Haley Park
Github: jung-hyeon
ID: 923812276
CSC415 Operating Systems

Assignment 2 – Structure in Memory and Buffering

Description:
This assignment is to allocate an instantiation of the personInfo structure and to populate it then write personal information structure.
Also getting a series of C strings by using string Buffer. When the buffer is filled, flush out Buffer. Plus, reading hexadecimal dump and understanding binary representation of data in memory and how it can be used for debugging and ensuring code works.

Approach:

1. Allocate an instantiation of the personInfo structure and populate it.
2. Initialize and assign values (firstName, lastName, studentID, level, languages and message).
    a. **firstName** is char* type. So, Allocate the memory which size is (size of char) * (argument[1]'s length + 1). The reason why adding plus 1 to length is by '\n'. String must have '\n' at the end.
    b. **lastName** is char* type. So, Allocate the memory which size is (size of char) * (argument[2]'s length + 1). The reason why adding plus 1 to length is by '\n'. String must have '\n' at the end.
    c. **firstName** and **lastName** are assigned by using strcpy. And print their address. (It will be described in **Analysis** part below).
    d. **studentID** is int type. So, Assign decimal number directly.
    e. **level** is enum type (gradelevel). So assign enum gradelevel directly.
    f. **languages** is int type. There are macro values(#define) that are already defined. So, Assign macro values. And to write at least 3 languages, use the '|' operator.
    g. **message** is char array. message's length is limited by 100. So it is assigned by using strncpy.
3. Write personal information structure by calling writePersonalInfo.
4. After using personalInfo struct, deallocate memory of itself and its member variables too if needed.
5. Get a series of C strings.
    a. Allocate buffer memory which size is BLOCK_SIZE.
    b. Gather string in buffer.
        i. Copy the data into the buffer in chunks using memcpy.
    c. CommitBlock if the buffer is full.
        i. If string length does not fit the buffer size, assign overflowing(remaining) string to flush the buffer newly.
    d. Repeat until the string is null.
6. After using the buffer , deallocate memory.
7. Exit main returning the same value as returned from checkIt.

Issues and Resolutions:

1. **Parameter issues (message assignment)**

At first time, I thought I had to write the command line parameter manually. eg) make run RUNOPTIONS=”Haley Park This is a message”
But in this way, argv[3] is “This” not “This is a message”. So I thought that I had to connect the parameters(except lastname and firstname) using a for loop. But in the description of assignment2, we have to use a third command line parameter to assign a message.
As a result, I deleted my RUNOPTIONS, and then the program ran well. I find that RUNOPTIONS already exist in the Makefile.

```
42   # DO NOT CHANGE the RUNOPTIONS
43   RUNOPTIONS=$(FIRSTNAME) $(LASTNAME) "Four score and seven years ago our fathers brought forth o
44
```

I can just use strncpy to assign message.

```
strncpy(pInfo->message, argv[3], MESSAGE_LENGTH);
```

2. Languages assignment issues

Languages variable of personalInfo struct is int type. But in the description of assignment2, we have to specify every language I have Knowledge of and there must be at least three. At first, I didn't know how to assign more than one language in the one variable.
I find that I can use ‘|’ operation because macro is a constant value.

```
pInfo->languages = KNOWLEDGE_OF_C | KNOWLEDGE_OF_JAVA | KNOWLEDGE_OF_CPLUSPLUS;
```

3. Buffer usage issues

At first, I just got a string by calling getNext. But the buffer size is 256. Each string length is different and does not fit on the buffer size.
To use the buffer effectively, I have to flush out the buffer only when it is full. So, in the While loop, I get a string in a row and then, if the buffer is full, commit buffer. If string length does not fit the buffer size, the back part of the string(remaining string) has to be assigned in the buffer after calling commitBlock(after flushing out the buffer). Finally, if getNext() is null, commit the buffer for the remaining string in the buffer.

Analysis:
[Time Analysis]
One for loop: O(N)
One while loop: O(N)
=> Time complexity: O(N)

[hexdump output Analysis]
END-OF-ASSIGNMENT

```
000000: 30 13 4A C2 83 55 00 00  50 13 4A C2 83 55 00 00 | 0.J??U..P.J??U..
000010: B4 41 10 37 14 00 00 00  13 00 00 00 46 6F 75 72 | ?A.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 20 | tion, conceived

1. Bytes at address **000000 - 000007** highlighted PINK represent the field **firstName address** which I know is right and has the value **0x5583C24A1330**
    a. It is represented by Little-endian.
    b. It is right. (I prove it by code)

```
printf("firstName address: %p\n", pInfo->firstName);
```

```
firstName address: 0x55c9014e4330
```

2. Bytes at address **000008 - 00000F** highlighted GREEN represent the field **lastName address** which I know is right and has the value **0x5583C24A1350**
    a. It is represented by Little-endian.
    b. It is right. (I prove it by code)

```
printf("lastName address: %p\n", pInfo->lastName);
```

```
lastName address: 0x55c9014e4350
```

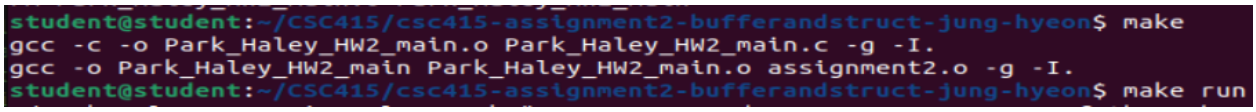
3. Bytes at address **000010 - 000013** highlighted BLUE represent the field **studentID** which I know is right and has the value **0x371041B4**
    a. It is represented by Little-endian.
    b. 0x371041B4 => 923812276(Decimal Number): RIGHT

4. Bytes at address **000014 - 000017** highlighted ORANGE represent the field **gradelevel** which I know is right and has the value **0x14**
    a. It is represented by Little-endian.
    b. 0x14 => 20 (Decimal Number) => Senior: RIGHT

5. Bytes at address **000018 - 00001B** highlighted YELLOW represent the field **languages** which I know is right and has the value **0x13**
    a. It is represented by Little-endian.
    b. 0x13 => KNOWLEDGE_OF_C(0x00000001) + KNOWLEDGE_OF_JAVA(0x00000002) + KNOWLEDGE_OF_CPLUSPLUS(0x00000010) => 19 (Decimal Number) => : RIGHT

6. Bytes at address **00001C - 00007F** highlighted PURPLE represent the field **message** which I know is right and has the value of (below).
    a. value
    46 6F 75 72 => Four
    20 73 63 6F 72 65 20 61 6E 64 20 73 65 76 65 6E => score and seven
    20 79 65 61 72 73 20 61 67 6F 20 6F 75 72 20 66 => years ago our f
    61 74 68 65 72 73 20 62 72 6F 75 67 68 74 20 66 => athers brought f
    6F 72 74 68 20 6F 6E 20 74 68 69 73 20 63 6F 6E => orth on this con
    74 69 6E 65 6E 74 2C 20 61 20 6E 65 77 20 6E 61 => tinent, a new na
    74 69 6F 6E 2C 20 63 6F 6E 63 65 69 76 65 64 20 => tion, conceived

Screen shot of compilation:



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