# COMP 3000A: Operating Systems

Carleton University
Fall 2021 Midterm Exam

October 19, 2021

There are 15 questions on **2 pages** worth a total of 30 points. Answer all questions in the supplied text file template (available on Brightspace, titled `comp3000-midterm-template.txt`). Please do not corrupt the template as we will use scripts to divide up questions amongst graders. Your uploaded file should be titled `comp3000-midterm-username.txt`, replacing username with your MyCarletonOne username. Make sure you submit a text file and not another format (such as MS Word or PDF). Use a code editor, not a word processor! (You won't be graded for spelling or grammar, just try to make it clear.)

This test is open book. You may use your notes, course materials, the course textbook, and other online resources. If you use any outside sources during the exam, you **must cite** the sources. Your citation may be informal but should be unambiguous and specific (i.e., if you refered to the textbook, indicate what chapter and page you looked at rather than just citing the textbook). You **may not** collaborate with any others on this exam. This exam should represent your own work.

**Do not share this exam or discuss it with others who have not taken it.** Some students will be taking it at other times due to accommodations. Solutions will be released once everyone has finished th exam.

All explanations should be concise and to the point (generally no more than a few sentences, sometimes much less). If you find a question is ambiguous, explain your interpretation and answer the question accordingly.

You have 80 minutes. Good luck!

1. [2] Can you make a regular file that consistently behaves the same as /dev/null? Why or why not? Explain.

2. [2] What is a C statement or declaration that could have generated the following assembly language code? Explain how each line is accounted for in your C code.

```
defaultName     .quad    .LC0
.LC0            .string  "Jane"
```

3. [2] On x86-64 systems (the type of systems we have been using in class), is it possible for the stack and heap to grow to the extent that they would both run into each other? Do you think this is likely to happen in practice? Explain.

4. [2] Describe the data structure that holds command line arguments. Is it a simple array? Be precise using the C type definition and explaining what it means.

5. [2] A friend of yours says that environment variables are an inefficiently stored key-value store. Is your friend correct? Explain briefly.

6. [2] Is it easy for a program to change a value in its argv array? Is it similarly easy to add elements to argv? Explain briefly.

7. [2] As shown by 3000memview, memory layout is randomized every time a program is run. We can disable this by using setarch. If we do an strace of `setarch -R`, we notice it makes the following system call just before execve'ing a program:

   ```
   personality(PER_LINUX|ADDR_NO_RANDOMIZE) = 0 (PER_LINUX)
   ```

   What does this system call tell us about what part of the system randomizes process memory layout? Explain briefly.

8. [2] You're writing a program on a new version of Linux that has a new system call, `fastread`. Libraries have not been updated to support `fastread`. Can you make a pure, standards-compliant C program that calls the `fastread` system call? Why or why not?

9. [2] Which is bigger *in process memory*, a statically-linked binary or a dynamically-linked binary? Why? Ignore any sharing of memory between processes or other multi-process memory saving techniques.

10. [2] `3000shell` can make many stat system calls for every command entered. What is the maximum number of stat calls `3000shell` will make in a given situation (while executing a command)? Explain.

11. [2] System calls are much more expensive than function calls (in terms of execution time and in other ways). At a high level, how does the C library minimize the number of write system calls it makes by default? And, what is one way you can force C library functions to make more write system calls?

12. [2] In the C standard library, there are multiple functions for outputting formatted string data from a program. Here is the declaration of four of them:

    ```
    int printf(const char *format, ...);
    int fprintf(FILE *stream, const char *format, ...);
    int dprintf(int fd, const char *format, ...);
    int sprintf(char *str, const char *format, ...);
    ```

    What is the difference between these four functions? Are any of them inherently less portable than the others?

13. [2] What standard library function is used to associate a signal with a signal handler? Does this library function actually call the signal handler? Explain briefly.

14. [2] Tools like `gdb` and `strace`, that use the ptrace system call, have several significant limitations compared to `bpftrace` and other tools based on eBPF.

    What is one thing you can do with eBPF that you can't do with ptrace? And, what key restriction is placed on eBPF programs that isn't there for ptrace programs?

15. [2] What system call is used to send a signal? Can signals be sent without using this system call? Explain briefly.