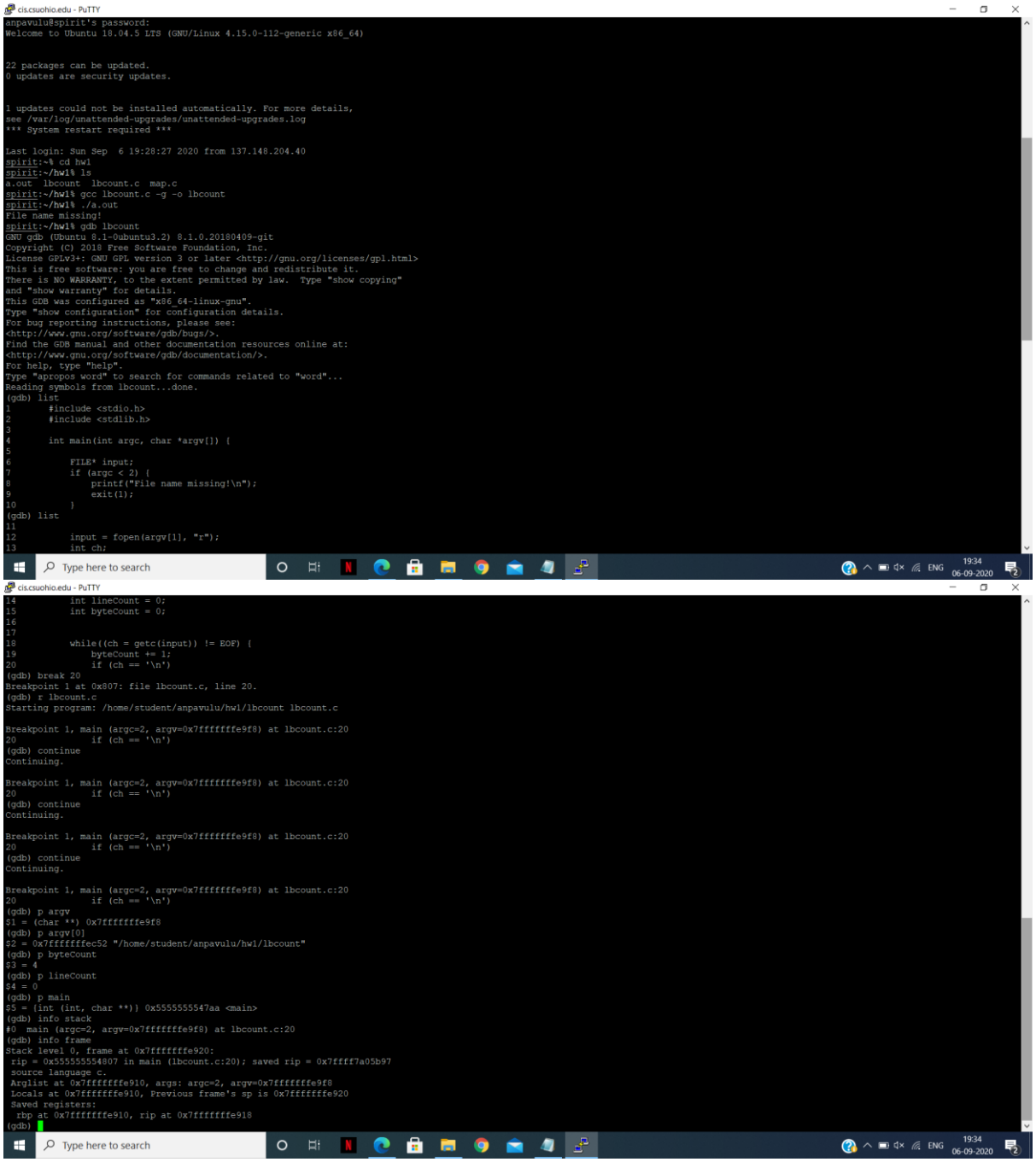
Name: Anil Pavuluru

Login id:anpavulu

Password:2782551P



1) What is the value of argv? (hint: print argv)
   Ans: $1 = (char **) 0x7fffffffe9f8
2) What is pointed to by argv? ? (hint: print argv[0])
   Ans: $2 = 0x7fffffffec52 "/home/student/anpavulu/hw1/lbcount"
3) What is the value of byteCount? lineCount?
   Ans: $3 = 4 byteCount
   $4 = 0 lineCount
4) What is the address of the function main?
   Ans: $5 = {int (int, char **)} 0x5555555547aa <main>

5) Try info stack. Explain what you see.
   Ans: #0 main (argc=2, argv=0x7fffffffe9f8) at lbcount.c:20
   Info stack like wise back trace prints stack frame here the frame #zero it means current frame followed by the argv pointer address of array and string index address and break count number.
6) Try info frame. Explain what you see.
   Ans: Stack level 0, frame at 0x7fffffffe920:
   rip = 0x555555554807 in main (lbcount.c:20); saved rip = 0x7ffff7a05b97
   source language c.
   Arglist at 0x7fffffffe910, args: argc=2, argv=0x7fffffffe9f8
   Locals at 0x7fffffffe910, Previous frame's sp is 0x7fffffffe920
   Saved registers:
   rbp at 0x7fffffffe910, rip at 0x7fffffffe918
   based on my observation of info frame and explaining about what I seen spirit here in the below.
   Info frame: nothing but display advanced info about stack frame parameters like frame number ,frame address

stack level 0

- Zero is current executing frame

frame at 0x7fffffffe920

- starting memory address of this stack frame

rip = 0x555555554807 in main (lbcount.c:20); saved rip = 0x7ffff7a05b97

- rip is the register like program counter for next instruction to execute.so at this moment, the next to execute is at 0x555555554807, which is line 20 of testing.cpp.

- **saved rip== 0x7ffff7a05b97**

we know instruction pointer normally located on the microprocessor which increments with 8 for 64bit system and 4 for 4 bytes system so that it points to next step of instruction.
When program calls main function, the address is saved in register here rip is that register pointer always it returns the address, function jump back after completed.

Source language c

- here “c” language is used.

Arglist at 0x7fffffffe910, args: argc=2, argv=0x7fffffffe9f8

- Starting address of the argument.args and argc are the command line arguments. Argv it is the pointer point to array.

Locals at 0x7fffffffe910

- Local variables address.

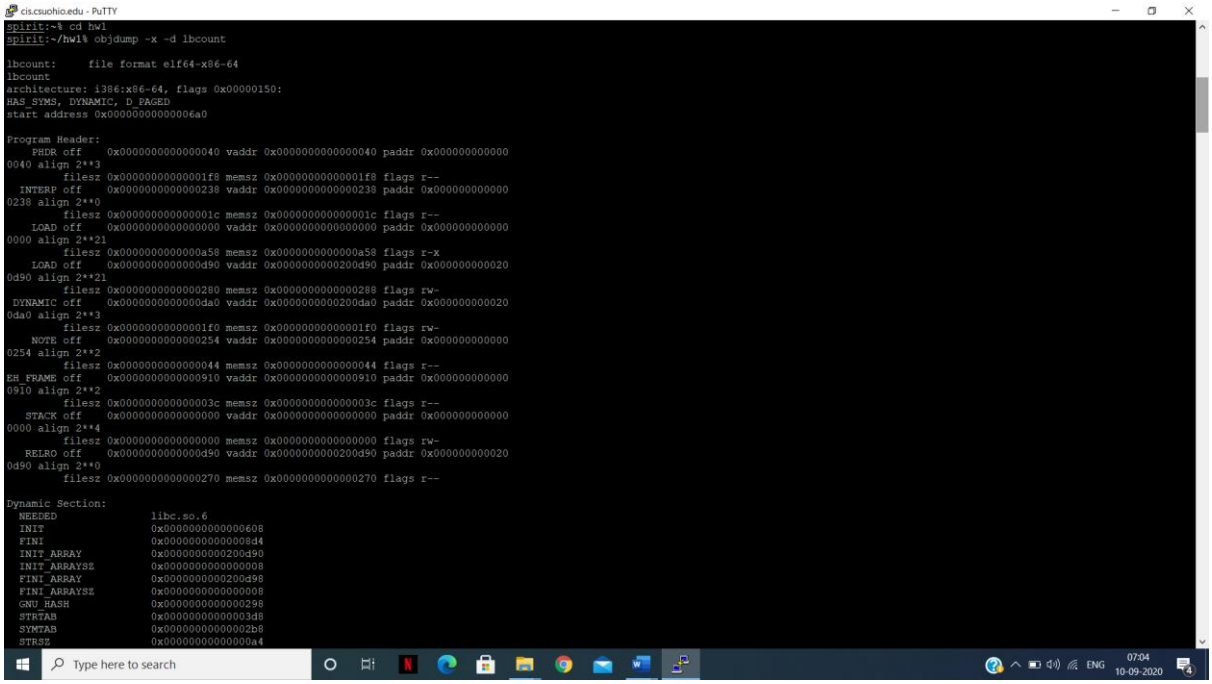
Previous frame's sp is 0x7fffffffe920

- Where previous frame pointer calls the frame while calling at that moment it become first memory address of stack frame.

Saved registers: save the address of the stack

rbp at 0x7fffffffe910, rip at 0x7fffffffe918

- **RIP** is the register automatically it pushed to the stack when you call.
- **RBP** is the register automatically popped from the stack when you return.
- rbp at 0x7fffffffe910 that is the address where the "rbp" register of the caller´s stack frame saved.
- rip at 0x7fffffffe918 as mentioned before, but here is the address of the stack (which contains the value "0x555555554807")



7)What file format is used for this binary? And what architecture is it compiled for?

Ans: lbcount: file format elf64-x86-64

Here elf is used in the file format ELF stands executable and linkable file format

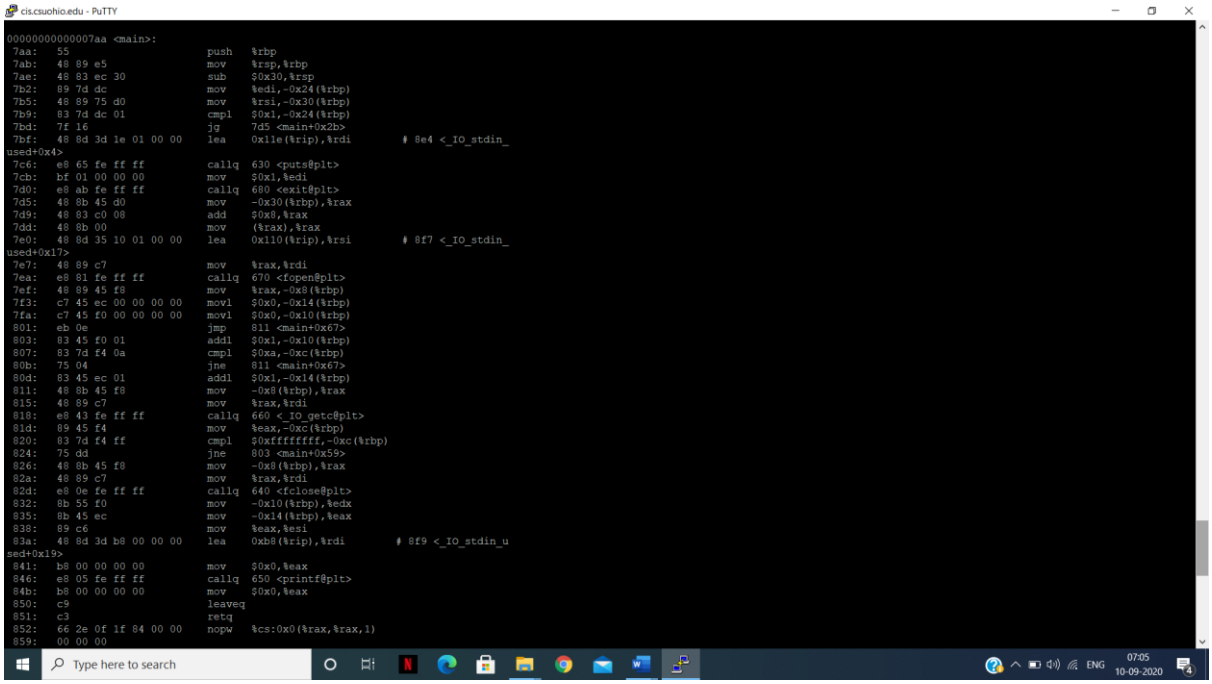
ELF is used as standard file format for object files on Linux.

Now we are using the a.out file format was being used as a standard but lately ELF is used to standard the file format.

64 means number of bits.

x-86 means The regular Program Files folder holds 64-bit.

architecture: i386:x86-64,

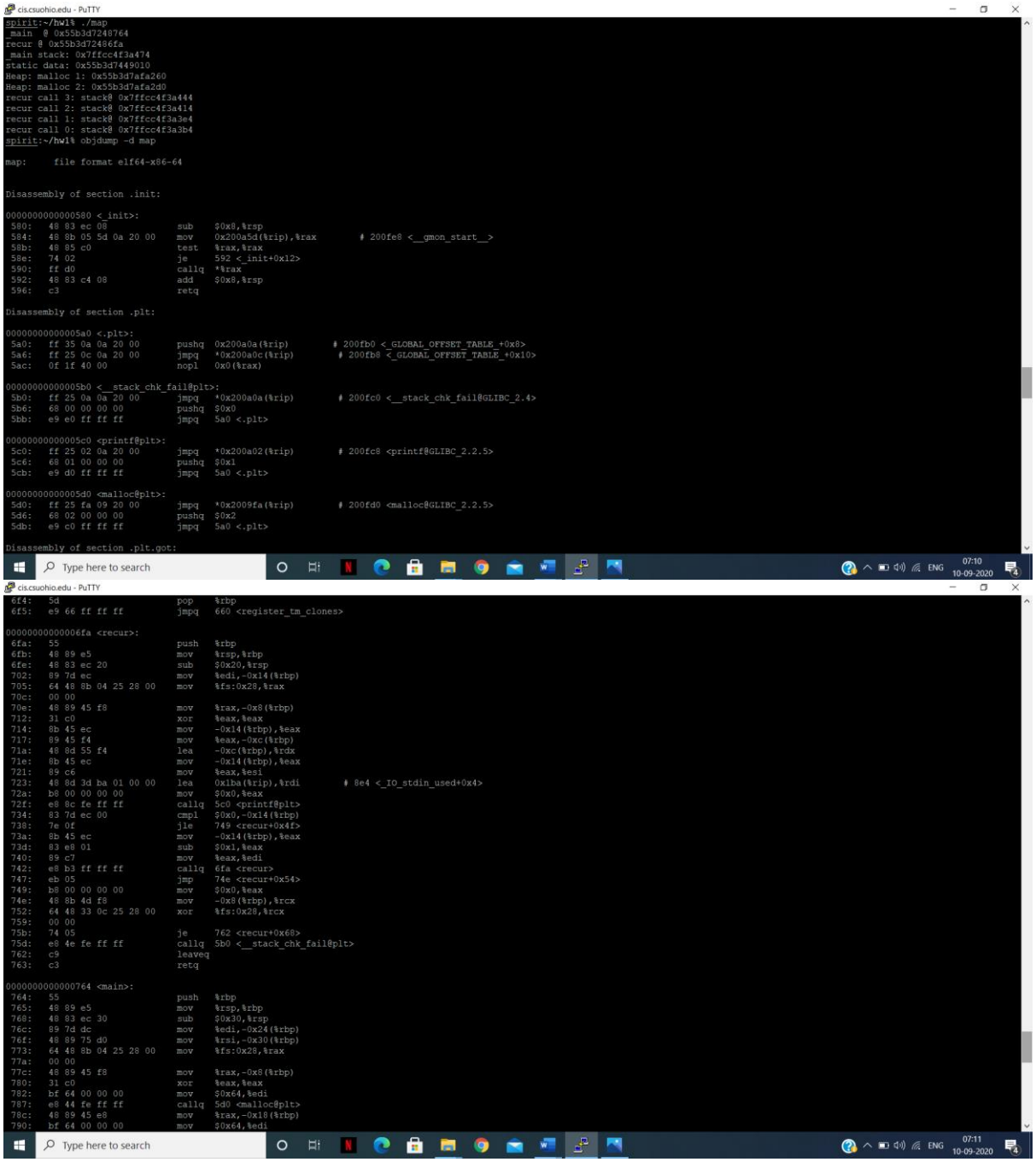


8)What segment/section contains main (the function) and what is the address of main? (The last few hex digits should be the same as what you saw in gdb)

Ans:00000000000007aa <main>:it is located under .txt

9)Do you see the stack segment anywhere? What about the heap? Explain

Ans: stack segment and heap are created while running the program we can say dynamically allocated the memory. But, here we don't execute any output we just examine the object file code. Here there is no chance to create heap and stack segment in object file part.



10)Use objdump with the -D flag on the map executable. Which of the addresses from the output of running ./map are defined in the executable, and which segment/section is each defined in?

Ans: _main @ 0x55f30ee6f764

recur @ 0x55f30ee6f6fa remember these two address and check this address matches by using obj dump also .

this two address are located under disassembly section.text;

0000000000000764 <main>:

00000000000006fa <recur>:

11)Where is the heap? What direction is it growing in?

Ans: based on the examine the code the heap is allocated with uninitialized variables and after creating the memory and it delete and basically it is on the ram side. I think here in the code it is located under the .txtfile like .bss and .relocate the heap is created in that place.

As we see in the direction of heap in malloc is increasing (Upward) whereas recur call it is decreasing (going downward).

12) Are the two malloc()ed memory areas contiguous? (e.g. is there any extra space between their addresses?)

Ans: We allocate memory of malloc100 in the program but when we type the command ./map we can see that malloc1 start address at 60 it occupies spaces from 60 to 6f(16),70 to 7f(32),80 to 8f(48),90 to 9f(64),a0 to af(80),b0 to bf(96),c0,c1,c2,c3(4) total 100bits.

Coming to malloc2 start address d0 here the continued is not there we can 12 bit space .

13) What direction is the stack growing in?

Ans: Downward (e4 to b4 to 84 to 54).

14) How large is the stack frame for each recursive call?

Ans: difference between two frames is 48bits(e4 to b4 or else b4 to 84 or else 84 to 54).