Ben Williams
CPRE 308
Fridays 10-1

The main purpose of this lab was to introduce basic concepts of debugging, version control software, and principles of Linux to students for a simple refresher before we got into the hardcore operating systems topics. To start it off, we logged into the computers in the OS lab and created a new directory in the filesystem using the mkdir command. This is one of the core commands of the Linux shell, as it allows users to customize the storage to their own liking. We then worked on several common unix commands to re familiarize ourselves with the Unix shell and its operations.

We then created a ssh key pairing with github to create a secure, two way connection that is difficuilt to forge. We accessed the classes Github and downloaded the lab1 repository. We each created our own individual repo on the class Github so that the teacher can see each of our submissions.

On to the lab. In the first portion, we used the GDB (GNU debugging tool.) It is a very powerful debugger that allows single stepping thorugh code, setting breakpoints, and viewing variables. I compiled csv_avg.c using gcc (GNU C compiler) command and ran it using the provided comma delimited file.

We started by inserting a breakpoint at line 46, and then running the code under the debugger. We can now print out the values of variables at run time and check on the status of the stack and buffer whenever we need to. Since this was inside a loop, this was repeated several times. If we wanted to pass to the next stopping point, we can use the continue command to skip past the current break point. Step is another way to move through the code without setting breakpoints.

On part b, I compiled and ran the code for part b, enabling debugging. It threw a seg fault, so I opened up the code to try and figure out what was wrong. Since the code looked and compiled fine, I decided it was a job for the GDB. I got the backtrace from the error and saw that it occurred at line 22. Since it happened at this point, I inserted breakpoints at line 21 and various other places earlier in the program. I looked at the sack frame and realized that the input buffer gets messed up at line 22, which makes me feel like the problem was a memory allocation error. Sure enough, when I went to re look at the code, I saw that the file was attempting to allocate memory to the buffer 1 more time than it should have (=length instead of >length.) I made the simple change and the program ran fine.

We then moved on to the Valgrind Memory tool. It tracks dynamics memory allocations and can detect memory leaks or other memory problems such as buffer overflows. We tested valgrind using the simple malloc test. We saw that the first message indicates that the write of size 1 was invalid (1 byte past the end of the buffer.)

We looked at the heap summary to see the allocates and frees used over the life of the program, and then number of bytes allocated at exit. This eventually led to me discovering how many bytes and blocks are gained or lost when programs start or exit.

I then looked at rand_string.c. I started debugging with the GDB and saw that the loop tried allocating memory past the end of the string (=length instead of >length for the loop conditional.) I changed this and the program ran just fine.