 points)

1

The associativity of the TLB can be configured by modifying kernel source code.

◯ True

◯ False

2

Explain.

j) (2.0 points)

1

Swapping is the term denoting the process of swapping PCBs and other housekeeping such as switching page table pointers.

◯ True

◯ False

2

Explain.

k) (2.0 points)

1

Thrashing is characterized by slow performance and high CPU utilization.

◯ True

◯ False

2

Explain.

l) (2.0 points)

1

Thrashing is characterized by slow performance and low CPU utilization.

◯ True

◯ False

2

Explain.

m) (2.0 points)

1

Killing the process with the largest working set is a guaranteed solution to thrashing.

◯ True

◯ False

2

Explain.

n) (2.0 points)

1

Write through caches do not need a dirty bit.

◯ True

◯ False

2

Explain.

o) (2.0 points)

1

Write through caches need a dirty bit.

◯ True

◯ False

2

Explain.

p) (2.0 points)

1

In general, larger caches or caches with higher associativity have a higher hit rate than smaller, direct-mapped caches.

◯ True

◯ False

2

Explain.

q) (2.0 points)

1

In general, larger caches or caches with higher associativity have a lower hit rate than smaller, direct-mapped caches.

◯ True

◯ False

2

Explain.

r) (2.0 points)

1

In deadlock, one process continually responds to another process's changes but is unable to complete any work.

◯ True

◯ False

2

Explain.

s) (2.0 points)

1

In deadlock, one process can't respond to another process's operating calls and is unable to complete any work.

◯ True

◯ False

2

Explain.

t) (2.0 points)

1

Pre-emptive schedulers fix the problem of deadlock in a system.

◯ True

◯ False

2

Explain.

u) (2.0 points)

1

A cyclic use of resources leads to deadlock.

◯ True

◯ False

2

Explain.

v) (2.0 points)

1

It's not possible to use the Banker's algorithm to guarantee the completion of tasks in a real-life operating system.

◯ True

◯ False

2

Explain.

w) (2.0 points)

1

The only way to prevent deadlock caused by cyclic use of resources is to use a dynamic algorithm such as the Banker's algorithm to mediate all resource acquisition.

◯ True

◯ False

2

Explain.

x) (2.0 points)

1

Banker's Algorithm can find more than one potential order of processes that result in a safe state.

◯ True

◯ False

2

Explain.

y) (2.0 points)

1

Banker's Algorithm can only find one order of processes that results in a safe state.

◯ True

◯ False

2

Explain.

z) (2.0 points)

1

Multiprocessing networks with wormhole routing must use a dynamic scheduler built in hardware to implement the Banker's Algorithm in order to avoid deadlock.

◯ True

◯ False

2

Explain.

aa) (2.0 points)

1

Assuming that proper feasibility checks have been performed on the workload, a real-time scheduler is not prone to starvation.

◯ True

◯ False

2

Explain.

ab) (2.0 points)

1

Real-time schedulers are prone to starvation even if proper feasibility checks have been performed on the workload.

◯ True

◯ False

2

Explain.

ac) (2.0 points)

1

There are scheduler workloads where a non-preemptive scheduler has a better a verage wait time than a preemptive scheduler.

◯ True

◯ False

2

Explain.

ad) (2.0 points)

1

The SRTF Algorithm is an example of a scheduler algorithm that can't be implemented in a real-life system.

◯ True

◯ False

2

Explain.

ae) (2.0 points)

1

The SRTF Algorithm is an example of a scheduler algorithm that can be implemented in a real-life system.

◯ True

◯ False

2

Explain.

af) (2.0 points)

1

Priority Donation can help to prevent priority inversion under some circumstances.

◯ True

◯ False

2

Explain.

ag) (2.0 points)

1

It is possible to build a scheduler that approximates SRTF using a moving average.

◯ True

◯ False

2

Explain.

ah) (2.0 points)

1

It is possible to build a scheduler that approximates SRTF using a moving average.

◯ True

◯ False

2

Explain.

2. (16.0 points) Multiple Choice

a) (2.0 pt)

Select all that apply: It is possible for the addition of physical memory to decrease performance when our page replacement policy is:

- ☐ LRU
- ☐ FIFO
- ☐ Random
- ☐ MRU
- ☐ None of the above
- ☐ It is never possible for more physical memory to be hurtful

b) (2.0 pt)

Suppose we have a 512 B single-page page table where each page table entry is 4 bytes. How big is the virtual address space?

- ◯ 256 KB
- ◯ 64 KB
- ◯ 2 KB
- ◯ 512 B
- ◯ 128 B
- ◯ Not enough information
- ◯ None of the above

c) (2.0 pt)

Suppose we have a 512 B single-page page table where each page table entry is 4 bytes. How big is the physical address space?

- ◯ 256 KB
- ◯ 64 KB
- ◯ 2 KB
- ◯ 512 B
- ◯ 128 B
- ◯ Not enough information
- ◯ None of the above

d) (2.0 pt)

Suppose that pages are 512 B and each page table entry is 4 bytes. Assume that somehow the virtual and physical address spaces were both 4 GB and that the page table begins at address 0x10000000. If we wanted to access the virtual address 0x00000345, what is the address of the PTE we would look at?

- ◯ 0x10000000
- ◯ 0x10000001
- ◯ 0x10000004
- ◯ 0x10000345
- ◯ Not enough information
- ◯ None of the above

e) (2.0 pt)

Select all that are true regarding inverted page tables.

- ☐ It would be smart to use an inverted page table when our physical memory space is very large.
- ☐ Inverted page tables make it difficult to implement shared memory.
- ☐ Lookup times are generally longer than standard page tables.
- ☐ Inverted page tables save memory when compared to a standard page table.
- ☐ None of the above

f) (2.0 pt)

Which of the following are true regarding virtual memory and address translation?

- ☐ It is possible to have a larger virtual memory space than physical memory space
- ☐ It is possible to have a larger physical memory space then virtual memory space
- ☐ Physical memory pages and virtual memory pages usually differ in size
- ☐ Modern processors generally include dedicated hardware to assist with address translation
- ☐ Address translation is managed entirely in hardware on x86

g) (2.0 pt)

Which of the following are true regarding virtual memory?

- ☐ Adjacent bytes within the same virtual page are always also adjacent in physical memory
- ☐ Adjacent bytes within the same physical page are always also adjacent in virtual memory
- ☐ Adjacent virtual pages are always stored in adjacent physical pages
- ☐ Adjacent physical pages always correspond to adjacent virtual pages

h) (2.0 pt)

Suppose a thread in Pintos is holding a lock called lock A. Which of the following could change the thread's effective priority? Select all that apply.

- ☐ The thread tries to acquire another lock, lock B, but has to wait.
- ☐ The thread releases lock A.
- ☐ Another thread tries to acquire lock A.
- ☐ Some other thread dies.
- ☐ The thread calls thread_set_priority and passes in a value less than its current base priority.
- ☐ None of the above

i) (2.0 pt)

Which of the following are true?

- ☐ Deadlock will always cause starvation of CPU resources
- ☐ Livelock will always cause starvation of CPU resources
- ☐ Shortest Run Time First scheduling may cause starvation of CPU resources
- ☐ Longest Run Time First scheduling may cause starvation of CPU resources

j) (2.0 pt)

Consider the following simple alternative to demand paging: processes must declare ahead of time (i.e. by including this information in each executable) how much memory they will use, and the OS must hold this much physical memory in reserve for exclusive use by the process from the time the process begins running until the time it exits. Select all of the following that are true regarding this alternative as compared to demand paging:

- ☐ It will result in lower memory access latency on average for processes
- ☐ It will result in overall higher CPU utilization on average for the system as a whole
- ☐ It will reduce the amount of disk I/O performed on average for the system as a whole
- ☐ It would require additional dedicated hardware support/features in order to implementable

k) (2.0 pt)

Consider an OS running on a single-core processor which handles page faults using the following approach: when a process encounters a page fault, the OS waits for the page to be brought in from disk and then resumes the process, without ever putting the process to sleep. Select all of the following that are true regarding this approach, compared to the standard approach of putting a process to sleep when a page fault is encountered, then waking it when the page has been brought into physical memory.

- ☐ It will result in higher data-cache hit rates on average for processes
- ☐ It will result in overall higher concurrency in the system as a whole
- ☐ It will result in higher throughput in terms of processes serviced in the system as a whole
- ☐ It will more highly stress the TLB

l) (2.0 pt)

When trying to cache data which follows a Zipfian distribution, which of the following are true:

- ☐ Increasing the size of the cache yields diminishing returns
- ☐ Caching is completely ineffective
- ☐ Caching is worthwhile only when the cache is large relative to the size of the domain (>= 50%)
- ☐ None of the above

m) (2.0 pt)

Which of the following are true regarding page replacement policies?

- ☐ Using MIN always results in fewer cache misses than using FIFO for the same workload
- ☐ Using MIN always results in the fewest possible cache misses for any workload
- ☐ Using LRU never results in more cache misses than using FIFO for the same workload
- ☐ Clock replacement generally has lower computational overhead than LRU replacement

n) (2.0 pt)

Which of the following are true regarding Banker's Algorithm?

- ☐ It only grants resource requests that will keep the system in a "safe" state so that it will have the potential to complete without deadlock.
- ☐ It can always find an allocation of resources that avoids deadlock
- ☐ If it detects an "unsafe" state, then the system is guaranteed to deadlock
- ☐ It is capable of handling threads whose maximum possible resource consumption for a particular resource changes over time

3. (19.0 points) Short Answer

a) (2.0 points) TLB During Context Switch

1

Describe two mechanisms that would ensure that the TLB functions correctly after a system context switches between two processes.

b) (2.0 points) Page Table Size

1

If we had a three level page table with each chunk of the table having 2^10 entries, how many total page table entries would there be among the second level page tables?

c) (2.0 points) Virtually-Indexed Caches

1

What is the difference between a system that has a virtually indexed cache and one which has a physically indexed cache. Assume that both systems have virtual memory and make sure to indicate how use of the TLB differs in each case.

d) (2.0 points) Page Replacement Policies

1

For the following page replacement policies - FIFO, LRU, MIN, Clock - list out the relationships between these policies. Specifically, list which policies are approximations of other policies using brackets and arrows. For example, {A→B→C→D} {E} means that A is an approximation of B, which is an approximation of C, which is an approximation of D. E is not an approximation of anything.

e) (2.0 points) Conflict Misses

1

Although any virtual page number can be mapped to any physical page number by a typical address translation scheme (thereby providing a fully-associative mapping), explain why the wrong choice of replacement policy could still lead to a high rate of conflict misses.

f) (2.0 points) Local Allocation Policy

1

Explain why a Local Allocation policy for pages might make sense for a real-time OS?

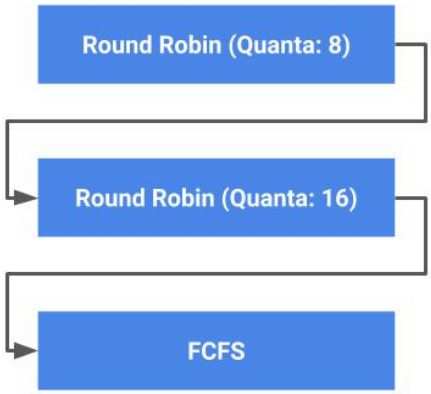
g) (2.0 points) Use-Bit Emulation

1

Explain how an operating system could make up for the lack of a use bit in the hardware-supported page table entry (PTE).

h) (2.0 points) MLFQS Scheduler

1



Assume our system uses the MLFQS scheduler depicted above. As the diagram shows, the highest priority queues are Round Robin with quantas of 8 and 16 ticks respectively. The lowest priority queue is FCFS. Explain three ways to write a program such that the program thread always remains in the highest priority queue.

i) (2.0 points) Lottery Scheduling

1

Explain how lottery scheduling can prevent starvation among low-priority tasks.

j) (2.0 points) Round-Robin Scheduling

1

Explain how Round Robin scheduling can prevent starvation among low-priority tasks.

k) Priority Donation

Consider a system that supports priority donation for locks. We have Thread A, with base priority 30, that holds Lock B and Lock C. There is a Thread B with effective priority 20 waiting on Lock B, and a Thread C with effective priority 40 waiting on Lock C.

1

Assuming these are the only three threads running on the system, what is the effective priority of Thread A?

2

For some thread T, our implementation of priority donation stores a list of donor threads that are waiting on a lock held by thread T. We calculate Thread T's effective priority using the list of donor threads and thread T's base priority. Monty Mole thinks that, because Thread B's effective priority is less than Thread A's base priority, Thread B never needs to be stored on Thread A's donor list. Is Monty Mole right? Why or why not?

l) (2.0 points) Timer

1

For the Project 2 timer, it is possible to insert threads on the sleeping list in sorted order, and peek/pop from the front of the sleeping list when waking up threads without any additional sorting. However, if we use a single ready list for priority scheduling, inserting threads onto the ready list in sorted order and popping from the front of the ready list when scheduling the next thread will cause threads to be scheduled in the wrong order. Why is this?

m) (4.0 points) Average Memory Access Time

Parameter	Value
TLB Hit Rate	0.4
TLB Lookup	5ns
L1 Cache Hit Rate	0.2
L1 Cache Lookup Time	5ns
Memory Access Time	50ns

1

Assume that our system uses a 3-level page table for address translation, in addition to a TLB and an L1 cache (assume this is the only memory cache). Given the data above, what is the Average Memory Access Time? Show your work (e.g. an equation).

2

If you could either double the TLB Hit Rate or the Memory Cache Hit Rate, which would you choose? In addition to a quantitative analysis, please provide a qualitative reason for why this is the case.

4. (28.0 points) Potpourri

a) (9.0 points) The Western Galactic Floopy Corporation

The original Central Galactic Floopy Corporation's Galaxynet server from Discussion 2 had an issue where transactions were not properly synchronized. Here, implement a new system with proper synchronization that is not prone to exploitation or deadlock.

In this new system, transactions can now involve multiple accounts and can be performed with `transact(galaxy_net_t *galaxy_net, int *accounts, int num_accounts)`. Accounts are represented with an `account_id` and are assigned in increasing order, starting at 0. `transact` is not thread safe, so users will call `transact_safe` instead.

given account in a `galaxy_net_t` can only be under one transaction at a time, but the system should allow multiple transfers that involve different acccounts to run concurrently.**Any given account in a `galaxy_net_t` can only be under one transaction at a time, but the system should allow multiple transfers that involve different acccounts to run concurrently.**

For example, say we want to initalize a `galaxy_net_t` system with 10 total accounts and wish to perform a transaction involving accounts 3, 7, 1, and 2. We would do the following:

```
galaxy_net_t *net = init_galaxy_net(10);

int accounts[4] = {3, 7, 1, 2};

transact(galaxy_net, accounts, 4); // we can also call transact_safe here
```

In the following problems, assume that you have `lock_acquire(lock_t *lock)`, and `lock_release(lock_t *lock)` to manipulate locks, and `lock_init(lock_t *lock)` to init a lock. You may also find `calloc()` or `malloc()` useful.

1

Fill in missing lines of code for data structure definitions (you do not have to fill all the lines):

```
typedef struct galaxy_net {

    // other members (not relevant)

    int num_accounts;

    ----------------------------------------

    ----------------------------------------

} galaxy_net_t;
```

2

Fill in missing lines of code for `init_galaxy_net()` (you do not have to fill all the lines):

```
galaxy_net_t *init_galaxy_net(int num_accounts) {

    galaxy_net_t *galaxy_net = malloc(sizeof(galaxy_net_t));

    // init other members (not relevant)

    galaxy_net->num_accounts = num_accounts;

    ----------------------------------------

    ----------------------------------------

    ----------------------------------------

    ----------------------------------------

    return galaxy_net;

}
```

3

Implement `transact_safe` such that it is thread-safe and properly synchronized, following the system description above. Make sure your implementation is not prone to deadlock. Runtime is not an issue. (You do not have to fill all the lines.):

```
void transact_safe(galaxy_net_t *galaxy_net, int *account_ids, int num_accounts) {

    ----------------------------------------

    ----------------------------------------

    ----------------------------------------

    ----------------------------------------

    ----------------------------------------

    ----------------------------------------

    ----------------------------------------

    ----------------------------------------

    transact(galaxy_net, account_ids, num_accounts);

    ----------------------------------------

    ----------------------------------------

    ----------------------------------------

    ----------------------------------------

    ----------------------------------------

    ----------------------------------------

    ----------------------------------------

    ----------------------------------------

}
```

b) **(6.0 points) Page Replacement**

For the following problem, assume a hypothetical machine with 4 pages of physical memory and 7 pages of virtual memory. Given the access pattern:

E]G D C A D F F G A B D F A F B A

E

Indicate which virtual pages remain resident in memory after completing the access pattern, for each of the following policies. This is equivalent to filling out the entire table, and providing the last column of the table as your final answer. You may copy the tables below onto scratch paper and fill them out, but you will only be graded on the final answer you provide.

We have given the FIFO policy as an example. If there are any ties, break them numerically, or alphabetically. When allocating a new virtual page, if there are multiple places the page can go, choose the lowest number physical page. When choosing a virtual page to evict, if any of them are equally good to evict, evict the virtual page with the smallest letter.

Format your final answer as a sequence of 4 capital letters, with no spaces. The first letter is the virtual page stored at the first physical page, the second letter is the virtual page stored at the second physical page, and so on. We also expect you to calculate the number of hits for each access pattern. Format your final answer as an integer, with no spaces.

FIFO

	G	D	C	A	D	F	F	G	A	B	D	F	A	F	B	A	E
1	G	G	G	G	G	F	F	F	F	F	F	F	A	A	A	A	A
2		D	D	D	D	D	D	G	G	G	G	G	G	F	F	F	F
3			C	C	C	C	C	C	C	B	B	B	B	B	B	B	E
4				A	A	A	A	A	A	A	D	D	D	D	D	D	D

Resident Pages (FIFO): AFED

Number of Hits (FIFO): 6

G	**D**	**C**	**A**	**D**	**F**	**F**	**G**	**A**	**B**	**D**	**F**	**A**	**F**	**B**	**A**	**E**

1

MIN

	G	**D**	**C**	**A**	**D**	**F**	**F**	**G**	**A**	**B**	**D**	**F**	**A**	**F**	**B**	**A**	**E**
1																	
2																	
3																	
4																	

Resident Pages (MIN):

2

Number of Hits (MIN):

3

LRU

	G	**D**	**C**	**A**	**D**	**F**	**F**	**G**	**A**	**B**	**D**	**F**	**A**	**F**	**B**	**A**	**E**
1																	
2																	
3																	
4																	

Resident Pages (LRU):

4

Number of Hits (LRU):

c) (4.0 points) Banker's Algorithm

Suppose we have the following resources `{A, B, C}` **and threads** `{T1, T2, T3, T4}`**.**

The total number of each resource available in the system is:

Total

A	B	C
11	13	12

The threads have maximum resource requirements and current allocation of resources as follows:

Currently Allocated

Thread ID	A	B	C
T1	1	2	1
T2	0	2	0
T3	4	3	0
T4	3	0	5

Maximum Required

Thread ID	A	B	C
T1	5	9	7
T2	0	3	0
T3	7	5	2
T4	10	8	10

1

If the system is in a safe state give a non-blocking sequence of thread executions. If no such sequence exists, write 'N/A' and provide a proof that the system is unsafe.

Answer Format: if you believe a correct execution order is (Thread 1, Thread 4, Thread 2, Thread 3), for example, input your answer in the format `[T1, T4, T2, T3]`**. Otherwise, write 'N/A', followed by a proof of why there is no such sequence.**

first*Break ties by choosing the thread with a lower ID to execute first.*

d) (9.0 points) Scheduling Potpourri

Here is a table of processes and their associated arrival and running times.

Name	Arrival Time	CPU Running Time
A	0	2
B	1	6
C	4	1
D	7	4
E	8	3

Show the scheduling order for these processes under 3 policies: First Come First Serve (FCFS), Shortest-Remaining-Time-First (SRTF), Round-Robin (RR) with quantum = 2. Assume that context switch overhead is 0.

Incoming jobs are appended to the back of the ready queue. However, if an existing job is preempted on a time slice, it will be processed before incoming jobs, which will be added after the originally running job is preempted and added to the back of the ready queue.

Priorities correspond to the lexicographical value of a job's name, so "John" has lower priority than "Kubi". When breaking ties, let the job with lowest priority win.

Please put your final answers for each scheduling algorithm in the corresponding answer slots below.

the following form:Your answer MUST take EXACTLY the following form:

`A, A, D, C, B, B, E,..., A`

with a single comma then single space delimiting the job at each time slice. We recommend you write down your answers on paper in their entirety in something like the following tabular format. Copy them over *carefully*.

Time	FCFS	SRTF	RR
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			

Time	FCFS	SRTF	RR
10			
11			
12			
13			
14			
15			

1

First Come First Serve:

2

Shortest-Remaining-Time-First:

3

Round Robin:

5. (19.0 points) Address Translation

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

Virtual Page #1	Virtual Page #2	Offset
10 bits	10 bits	12 bits

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

Physical Page #	Offset
20 bits	12 bits

Page table entries (PTE) are 32 bits in the following format, **stored in big-endian form** in memory (i.e. the MSB is first byte in memory):

Physical Page #	OS Defined	0	0	Dirty	Accessed	Nocache	Write Through	User	Writable	Valid
20 bits	3 bits	1 bit	1 bit	1 bit	1 bit	1 bit	1 bit	1 bit	1 bit	1 bit

Here, 'Valid' means that a translation is valid, 'Writeable' means that the page is writeable, 'User' means that the page is accessible by the User (rather than only by the Kernel).

a) Physical Memory

For the following two questions, please use the memory contents below.

Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+a	+b	+c	+d	+e	+f
0x0002f210	26	00	9d	25	00	65	ed	b2	47	c5	f3	10	00	00	5e	06
...																
0x00030070	ef	00	94	ca	01	3f	7e	00	00	a7	b3	ee	00	02	f0	67
...																
0x00040000	00	08	30	5f	83	00	4f	d1	64	d1	27	5c	00	44	00	00
0x00040010	00	04	40	7f	00	82	59	90	50	c2	6b	86	1c	8a	28	4a
...																
0x00040060	d1	b9	18	db	a7	00	00	c1	00	03	00	4d	35	00	e5	00
0x00040070	00	04	40	0f	4a	97	86	b8	e8	a8	00	00	13	00	28	4c
...																
0x00044010	00	c7	c8	2d	b2	05	2e	78	31	12	d1	9d	00	0a	10	4f
...																
0x00044060	c1	00	02	83	0c	7c	00	bf	00	00	5f	41	00	0a	10	57
...																
0x00083020	02	00	b2	7e	00	f7	00	00	e2	e1	61	f3	00	0d	f0	3b
0x00083030	00	0d	e0	17	00	0f	00	ab	73	00	77	6f	44	7d	00	42
...																

Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+a	+b	+c	+d	+e	+f
0x000a1b50	00	73	00	00	b2	cb	00	8c	a0	be	00	2b	29	4a	f1	e9
0x000a1b60	be	00	3e	21	1d	d2	32	64	e4	00	00	67	3d	00	e7	28
...																
0x000def20	00	02	eb	78	d0	f4	41	4c	4a	fa	4a	20	77	00	00	ff
...																
0x000dff10	60	00	96	a5	45	d3	23	99	e3	23	00	04	85	a7	00	76

b) Translation Table

The base table pointer for the current **user level process** is `0x00040000`. Translate the following virtual addresses to physical addresses, using the memory contents given above. We have filled in some of the boxes for you; you should fill in the boxes marked *blank*.

with `0x`. (Hint: remember that hexadecimal digits contain 4 bits!)**All entries should be in hexadecimal, beginning with `0x`. (Hint: remember that hexadecimal digits contain 4 bits!)**

Virtual Address	VPN #1	VPN #2	First-Level PTE	2nd-Level Page Table Address	Second-Level PTE	Physical Address
`0x0000cf2b`	`0x0`	***A***	***B***	`0x00083000`	`0x000de017`	`0x000def2b`
`0x01007b63`	`0x4`	`0x7`	***C***	***D***	***E***	***F***
`0x0681f213`	***G***	`0x1f`	***H***	***I***	`0x0002f067`	***J***
`0x0701bb5b`	`0x1c`	`0x1b`	***K***	`0x00044000`	***L***	***M***

1

Blank A (Row 1, VPN #2)

2

Blank B (Row 1, First-Level PTE)

3

Blank C (Row 2, First-Level PTE)

4

Blank D (Row 2, 2nd-Level Page Table Address)

5

Blank E (Row 2, Second-Level PTE)

6

Blank F (Row 2, Physical Address)

7

Blank G (Row 3, VPN #1)

8

Blank H (Row 3, First-Level PTE)

9

Blank I (Row 3, 2nd-Level Page Table Address)

10

Blank J (Row 3, Physical Address)

11

Blank K (Row 4, First-Level PTE)

12

Blank L (Row 4, Second-Level PTE)

13

Blank M (Row 4, Physical Address)

c) Instructions

Using the same assumptions and memory contents, predict results for the following instructions. Addresses are virtual. The return value for a `load` is an 8-bit data value (which should be written in hexadecimal), or an error. The return value for a `store` is `ok`, or an error. Possible errors are: `invalid`, `read-only`, `kernel-only`.

Instruction	Result
`Load 0x0701bb5b`	`0x2b`
`Store 0x01007b63`	`ok`
`Store 0x0681f213`	`ERROR: read-only`
`Store 0x0000bf19`	***N***
`Load 0x01007b5c`	***O***
`Test-And-Set 0x0681f210`	***P***

1

Blank N (`Store 0x0000bf19`)

2

Blank O (`Load 0x01007b5c`)

3

Blank P (`Test-And-Set 0x0681f210`)

6. Reference Sheet

```
/****************************** Threads ******************************/

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,

                      void *(*start_routine) (void *), void *arg);

int pthread_join(pthread_t thread, void **retval);

int pthread_mutex_init(pthread_mutex_t *restrict mutex,

                      const pthread_mutexattr_t *restrict attr);

int pthread_mutex_lock(pthread_mutex_t *mutex);

int pthread_mutex_unlock(pthread_mutex_t *mutex);

int sem_init(sem_t *sem, int pshared, unsigned int value);

int sem_post(sem_t *sem);

int sem_wait(sem_t *sem);


/***************************** Processes *****************************/

pid_t fork(void);

pid_t wait(int *status);

pid_t waitpid(pid_t pid, int *status, int options);

int execv(const char *path, char *const argv[]);


/************************** High-Level I/O ***************************/

FILE *fopen(const char *path, const char *mode);

FILE *fdopen(int fd, const char *mode);

size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);

size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);

int fclose(FILE *stream);


/******************************* Sockets *****************************/

int socket(int domain, int type, int protocol);
```

```
int bind(int sockfd, struct sockaddr *addr, socklen_t addrlen);
int listen(int sockfd, int backlog);
int accept(int sockfd, structure sockaddr *addr, socklen_t *addrlen);
int connect(int sockfd, struct sockaddr *addr, socklen_t addrlen);
ssize_t send(int sockfd, const void *buf, size_t len, int flags);

/************************** Low-Level I/O ****************************/
int open(const char *pathname, int flags);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
int dup(int oldfd);
int dup2(int oldfd, int newfd);
int pipe(int pipefd[2]);
int close(int fd);

/****************************** PintOS *********************************/
void list_init(struct list *list);
struct list_elem *list_head(struct list *list);
struct list_elem *list_tail(struct list *list);
struct list_elem *list_begin(struct list *list);
struct list_elem *list_next(struct list_elem *elem);
struct list_elem *list_end(struct list *list);
struct list_elem *list_remove(struct list_elem *elem);
bool list_empty(struct list *list);
#define list_entry(LIST_ELEM, STRUCT, MEMBER) ...
void list_insert(struct list_elem *before, struct list_elem *elem);
void list_push_front(struct list *list, struct list_elem *elem);
```

```
void list_push_back(struct list *list, struct list_elem *elem);

void sema_init(struct semaphore *sema, unsigned value);

void sema_down(struct semaphore *sema);

void sema_up(struct semaphore *sema);

void lock_init(struct lock *lock);

void lock_acquire(struct lock *lock);

void lock_release(struct lock *lock);

void *memcpy(void *dest, const void *src, size_t n);

void *memmove(void *dest, const void *src, size_t n);
```

No more questions.