CMPU 334 Quiz1

Fall 2022

Name: ______________________________

Instructions:

This is a closed book, closed notes exam.   No electronic devices, including calculators, are allowed. You have 75 minutes. There are 9 problems and 15 pages to this exam.

Good Luck!

1. (5 Points) The Linux Completely Fair Scheduler (CFS)

Consider Linux's Completely Fair Scheduler (CFS) as described in lecture and in the textbook, on a single core system. Suppose CFS is configured such that:

- The `sched_latency` (scheduling latency) is set to 48 ms.
- There is no `min_granularity` parameter used, so the time slice for a process is determined by the number of runnable processes and the `sched_latency` parameter.
- There are 4 processes currently able to run: A, B, C, and D.
- Processes A, B, C, and D all have the same priority.
- The the processes currently have the following Virtual Runtimes (`vruntime`)

Process	**vruntime**
A	22
B	6
C	32
D	36

The CFS will run the process that has the lowest vruntime. Since all 4 processes have the same priority, the vruntime will be the same as the actual runtime. The time quanta for each process is the sched_latency / number of processes, 48 / 4 = 12. So B runs first, as it has the lowest vruntime. After B runs for 12 ms, another scheduling decision is made. B still has the lowest runtime (18) so it is run again for another 12 ms. After the time quanta, B has vruntime of 30, so now A has the lowest vruntime (22) so it will run next, so on and so forth.

(a) The first process that runs is process ____B_____.

(b) This process runs for ___24_______ ms.

(c) The second process that runs is process ___A_______ (should be different from the previous process).

(d) This process runs for ___12_______ ms.

(e) The third process that runs is process ___B_______ (should be different from the previous process).

(f) This process runs for ___12_______ ms.

(g) The fourth process that runs is process __C_______ (should be different from the previous process).

(h) This process runs for ___12_______ ms.

(i) The fifth process that runs is process __A_______ (should be different from the previous process).

(j) This process runs for ___12_______ ms.

2. (10 points) The TLB

Which of the following are **true** statements about the translation lookaside buffer (TLB). **Circle all true statements**.

(a) Using an address space identifier (ASID) in the TLB prevents having to flush the TLB on a context switch.

Yes, that's why an ASID is used.

(b) The TLB is used to cache pages read from disk.

No, the TLB acts as a cache for page table entries. It has nothing to do with the page cache.

(c) The TLB speeds up virtual address translation.

Yes, the TLB is why we can use page tables with reasonable performance.

(d) If the valid bit in a TLB entry is set to zero, a page fault will occur when that page is accessed.

No, a page fault occurs when the page is on disk, but not in memory. If the valid bit in the TLB is not set, it means the page table entry does not exist for that page, and the page is invalid.

(e) The TLB removes the need for having full-sized page tables.

No, the TLB is only a cache for recently used page table entries.

(f) TLB misses are handled by the hardware in the Intel x86 architecture.

Yes, all OS has to do is place the start of the page table directory in a control register. The hardware automatically walks the page table.

(g) The impact of TLB misses is reduced by having a multi-level page table.

No, the cost of a TLB miss is increased by multi-level page tables due to the increased memory accessed to walk the page table.

(h) The replacement policy for TLB evictions isn't critical for performance due to the high-associativity of the TLB.

No, even in a fully associative TLB, the replacement policy has a huge impact on TLB performance due to the large difference in the cost between a TLB hit and a TLB miss.

(i) Pressure on the TLB is reduced with larger page sizes.

Yes, a larger page size means less page table entries overall, which means fewer evictions.

(j) Protection bits are not stored in the TLB because they are stored in the Page Table Entry (PTE) for a page.

No, they need to be stored in both places. If the entry is in the TLB, the PTE is not consulted, so they need to be in the TLB.

3. (10 points) Processes

Which of the following are **true** statements about processes?. **Circle all true statements**.

(a) A blocked process waiting on I/O moves to running when the I/O is complete.

False, it moves to Runnable to be eventually be scheduled and then made Running

(b) On a single processor machine, there can only be one process in the RUNNING state.

True

(c) Each process has its own page table.

True. This is how memory isolation between processes and the kernel is performed.

(d) New processes are created with the `exec()` system call.

False, new processes are created with the fork() system call.

(e) `fork()` and `exec()` are two separate system calls to make copy on write (COW) easier to implement.

False, separating fork and exec has no bearing on the implementation of COW. Separating fork and exec makes file redirection easy to implement.

(f) A child process can wait on for its parent to terminate by calling `wait()` and passing in the pid of the parent process.

False, a child process can not wait on a parent. However, a parent can wait on its children.

(g) Having processes use virtual memory addresses provides memory isolation between processes.

True.

(h) `fork()` returns the value 0 to the parent process.

False, fork returns the PID of the child to the parent process. It returns 0 to the child process.

(i) When a process calls `exec()` will not return unless there is an error.

True.

(j) The concept of Limited Direct Execution means a process runs directly on the cpu unless it needs to perform a privileged operation.

True.

4. (5 points) Traps

Which of the following are **true** statements about the **trap** instruction? **Circle all true statements**.

(a) A trap instruction causes a switch from user mode to kernel mode.

True, the purpose of the trap is to "trap" into the kernel.

(b) A trap instruction causes a switch from kernel mode to user mode.

False. See above.

(c) The trap instruction is an unprivileged instruction.

False. It is very important that the trap instruction is privileged. If the trap wasn't privileged, there would be no mechanism to stop the running process and enter into the kernel. That is the point of LDE. All unprivileged instructions will run directly on the CPU without any interference from the OS.

(d) The code the trap handler jumps to during is set up by the OS at boot time.

True. The trap table is set at boot time.

(e) The trap is entirely handled by the hardware.

False. The trap handler works with both the hardware and OS.

5. (16 points) Scheduling

(a) If you were most concerned about **fairness**, which scheduler would you pick? **Circle One.**

A. First In, First Out

B. Shortest Job First

C. Shortest Time-to-Completion First

D. Round Robin

(b) If you were most concerned about **turnaround time**, which scheduler would you pick? **Circle One.**

A. First In, First Out

B. Shortest Time-to-Completion First

C. Round Robin

D. Linux CFS scheduler

(c) If you were most concerned about **response time**, which scheduler would you pick? **Circle One.**

A. First In, First Out

B. Shortest Job First

C. Shortest Time-to-Completion First

D. Round Robin

(d). Decreasing the length of time quanta for a round robin scheduler most likely? **Circle all that apply.**

A. Increases the number of context switches

B. Increases fairness

C. Increases the amount of time between when a program becomes runnable after I/O and when it first runs

D. Increases the amount of storage required to implement the scheduler

6 (10 points) MLFQ Scheduling.

Which of the following are **true** statements about the Multi-level Feedback Queue scheduling? **Circle all true statements**.

(a) Priority-boost prevents a process from starving (unable to make any progress).

True. Priority-boost means all processes will be promoted to the highest priority queue which guarantees all processes will get a chance to run.

(b) When a job enters the system it is placed in the lowest priority (the bottom most queue).

False. It is placed in the highest priority queue.

(c) If process A is in a higher priority queue than process B, A runs and B does not.

True.

(d) If A and B are the same priority, they are run in proportion to the number of tickets allocated to each process.

False. Tickets are not used in MLFQ.

(e) Processes are moved to a lower priority queue once its time allotment is up, regardless of how many times it has given up the CPU (e.g., because of an I/O request).

True.

(f) It is a common optimization to have lower priority queues have a longer time quanta than higher priority queues.

True. Lower priority processes tend to be CPU bound (non-interactive) and benefits from running longer.

(g) One issue with MLFQ is that if a process changes from being CPU-bound to interactive, MLFQ can not adjust to this because processes can only move from a higher-level queue to a lower-level queue.

False. Priority boost moves a process to the highest priority queue.

(h) Periodically, the priority of each process is boosted by moving it up one level, unless it was already at the highest level. In that case, the process stays in the highest priority queue.

False. Priority boost moves a process to the highest priority queue.

(i) A design goal for MLFQ scheduling is to minimize response time for interactive jobs while minimizing turn around time for processes.

True.

(j) A process is in exactly one queue at any given time.

True

7. (10 points) Memory

Which of the following are **true** statements about memory? **Circle all true statements**.

(a) **`malloc()` returns a null pointer if it can not fulfill the request.**

True.

(b) **`malloc()` returns a pointer to the start of the memory that was allocated.**

True.

(c) `malloc()` returns the actual number of bytes you were allocated.

False. See above.

(d) **`free()` takes a pointer previously returned by malloc.**

True.

(e) `free()` will return an error if you free the same pointer twice.

Fasle. Unfortunately, it will not, but it could possibly corrupt your heap.

(f) Using `free()` is recommended but not strictly necessary because C will eventually garbage collect any memory that is no longer in use by a process.

False. C does not have garbage collection.

(g) **When using segments to implement virtual memory, bounds checking can be done in either the physical address space or the virtual address space.**

True. Both will work

(h) **Larger page sizes may lead to more internal fragmentation.**

True.

(i) One advantage of segments over page tables is that allocating segments leads to less external fragmentation than allocating pages.

False. Segments suffer from external fragmentation, while pages do not (because all pages are the same size).

(j) **It is more expensive to evict dirty pages than clean pages.**

True. Dirty pages must be written out to disk, while clean pages do not.

8. (8 points) Odds and Ends

Which of the following are **true** statements? **Circle all true statements**.

(a) **An OS mechanism generally answers the "how" question.**

True.

(b) An example of a policy is using a context switch to stop one process running to run another process.

False. A context switch is a mechanism. It's how we switch processes.

(c) **The clock algorithm tries to approximate an LRU algorithm but with lower overhead.**

True.

(d) **The VAX/VMS virtual memory system is an example of a hybrid system.**

True. It uses both pages and segments.

(e) A multi-level page table speeds up PTE lookups in the event of a TLB miss.

False. A multi-level page table slows down PTE lookups because each level requires another memory address. However, it greatly reduces the space needed to store the page table.

(f) **A mult-level page table is typically much smaller than a linear page table.**

True.

(g) An example of external fragmentation is when a program does not use all the memory in a page that was allocated to it.

False. This is an example of internal fragmentation.

(h) The OS typically terminates a process when the process generates a page fault.

False. A page fault occurs when a valid address is not in physical memory. A page fault will cause the page to be read in from the disk, and the instruction will be restarted, where it will then succeed.

9. (26 points) Address Translation

The following problem concerns the way virtual addresses are translated into physical addresses.

- The memory is byte addressable.
- Virtual addresses are 20 bits wide.
- Physical addresses are 18 bits wide.
- The page size is 1024 bytes.
- The TLB is 8-way set associative, with 16 total entries.

In the following tables, **all numbers are given in hexadecimal**. The contents of the TLB and the page table for the first 32 pages are as follows:

TLB			
Index	**Tag**	**PFN**	**Valid**
0	0F2	C6	1
	0DC	48	1
	036	58	0
	00C	3D	1
	03B	8D	0
	113	9E	1
	0C5	80	0
	02E	9A	0
1	0C5	93	0
	113	A3	1
	00C	61	0
	0F2	75	1
	0DC	CD	0
	02E	E1	0
	036	9C	1
	03B	81	0

Page Table (first 32 pages only)					
VPN	PFN	Valid	VPN	PFN	Valid
00	8B	1	10	84	1
01	D8	0	11	70	1
02	E4	1	12	09	0
03	AC	0	13	23	0
04	A4	1	14	61	0
05	E4	0	15	9E	1
06	2E	1	16	8F	0
07	03	0	17	A1	0
08	58	1	18	A0	1
09	1F	1	19	25	1
0A	8B	1	1A	75	0
0B	C9	1	1B	CA	1
0C	29	0	1C	11	0
0D	5E	1	1D	09	0
0E	D8	0	1E	33	0
0F	BF	1	1F	8C	1

1. (3 points) How many total **physical** frames (PFNs) are there in this system? __**256**___________________

2. (3 points) How many total **virtual** pages (VPNs) are there in this system? ____**1024**___________________

For the given virtual address, indicate the TLB entry accessed and the physical address below. Indicate whether the TLB misses and whether a page fault occurs.

If there is a page fault, enter "N/A" for "PFN" and put N/A for the physical address.

Virtual address : `0x793D5`

3. (2 points) Virtual address binary (one bit per box)

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**1**

4. (6 points) Address translation:

Parameter	**Value**
VPN	**0x1E4**
TLB Index	**0x0**
TLB Tag	**0xF2**
TLB Hit? (Y/N)	**Y**
Page Fault? (Y/N)	**N**
PFN	**0xC6**

5. (2 points) Physical address: **0x31BD5**______________________

For the given virtual address, indicate the TLB entry accessed and the physical address below. Indicate whether the TLB misses and whether a page fault occurs.

If there is a page fault, enter "N/A" for "PFN" and put N/A for the physical address.

Virtual address : `0x6716`

6. (2 points) Virtual address binary (one bit per box)

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**

7. (6 points) Address translation:

Parameter	**Value**
VPN	`0x19`
TLB Index	`0x1`
TLB Tag	`0xc`
TLB Hit? (Y/N)	`N`
Page Fault? (Y/N)	`N`
PFN	`0x25`

8. (2 points) Physical address: `0x9716`______________________