Mark Kim ID: 918204214
Github: mkim797 CSC415 Operating Systems

Assignment 2 – Struct and Buffer

Description:
This assignment is to write a C program that accepts arguments via the command line and then populates a struct provided with personal information and a message. Then it retrieves a set of different length strings which are copied to a buffer, then committed only after the buffer is completely filled.

Approach / What I Did:
After cloning the GitHub repo, I carefully read the assignment2.h file. Within this, noticed the way the typedef was defined. Then, I created the c file and instantiated the struct provided and then populated it with my personal information (taking care to use the executable's arguments to populate my name). This was a relatively simple process. Since constants were provided for the language, I was able to just type in the names of the constants rather than typing the hexadecimal values. I used strncpy() to copy the final argument into the message field of the struct since the size of the destination was a known and static size. Of note is the appending of a null character to the message embedded in the struct so that the end of the string could be found.

The largest part of the assignment was the string copies into a buffer. The way I approached this project was to first examine the string data by iterating through the set to see how it was structured. I then wrote down my thought process and then coded the program. I tested iteratively to ensure that the program behaved correctly. I wished to program this with all edge cases in mind, but failed. The string copy was done through a while loop since the stop condition was a NULL return rather than a string. Within the loop, I used conditionals to determine whether the string was larger than the available space in the buffer. If the string was larger, the head of the string would be appended to the buffer via strncat(). A pointer would be moved to the next character, the full buffer committed, then the rest of the string is copied to the newly cleared buffer with strcpy(). If a string was smaller than the available space in the buffer, the entire string was appended to the buffer via strcat().

Issues and Resolutions:

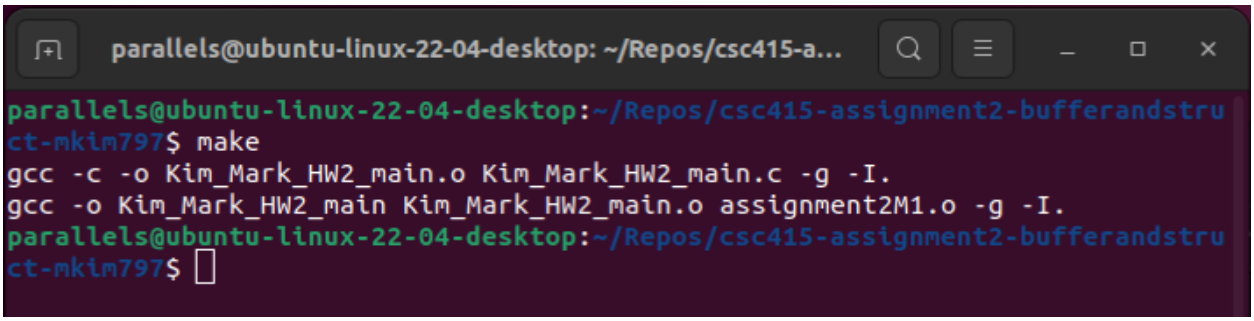
My first issue was a malloc problem from using the incorrect variable for comparison to copy the contents of a string that was longer than the available free buffer space.

The most time consuming and problematic issue was trying to program for the edge case of having a disproportionately large string. I attempted to use nested while loops to iterate through a hypothetical C string that would require multiple commits from the string buffer. I encountered constant errors that I was unable to resolve. Fortunately, the data set did not contain strings that would require such an implementation.

Analysis:

Since I am using a Mac with an M1 chip, the numbers are in Little Endian format. Armed with the knowledge that the typedef of personalInfo only utilizes two pointers (with each being a part of my name), I suspect that the first line of the bytes output is the first and last name. Each takes up 8 bytes, which is consistent with a pointer. The next four bytes "36 AF BA 36" is the little endian form of the hexadecimal number "36 BA AF 36", which matches my student ID number (918204214). Similarly, "9F 0C 44 01" matches the hexadecimal number "01 44 0C 9F" which corresponds to the programming languages I know. Finally, the 4 bytes "03 00 00 00" matches the decimal value 3, which I suspect corresponds with index 3 of the enum and corresponds with my Senior standing with the school. Finally, the rest of the output corresponds with the message.

Screen shot of compilation:



Screen shot(s) of the execution of the program: