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:	
SID:	
TA Name:	
Discussion Section Time:	

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?

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

TRUE FALSE

Why?

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?

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

TRUE FALSE

Why?

e) Priority scheduling can lead to starvation.

TRUE FALSE

Why?

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

TRUE FALSE

Why?

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

TRUE FALSE

Why?

h) Priority donation is a method of preventing deadlocks.

TRUE FALSE

Why?

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								
B								
C								
D								

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.

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.

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?

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?

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?

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

d) (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)

e) (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.)?

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?

b) (3 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.

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.

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.

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?

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

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 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															
T2															
T3															

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.

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															
T2															
T3															

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.

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															
T2															
T3															

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.

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.