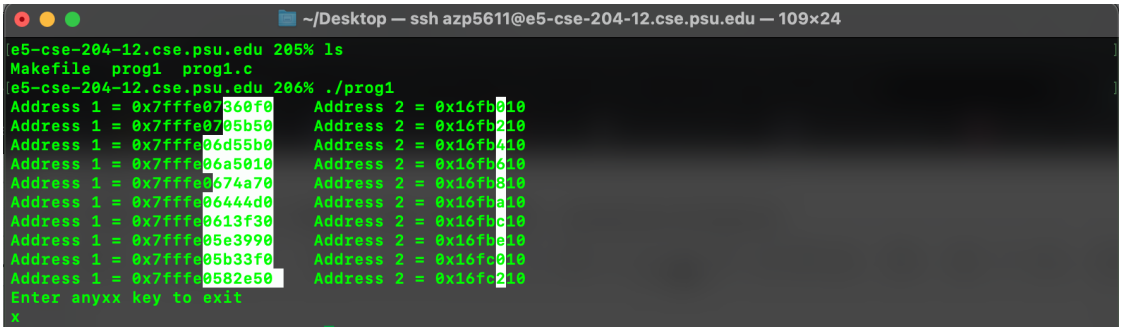
Ahmet Miles Putun
Project 1 Report
Dr. Urgaonkar
06/26/2021

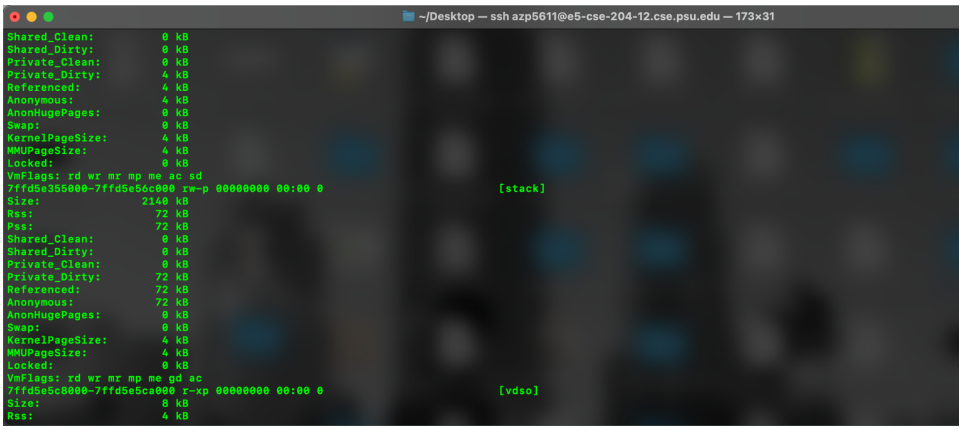
Prog1

1) Stack, Heap and System Calls

a) In address 1 section that is being recursively printing the new addresses, the addresses extend towards the 0x7fffe0000000, leaning towards 0x0000 direction. Address 1 refers to local variables and the data structure stored in is stack.
In address 2 section that is being recursively printing the new addresses, the addresses go towards the 0x16ff, growing towards 0xffff direction. Address 2 refers to dynamically allocated variables and the data structured stored in is heap.



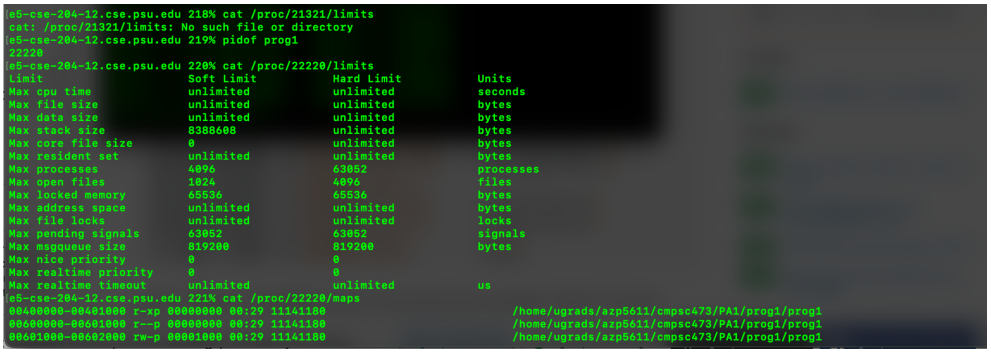
b) For this question, while running the program in a separate terminal, do pidof prog1, and get pid, then display process details with /proc/pid/smaps
Stack size is 2140 KB.

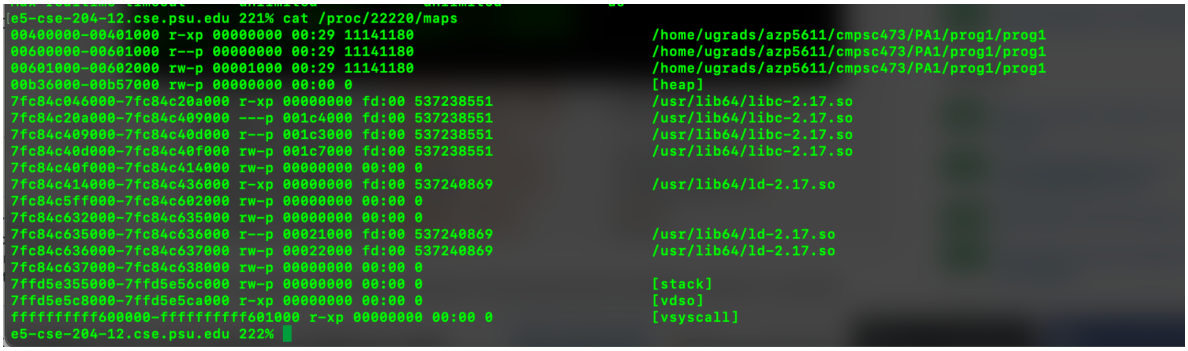


c) Heap size is 132 KB.

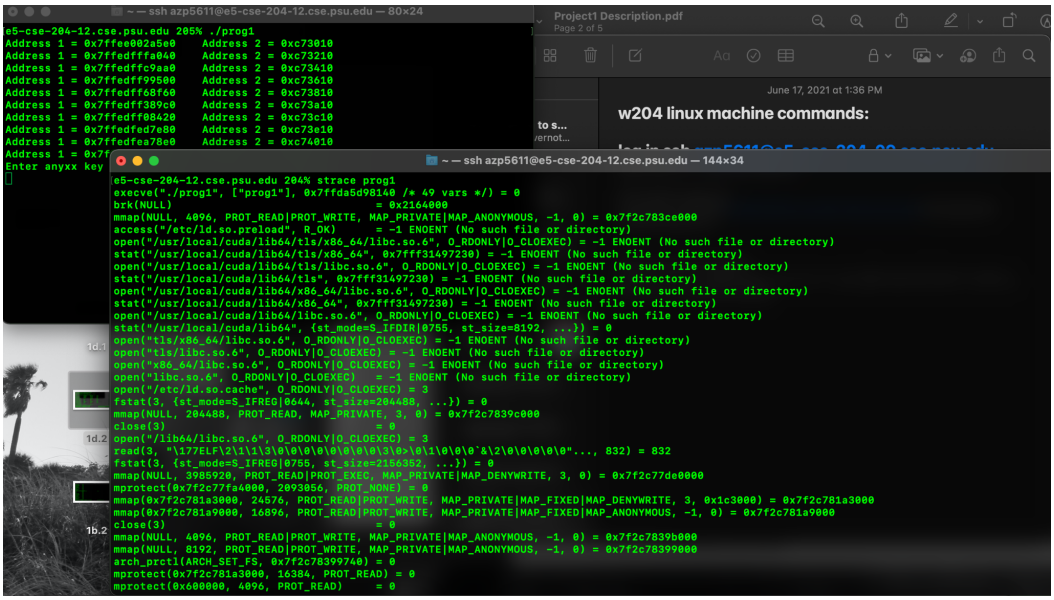
d) Limits -> Stack: 7ffd5e355000 - 7ffd5e56c000

-> Heap: 00b36000 - 00b57000





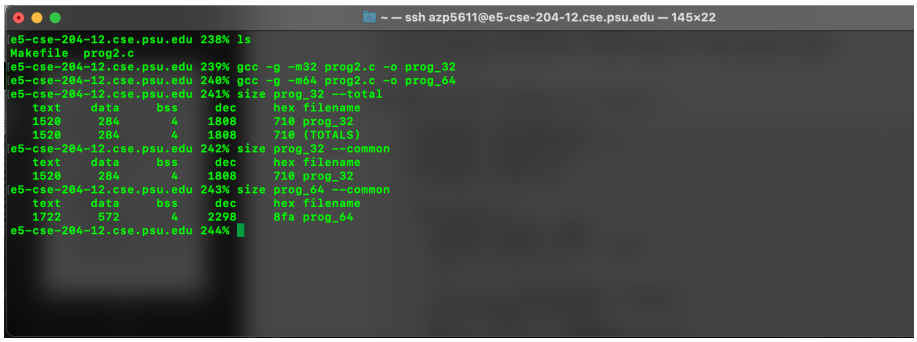
e) Strace command

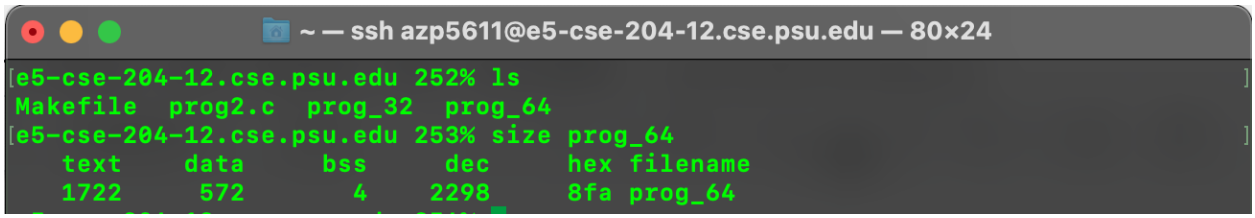


- According to man strace command, strace intercepts and records the system calls which are called by a process and the signals which are received by a process
  Execve: executes the program “./prog1, [“prog1”]
  Brk: it is a call to change the location of the program break. Apparently, if you do increase it, it has an effect of allocating more memory to the given process, the deallocation for decreasement. First it is defined to be NULL as it starts.
  Mmap: it is a system call that maps files or devices into memory. And according to Wikipedia, “it implements demand paging because file contents are not read from disk directly and initially do not use physical RAM at all”.
  Open: Allocates resources associated with the file descriptor and returns a handle that the process will use to refer to that file.
  Stat: It returns the information about a file. Fstat is similar to retrieving information about a file.
  Access: checks whether the calling process can access the file path.
  Close: closes a file descriptor

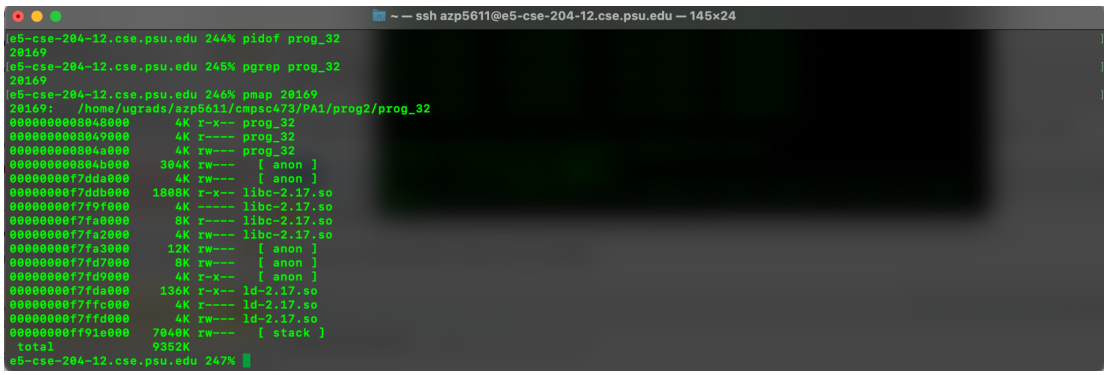
2) Debugging Refresher

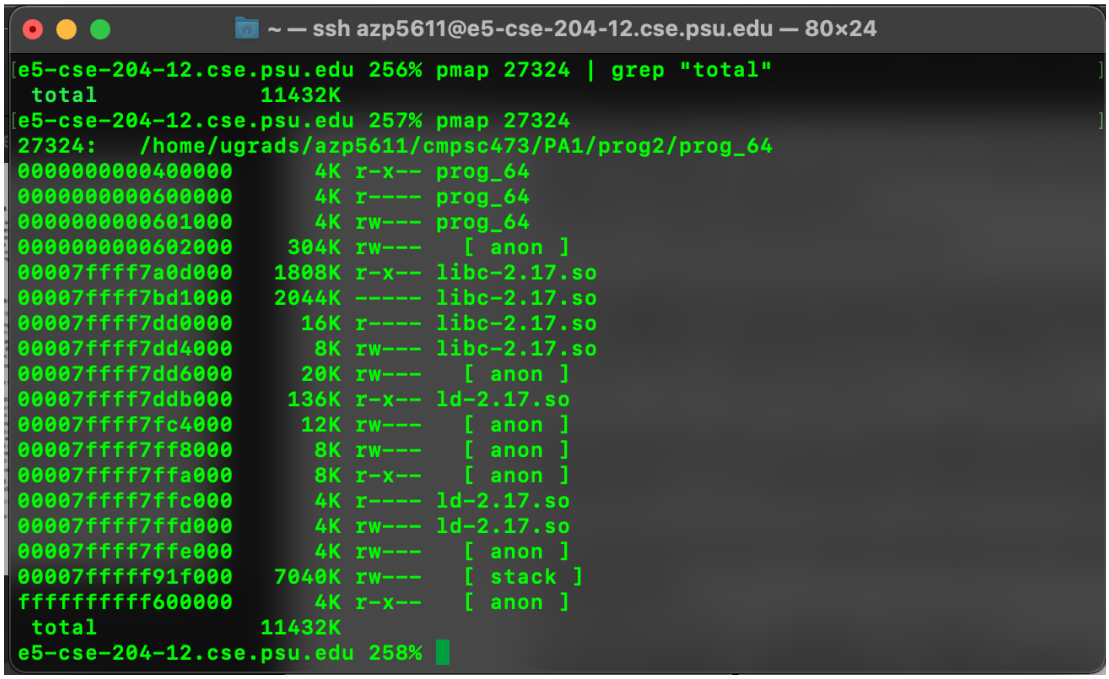
a) **i)** For prog_32 (32 bits exec), size of compiled code in “text” is 1520 and size of compiled code for prog_64 (64 bits exec), size of compiled code in “text” is 1722.





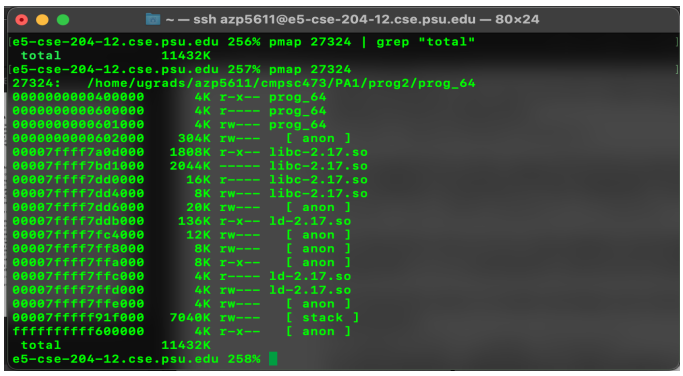
ii) For prog_32, size of code during run time, total is 935k, stack is 7040k. Please see the screenshot below. For prog_64, total size during run time is 11432K and stack is 7040k.





iii) For prog_32, Library sizes (libc-2.17.so, ld-2.17.so …), total is 1968 K. See the screenshot below for details

For prog_64, Library, library size 3884 K (adding up libc-2.17.so, ld-2.17.so …). See screenshot.



b) and c)

gdb to find the program statement that caused the error, explain the cause of this error.

C = malloc(300);

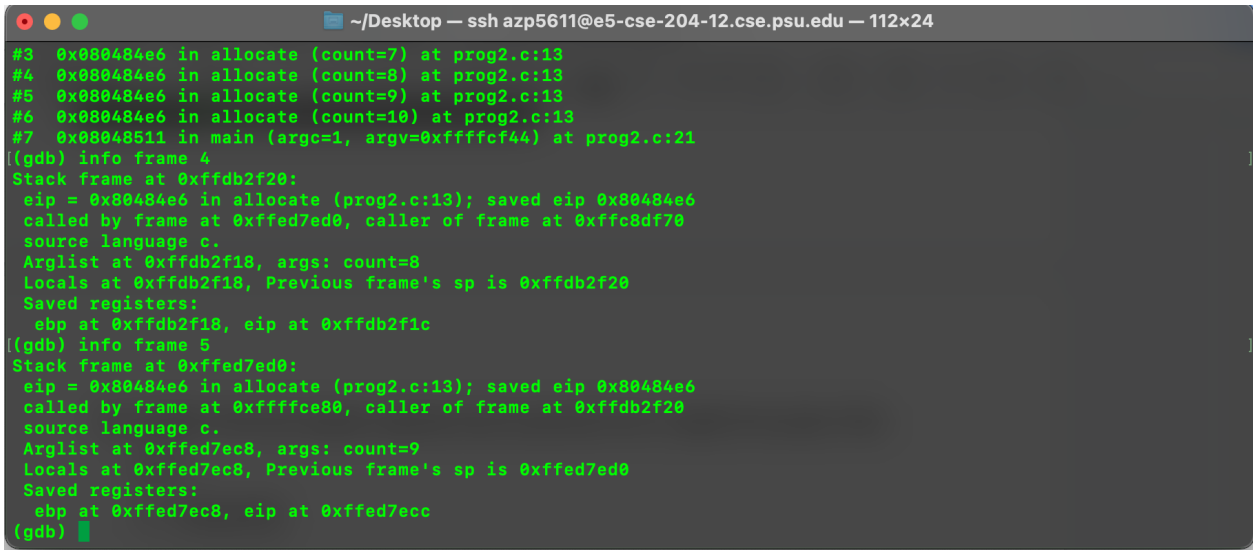
Int x[300000];

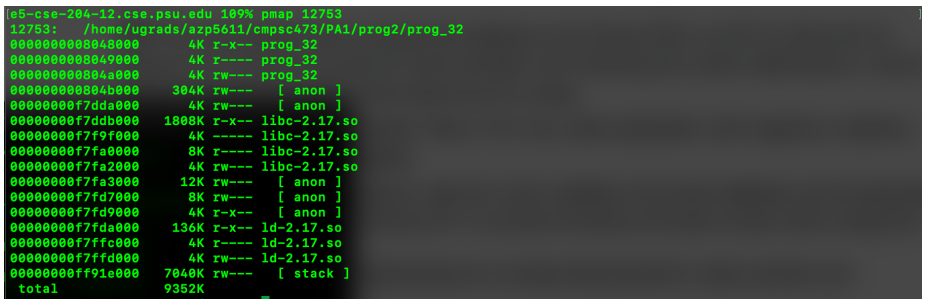
Size of each frame:

(Stack Frame 5 at - Stack Frame 4 at)

-> 0xffed7ed0 - 0xffdb2f20 = 124FB0 (1200048 in decimal)

X exceeds the bounds of the stack when reached for, so overflow.





d) It seems like it is invoked 7 times. I am not sure about the last invocation, meaning the 7th, since it goes back to main with argv, it may mean it fails the 7th time so 6 invocations could be the correct answer as well. But my count starts from 4.

```
~ — ssh azp5611@e5-cse-204-12.cse.psu.edu — 80×24

Stack Address  = 0xffc8df8c    Heap Address = 0x8059a78

Stack Address  = 0xffb68fdc    Heap Address = 0x8060fb0

Stack Address  = 0xffa4402c    Heap Address = 0x80684e8

Stack Address  = 0xff91f07c    Heap Address = 0x806fa20


Program received signal SIGSEGV, Segmentation fault.
allocate (count=4) at prog2.c:11
11          c = malloc (30000);
Missing separate debuginfos, use: debuginfo-install glibc-2.17-324.el7_9.i686
(gdb) backtrace
#0  allocate (count=4) at prog2.c:11
#1  0x080484e6 in allocate (count=5) at prog2.c:13
#2  0x080484e6 in allocate (count=6) at prog2.c:13
#3  0x080484e6 in allocate (count=7) at prog2.c:13
#4  0x080484e6 in allocate (count=8) at prog2.c:13
#5  0x080484e6 in allocate (count=9) at prog2.c:13
#6  0x080484e6 in allocate (count=10) at prog2.c:13
#7  0x08048511 in main (argc=1, argv=0xffffcf44) at prog2.c:21
(gdb)
```

```
~ — ssh azp5611@e5-cse-204-12.cse.psu.edu — 127×24
11          c = malloc (30000);
Missing separate debuginfos, use: debuginfo-install glibc-2.17-324.el7_9.i686
(gdb) backtrace full
#0  allocate (count=4) at prog2.c:11
        x = <error reading variable x (Cannot access memory at address 0xff7fa0cc)>
        c = 0xf7e87928 <malloc+8> "\201\303\324\\"
#1  0x080484e6 in allocate (count=5) at prog2.c:13
        x = {0 <repeats 299607 times>, -134599968, 57, -134385664, -135501485,
          -134602752, -135996991, 1, -134385664, 57, 0, -134599968,
          -134608704, -1, -135990102, -134599968, -134385664, 57, 0, 0,
          -134385664, 0, 0, 0, 0, 0, -134602752, -134599968, 57, -1,
          -135988806, -134599968, -134385664, 57, 0, 0, 0, 0, -134602752,
          -134599968, 2, 0, -135994633, -134599968, -1, 2, 0, 0, 0, 0, 2004,
          0, -134610336, -6012952, -135986564, -134599968, 120, -136188995,
          -135955280, -134602752, -134599968, 2147483592, -136186177, 0,
          134514174, 2, 0 <repeats 18 times>, -7, 0, 134644968, 0, 1, 0, 16,
          0, 0, 0, 0, 0, 0, -1, 1, 120, 2097152, -6012983, -6012980, 0,
          -136178442, -134610336, 0, 0, 0, 0, 0, 134514149, 0, 2004, 57, 0, 0,
          134514174, -6012908, 0, 0, 0, 0, 0, -136054592, -134599968,
          0 <repeats 224 times>, -135965690, 0, 0, 0, 0, 0, 0, 0, 0,
          134674968, 30808, -134601648, 0, 10000, 30000, 117, 3751, 468, 524,
          58, 149, 7, 117, 0, 119, 0, 124, -134889893, 0, 0, 0, 98129830,
          946156400, 0, 0, -136186533, -134602752, 0, 0, -4812872, -136145489,
          -134599968, 134514132, -6012984}
```

e) Size of each frame:

(Stack Frame 5 at - Stack Frame 4 at)

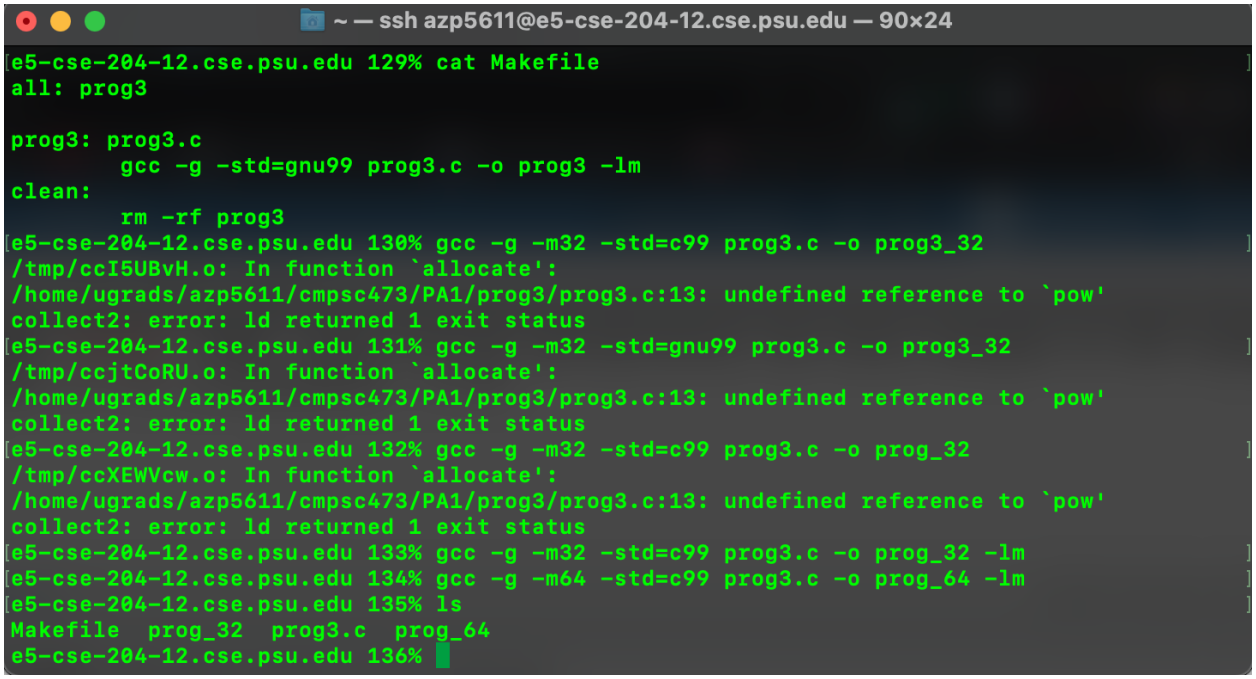
-> 0xffed7ed0 - 0xffdb2f20 = 124FB0 (12.000.48 bytes).

```
(gdb) frame
#1  0x080484e6 in allocate (count=5) at prog2.c:13
13          allocate(count - 1);
(gdb)
```

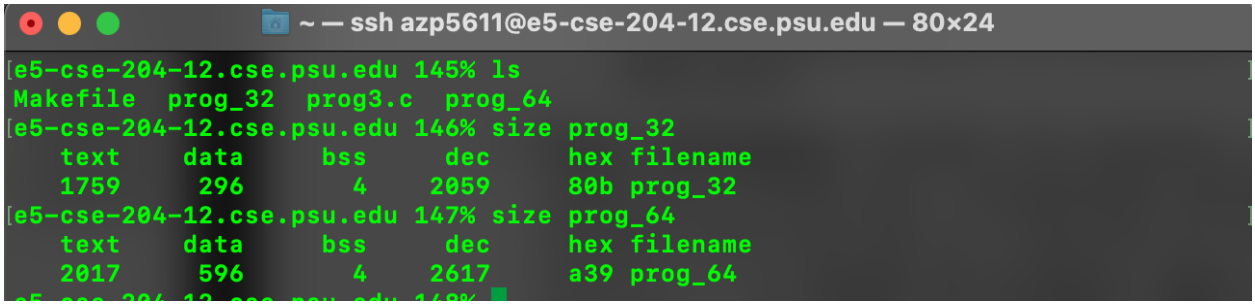
It has the information about saved registers, local variables, the caller and callee of the frames, argument lists, displaying the contents of the frame, and gives details about it.

```
(gdb) info frame 5
Stack frame at 0xffed7ed0:
 eip = 0x80484e6 in allocate (prog2.c:13); saved eip 0x80484e6
 called by frame at 0xffffce80, caller of frame at 0xffdb2f20
 source language c.
 Arglist at 0xffed7ec8, args: count=9
 Locals at 0xffed7ec8, Previous frame's sp is 0xffed7ed0
 Saved registers:
  ebp at 0xffed7ec8, eip at 0xffed7ecc
(gdb)
```

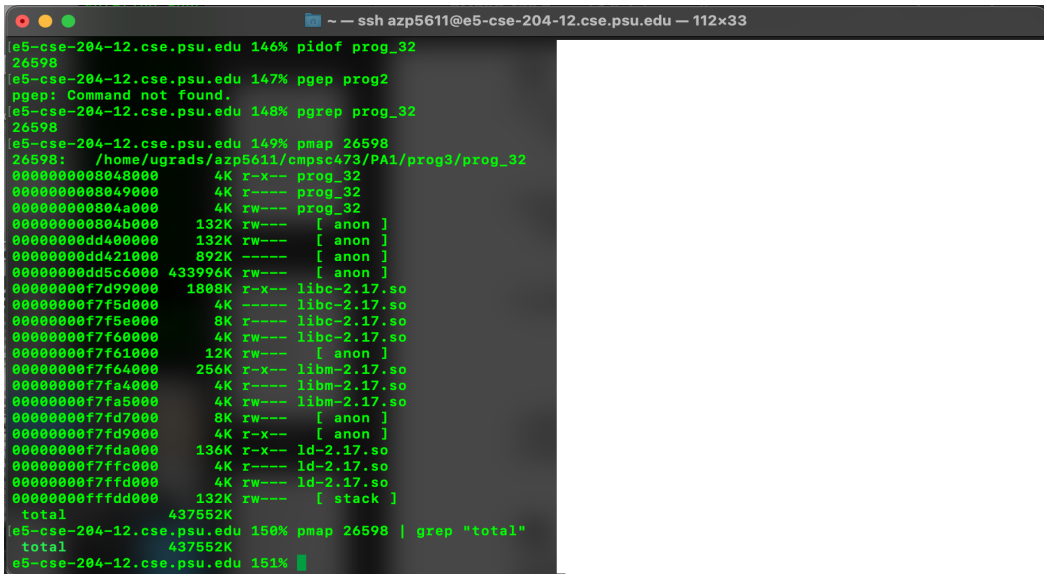
3)

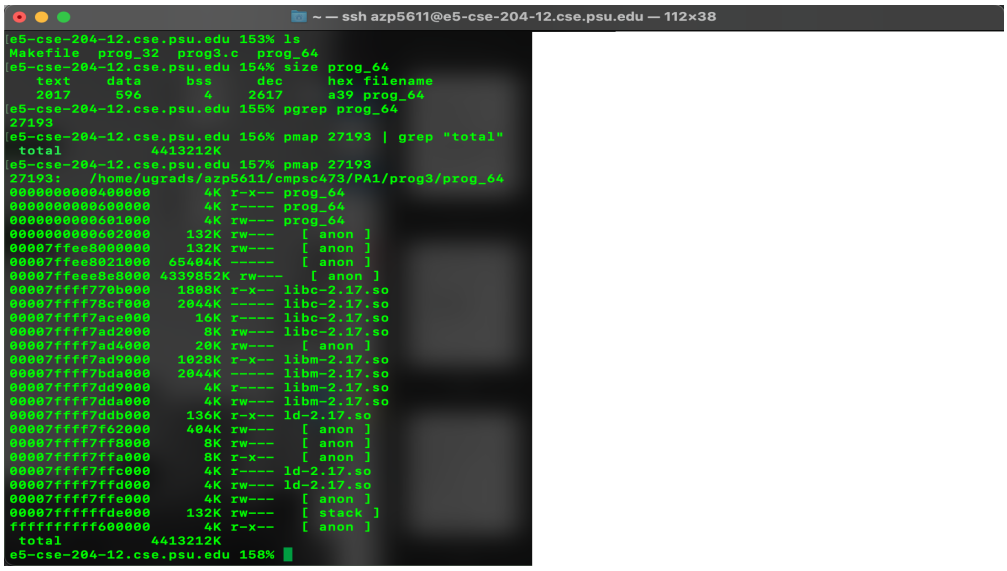


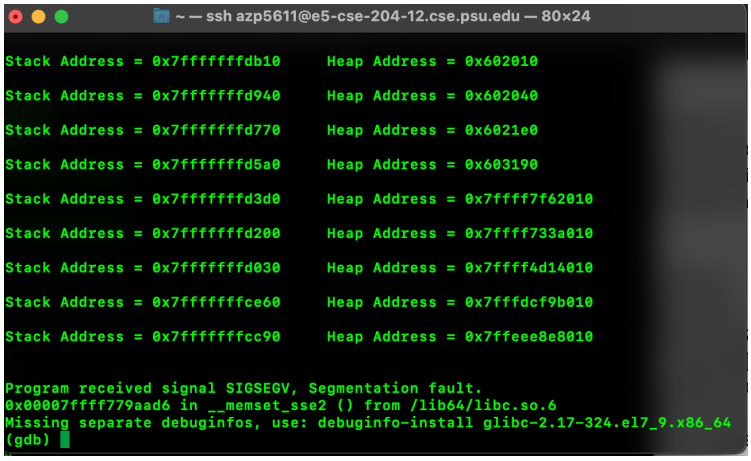
a) **i)** For prog_32 (32 bits exec) of prog3, size of compiled code in “text” is 1759 and size of compiled code for prog_64 (64 bits exec), size of compiled code in “text” is 2017.



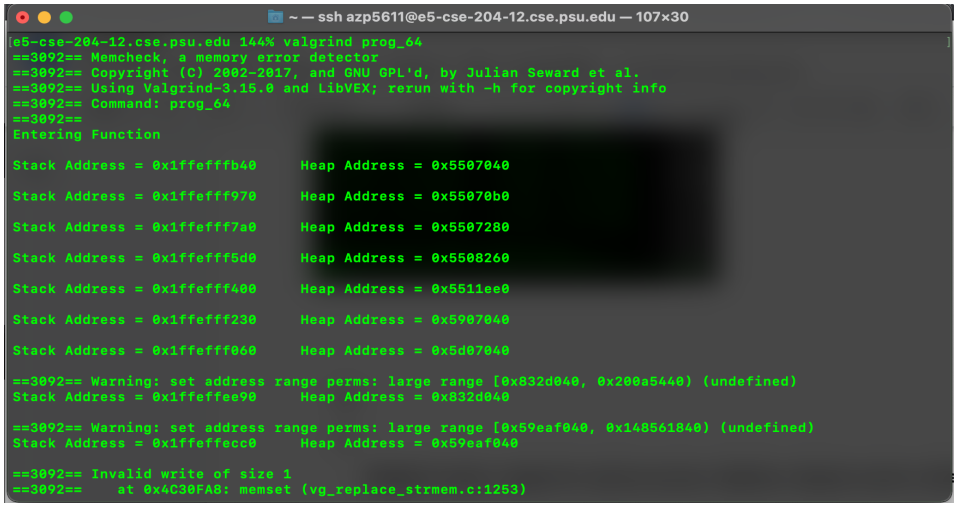
ii) For prog_32, size of code during run time, total is 437552K, stack is 132K. Please see the screenshot below. For prog_64, total size during run time is 4413212K and stack is 132K with read and write. See the screenshots below. Linked libraries for the 32 bit executable prog_32 (if it is only libc and libm) is total 2088K (libc+limb value). Its value is (136+4+4) = 144K Linked libraries for the 64 bit executable prog_64 is (libc_limb value) 6956K. Its value is (136+4+4) = 144K.







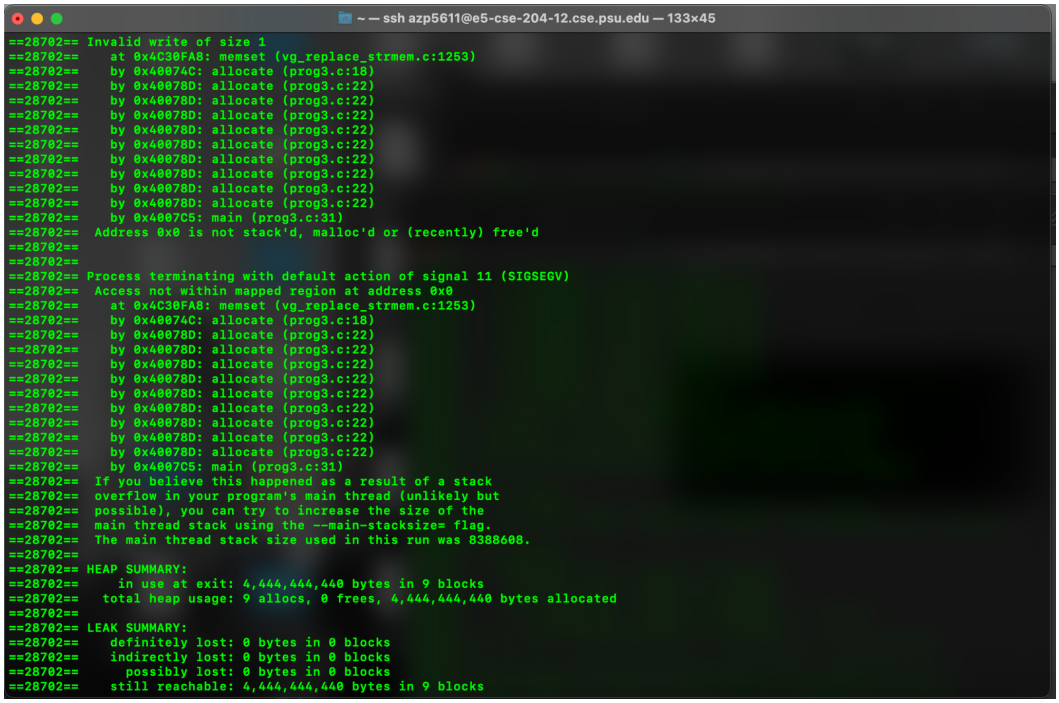
b) According to google, SigSegV is a signal that is received when there is a memory access violation while trying to read or write from/to a memory area that your process does not have access to.



From the screenshot above, it seems that there is invalid write of size 1 which apparently is the size of the written data and in this case it is 1 byte, size of a character.

```
{
  b = malloc(pow(10,r)*sizeof(int));
  for (int i = 0 ;i<sizeof(b) ; i++)
  {
    char *ch1;
    ch1 = &b[i];
    memset(ch1,'*',sizeof(b[0])+i);
  }

  printf("Stack Address = %p     Heap Address = %p  \n\
  allocate(count-1,r);
```



There is a memset error which is because we allocated memory and tried accessing its addresses that exceeded the range that was allocated.

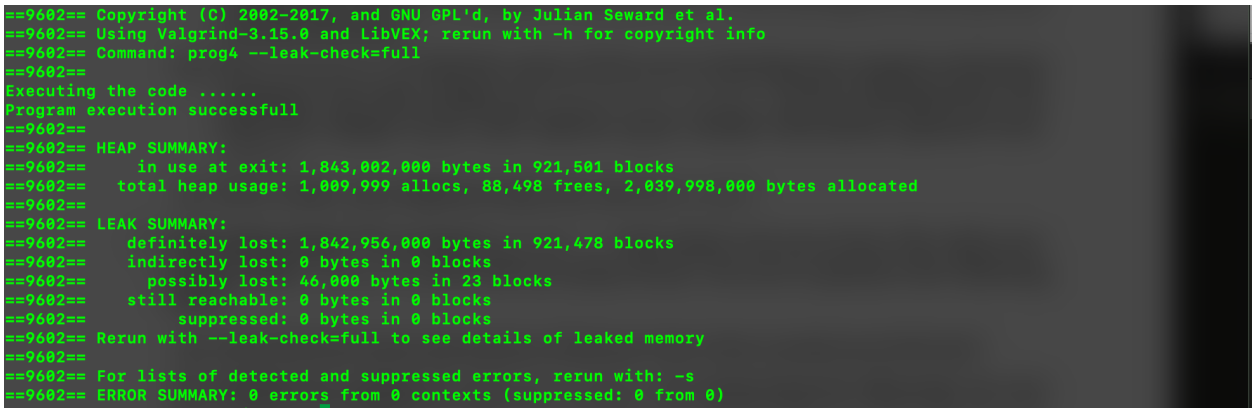
Valgrind gives details about the referring locations starting with 0x4C30FA8 and then refers to the lines, as line 22 in prog3.c: “allocate (count-1,r); line 18: **memset**(ch1,’*’,sizeof(b[0]+i) and line 31 that corresponds to function main, “allocate(count,r)”.

c) This bug is caused by trying to reach memory it was not supposed due to allocation of the memory. Prog3 is accessing invalid memory range and prog2 stack being overflown.

4)

a) The program frees the memory conditionally except for the allocate2 function. The allocation of the memory seems absurd because the for loop may allocate more than it should have, or it is trying to overwrite where it was allocated before with “if conditions” or switch , memory leak could happen.

The if statements bothered me while following code, I am not exactly sure what I would do to fix the allocation-deallocation of the memory itself but I would remove the if statements and after it finishes the first execution, I would change it to free it immediately.



b) User CPU time is the time spent executing the user program, it is on the processor running your program's code (or code in libraries). System CPU time is the time spent executing the code in the operating system kernel on behalf of your program.

(Reference for this question 4b.

https://stackoverflow.com/questions/4310039/user-cpu-time-vs-system-cpu-time)

The User CPU time is 1.099468 seconds, and 0.498523 seconds System CPU time.