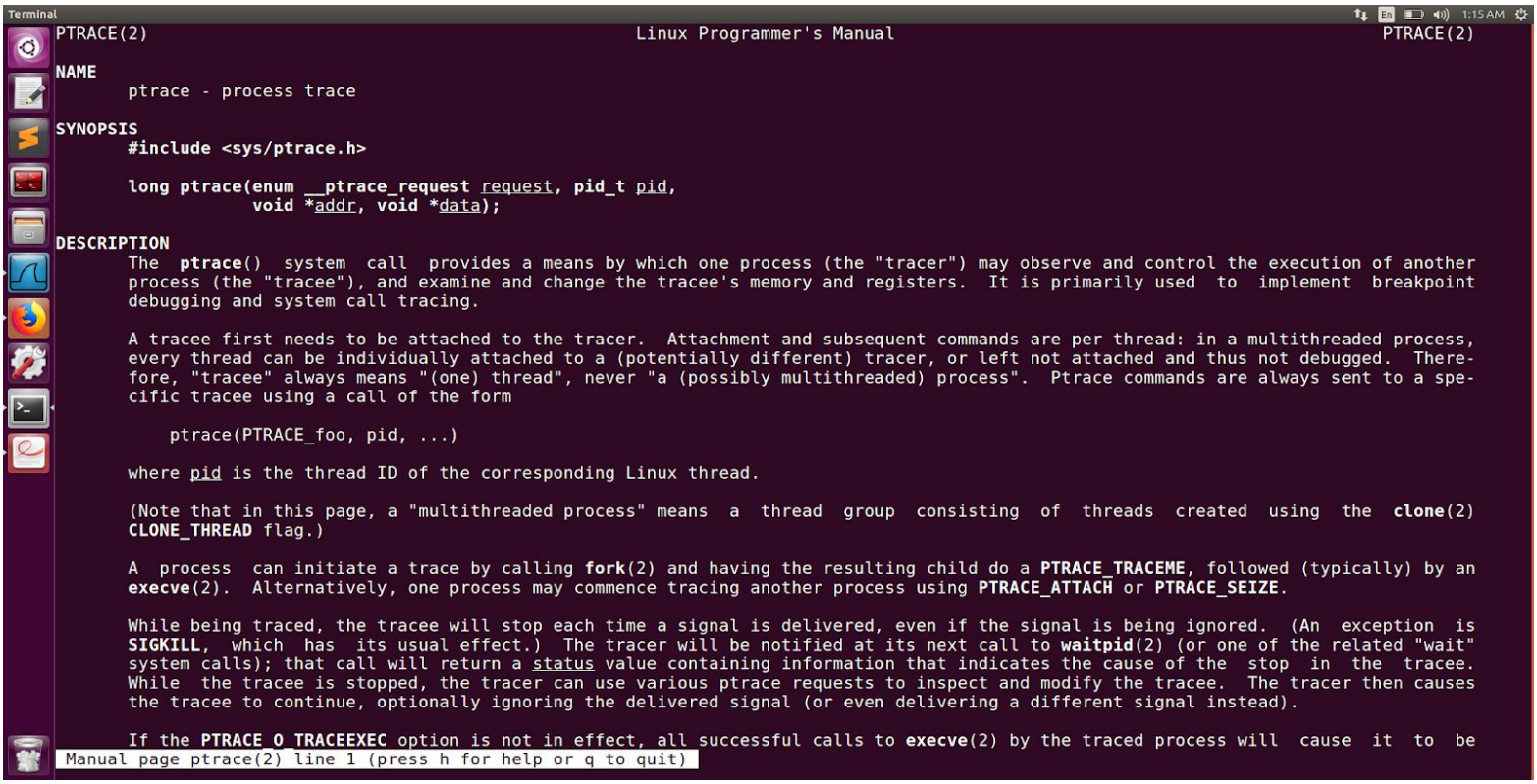
Lab 6: ELF Hijacking via PLT/GOT Poisoning Darshan K(A20446137)

*Task 1.1: ptrace is used by *nix debuggers. With your knowledge of ptrace, explain how gdb can attach to a running process and print out its register contents at a particular point of execution.*



As per the man page of **ptrace** system call, the description part says "***The ptrace() system call provides a means by which one process (the "tracer") may observe and control the execution of another process (the "tracee"), and examine and change the tracee's memory and registers. It is primarily used to implement breakpoint debugging and system call tracing***".

By using this command, which means process trace, one process can control another process enabling the controller to inspect and manipulate the internal state of it's target. It supports quite a bunch of actions. To observe and interfere with a tracee, it follows the following pattern and sends the command,

long ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *data);

Lab 6: ELF Hijacking via PLT/GOT Poisoning **Darshan K(A20446137)**

Task 1.2: Explain how you could build gdb's memory dump (e.g. x/32x) command.

As mentioned in the answer to the previous question, of all the actions it supports, **PTRACE_PEEKTEXT, PTRACE_PEEKDATA,** to get a dump of tracee's memory, we can run **PTRACE_PEEKDATA,** in loop.

```
for(int i=0; i<32; i++){
        Int val = ptrace(PTRACE_PEEKDATA, pid, addr+i, NULL);
        printf("%8x",val);
}
```

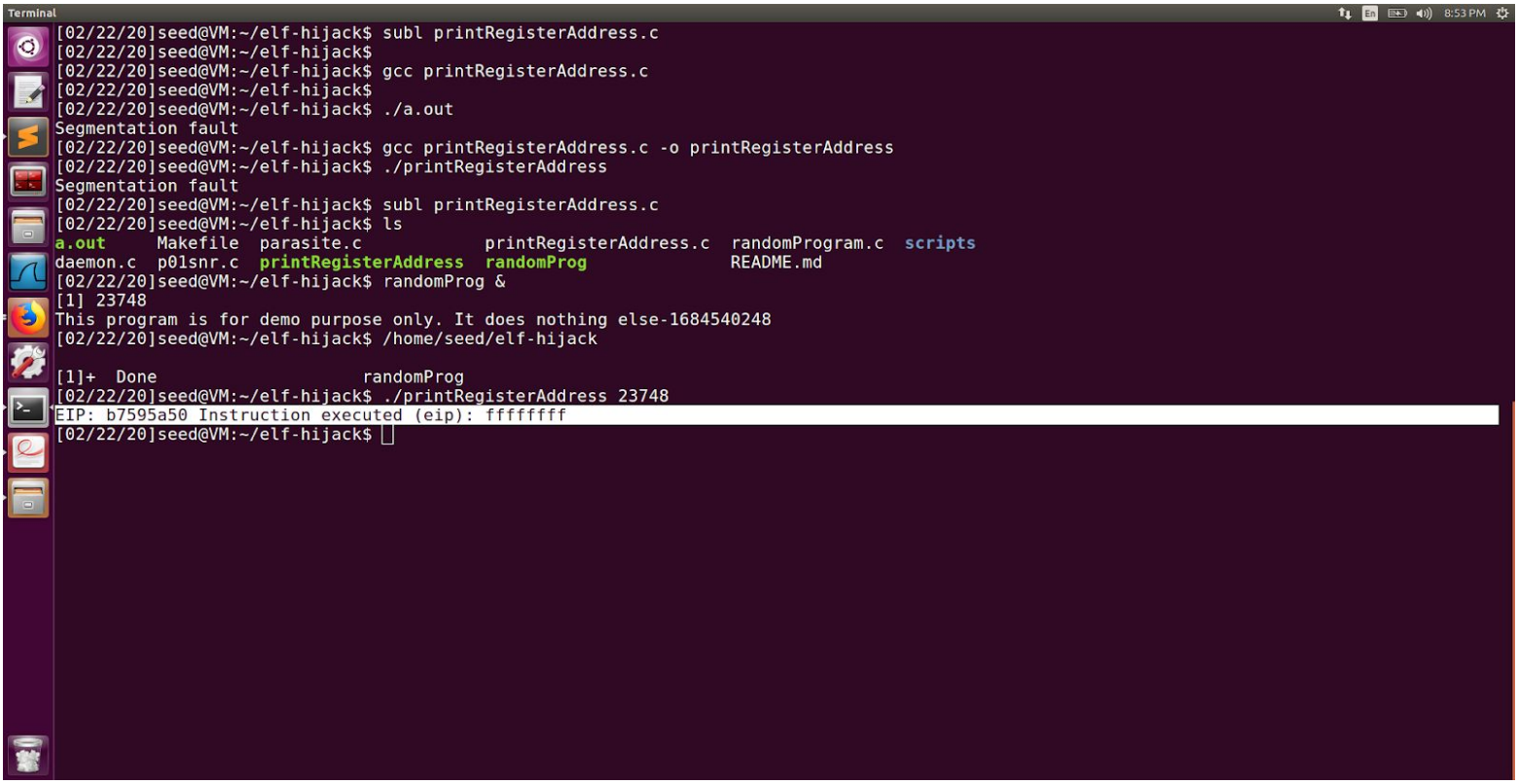
Task 1.3: Write a small program that uses ptrace and accepts a PID as an argument to attach to a running process and print out its current register values. Include the code in your writeup.

Here is simple C program that takes the pid of a random program written for the purpose of executing this task. When run this program, we get to see the output attached below. Here's the code:

```
#include<stdio.h>
#include<sys/ptrace.h>
#include<signal.h>
#include<stdlib.h>
#include<sys/user.h>
#include<sys/types.h>
#include<sys/wait.h>
int main(int argc, char const *argv[])
{
   pid_t tracedProcess = atoi(argv[1]);
   struct user_regs_struct userRegisters;
   ptrace(PTRACE_ATTACH, tracedProcess,NULL, NULL);
   wait(NULL);
   ptrace(PTRACE_GETREGS, tracedProcess,NULL, &userRegisters);
```

```
        long ins = ptrace(PTRACE_PEEKTEXT, tracedProcess,userRegisters.eip, NULL);
        printf("EIP: %lx Instruction executed (eip): %lx\n",userRegisters.eip, ins);
        ptrace(PTRACE_DETACH, tracedProcess,NULL, NULL);
        return 0;
}
```

Output:



Lab 6: ELF Hijacking via PLT/GOT Poisoning **Darshan K(A20446137)**

Task 1.4: Why should ptrace only be available to a privileged process?

It has a capability to change the way the program behaves and also change the memory space of the process. These are the few functionalities that shouldn't be accessible to the unprivileged processes which means only the root can perform these operations.

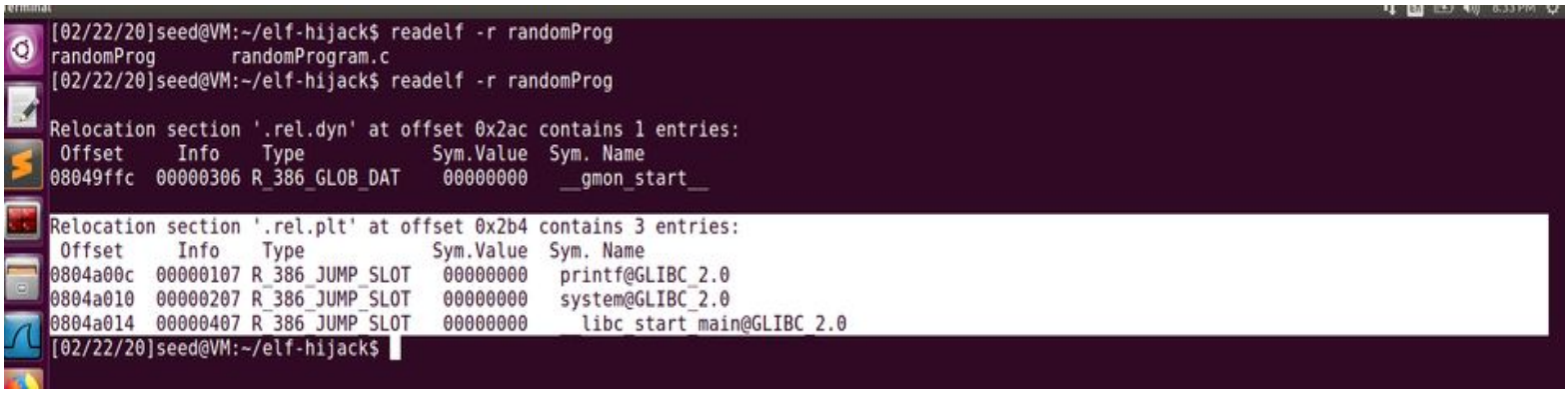
Task 2.1: What is the purpose of PLT?

PLT is Procedure linkage table which is used to call external procedures/functions whose address at the time of linking is not known and it is left to be resolved by the dynamic linker at the run time. It works alongside GOT to reference and relocate the function resolution as needed. Here is a simple program to demonstrate what PLT does..

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int main(int argc, char const *argv[])
{
        printf("This program is for demo purpose only. It does nothing else");
        printf("%d\n", sqrt(5));
        system("pwd");
        return 0;
}
```

The output of the program when run, you can see the addresses of printf and the call to the system function are computed dynamically. Refer to the highlighted section of the output.

Lab 6: ELF Hijacking via PLT/GOT Poisoning Darshan K(A20446137)



Task 2.2: What is the purpose of GOT?

GOT is Global Offset Table. It redirects position independent address calculations to an absolute location and it is located in the .got section of an elf executable or shared object. It stores the final location of a function calls symbol used in the dynamically linked code.

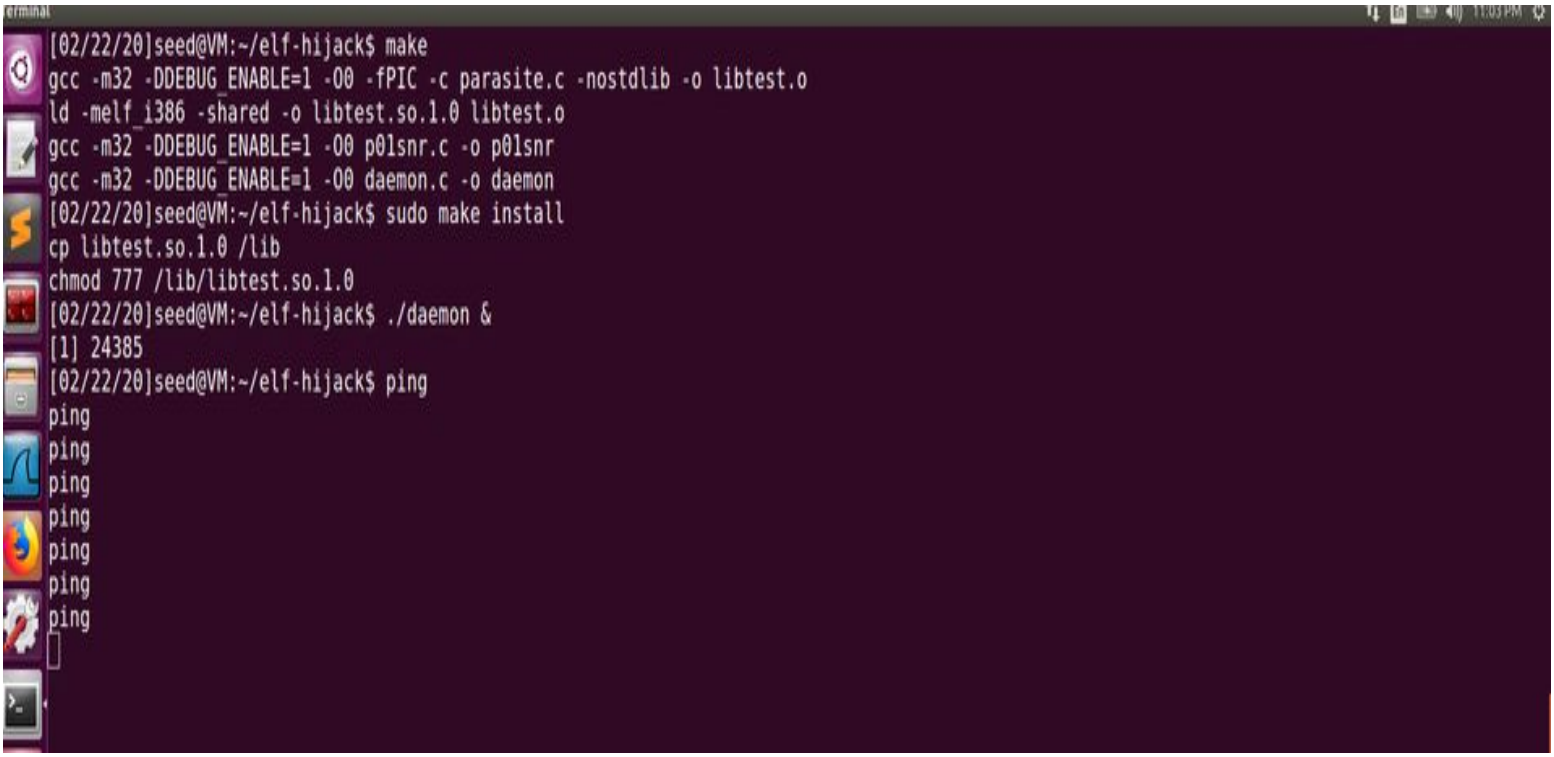
For instance when the program requests to use printf() after the rtld locates the symbol, the location is then relocated in the GOT, and allows for the executable via the PLT to directly access the symbol location.

Task 3

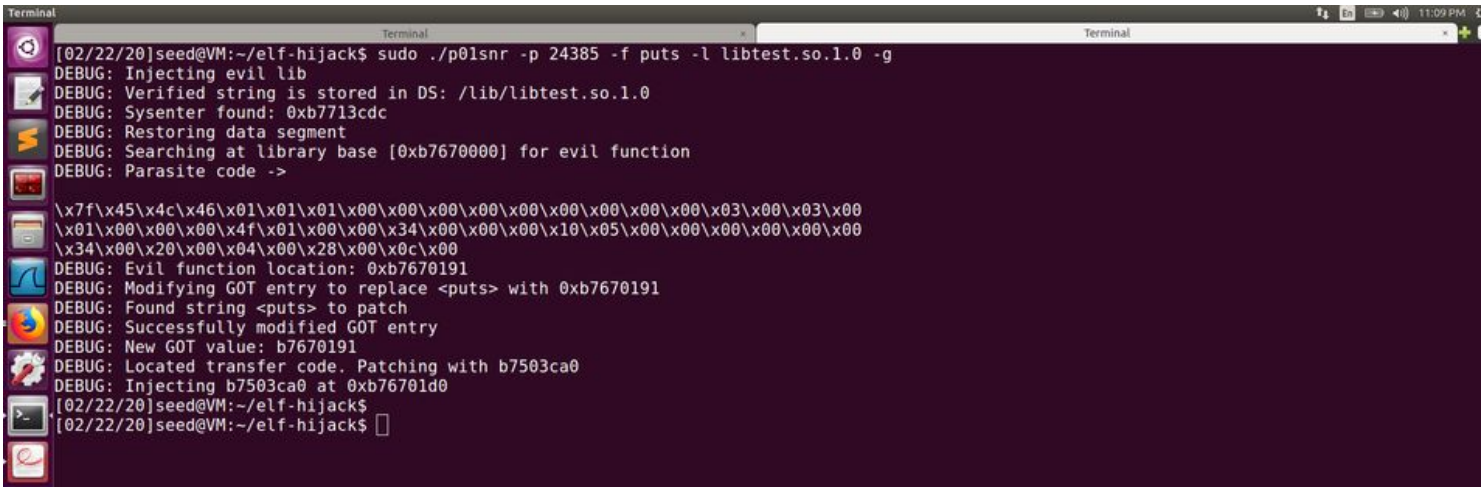
For this task, we have 3 components. The first is the simple target daemon process that prints a string “ping” every few seconds. The second is the parasite library and the third is the injection tool. On executing **make** it builds all the code, the second command **sudo make install** installs the parasite library into a directory and the third one invokes the target as a background . Here’s the screenshot of what it looks like on executing these commands.

Lab 6: ELF Hijacking via PLT/GOT Poisoning

Darshan K(A20446137)



On executing **sudo ./p01snr -p 24385 -f puts -l libtest.so.1.0 -g** with 24385 being the pid of the **daemon** process. Here is the output of it.



The execution of this code prints “I am evil” and here’s how it looks.

Lab 6: ELF Hijacking via PLT/GOT Poisoning **Darshan K(A20446137)**

Creating a logic bomb:

I'm not very sure of how things are to be implemented to launch the attack.