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

Fall 2017 Ion Stoica

**Second Midterm Exam**
October 23, 2017
CS162 Operating Systems

| | |
|---|---|
| **Your Name:** | Kazuto Kirigaya (because Kirito is always right) |
| **SID:** | 13371337 |
| **TA Name:** | Asuna Yuuki |
| **Discussion Section Time:** | 7-8pm, Funday |

General Information:
This is a **closed book and one 2-sided handwritten note** 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 |
|---|---|---|
| **1** | **24** | |
| **2** | **18** | |
| **3** | **18** | |
| **4** | **20** | |
| **5** | **20** | |
| **TOTAL** | **100** | |

**P1** (24 points total) True/False and Why? **CIRCLE YOUR ANSWER**. For each question: 1 point for true/false correct, 2 points for explanation. An explanation cannot exceed 2 sentences.

a) Adding a TLB will always improve memory access latency.

TRUE FALSE

Why?

Any workload that doesn't visit the same page twice within the TLB's capacity (or less if N-way associative) will make the workload perform worse than not having a TLB. We also accepted answers explaining on single TLB misses, the added overhead of the TLB check would result in worse latency.

b) Memory protection can be implemented without using an address translation mechanism.

TRUE FALSE

Why?

Each process can be associated a BaseAddr and a LimitAddr, and the CPU or OS can check every access of the process whether it falls between BaseAddr and LimitAddr (see slide 15 in lecture 12).

c) Segmentation does not change the value of the bits in the offset field of the virtual address when translating to a physical address.

TRUE FALSE

Why?

With segmentation we add base to virtual address, which can modify the offset (unlike in paging).

d) Two distinct page tables can have the same page table entry despite existing in separate processes.

TRUE FALSE

Why?

This happens when processes share pages with each other.

e) Priority scheduling can lead to starvation.

TRUE FALSE

Why?

Yes, if there are enough high priority threads, the lower priority threads will be denied the service for ever.

f) Under demand paging, if we think of memory as a cache for disk, conflict misses are not possible.

TRUE FALSE

Why?

Any page can be placed anywhere in memory, making the "cache" fully-associative.

g) Each syscall has its own unique entry in the interrupt vector in Pintos.

TRUE FALSE

Why?

All syscalls share the single interrupt 0x30, and are distinguished in the syscall handler by the syscall number passed as an argument.

h) Priority donation is a method of preventing deadlocks.

TRUE FALSE

Why?

Priority donation prevents priority inversion, not deadlocks (it does not prevent circular wait).

**P2** (18 points) **Synch Art Online II – Lawyer's Rosario:** In the dining lawyers problem, four lawyers sit around a table, with IDs A-D from left to right. There is a chopstick between each lawyer, so that chopstick a is to the left of lawyer A, and so on. The chopsticks loop around (e.g. the right chopstick of lawyer D is chopstick a). Each lawyer thinks and then eats, repeatedly. To eat, each lawyer needs two chopsticks. Each lawyer can only reach the chopstick immediately to their left and right (e.g., lawyer A needs chopsticks a and b).

| Total resources | | | |
|---|---|---|---|
| Chopstick a | Chopstick b | Chopstick c | Chopstick d |
| 1 | 1 | 1 | 1 |

| Lawyer ID | Current chopstick allocation | | | | Maximum chopstick allocation | | | |
|---|---|---|---|---|---|---|---|---|
| | a | b | c | d | a | b | c | d |
| A | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| B | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| C | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| D | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

a) (5 points) Fill out the maximum chopstick allocation table (aka only the right, shaded half of the table). More precisely, for every lawyer indicate the chopsticks it needs to acquire in order to eat.

b) (5 points) Fill out the current chopstick allocation table (the left half of the table) so that we are in an unsafe state, that is, a state in which none of the lawyers can proceed, so all are starving. In addition, give a sequence of requests that will lead to the unsafe state you filled in in the table. Each request should be a pair of a lawyer and the requested chopstick.

Lawyer A requests chopstick a -> Lawyer B requests chopstick b -> Lawyer C requests chopstick c -> Lawyer D requests chopstick d (aka everyone requests left, then right)

ALTERNATIVELY: Lawyer D requests chopstick a. Lawyer C requests chopstick d. Lawyer B requests chopstick c. Lawyer A requests chopstick b. (aka right, then left)

The table is filled in for the left->right solution, the right->left solution for the table would be valid as well.

c) (5 points) Consider the order of requests that led to the unsafe state at point (b), but now use the Banker algorithm to avoid getting into the unsafe state. Give the order in which the lawyers will finish eating.

C → B → A → D
Alternatively:
B → C → D → A

d) (3 points) Suppose we implemented each lawyer as a thread and we use a critical section for every chopstick (i.e., a chopstick can be picked or dropped only within its critical section). Suppose we do not use the banker's algorithm, and instead detect the deadlock and break it by abruptly "killing" a thread (lawyer). However, to our surprise this doesn't break the deadlock. What happened?

Chopstick acquisition is most likely implemented with a synchronization primitive. Shooting one of the lawyers will leave the system in an inconsistent state. For example, if chopsticks are implemented with semaphores, then the killed lawyer can't up the semaphore that another lawyer is waiting on.

**P3** (18 points) **Address Translation for RAM (& Rem):** Suppose we have split our 64-bit Virtual Address Space as follows:

| Y bits [Seg. Index] | X bits [Table Index] | X bits [Table Index] | X bits [Table Index] | X bits [Page Index] | 12 bits [offset] |
|---|---|---|---|---|---|

a) (2 points) How big is a page in this system?

We have 12 offset bits, so a page can have 2^12 = 4K bytes.

b) (4 points) Assume that the page tables are divided into page-sized chunks (so that they can be paged to disk). How many PTEs are fit in a page table? What is the value of X?

A PTE should contain one address, so its length is 8 bytes (64 bits). A page table has 2^12/2^3 = 2^9= 512 entries.
We also gave full points to the following alternative answer:
If the PTE was assumed to be 7 bytes (as PPN = 52 bits) then X=2^12/7=10 (rounded up from 9.19), and the corresponding number of PTEs = 2^10=1024.

c) (3 points) What is the size of the segment index (Y)? How many segments can the system have?

If answer to part b was X=9, then Y=64-12-4*9=16 bits.
number of segments=2^Y (2^16 = 64K)
If answer to part b was X=10, then Y=64-12-4*10=12
number of segments=2^Y = 2^12 = 4KB

a) (4 points) What would be the format of a PTE (page table entry)? Show the format of including bits required to support the clock algorithm for page replacement and copy-on-write optimizations, i.e., a dirty bit, a valid bit, and used bit. (Assume: 64-bit Physical Address space)

We have 12-bit offset, so in a 64-bit physical address space, we will have 2^64/2^12=2^52 pages. Hence, we will need 52 (64-12) bits to index one of those pages.

| 52 bits [Physical Page Number] | 9 bits [Other/OS] | D | A | V |
|---|---|---|---|---|

b) (5 points) If a user's heap segment is completely full, how much space is taken up by the page tables for this segment? (It is fine if you just provide the numerical expression without calculating the final result.)?

Number of pages taken by the page table across different levels
=1 (top-level) + 2^X (second-level) + 2^X*2^X (third-level) + 2^X*2^X*2^X (fourth-level)
So, the page table for the heap segment in this case will occupy (1 + 2^X + 2^X*2^X + 2^X*2^X*2^X) pages)
And the size of the heap segment will be
= (number of pages in the corresponding page table) * size of a page
= (1 + 2^X + 2^X*2^X + 2^X*2^X*2^X) * 2^12 bytes

**P4** (20 points total) **Caching – Bad Day at Work:** Your company designs caches for servers with 32-bit addresses. You are designing a 1KB cache with 32B blocks, and your hardware designer suggests a 64-way set associative cache.

a) (4 points) Why would 64-way set associativity not make sense for the cache?

The total number of cache blocks is 1KB/32B = 32. Therefore it doesn't make sense to have a 64-way set associative cache, since #cache blocks < associativity. In fact, the cache will already be fully associative if you have 32-way set associativity.

b) (4 points) You decide to design a fully associative cache instead, and have to come up with a replacement algorithm. Given that the common use-case is repeatedly iterating through arrays that are only slightly larger than the cache capacity, which of LRU (Least Recently Used) or MRU (Most Recently Used) policies yield higher hit rates and why? As the names implies MRU replaces the entry that was accessed last. Answer in at most 2 sentences.

MRU; with LRU, we would keep evicting entries earlier in the array, resulting in cache misses for repeated iterations. With MRU, we would evict entries later in the array, leaving most of the earlier array entries in the cache, resulting in better hit rates for repeated iterations.

c) (4 points) Unfortunately, the hardware engineer introduced a bug in the cache where the cache tag comparison always yields a mismatch. How would this bug impact (i) correctness of end-to-end memory lookups and (ii) effective access time for memory lookups, regardless of correctness? Answer in at most 2 sentences.

The bug would not impact the correctness of end-to-end memory lookups, since tag mismatches in the cache be corrected by lookups to the memory.

The bug would increase effective access time, since every access would result in a cache miss.

d) (4 points) Your hardware engineer fixes the bug in (c), but introduces another bug where the cache only compares the first N-1 bits of the N-bit cache tag. How would this bug impact (i) the correctness of end-to-end memory lookups and (ii) effective access time for memory lookups, regardless of correctness? Answer in at most 2 sentences.

The bug would result in an incorrect lookup for a requested address whose tag matches N-1 bits but not all N bits with the tag corresponding to a cache entry.

The bug would reduce effective access time, since it is now twice as likely to find entries in the cache due to partial (and potentially incorrect) tag matches.

e) (4 points) Your hardware engineer fixes the bug in (d) but quite unsurprisingly, introduces yet another bug, where the cache randomly reads the valid bit of any cache entry as 0 with probability p. In other words, if a cache entry is valid, the cache will assume this entry is invalid with probability p. On the other hand, if the cache entry is invalid, the cache will correctly assume it is invalid. The memory access time is 40 ns, cache access time is 10ns, and probability of finding an entry in cache (before reading valid bit) is 0.9. If effective access time (after reading the valid bit) is 41 ns, what is the value of probability p?

Cache hit rate = Probability of finding the entry in the cache * Probability of reading valid bit as 1 = 0.9 * (1-p) = 0.9 - 0.9p

Cache miss rate = 1 - Cache hit rate = 0.1 + 0.9p

Effective Access Time = (0.9 - 0.9p) * 10 + (0.1 + 0.9p) * (10 + 40) = 41
=> p = 0.75

f) (0 points) Would you fire your hardware engineer?

Think of their family. Have mercy.

**P5** (20 points) **Real-Time Exam Scheduling:** Assume 3 threads with period (P) and computation time per period (C) defined as below:

| | Period (P) | Computation Time (C) |
|---|---|---|
| T1 | 5 | 2 |
| T2 | 6 | 2 |
| T3 | 3 | 1 |

Assume all threads arrive at time 0.

a) (5 points) Fill out the table below according to the the Earliest Deadline First scheduling algorithm. As the name implies, EDF schedules the ready thread with the earliest deadline, where a thread is ready, if the requested computation, C, in the current period has not been satisfied yet. Fill out a box with an 'X' if the thread of that row is running at the time of the column. If two threads have the same deadline, break ties in numerical order i.e., T1 takes precedence before T2. A time slice is one time unit (the table below shows the first 15 time slices). Threads can only be preempted at time slice boundaries.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| T1 | | X | X | | | | | X | X | | | | X | X | |
| T2 | | | | X | X | | | | | X | X | | | | |
| T3 | X | | | | | X | X | | | | | X | | | X |

b) (1 point) Were any of the deadlines missed? If so, give the thread(s) and time period(s) during which the deadline was not met.

None were missed.

c) (5 points) Now consider Rate Monotonic (RM), another real-time scheduler which always schedules the thread with the highest ratio C/P. In other words RM is a fixed priority scheduler where the priority of a thread is C/P, and highest priority thread are scheduled first, i.e., T1 is scheduled before T2 if C1/P1 > C2/P2. Given the RM scheduler, fill in the table below assuming the same threads as in (a). Ties are again broken by the thread number with the lower number running first, and threads can be preempted at the time slice boundaries.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| T1 | X | X | | | | X | X | | | | X | X | | | |
| T2 | | | X | X | | | | X | X | | | | X | X | |
| T3 | | | | | X | | | | | X | | | | | X |

d) (1 point) Were any of the deadlines in the previous part missed? If so, give the thread(s) and time period(s) during which the deadline was not met.

Yes. The deadlines of T3 were missed during every time period.

e) (5 points) Now consider Round-Robin (RR), where each thread is scheduled for a time slices in a round-robin fashion, T1, T2, T3, T1, T2, … If a thread is not ready (i.e., its request in a time period has been already satisfied), the thread is simply skipped and the next time slice is allocated to the next ready thread. Fill in the table below assuming the same threads as in (a).

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| T1 | X | | | X | | | X | | | X | | | X | | |
| T2 | | X | | | X | | | X | | | X | | | X | |
| T3 | | | X | | | X | | | X | | | X | | | X |

f) (1 point) Were any of the deadlines in the previous part missed? If so, give the thread(s) and time period(s) during which the deadline was not met.

T1 misses its 3rd period deadline in the time interval [11, 15].

g) (2 points) Assume the three tasks at A are running forever. Does EDF guarantee that all the deadlines will be always met? Why, or why not? No more than 2 sentences.

No. Consider time 3*6*5 = 90. By time 90 we need to schedule T1 18 times so we need 2*18 = 36 time slices, T2 15 times so we need 30 times slices, and T3 30 times so we need 30 time slices. Thus, to meet all their deadlines the threads need to run for a total of 36+30+30 times slices during 90 time slices, which is impossible.