Exercise 2: Process Address Space

## Code: exercise2.c

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

int data1;

void RecuresiveFunc1 (int i)

{

data1= 1;

int data2 = 1;

int data3[2] = {1,1};

int \*data4 = malloc(1000\*sizeof(int));

data4[0] = data4[1] = 1;

static int data5 =1;

printf("Addresses which fall into:\n");

printf("1) datal = %p. \n", &data1);

printf("2) data2 = %p. \n", &data2);

printf("3) data3 = %p. \n", &data3);

printf("4) data4 = %p. \n", &data4);

printf("5) data4[0] = %p. \n", &data4[0]);

printf("6) data5 = %p. \n", &data5);

printf("i is %d.\n\n",i);

if(i < 2)

{

RecuresiveFunc1 (++i) ;

}

else

{

while (1)

{

printf("[while loop)Process ID: %d.\n",getpid());

sleep (100);

}

free(data4);

}

}

int main (int argc, char \*argv[])

{

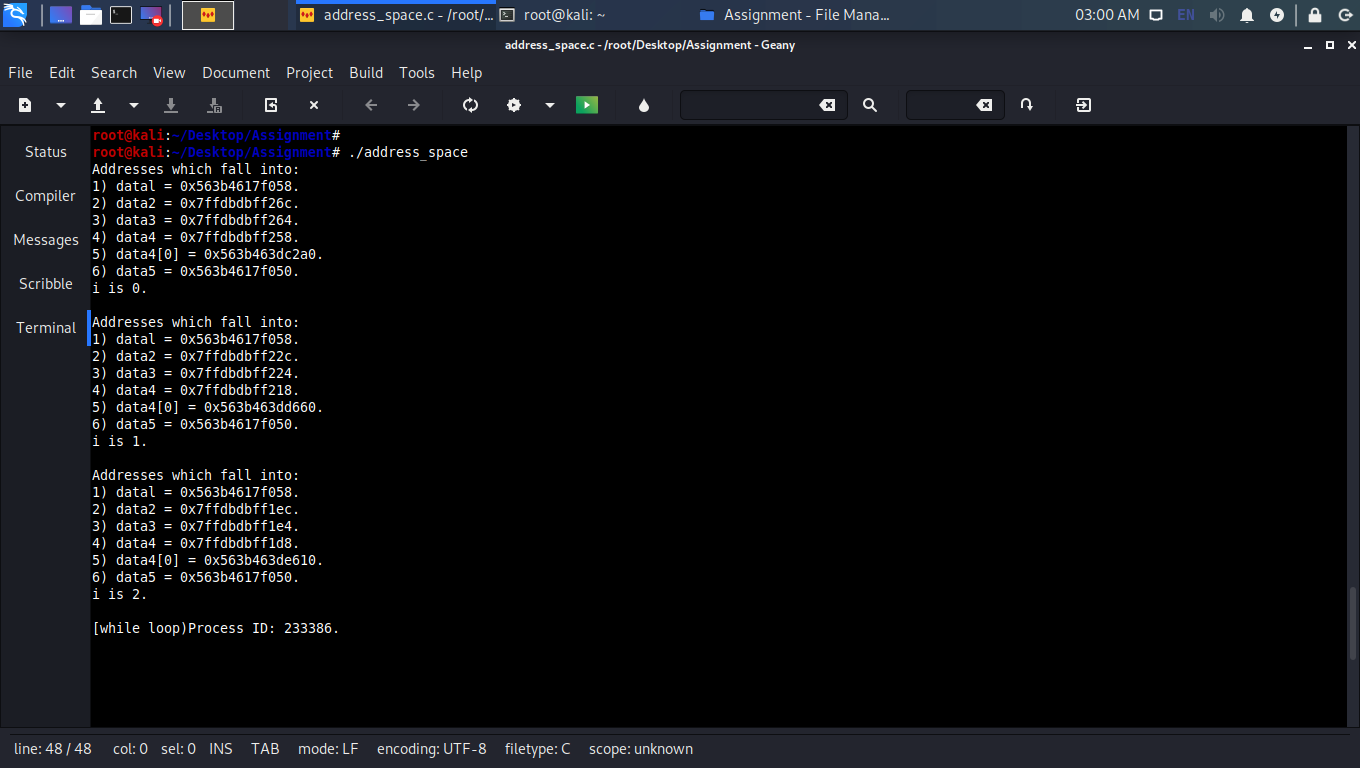
int i = 0;

RecuresiveFunc1(i);

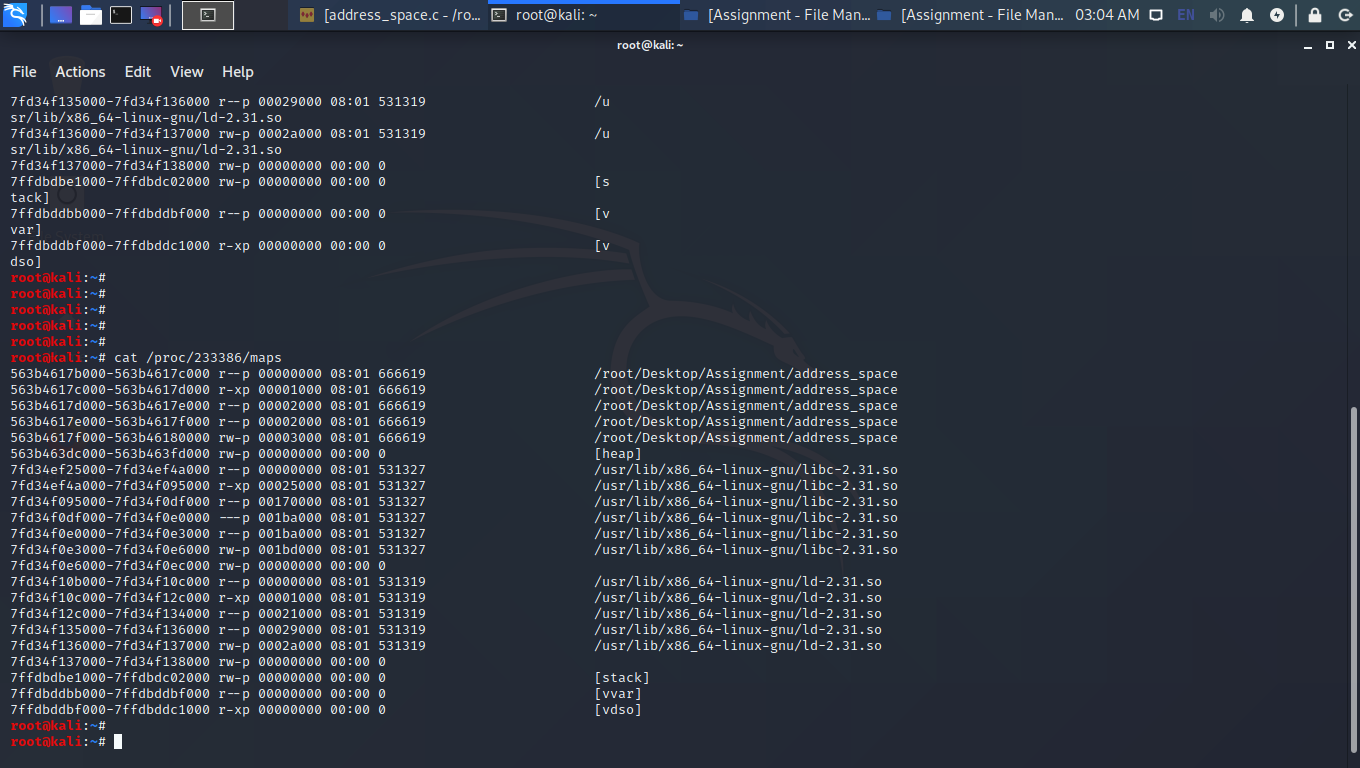
return 0;

}

## Screenshot 1:



## Screenshot 2:



## Code Output

root@kali:~/Desktop/Assignment# ./address\_space

Addresses which fall into:

1) datal = 0x563b4617f058.

2) data2 = 0x7ffdbdbff26c.

3) data3 = 0x7ffdbdbff264.

4) data4 = 0x7ffdbdbff258.

5) data4[0] = 0x563b463dc2a0.

6) data5 = 0x563b4617f050.

i is 0.

Addresses which fall into:

1) datal = 0x563b4617f058.

2) data2 = 0x7ffdbdbff22c.

3) data3 = 0x7ffdbdbff224.

4) data4 = 0x7ffdbdbff218.

5) data4[0] = 0x563b463dd660.

6) data5 = 0x563b4617f050.

i is 1.

Addresses which fall into:

1) datal = 0x563b4617f058.

2) data2 = 0x7ffdbdbff1ec.

3) data3 = 0x7ffdbdbff1e4.

4) data4 = 0x7ffdbdbff1d8.

5) data4[0] = 0x563b463de610.

6) data5 = 0x563b4617f050.

i is 2.

[while loop)Process ID: 233386.

## Memory Map

root@kali:~# cat /proc/233386/maps

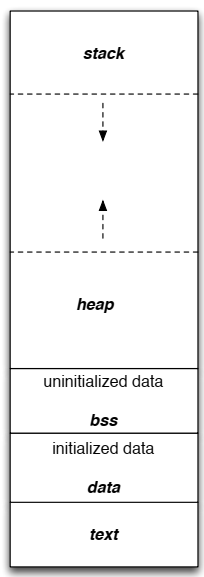
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sr. No. | Address | Permissions | Offset | Device | Inode | Pathname |
| 1 | 563b4617b000-563b4617c000 | r--p | 00000000 | 08:01 | 666619 | /root/Desktop/Assignment/address\_space |
| 2 | 563b4617c000-563b4617d000 | r-xp | 00001000 | 08:01 | 666619 | /root/Desktop/Assignment/address\_space |
| 3 | 563b4617d000-563b4617e000 | r--p | 00002000 | 08:01 | 666619 | /root/Desktop/Assignment/address\_space |
| 4 | 563b4617e000-563b4617f000 | r--p | 00002000 | 08:01 | 666619 | /root/Desktop/Assignment/address\_space |
| 5 | 563b4617f000-563b46180000 | rw-p | 00003000 | 08:01 | 666619 | /root/Desktop/Assignment/address\_space |
| 6 | 563b463dc000-563b463fd000 | rw-p | 00000000 | 00:00 | 0 | [heap] |
| 7 | 7fd34ef25000-7fd34ef4a000 | r--p | 00000000 | 08:01 | 531327 | /usr/lib/x86\_64-linux-gnu/libc-2.31.so |
| 8 | 7fd34ef4a000-7fd34f095000 | r-xp | 00025000 | 08:01 | 531327 | /usr/lib/x86\_64-linux-gnu/libc-2.31.so |
| 9 | 7fd34f095000-7fd34f0df000 | r--p | 00170000 | 08:01 | 531327 | /usr/lib/x86\_64-linux-gnu/libc-2.31.so |
| 10 | 7fd34f0df000-7fd34f0e0000 | ---p | 001ba000 | 08:01 | 531327 | /usr/lib/x86\_64-linux-gnu/libc-2.31.so |
| 11 | 7fd34f0e0000-7fd34f0e3000 | r--p | 001ba000 | 08:01 | 531327 | /usr/lib/x86\_64-linux-gnu/libc-2.31.so |
| 12 | 7fd34f0e3000-7fd34f0e6000 | rw-p | 001bd000 | 08:01 | 531327 | /usr/lib/x86\_64-linux-gnu/libc-2.31.so |
| 13 | 7fd34f0e6000-7fd34f0ec000 | rw-p | 00000000 | 00:00 | 0 |  |
| 14 | 7fd34f10b000-7fd34f10c000 | r--p | 00000000 | 08:01 | 531319 | /usr/lib/x86\_64-linux-gnu/ld-2.31.so |
| 15 | 7fd34f10c000-7fd34f12c000 | r-xp | 00001000 | 08:01 | 531319 | /usr/lib/x86\_64-linux-gnu/ld-2.31.so |
| 16 | 7fd34f12c000-7fd34f134000 | r--p | 00021000 | 08:01 | 531319 | /usr/lib/x86\_64-linux-gnu/ld-2.31.so |
| 17 | 7fd34f135000-7fd34f136000 | r--p | 00029000 | 08:01 | 531319 | /usr/lib/x86\_64-linux-gnu/ld-2.31.so |
| 18 | 7fd34f136000-7fd34f137000 | rw-p | 0002a000 | 08:01 | 531319 | /usr/lib/x86\_64-linux-gnu/ld-2.31.so |
| 19 | 7fd34f137000-7fd34f138000 | rw-p | 00000000 | 00:00 | 0 |  |
| 20 | 7ffdbdbe1000-7ffdbdc02000 | rw-p | 00000000 | 00:00 | 0 | [stack] |
| 21 | 7ffdbddbb000-7ffdbddbf000 | r--p | 00000000 | 00:00 | 0 | [vvar] |
| 22 | 7ffdbddbf000-7ffdbddc1000 | r-xp | 00000000 | 00:00 | 0 | [vdso] |

## Solution

For the variables *data1, data2, data3, data4, data4[0], data5*:  
**(1) Where are these variables stored? Give you reasons.**A. Code segment  
B. Data segment  
C. BSS  
D. Heap  
E. Stack segment

## **Answer:**

Following diagram shows typical layout of a simple computer's program memory with the text, various data, and stack and heap sections.



Lower Address Space

Higher Address Space

* data1 memory address is 0x563b4617f058 in all three recursive calls. This address lies in address range [563b4617f000-563b46180000] (Serial No. 5) which is **BSS** because BSS section is immediately before Heap section (Serial No. 6).
* data2 memory address are 0x7ffdbdbff26c, 0x7ffdbdbff22c, and 0x7ffdbdbff1ec in three recursive function calls. All of these address lie in **Stack section** [7ffdbdbe1000-7ffdbdc02000] (Serial No. 20). Because this is a local variable.
* data3 memory address are 0x7ffdbdbff264, 0x7ffdbdbff224, and 0x7ffdbdbff1e4 respectively for three recursive function calls. All of these address lie in **Stack section** [7ffdbdbe1000-7ffdbdc02000] (Serial No. 20). Because this is a local variable.
* data4 memory address are 0x7ffdbdbff258, 0x7ffdbdbff218, and 0x7ffdbdbff1d8 respectively for three recursive function calls. All of these address lie in **Stack section** [7ffdbdbe1000-7ffdbdc02000] (Serial No. 20). Because this is a point type variable and is a local variable.
* data4[0] memory address are 0x563b463dc2a0, 0x563b463dd660, and 0x563b463de610 respectively for three recursive function calls. All of these address lie in **Heap section** [563b463dc000-563b463fd000] (Serial No. 6). Because this is dynamically allocated space used by all three recursive function calls.
* data5 memory address are 0x563b4617f050, 0x563b4617f050, and 0x563b4617f050 respectively for three recursive function calls. All of these address lie in **BSS** [563b4617f000-563b46180000] (Serial No. 5). Because this is static variable.

**(2) Estimate the stack size of RecursiveFunc1 and give your reasons.**

Stack Frame on First Function Call

|  |
| --- |
| Locals of RecursiveFunc1 |
| Return Address to Main Function |
| Parameters for RecursiveFunc1 |
| Locals of Main Function |
| Return Address to OS |
| Parameters for Main Function |
| ……………………….. |
| ……………………….. |

Because 1 parameter of type integer is passed to **RecursiveFunc1** function and data2, data3, \*data4 are local variables. So estimated size of stack for **RecursiveFunc1** is

Parameter (4 bytes)+ return address (4 bytes) + 3 local variables (12 bytes) = 20 bytes

For three recursive calls 20 bytes \* 3 = 60 bytes.