CS502 - Operating Systems
2022 Spring Midterm Exam
Maximum Points: 112.

High-level Instructions

Time Limit: 1 hours 20 mins. The exam will start at 4PM. Canvas site will close at 5:20PM. Please plan ahead to submit.

- There are seven total numbered pages, with 8 questions.
- Please read all questions carefully!
- Please use the provided answer sheet (**2022S_midterm_worksheet.txt**) to answer the following questions. It is recommended that you do the work **locally** in your computer and save periodically.
- Please submit the worksheet via Canvas Midterm Page.
- Open-book policy: **You are free to reference any resources you can find yourself.** However, during the exam, you **must not** engage in any form of communication with other people, regardless of whether they are currently enrolled in this class. Any questions and concerns should be directed to Prof. Guo.
- Good luck!

Midterm Exam: Questions

Question 1: Processes

(10 Points) We have a simple program that looks like as follows:

```
1  int main(int argc, char *argv[]) {
2    char *str = argv[1];
3    while (1)
4      printf("%s", str);
5    return 0;
6  }
```

The programs were invoked as follows, assuming the `&&` operator as per Project 2 Shell Project:

```
1  wshell> main a && main b
```

Recall, this operator `&&` tells the shell to execute the first command and then immediately execute the second command.

Below are possible (or impossible?) screen captures of some of the output from the beginning of the run of the programs. Which of the following are possible? **To answer:** Fill in **A** for possible, **B** for not possible.

(a) abababab ...

(b) aaaaaaaa ...

(c) bbbbbbbb ...

(d) aaaabbbb ...

(e) bbbbaaaa ...

Question 2: Fork Exec Wait What?

(15 Points) Remember `fork()`, `exec()`, and `wait()`. In this question, we will examine these system calls' action by looking at a simple program and trace some scenarios.

(a) We have another program that looks like as follows:

```
1    int main(int argc, char *argv[]) {
2      printf("a");
3      fork();
4      printf("b");
5    return 0; }
6
```

Assuming `fork()` might fail (by returning an error code and not creating a new process) and `printf()` prints its outputs immediately (no buffering occurs), what are possible outputs of the program as above? **To answer:** Fill in **A** for possible, **B** for not possible.

1. ab
2. abb
3. bab
4. bba
5. a

(b) Next, we will trace when each of these three system calls is CALLED, and when they RETRUN. **To answer:** write down either "syscall called" or "syscall return" for each scenario. For example, if `fork()` has been called, write down "fork called" and if `fork` returns, write down "fork returned". A particular scenario may have more than one action.

1. Process A is running; it starts to make a child process B.
2. Process A then continues running just after the OS has been told to create B.
3. Process B runs for the first time.
4. Process B now overlays its address space with a new program and starts running in its main().
5. Process A now ensures that it will be notified when B exists, blocking until that is the case.
6. B now creates C.
7. C starts running.
8. C tries to overlay its address space but fails to do so and decides to exit.
9. B runs again, doing some disk I/O.
10. B finally exits and A runs.

Question 3: Limited Direct Execution

(12 Points) This question is about the limited direct execution. Some of these steps are performed by the OS; some are handled by hardware; some are in the user program itself. In the timeline below, which steps are taken by OS, hardware, or user program in this example of a process being created, running, issuing a system call, and exiting. **To answer:** Fill in **A** for OS, **B** for hardware, and **C** for user program.

(a) Create entry for process list.

(b) Allocate memory for program.

(c) Load program into memory.

(d) Setup user stack with argv.

(e) Fill kernel stack with reg/PC.

(f) Call the return-from-trap instruction.

(g) Restore regs from kernel stack.

(h) Switch to user mode.

(i) set PC to main().

(j) Start running in main().

(k) Call a system call.

(l) Call the trap instruction.

(m) Save regs to kernel stack.

(n) Switch to kernel mode.

(o) Set PC to OS trap handler.

(p) Handle trap.

(q) Do work of system call.

(r) Call the return-from-trap instruction.

(s) Restore regs from kernel stack.

(t) Switch to user mode.

(u) set PC to instruction after the earlier trap.

(v) Call exit() system call.

(w) Free memory of the process.

(x) Remove from the process list.

Question 4: MLFQ

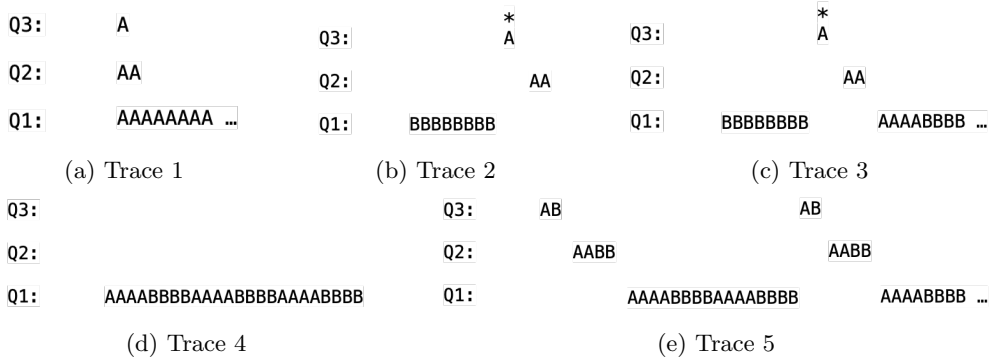
(15 Points) The Multi-level Feedback Queue (MLFQ) is a fancy scheduler that does lots of things. It consists of a number of rules. Here are a list of rules.

(a) Can you identify which rules are actually part of the final MLFQ policy? **To answer:** Write down the rule numbers in the order they appear below.

1. If Priority(A) > Priority(B), A runs (B does not).
2. If Priority(A) < Priority(B), A runs (B does not).
3. If Priority(A) = Priority(B), A and B run in round robin fashion.
4. If Priority(A) = Priority(B), A and B run in first come first serve fashion to completion.
5. When a job enters the system, it is placed at the highest priority (the topmost) queue.
6. When a job enters the system, it is placed in a random queue.
7. When a job enters the system, it is placed in the lowest priority queue.
8. Once a job uses up its time slice at a given level, its priority is reduced.

9. Once a job uses up its time slice at a given level, it moves to the end of the round-robin queue.
10. Once a job uses up its time slice at a given level, it exits.
11. After some time period, move each job up to a higher priority queue.
12. After some time period, move each job down to a lower priority queue.
13. After some time period, move all the jobs in the system to the highest priority (the topmost) queue.

(b) Now, write down the rule (or rules) that come into play in each of the following example traces of MLFQ behavior (use the rule numbers from above). Note that * marks when A arrives, if the information is relevant. **To answer:** Write down the rule numbers.



(a) Trace 1 (b) Trace 2 (c) Trace 3

(d) Trace 4 (e) Trace 5

Figure 1: Five example traces of MLFQ.

Question 5: Segmentation

(12 Points) Segmentation is an approach to supporting virtual memory. In this question, you will examine some timelines of the virtual memory addresses and try to set the base and bounds registers, per segment, correctly so as to **never generate a segmentation fault**, and to make sure that the virtual addresses in the trace get translated to the proper physical addresses.

All other virtual addresses (not seen in the trace) should generate a segmentation fault.

Here, we assume a simple segmentation approach that splits the virtual address space into two segments. The top bit of the virtual address determines which segment it is in. Segment 0 acts like a code and heap segment; the heap grows towards higher addresses. Segment 1 acts like a stack segment; it grows backwards towards lower addresses. For this segment, we follow convention that the book uses: the base register points to the physical address one past the last byte of the stack.

In both segments, the bounds (or limit) register just contains the size of the segment, i.e., the number of bytes valid.

Trace 1: Assume a 16-byte (4-bit) virtual address space. All of the following virtual addresses are valid: 0, 1, 2, 3, 15, 14, 13. We also know that virtual address 1 translates to physical address 101; and virtual address 13 translates to physical address 998.

(a) What is the base for segment 0?

(b) What is the limit for segment 0?

(c) What is the base for segment 1?

(d) What is the limit for segment 1?

Trace 2: Assume a 64-byte (6-bit) virtual address space. All of the following virtual addresses are valid: 0, 1, 63. We also know that virtual address 1 translates to physical address 1001; and virtual address 63 translates to physical address 899.

(a) What is the base for segment 0?

(b) What is the limit for segment 0?

(c) What is the base for segment 1?

(d) What is the limit for segment 1?

Trace 3: Assume a 8-byte (3-bit) virtual address space. All of the following virtual addresses are valid: 0, 1, 2, 3. We also know that virtual address 3 translates to physical address 100.

(a) What is the base for segment 0?

(b) What is the limit for segment 0?

(c) What is the base for segment 1?

(d) What is the limit for segment 1?

Question 6: Reverse Engineering the Page Table

(12 Points) In this question, we consider address translation in a system with a simple linear page table (an array of page table entries, or PTEs). Assume the following setup: **(1)** Virtual address space size is 32KB; **(2)** Page size is 4KB; and **(3)** Physical memory size is 64KB.

Here is a trace of virtual addresses and the physical addresses they translate to (or perhaps an invalid access):

VA	PA
0x1063	0x2063
0x67b4	0x67b4
0x584a	0xe84a
0x4dfe	invalid
0x388a	invalid
0x1c6b	0x2c6b
0x50a9	0xe0a9
0x0bc6	invalid
0x2a9f	0x9a9f
0x742b	invalid
0x4b5e	invalid
0x5597	0xe597

Can you reconstruct the page table entries from the above information? For each entry that you can construct, please do so; otherwise, mark down the entry as "UNKNOWN". The page table entry is formatted as a valid bit followed by the physical frame number (PFN). If valid the bit is 1, the rest of the entry is the PFN. If the bit is 0, the page is not valid. **To answer:** Write down the valid bit, followed by the PFN.

	Valid	PFN
(a) Page table entry 0:		
(b) Page table entry 1:		
(c) Page table entry 2:		
(d) Page table entry 3:		
(e) Page table entry 4:		
(f) Page table entry 5:		
(g) Page table entry 6:		
(h) Page table entry 7:		

Question 7: TLB

(18 Points) TLBs are a critical part of modern paging systems. Assume a TLB that can hold 4 entries, and an LRU (Least-recently-used) replacement policy. Further assume that no context switch happens during these memory accesses.

(a) The first four references to memory are to virtual pages 6, 7, 7, 9. Assuming the next five accesses are to virtual pages 7, 9, 0, 4, 9, which of those will hit in TLB? (and which will miss?) **To answer:** Fill in **A** for cache hits, **B** for misses.

1. virtual page 7
2. virtual page 9
3. virtual page 0
4. virtual page 4
5. virtual page 9

(b) Further assume that the page size is 64 bytes, and you were given a virtual memory address trace, i.e., a set of virtual memory addresses referenced by a program. In which of the following traces will the TLB possibly help speed up execution? You should assume that each trace starts with an empty TLB. **To answer:** Fill in **A** for Speedup, **B** for No Speedup.

1. **Trace A:** 0, 100, 200, 1, 101, 201, ... (repeats in this pattern)
2. **Trace B:** 0, 1000, 2000, 3000, 4000, 0, 1000, 2000, 3000, 4000, ... (repeats)
3. **Trace C:** 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, ... (repeats)
4. **Trace D:** 0, 100, 200, 300, 0, 100, 200, 300, ... (repeats)

Question 8: Multi-level page table

(18 Points) Assume you have a 15-bit virtual address space, with page size of 32 bytes. Assume further a two-level page table, with a page directory which points to pieces of the page table. Each page directory entry is 1 byte, and consists of a valid bit and PFN of the page of the page table. Each page table entry is similar: a valid bit followed by the PFN of the physical page where the desired data resides.

The page directory resides in physical page 18. The following physical page contents are made available to you:

```
page 10:   7f 7f 7f 7f 7f 7f 7f 7f 7f f0 7f a4 7f 7f 7f 7f
           7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f

page 18:   7f c0 ea f9 ed 8b db ba d9 c1 84 8a b3 7f da eb
           9a 85 ab 87 e5 97 b1 df 86 ec e7 ad f2 b9 d5 f8

page 30:   13 1b 03 11 1e 12 16 18 0f 08 12 10 0a 1a 0b 0e
           17 19 1b 14 07 1a 1c 16 17 0f 0f 12 04 14 1a 05

page 57:   a1 7f 7f 7f 7f 7f 7f 9e 7f 7f 7f 7f 7f 7f 7f 7f
           7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f e0 7f 7f

page 90:   7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
           7f e2 7f 7f 7f 7f 7f c7 7f 7f 7f 7f 7f 7f 7f 7f

page 98:   16 0d 18 10 02 0e 01 1c 1d 0a 09 17 06 05 05 0a
           13 1d 06 1d 11 1b 19 04 14 03 00 0c 17 11 05 1a

page 126:  16 0e 14 07 07 01 0c 11 03 05 0c 00 19 05 1c 11
           09 02 13 01 0a 1e 19 16 12 13 17 1b 03 1b 1e 12
```

For example, the first (0th) byte of page 30 is 0x13 and the last (31th) byte of page 126 is 0x12.

(a) In translating virtual address 0x3a3a, which physical pages are accessed?

(b) what is the final value returned?

(c) In translating virtual address 0x74f6, which physical pages are accessed?

(d) what is the final value returned?