Collaborators: Varun Jani and Trisha Mandal

**CMPSC 473 Project 1 Report**

Prog1

1. Variable b should be the only dynamically allocated variable since we are using malloc to allocate memory to the heap and it is a pointer. The heap uses the lowest address and then grows upwards. Variable a is an array that is stored in the stack. The stack is growing downwards in addresses. The local variables are a, b and count. The addresses of a and b are 0x7ffc3465b610 and 0xcf3010 respectively.
2. Process id: 10293, Stack size: 132 kB
3. Process id: 10293, Heap size: 620 kB
4. Address limit of stack: 7f2be32d4000-7f2be32d5000, Address limit of heap: 0164f000-016ea000
5. Functions:

execve(): executes the program thay is gotten by the path name

brk(), sbrk(): It is the end of the uninitialized data segment gotten from the given process id

mmap(), munmap(): It creates new mapping in the virtual address space for that particular process.

access(): it checks if access to the file is not allowed or not

open(): It opens file

stat(): It shows the file status

fstat(): gives some information about a file using a buffer pointer

close(): closes a file

mprotect(): changes the access protection for the memory pages of the running process

write(): allows you to communicat with other users on the server

read(): counts the total bytes

exit\_group(): it terminates all the threads in the calling process

**Prog2**



|  |  |
| --- | --- |
| **32 bit** | **64 bit** |
| Size of code during compile: 1520K | Size of code during compile: 1722K |
| Size of code during runtime: 2312k | Size of code during runtime: 4392K |
| Size of linked libraries: 1968K | Size of linked libraries: 4020K |

1. Line 21-> allocate(count);
2. When the value of count is passed into allocate as 0, when we enter the if statement, allocate function takes in count-1, if that happens allocate function will stop as the if statement works only if count is greater than equal to 0.

Starting address of stack: 7ffe610ae000

Ending address of stack: 7ffe610cf000

The address size is limited in this region. Within this stack size the memory does not exist. This justifies the error with the given addresses.

1. Size of frame 1: 0x124fb0 (0x7fffffa44ce0 – 0x7fffff91fd30)

Size of frame 2: 0x124fb0

Size of frame 3: 0x124fb0

Size of frame 4: 0x124fb0

Size of frame 5: 0x124fb0

Size of frame 6: 0x124fb0

Size of frame 7: 0x30 (0x7fffffffdb80-0x7fffffffdb50)

11 recursions should be possible on the system but actually only 6 recursions are happening when the program is being executed.

1. The frame consists of the different type of register addresses like the addresses of the rip and rbp registers. It allows us to see the process of registers and allocate memory from one register to another register when the process is called and it shows us the addresses for the registers.

Prog 3

|  |  |
| --- | --- |
| **32 bit** | **64 bit** |
| Size of code during compile: 1763K | Size of code during compile: 2017K |
| Size of code during runtime: 437420K | Size of code during runtime: 4413080K |
| Size of linked libraries: 2232K | Size of linked libraries: 6964K |

1. Line 22 is giving as issue according to the valgrind. Around 4,444,444,440 bytes of heap memory. So, I think the segmentation fault is due to a memory leak as the power function in the code allocates huge amounts of memory.
2. In prog2, there was no memory leak. Segmentation fault was caused due to a logical error. Whereas, in prog3, there is a memory leak which caused is a SIGSEGV.

Prog4

1. Valgrind command gives us the following information:

Heap memory information:

definitely lost: 1,842,952,000 bytes in 921,476 blocks

possibly lost: 50,000 bytes in 25 blocks

This means that there is a memory leak even though the program execution was successful.

We fix it by freeing the memory in the right place.



Text

Description automatically generated

(i) User CPU time: 1.75979s

(ii) System CPU time: 0.446958s

Difference between user cpu time and system cpu time is: 1.312832s

(iii) Max resident size = 1814736 (Approx difference)

(iv)

Text, letter

Description automatically generated

signals sent: SIGKILL and SIGSEGV

The CPU sends the signals.

(v) Voluntary context switches : 4

(vi) Involuntary Context switches: 5

Prog5

1. (i) Process id is different for child and parent process. They are 7094 and 6688 respectively. (ii) The address of the child and parent processes are the same if you look at the screenshots. It is 0x602010.
2. Before exec function the parent and child process will have the same address space since the child gets the exact copy of the parent’s address space after the fork function. After exec function has been run, the current process image will be replaced by a new process image so the address space will change.

Text, application

Description automatically generated

We see that after exec, the address changes from 0x7fffffffdd88 to 0x7fffffffddf8.

Table

Description automatically generated

Stack frame before signal handling is the above image.

41 is the ascii value of a which is there in the stack.

Table

Description automatically generated with medium confidence

Stack frame after exec is the above image