Operating Systems/Architecture Candidacy Exam FA 2021

Name: _____

Instructions: This exam contains a collection of questions from two areas (*OS* and *Arch*), each organized into four sub-topics. Each area has 100 points worth of questions. You will have **3 hours** to answer **60** points worth of questions **from each section**, satisfying the following constraint: ***No more than 60 points within a section may be attempted and, accordingly, no more than 60 points per section will be evaluated.*** *Submission of more than 60 points worth of answers for a section will result in a two point deduction from the score for every extraneous question submitted. Down-selection of >60 points of submitted answers for grading will be performed at random.* Therefore, please try to answer only those questions whose answers you are confident about. Please separate answers to different questions onto distinct pages.

The exam is closed book, so no notes, phones, etc. Good luck!

Categories	**Attempted**	**Score**
OS-1) Virtual Memory (25 pts)		
OS-2) Kernel (25 pts)		
OS-3) Storage (25 pts)		
OS-4) Concurrency and Synchronization (25 pts)		
Arch-5) Caching (25 pts)		
Arch-6) Scheduling, Dependencies, and Hazards (25 pts)		
Arch-7) Performance, Efficiency, and Parallelism (25 pts)		
Arch-8) Multicore and Parallel Processors (25 pts)		
Total	/ 120	

Operating Systems

Virtual Memory

(**VM-1**: 5 points) Do pointers in a typical user-space program represent virtual or physical addresses? Who is going to perform address translation? Explain your answer.

(**VM-2**: 5 points) Typically, a special "dirty" bit is set by the CPU in a page table entry, when there is first write access against a corresponding memory page. What is the benefit of this?

(**VM-3**: 5 points) Ordinarily, when translating a virtual address, only a virtual page number needs to be translated to a physical page number, whereas a virtual page offset is identical to its physical page offset. Please explain how using power-of-2 numbers (e.g., 2^{12} = 4096 = 4KB) for page sizes simplifies the CPU logic.

(**VM-4**: 5 points) Aside from physical addresses, what do page table entries typically contain? Please give a couple of examples.

(**VM-5**: 5 points) What are multi-level page tables? How many levels are needed for a 32-bit CPU with 4KB pages to map 4GB of virtual memory, assuming that each page table entry in every level occupies 4 bytes? Please show step-by-step how you arrived at the answer.

Kernel

(**K-1:** 5 points) To allow user programs execute some functions that reside inside the kernel (e.g., open(), read(), mmap()), the OS provides a special mechanism. What is this mechanism? Is it different from ordinary function calls? Why "yes" or why "no"? Explain your answer.

(**K-2:** 5 points) In typical OSs, when handling an interrupt or trap while the CPU is in user mode, the CPU switches to a dedicated kernel stack. Why cannot the CPU just use the existing stack to execute a corresponding interrupt or trap handler?

(**K-3:** 5 points) In the x86-64 architecture (prior to the discovery of the "Meltdown" bug), an ordinary system call would not need to change page tables while switching from user space to kernel space. Despite this, the CPU would still be able to execute kernel code without violating any kernel-user isolation requirements while in user space. How could that be possible? What mechanisms in page tables enable that? Explain your answer.

(**K-4:** 5 points) Why do OSs need to support interrupts? Why is it problematic for the CPU and OSs to just wait until after I/O requests complete?

(**K-5:** 5 points) Each process typically has its own page table. Why is that the case and where in the OS virtual memory is this page table located? Can a process update its own page table without involving the OS kernel, i.e., directly from user space? Please explain why "yes" or why "no".

Storage

(**SS-1:** 5 points) What is DMA? How can DMA be useful when using with storage devices, e.g., disks? Do DMA engines work with virtual or physical addresses? Please explain.

(**SS-2:** 5 points) Journaling file systems (e.g., ext4) always write metadata to a journal but may support different modes for data (e.g., not writing data to a journal, writing data to a journal, or using some tricks such as ordering data writes). Why cannot they just always write data to a journal and what is the trade-off here? What is the difference between metadata and data in the context of file systems? Please give some specific examples.

(**SS-3:** 5 points) What is the superblock of a file system? What relevant information does it contain? What will happen if the superblock is damaged and no backup copy exists?

(**SS-4:** 5 points) What is the difference between a hard and soft (symbolic) link? Which one of them needs a separate inode?

(**SS-5:** 5 points) What is the /dev filesystem and what does it contain? How does it help implement UNIX's vision of "everything is a file"? Please give some specific examples with respect to storage devices and partition tables.

Concurrency and Synchronization

(**CaS-1:** 5 points) What is the difference between a mutex and a semaphore? Is it possible to replace a mutex with a semaphore? If possible, what value must this semaphore be initialized to?

(**CaS-2:** 5 points) What is preemption? What problems does a non-preemptive scheduler have when used in general-purpose systems?

(**CaS-3:** 10 points) Suppose you have a program with three threads: two threads must obtain two mutexes to access a critical section, and one thread can obtain just one (specific) mutex:

Thread 1	**Thread 2**	**Thread 3**

lock(&mutexA);
lock(&mutexB);
// Access a critical section
unlock(&mutexB);
unlock(&mutexA);
	lock(&mutexB);
	lock(&mutexA);
	// Access a critical section
	unlock(&mutexB);
	unlock(&mutexA);
		lock(&mutexB);
		// Access a critical section
		unlock(&mutexB);

a) [7pts] What is the problem with this program and what potentially can go wrong? Please also give the name of the problem and explain the solution to it. You can assume that by design requirements, Threads 1 and 2 need both mutexA and mutexB, and Thread 3 only needs mutexB. You can also assume that mutexes are not used elsewhere.

b) [3 pts] In the code above, does the order in which unlock() operations are called in Threads 1 and 2 matter with respect to correctness? Why “yes” or why “no”?

(**CaS-4:** 5 points) What is a reader-writer lock? What problem does it solve (compared to a regular lock)? What problem can potentially arise when using read-preferring reader-writer locks?

Caching

a) (5) Consider a cache for a system that does not support any form of virtualization of memory (i.e. no paging, no segments, no translation and a fully physical addressing scheme). The byte-addressable address space for this system spans Z bytes, the cache has S sets, W ways, and a block size of K bytes. Each block in the cache has P associated status bits (e.g. "Valid", "Dirty" "Shared", etc.)

In terms of the above variables, how many bits of storage are needed to implement the tag array for this cache?

b) (5) While having a deeper (e.g. L1, L2, L3) cache hierarchy is generally beneficial, adding additional cache layers is not guaranteed to improve performance. Given an existing uniprocessor data cache hierarchy (for simplicity, ignore instruction caching) with L1 access time $\mathbf{A_1}$ and miss rate $\mathbf{M_1}$, L2 access time $\mathbf{A_2}$ and miss rate $\mathbf{M_2}$, L3 access time $\mathbf{A_3}$ and miss rate $\mathbf{M_3}$, and main memory access time (for simplicity, assume no virtualization "misses" to main memory) **D**, what would the hit rate in an L4 cache have to be in order to improve performance if the L4 access time is **D/2**?

c) (15) For caches with associativity > 1, there is a wide space of possible replacement policies. Consider the possible replacement policies First-in-first-out (FIFO), Last-in-first-out (LIFO), Least-recently-used (LRU), and Least-Frequently-Used (LFU). For each of **three of these policies**, of your choosing:

Provide a capacity, in blocks (assume a fully associative cache), and a specific repeating sequence of block accesses which combine to form a scenario where that policy is the best replacement policy (highest hit rate) among the four policies above.

Scheduling, Dependencies, and Hazards

Consider the following RISC-style (in this case, MIPS) assembly-level representation of adding the values from one singly-linked list of integers A into the values of a second singly-linked list of integers B (terminating as soon as either A or B reaches its end), that returns the running sum of the elements from A that were added into B. Assume that each list node is an 8-byte struct consisting of a 32-bit unsigned integer value and a 32-bit pointer to the next node. (Note that, in MIPS, except for SW[writes to memory] and BEQ/BNE[PC update only] the leftmost identifier is the destination register)

```
1 |       LA   $s0, A              ; $s0 ← &A - pointer to list A
2 |       LA   $s1, sum            ; $s1 ← &sum
3 |       LA   $s2, B              ; $s2 ← &B - pointer to list B
4 |DW0:   XOR  $t0, $0, $0         ; $t0 ← 0 [sum_tmp=0]
5 |FL0:   BEQ  $s0, $0, DONE       ; while(A && B){
6 |       BEQ  $s2, $0, DONE       ;
7 |FL1:   LW   $t1, 0($s0)         ;    tmp_A = *A
8 |       LW   $t2, 0($s2)         ;    tmp_B = *B
9 |       ADD  $t0, $t0, $t1       ;    sum_tmp += tmpA
10|       ADD  $t1, $t1, $t2       ;    ABsum=tmpA+tmpB
11|       SW   $t1, 0($s2)         ;    *B = ABsum
12|       LW   $s0, 4($s0)         ;    A=A->next
13|       BEQ  $s0, $0, DONE       ;    // null check for A
14|       LW   $s2, 4($s2)         ;    B=B->next
15|       BNE  $s2, $0, FL1        ;    } // null check for B
16|DONE:  SW   $t0, 0($s1)         ;    *SUM = sum_tmp
```

a) (5) List all data dependencies and anti-dependencies among the instructions in the sequence starting at line 7 and ending at line 13, inclusive. You may assume that lists A and B are disjoint.

b) (10) Assuming a standard 5-stage (**F**etch-**D**ecode-e**X**ecute-**M**emory-**W**riteback) in-order processor pipeline with all viable forwarding paths and both hazard detection and branch resolution in stage **D**, schedule the instructions, starting from **FL1** (line 7), that would be executed when the length of both A and B = 2, until the completion of the SW at line 16. Assume perfect branch prediction and that all memory accesses are hits.

Provide the cycle by cycle schedule of all instructions that should execute in this sequence. This should take the form of a 2D grid, with time on the X axis (left=old) and instructions on the Y axis (top = old) . Indicate that an instruction completes a particular pipeline stage in a given cycle with the capital letter (FDXMW) associated with that pipeline stage. Indicate an instruction being stalled in a particular cycle with a lower-case letter of the pipeline stage that instruction is currently stalled in. Indicate value forwarding with a vertical arrow between the source pipeline stage and the destination stage, in the cycle in which forwarding occurs.

c) (10) Assume a 2-level branch predictor with a shared global history and per-branch 2-bit {N,n,t,T} saturating hysteresis counters for taken/not-taken decisions, i.e. tables are selected via the global history FIFO value and each table has PC-indexed counters. Assume that the global history is of length 3 and is currently [N,N,N]. Assume that all PC-indexed counters experience no aliasing and are initially in state “t”. Assume two lists of length 4 are being processed. If execution begins at **FL0**, and continues until **DONE**, indicate the sequence of values, predictions, and outcomes for the (active) branch predictors associated with the branch on line 13.

Performance, Efficiency, and Parallelism

a) (10) Consider a modern simultaneously-multithreaded multicore processor with AVX support. Describe how each of ILP, TLP, and DLP are exploited by such a multiprocessor within the execution of a single application. For each dimension of parallelism, to what degree is software responsible for identifying parallelism and to what degree is hardware responsible for identifying parallelism?

b) (5) Exclusive cache inclusion policies allow a multi-level cache hierarchy to store more distinct blocks of data than an inclusive policy. Discuss the tradeoffs in power usage between an inclusive and exclusive cache hierarchy and how which policy is better can depend on access patterns.

c) (5) The parallelism in GPUs is usually described as SIM**T** (single-instruction-multiple-thread) rather than SIM**D** (single-instruction-multiple-data). List two circumstances under which the SIMT and SIMD models diverge.

d) (5) GPUs support substantially higher throughput on data parallel codes than CPUs. Nonetheless, CPUs still continue to update their vector instruction sets with increasingly larger vectors and more complex operations, even on integrated CPU-GPU systems. Why?

Multicore and Parallel Processors

a) (10) Consider a program whose initial (serial) version is 90% parallelizable and 10% serial. Assume that the program is rewritten such that A) the parallel portion can run on any power-of-2 number of cores **N** B) there is new code that must be run, in addition to original program work, to synchronize the parallel portion, which takes (2% *(original serial time) * $\lg_2$**N**) time.

1) For what value of **N** is performance maximized, and what is the speedup relative to the original serial program?

2) If, due to per-core working set reductions, the parallel portion of the code exhibits ½ the CPI of the original serial version, and, by exploiting thermal headroom during serial computations, the core executing the serial portion can use DVFS to increase its clock frequency by 50% without impacting CPI, what would be the new speedup for the same value **N** computed in 1)?

b) (10) A common atomic memory primitive in modern processors is the load-linked/store-conditional (LL/SC) instruction pair.

List four key invariants for the store-conditional instruction in an LL/SC pair to succeed in executing in a cache-coherent chip-multi-processor with directory-based coherence.

c) (5) Consider a bus-based cache-coherent chip-multiprocessor. Describe an access pattern for a multithreaded program running on this processor where a VI coherence protocol will require fewer bus transactions than an MSI coherence protocol. Specify which transitions are occurring in this workload, and why this leads to that relative ordering.