

DIGGING INTO THE RUNTIME STACK

By Manuel Holguin 10-09-2022

Programming # 3

Table of Contents

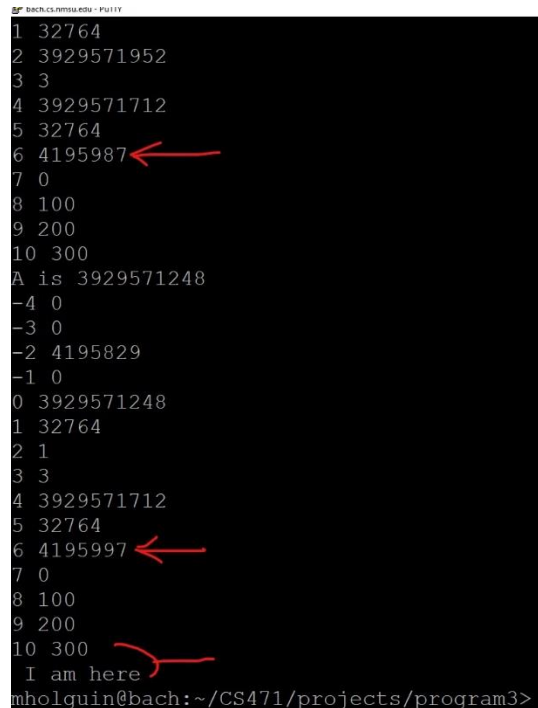
| | |
|---------------------------------------|---|
| Problem Description | 3 |
| Investigation..... | 3 |
| Extending the Program | 4 |
| Reconstructing the Initial Issue..... | 4 |
| Code Base..... | 5 |

Problem Description

For this programming assignment we were tasked with playing and understanding how memory is allocated and how it works on the runtime stack. We are given a piece of C code that gives us the contents of the stack segment, as programmers it is vital that we understand how it works. By the end of the assignment, we should have a better understanding of the memory segment it's constraints and manipulation.

Investigation

When initiating the program, I see a list of numbers my first realization is these represent addresses in the memory segment. I see the address for the main method and the function f within this array of addresses additionally I can see where most of the variables are listed withing the program. One other thing I notice is for the f function it's return address is also available for me to see. However, it has been shifted by ten somewhere within the code. The shifting has caused the program to skip over the first printf statement after main calls f.

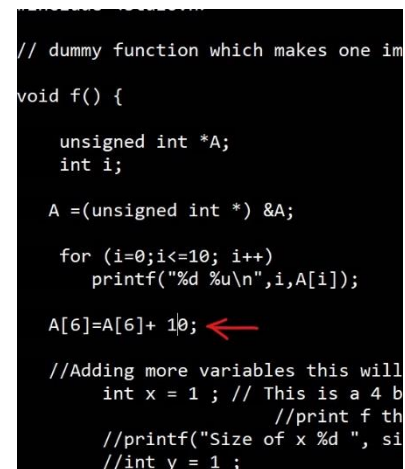


```

1 32764
2 3929571952
3 3
4 3929571712
5 32764
6 4195987 ←
7 0
8 100
9 200
10 300
A is 3929571248
-4 0
-3 0
-2 4195829
-1 0
0 3929571248
1 32764
2 1
3 3
4 3929571712
5 32764
6 4195997 ←
7 0
8 100
9 200
10 300
I am here
mholguin@bach:~/CS471/projects/program3>

```

Looking back at the actual code I can see where in the array the addition of 10 has been added to the return address. This line of code is manipulating the stack segment which is something we as a programmer should not have access to however C allows for these kinds of changes since there is no governing restrictions unlike other languages such as JAVA.



```

// dummy function which makes one im
void f() {
    unsigned int *A;
    int i;

    A =(unsigned int *) &A;

    for (i=0;i<=10; i++)
        printf("%d %u\n",i,A[i]);

    A[6]=A[6]+ 10; ←
    //Adding more variables this will
    int x = 1 ; // This is a 4 b
    //print f th
    //printf("Size of x %d ", si
    //int y = 1 ;

```

Extending the Program

Our second task in this assignment is to extend the code by adding additional variables to it. In doing so this will cause a segmentation fault we are to explain why is that the case. After adding one int variable I did not notice any difference within the execution of the program all was working as intended. However, after adding the secondary int variable the program did not skip over the first printf statement anymore, but I created a seg fault within the program. This tells me that the return address for the program has shifted and no longer accessible from the same array[index] as shown above. Also I am trying to edit something else which is outside of the stack.

```
//Adding more variables this will
int x = 1 ; // This is a 4
int y = 1 ;
```

```
2 0
3 1
4 1
5 5
//6 1820258138
7 32766
8 4195994
9 0
10 100
I called f
I am here
} Segmentation fault (core dumped)
mholguin@bach:~/CS471/projects/pr
```

Reconstructing the Initial Issue

Finally, to complete the assignment we must reconstruct the first issue in which the “I called f” printf statement is skipped over before the presence of the added variables. During the initial trials I that the return address for function f was located at A[6] however after adding the variables the return address has shifted and is no longer located there. As shown in the image above the new address is

located at 8 or A[8] in this case. After a simple modification of the code instead of targeting index six I target the new return address location at eight. In the image below you can see the

```
//return address of function f before creating two new variables.
//A[6] = A[6] + 10;

//Eventually the return address for the f function is shifted
//due to the addition of the two new variables therefore A[6]
//no longer points to the return address.
A[8]=A[8] + 10;

//Adding more variables this will eventually cause a runtime st
int x = 1 ; // This is a 4 bit addition proven by the bott
int y = 1 ;
```

values of the two new variables I created, the new index for the return address and only one printf statement the other one is skipped once again.

Concluding the assignment.

Code Base

/ Program to demonstrate how to over write the*

** return address inside of function*

** we will use a global variable to store*

** the address we want to go to on return*

** and we will use an array in the function to*

** seek the location and replace with the new value*

```
-2 0
-1 0
0 1697076824
1 32766
2 0
3 1 ←
4 1 ←
5 5
6 1697077296
7 32766
8 4196004 ←
9 0
10 100
I am here ←
mholguin@bach:~/CS471/projects/program3>
```

Shaun Cooper

2020 September

Modified By: Manuel Holguin

Date: 10-06-2022

Description: Learning about the runtime stack adding enough variables

to were the printf that is skipped over returns and create a seg fault.

Finally changing the runstack in order to return to the original state.

**/*

#include <stdio.h>

// dummy function which makes one important change

```
void f() {
```

```
    unsigned int *A;
```

```
    int i;
```

```
    A=(unsigned int *) &A;
```

```
    for (i=0;i<=10; i++)
```

```
        printf("%d %u\n",i,A[i]);
```

//return address of function f before creating two new variables.

//A[6] = A[6] + 10;

//Eventually the return address for the f function is shifted

//due to the addition of the two new variables therefore A[6]

//no longer points to the return address.

```
    A[8]=A[8] + 10;
```

//Adding more variables this will eventually cause a runtime stack

```
    int x = 1 ; // This is a 4 bit addition proven by the bottom
```

```
    int y = 1 ;
```

```
    printf("A is %u \n",A);
```

```
    for (i=-4;i<=10; i++)
        printf("%d %u\n",i,A[i]);
}

int main()
{

    int A[100];
    unsigned int L[4];
    L[0]=100;
    L[1]=200;
    L[2]=300;
    L[3]=400;
    for (int i=0; i < 100; i++) A[i]=i;

    printf("main is at %lu \n",main);

    printf("f is at %lu \n",f);
    printf("I am about to call f\n");
    f();
    printf("I called f\n");

    out: printf(" I am here\n");

}
```