Will Young
Github: Woyoung21
ID: 924230057
CSC415 Operating Systems

Assignment 2 – Struct and Buffer

Description:

This assignment is to write a C program that accepts arguments and other information to populate a struct with personalInfo using safe memory principles. Once our struct is populated, we create a buffer and fill it with an unknown number of C-Strings, we also don't know their size. At a high level, we fill the buffer to its capacity of 256 and commit the buffer using a provided function. Due to encapsulation, we don't know the implementation of this function. Once we commit, we start the process all over again until we finally reach NULL which breaks us out of the loop. There are special cases that ensure safe memory practices are followed in the event a C-String is longer than the available buffer size. There is also a final commit, pushing the last chunk of C-Strings out of the buffer. We then call a checkIt() function, which the implementation is also hidden from us. It in turn lets us know that we are successful and prints out the contents of out struct in plain text and also in the form of a hex dump.

Approach:
From the beginning of the assignment, I wanted to build this program in a modular manner, piece by piece. Since the assignment was broken up into steps, it made this a bit easier. Building off of assignment 1, this assignment added some concepts that I needed to brush up on in order to implement them in C. Going forward, I will discuss my approach to each step.

Starting off with step 4, we had to populate each variable in the struct personalInfo. To start this process, we had to use malloc (https://man7.org/linux/man-pages/man3/malloc.3.html) & (https://sourceware.org/glibc/manual/latest/html_mono/libc.html#Memory-Allocation-and-C section 3.2.1.1) to allocate new memory while returning a pointer to the beginning of the allocated memory. Once we had the new struct initialized, I started filling the variable fields with the requested data. First and last name came from the elements 1 & 2 of argv[]. I also assigned my student ID and grade level to the appropriate fields. Then I populated the languages field with all of the computer language I have knowledge of using the | operator (https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html#Bitwise-Logical-Operators section 3.9).

The last part was to fill the message field with the third element of argv[]. This is where I ran into an issue which I will speak more about in that section, but after referencing your guide, "Do note the length of the message field" I cut the size of the incoming message to fit in the hard coded memory you allocated in the header file. I used strlen() (https://man7.org/linux/man-pages/man3/strlen.3.html) to get the size of the incoming message, which confirmed it was larger than the message field. So I truncated the message to the size of the message field -1 and used strncpy() (https://man7.org/linux/man-pages/man3/strncpy.3.html) to copy to the message field from the 3rd element of argv[] at the size of message field -1. I then appended NULL to the last element of the message field. Again,

using debugging print statements and their supplemental variables allowed me to ensure that the right fields were being populated in a memory safe manner.

Moving on to step 5, I used the provided function to “write” the struct newpInfo. To ensure this was executed successfully, I assigned the returned value to a dummy variable and printed that dummy variable. It shows 0, which indicates a successful execution. The print statement clearly shows the results so I can move on to the next step.

Moving onto step 6, I allocated new memory for the buffer at BLOCK_SIZE. Since we are receiving an unspecified number of c-strings and we also don't know how big each one is, we created some while loops and conditional statements to help push these c-strings to the buffer while also making sure we don't run over the allotted memory. There is a special check if the c-string being pushed is bigger than what is available in the buffer. In that instance, the c-string is partially pushed to max out the buffer using memcpy() (https://man7.org/linux/man-pages/man3/memcpy.3.html), the buffer is committed, then the remaining c-string is pushed to the buffer starting at the beginning of the buffer. Once we hit NULL, we exit this loop and whatever is left in the buffer is committed. During step 6, I heavily relied on printf statements to ensure that I could keep track of how many iterations were performed until I reached NULL, as well as the size of the incoming c-string, the current index of the buffer and the amount of buffer remaining. In our rare case where the c-string needed to be split, we also kept track of what was being committed and what was left to be committed. These printf statements helped me visually track each iteration as well as when I finally broke out of the loop because NULL was reached and indicated there were ~21 iterations until NULL was reached.

The last step was calling the checkIt() function which was provided by the professor. It checks our program and provides a success or failure message. After this function is called, I used free on our newpInfo struct and the buffer to release the memory we allocated for this program. This follows our memory safe programming standards.

Issues and Resolutions:

My first issue was when I began filling the message field of my struct. Initially I copied the entire incoming third argument, but this caused some undefined behavior. The incoming argument was roughly 176 characters long, but the hard coded memory allotment only allowed for 100. Because I wrote over the allotted amount, it caused weird, unexpected text to be displayed.

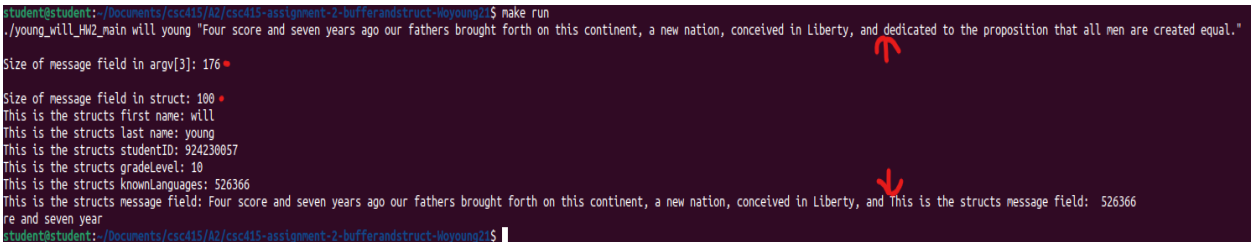

Figure 1: Unexpected Behavior

I resolved it by truncating the third argument to 99 characters and appending NULL to the end. I also added more debugging printf() statements to help track everything as I stepped through resolution.

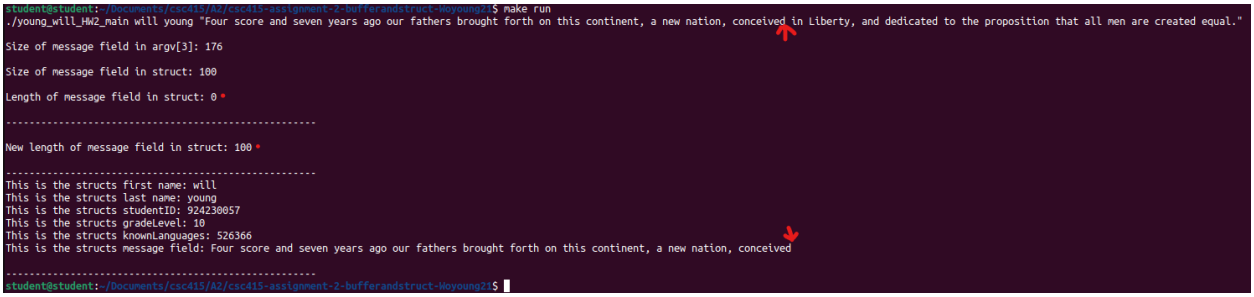

Figure 2: Truncated message

The next issue I had was running checkit() at the end. Initially when I ran it, I got a failure message.

```
student@student:~/Documents/csc415/A2/csc415-assignment-2-bufferandstruct-Woyoung21$ make run
./young_will_HW2_main will young "Four score and seven years ago our fathers brought forth on this
continent, a new nation, conceived in Liberty, and dedicated to the proposition that all men are cr
eated equal."
---------------------------------- CHECK ----------------------------------
Running the check for will young
Name check is 0 by 0
Student ID: 924230057, Grade Level: Senior
Languages: 526366
Message:
Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived

The Check Failed (-32, 1792)

END-OF-ASSIGNMENT
000000: 34 B3 FC 93 FD 7F 00 00  39 B3 FC 93 FD 7F 00 00 | 4????..9????..
000010: A9 A1 16 37 0A 00 00 00  1E 08 08 00 46 6F 75 72 | ??.7........Four
000020: 20 73 63 6F 72 65 20 61  6E 64 20 73 65 76 65 6E |  score and seven
000030: 20 79 65 61 72 73 20 61  67 6F 20 6F 75 72 20 66 |  years ago our f
000040: 61 74 68 65 72 73 20 62  72 6F 75 67 68 74 20 66 | athers brought f
000050: 6F 72 74 68 20 6F 6E 20  74 68 69 73 20 63 6F 6E | orth on this con
000060: 74 69 6E 65 6E 74 2C 20  61 20 6E 65 77 20 6E 61 | tinent, a new na
000070: 74 69 6F 6E 2C 20 63 6F  6E 63 65 69 76 65 64 00 | tion, conceived.

student@student:~/Documents/csc415/A2/csc415-assignment-2-bufferandstruct-Woyoung21$
```

Figure 3: Check Failed

To resolve this problem, I ended up going back through my code and the printf() statements to see what the issue was. It turns out that at the end of the run, before we hit NULL, the buffer is

not empty. It isn't full, which is why it didn't commit based on the logic I created, but it still has some c-strings in there. So I added a quick conditional statement to push the commit after NULL was reached to commit anything left in the buffer. Once I did this, I got a successful run.

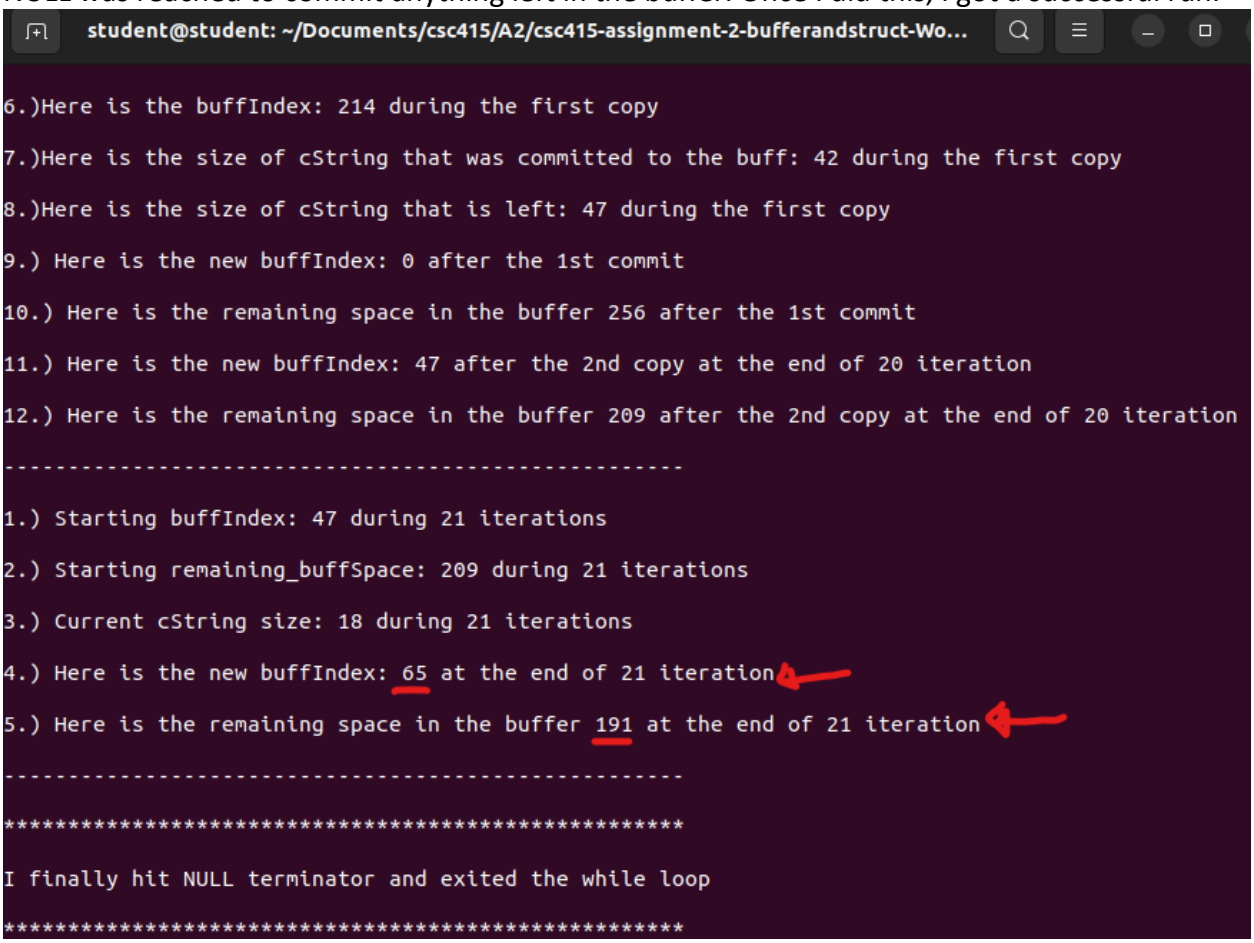

Figure 4: Proof of partial buffer at the end of run

```
student@student:~/Documents/csc415/A2/csc415-assignment-2-bufferandstruct-Woyoung21$ make run
./young_will_HW2_main will young "Four score and seven years ago our fathers brought forth on this
continent, a new nation, conceived in Liberty, and dedicated to the proposition that all men are cr
eated equal."
---------------------------------- CHECK ----------------------------------
Running the check for will young
Name check is 0 by 0
Student ID: 924230057, Grade Level: Senior
Languages: 526366
Message:
Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived

The Check Succeded (0, 0)

END-OF-ASSIGNMENT
000000: 34 93 64 1F FE 7F 00 00  39 93 64 1F FE 7F 00 00 | 4?d.?..9?d.?..
000010: A9 A1 16 37 0A 00 00 00  1E 08 08 00 46 6F 75 72 | ??.7........Four
000020: 20 73 63 6F 72 65 20 61  6E 64 20 73 65 76 65 6E |  score and seven
000030: 20 79 65 61 72 73 20 61  67 6F 20 6F 75 72 20 66 |  years ago our f
000040: 61 74 68 65 72 73 20 62  72 6F 75 67 68 74 20 66 | athers brought f
000050: 6F 72 74 68 20 6F 6E 20  74 68 69 73 20 63 6F 6E | orth on this con
000060: 74 69 6E 65 6E 74 2C 20  61 20 6E 65 77 20 6E 61 | tinent, a new na
000070: 74 69 6F 6E 2C 20 63 6F  6E 63 65 69 76 65 64 00 | tion, conceived.

student@student:~/Documents/csc415/A2/csc415-assignment-2-bufferandstruct-Woyoung21$
```

Figure 5: Successful run after commit of partial buffer

Analysis:
Here is the analysis of the hex dump we got after successful completion of our program. To start out, I color coordinated each struct field below in figure 6, and I will discuss that in depth. Starting at 0x00->0x07, we have the first name field at 8 bytes in size. 8 bytes is the size of the pointer (*) variable, and this is colored red.

Next, we move to 0x08->0x0F, which is the last name field at 8 bytes in size. 8 bytes is the size of the pointer (*) variable, and this is colored in blue.

Next, we move to 0x10->0x13, which is the studentID field at 4 bytes. 4 bytes is the size of the integer variable, and it is colored green. These 4 bytes hold the hex value of 0x3716A1A9 which in decimal is my actual studentID of 924230057.

Next, we move to 0x14->0x17, which is the Grade Level field at 4 bytes. 4 bytes is the size of the Enum variable (also known as named integer constant) and it is colored in purple. In addition, hex value 0x0A at position 0x14 is 10 in decimal, which is the numerical representation of SENIOR.

Next, we move to 0x18->0x1B, which is the languages field at 4 bytes. 4 bytes is the size of the integer variable, and it is colored orange. Starting at 0x18 position, the hex value is 0x0008081E which in decimal is 526366. This is the combined value of the languages I selected.

Lastly, we move to 0x1C->0x7F, which is the message field at 100 bytes. 100 bytes is the allocated memory space for this field and is colored in pink. This is a bit more direct, as starting

at 0x1C position, every byte's hex value correlates to an ASCII value. Using any ASCII table (https://www.asciitable.com/) we can see that 0x46 equals the character F. 0x6F equals the character o, 0x75 equals the character u and 0x72 equals the character r. We can continue to match the remaining characters to their hex equivalent in the ASCII table to form the entire message field, character by character.

```
Address    00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  ASCII

00000000:  34 23 64 EB FD 7F 00 00  39 23 64 EB FD 7F 00 00  4#d.....9#d.....
00000010:  A9 A1 16 37 0A 00 00 00  1E 08 08 00 46 6F 75 72  ...7........Four
00000020:  20 73 63 6F 72 65 20 61  6E 64 20 73 65 76 65 6E   score and seven
00000030:  20 79 65 61 72 73 20 61  67 6F 20 6F 75 72 20 66   years ago our f
00000040:  61 74 68 65 72 73 20 62  72 6F 75 67 68 74 20 66  athers brought f
00000050:  6F 72 74 68 20 6F 6E 20  74 68 69 73 20 63 6F 6E  orth on this con
00000060:  74 69 6E 65 6E 74 2C 20  61 20 6E 65 77 20 6E 61  tinent, a new na
00000070:  74 69 6F 6E 2C 20 63 6F  6E 63 65 69 76 65 64 00  tion, conceived.
```

Figure 6: Hex Dump

Adding all these fields gives you 128 bytes. This size is confirmed as the last memory address is 0x7F, which is 127 in decimal. We add 1 to that since the address starts at 0, and we get 128 in decimal.

Screen shot of compilation:

```
student@student:~/Documents/csc415/A2/csc415-assignment-2-bufferandstruct-Woyoung21$ make
gcc -c -o young_will_HW2_main.o young_will_HW2_main.c -g -rdynamic -I.
gcc -o young_will_HW2_main young_will_HW2_main.o assignment2.o -g -rdynamic -I.
student@student:~/Documents/csc415/A2/csc415-assignment-2-bufferandstruct-Woyoung21$
```

Figure 7: Screen shot of compilation

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

```
student@student:~/Documents/csc415/A2/csc415-assignment-2-bufferandstruct-Woyoung21$ make run
./young_will_HW2_main will young "Four score and seven years ago our fathers brought forth on this continen
t, a new nation, conceived in Liberty, and dedicated to the proposition that all men are created equal."
---------------------------------- CHECK ----------------------------------
Running the check for will young
Name check is 0 by 0
Student ID: 924230057, Grade Level: Senior
Languages: 526366
Message:
Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived

The Check Succeded (0, 0)

END-OF-ASSIGNMENT
000000: 34 E3 CB 10 FC 7F 00 00  39 E3 CB 10 FC 7F 00 00 | 4??.?..9??.?..
000010: A9 A1 16 37 0A 00 00 00  1E 08 08 00 46 6F 75 72 | ??.7........Four
000020: 20 73 63 6F 72 65 20 61  6E 64 20 73 65 76 65 6E |  score and seven
000030: 20 79 65 61 72 73 20 61  67 6F 20 6F 75 72 20 66 |  years ago our f
000040: 61 74 68 65 72 73 20 62  72 6F 75 67 68 74 20 66 | athers brought f
000050: 6F 72 74 68 20 6F 6E 20  74 68 69 73 20 63 6F 6E | orth on this con
000060: 74 69 6E 65 6E 74 2C 20  61 20 6E 65 77 20 6E 61 | tinent, a new na
000070: 74 69 6F 6E 2C 20 63 6F  6E 63 65 69 76 65 64 00 | tion, conceived.

student@student:~/Documents/csc415/A2/csc415-assignment-2-bufferandstruct-Woyoung21$
```

Figure 8: Screen shot of execution