Name: Pratyay Kumar
Aggie Id: 800811773
Date: 10/16/2022
Lab: 3

CS 471 – Programming #3 – digging into the runtime stack

Part 1:

1. The compiler allocates some memory to the main function in the runtime stack whenever it executes a program.
2. The hardware stores the value of the function's return address when the function call happens.
3. Inside the function f, we created an array A, which will be stored in the runtime stack.
4. The address of A is stored in A itself.
5. Adding and subtracting values to array A will help us get to different locations in the main function.
6. We have used this array to access the location where the hardware stored the function 'f' return value address and alter it.

Thus, array A can be used smartly in C language to get close to the main function in the stack and gain/alter the addresses. In the given code, by altering A [6] = A [6] + 10, we are changing the return address of the function, and hence we can skip the first printf ().

The output of the code is given below.

```
pkumar@kaiju:~> gcc lab3.c
pkumar@kaiju:~> ./a.out
main is at 4195835
f is at 4195655
I am about to call f
0 655617008
1 32767
2 655617712
3 3
4 655617472
5 32767
6 4195980
7 0
8 100
9 200
10 300
A is 655617008
-4 0
-3 0
-2 4195822
-1 0
0 655617008
1 32767
2 655617712
3 3
4 655617472
5 32767
6 4195990
7 0
8 100
9 200
10 300
 I am here
pkumar@kaiju:~>
```

Part 2:

By incrementally adding single variables to the function f (), the stack size is increasing, and the variable addresses go down in the memory segment. The address reaches a point where the stack is consumed, and the variable in my code tries to go down the memory address, which throws an error.

Since all the runtime stack is consumed, there will be no space for more variables to be declared in the function f ().

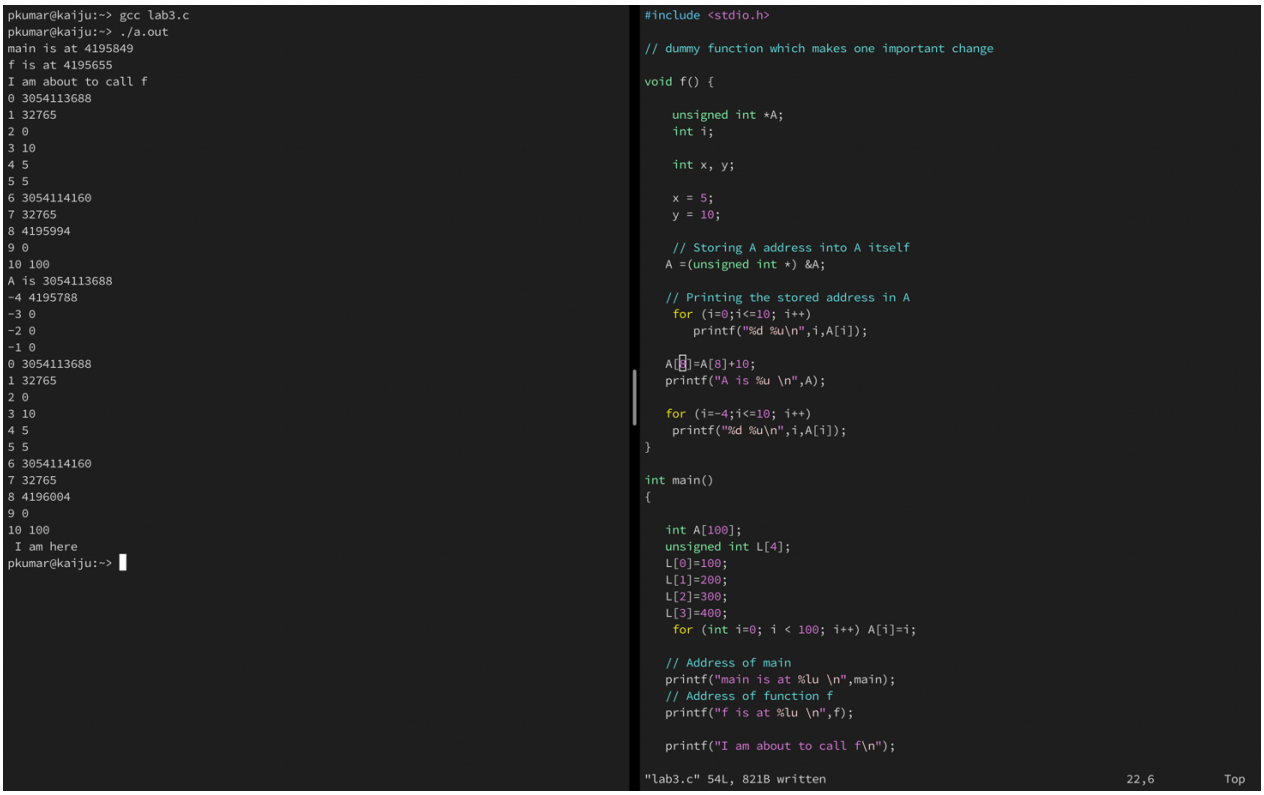
Part3:

By adding two variables, x and y, to the system. The system is giving a segmentation fault error, as shown in the below screenshot.

```
pkumar@kaiju:~> gcc lab3.c
pkumar@kaiju:~> ./a.out
main is at 4195849
f is at 4195655
I am about to call f
0 1951645704
1 32767
2 0
3 10
4 5
5 5
6 1951646176
7 32767
8 4195994
9 0
10 100
A is 1951645704
-4 4195788
-3 0
-2 0
-1 0
0 1951645704
1 32767
2 0
3 10
4 5
5 5
6 1951646186
7 32767
8 4195994
9 0
10 100
I called f
 I am here
Segmentation fault (core dumped)
pkumar@kaiju:~>
```

```
#include <stdio.h>

// dummy function which makes one important change

void f() {

    unsigned int *A;
    int i;

    int x, y;

    x = 5;
    y = 10;

    // Storing A address into A itself
   A =(unsigned int *) &A;

   // Printing the stored address in A
    for (i=0;i<=10; i++)
       printf("%d %u\n",i,A[i]);

   A[8]=A[6]+10;
   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;

   // Address of main
   printf("main is at %lu \n",main);
   // Address of function f
   printf("f is at %lu \n",f);

   printf("I am about to call f\n");

"lab3.c" 54L, 821B written                                    22,6          Top
```

To solve the Segmentation fault caused by the new variables initialized, I now alter the value stored at A[8] to A[8] + 10. In the screenshot below, we can see there is no segmentation error anymore.



```
pkumar@kaiju:~> gcc lab3.c
pkumar@kaiju:~> ./a.out
main is at 4195849
f is at 4195655
I am about to call f
0 3054113688
1 32765
2 0
3 10
4 5
5 5
6 3054114160
7 32765
8 4195994
9 0
10 100
A is 3054113688
-4 4195788
-3 0
-2 0
-1 0
0 3054113688
1 32765
2 0
3 10
4 5
5 5
6 3054114160
7 32765
8 4196004
9 0
10 100
 I am here
pkumar@kaiju:~>
```

```
#include <stdio.h>

// dummy function which makes one important change

void f() {

    unsigned int *A;
    int i;

    int x, y;

    x = 5;
    y = 10;

    // Storing A address into A itself
   A =(unsigned int *) &A;

   // Printing the stored address in A
    for (i=0;i<=10; i++)
       printf("%d %u\n",i,A[i]);

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

   // Address of main
   printf("main is at %lu \n",main);
   // Address of function f
   printf("f is at %lu \n",f);

   printf("I am about to call f\n");

"lab3.c" 54L, 821B written                                    22,6          Top
```

In this way, by making changes to function f() the first printf() is jumped over once again.

Final Code to solve the problem:

```
/*
* Name: Pratyay Kumar
* Date: 10/16/2022
* Lab: 3
* Purpose: Digging into runtime stack
* Program Description: Program demonstrates how to overwrite the
*                      return address inside of the function
*                       we will use a global variable to store
*                        the address we want to go to on the return
*                          and we will use an array in the function to
*                           seek the location and replace it with the new value
*/
#include <stdio.h>

// Dummy function, which makes one crucial change

void f() {
    unsigned int *A;
    int i;
```

```
    // Adding two new variables are initialized.
    int x, y;
    x = 5;
    y = 10;

    // Storing the address of A into A itself
    A =(unsigned int *) &A;

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

    A[8]=A[8]+10; // TO SOLVE THE PROBLEM
    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;

    // Printing address of main
    for (int i=0; i < 100; i++) A[i]=i;
        printf("main is at %lu \n",main);

    // Printing address of function f
    printf("f is at %lu \n",f);
    printf("I am about to call f\n");

    // Function call
    f();

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

}
```