CSCI 5573 **Homework Assignment 3** **Spring 2022**

Due Date: 11.59 pm, March 7 (Mon), 2022

1) Disclaimer

This homework is created based on our textbook, Operating Systems: Three Easy Pieces, written by Remzi and Andrea Arpaci-Dusseau at the University of Wisconsin.

2) Goal

In this homework, you will gain some familiarity with memory allocation.
First, you'll write some buggy programs (fun!). Then, you'll use some tools to help you find the bugs you inserted. Then, you will realize how awesome these tools are and use them in the future, thus making yourself more happy and productive. The tools are the debugger (e.g., gdb) and a memory-bug detector called **valgrind**.

3) Requirement

Write all codes with either C or C++, and you must test your code on the CSE department Linux cluster, csegrid.ucdenver.pvt.

4) What to submit & how to submit

a) Answer all questions and add print output for each question.
b) All source codes in tar format, DO NOT INCLUDE *.O and executable image files.
- Please use the following convention when you create a tar file

  ①. Letters of your last name + last 4 digits of your student ID
  ②. e.g., If your name is "James Bond" and your ID is 201812345, then your tar file name is "bon2345.tar".
  ③. If you want to know more about the "tar" command, type "man tar" at Linux prompt

c) Upload all required materials to the class Canvas.

5) Questions

1. First, write a simple program called null.c that creates a pointer to an integer, sets it to NULL, and then tries to dereference it. Compile this into an executable called null. What happens when you run this program?

2. Next, compile this program with symbol information included (with the -g flag). Doing so, let's put more information into the executable, enabling the debugger to access more useful information about variable names and the like. Run the program under the debugger by typing gdb null and then, once gdb is running, typing run. What does gdb show you?

3. Finally, use the valgrind tool on this program. We'll use the memcheck tool that is a part of valgrind to analyze what happens. Run this by typing in the following: valgrind --leak-check=yes null. What happens when you run this? Can you interpret the output from the tool?

4. Write a simple program that allocates memory using malloc() but forgets to free it before exiting. What happens when this program runs? Can you use gdb to find any problems with it? How about valgrind (again with the --leak-check=yes flag)?

5. Write a program that creates an array of integers called data of size 100 using malloc; then, set data[100] to zero. What happens when you run this program? What happens when you run this program using valgrind? Is the program correct?

6. Create a program that allocates an array of integers (as above), frees them, and then tries to print the value of one of the elements of the array. Does the program run? What happens when you use valgrind on it?

7. Now pass a funny value to free (e.g., a pointer in the middle of the array you allocated above). What happens? Do you need tools to find this type of problem?