CSCI 4210 — Operating Systems
Lecture Exercise 1 (document version 1.0)
Dynamic Memory Allocation, System Calls, and Files

- This lecture exercise is due by 11:59PM ET on Wednesday, February 3, 2021
- This lecture exercise consists of practice problems and problems to be handed in for a grade; graded problems are to be done individually, so **do not share your work on graded problems with anyone else**
- For all lecture exercise problems, take the time to work through the corresponding video lecture(s) to practice, learn, and master the material; while the problems posed here are usually not exceedingly difficult, they are important to understand before attempting to solve the more extensive homeworks in this course
- As with the homeworks, you **must** use C for this assignment, and all submitted code **must** successfully compile via `gcc` with no warning messages when the `-Wall` (i.e., warn all) compiler option is used; we will also use `-Werror`, which will treat all warnings as critical errors
- All submitted code **must** successfully compile and run on Submitty, which uses Ubuntu v18.04.5 LTS and `gcc` version 7.5.0 (`Ubuntu 7.5.0-3ubuntu1~18.04`)

Practice problems

Work through the practice problems below, but do not submit solutions to these problems. Feel free to post questions, comments, and answers in our Discussion Forum.

1. Run the `static-allocation.c` example from the January 25 video lecture on your own Linux platform (not necessarily Ubuntu). Do you see the same output? If not, why not?

   How would the addition of reading in input from the user potentially mimic the incorrect buffer behavior shown in this example?

2. Run the `dynamic-allocation.c` example from the January 25 video lecture on your own Linux platform (again, not necessarily Ubuntu). Do you see the same output? And if not, why not?

   Also, why does the output appear as shown in the video (and pasted below), in particular starting on the third line of output?

   ```
   path is /csci/goldsd/s21/
   path2 is AAAAAAAAAAAAAA
   path is /csci/goldsd/s21/blah/blah/blah/path is  AAAAAAAAAAAAAA
   /

   path2 is AAAAAAAAAAAAAA
   ```

Graded problems

Complete the problems below and submit via Submitty for a grade. Please do not post any answers to these questions. All work on these problems is to be your own.

1. Study the `reverse()` function shown below to understand how it works.

   ```
   char * reverse( char * s )
   {
     char buffer[1024];
     int i, len = strlen( s );
     for ( i = 0 ; i < len ; i++ ) buffer[i] = s[len-i-1];
     buffer[i] = '\0';
     strcpy( s, buffer );
     return s;
   }
   ```

   Rewrite the `reverse()` function by using dynamic memory allocation for `buffer`. Use `malloc()` for your dynamic memory allocation, and be sure there are no memory leaks.

   Submit this code in a file called `reverse.h` (and only include the function itself).

2. Write a program called `extract.c` that uses `open()`, `read()`, `lseek()`, and `close()` to extract every seventh byte from the input file specified as a command-line argument. Display these individual bytes to `stdout`. And only append one newline `'\n'` character to `stdout` at the end of your program execution.

   As an example, given an input file consisting of the English alphabet in uppercase (plus a newline character), your output should appear as shown below (i.e., the output of `./a.out`).

   ```
   bash$ cat infile.txt
   ABCDEFGHIJKLMNOPQRSTUVWXYZ
   bash$ wc -c infile.txt
   27 infile.txt
   bash$ ./a.out infile.txt
   GNU
   bash$
   ```

   As a hint, use the return value of `read()` to determine when to stop.

What to submit

Please submit two C source files called `reverse.h` and `extract.c`. They will be automatically compiled and tested against various test cases, some of which will be hidden.