

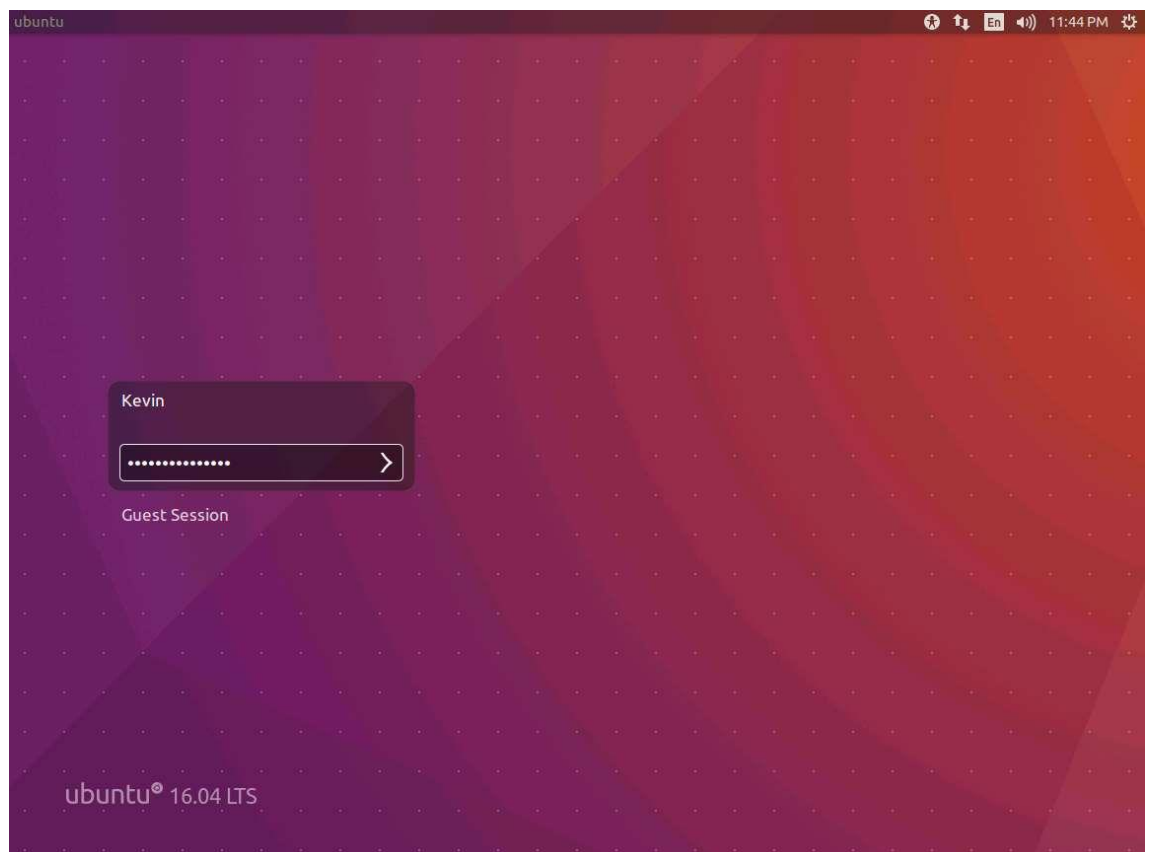
Exercise 2: Binary Analysis on a 64-bit Machine

Lab Objective:

In this lab, we will explore the creation and data for assembly language.

Lab Tasks

1. ☐ Login to the [Software-Test-Linux](#) machine using **studentpassword** as Password.



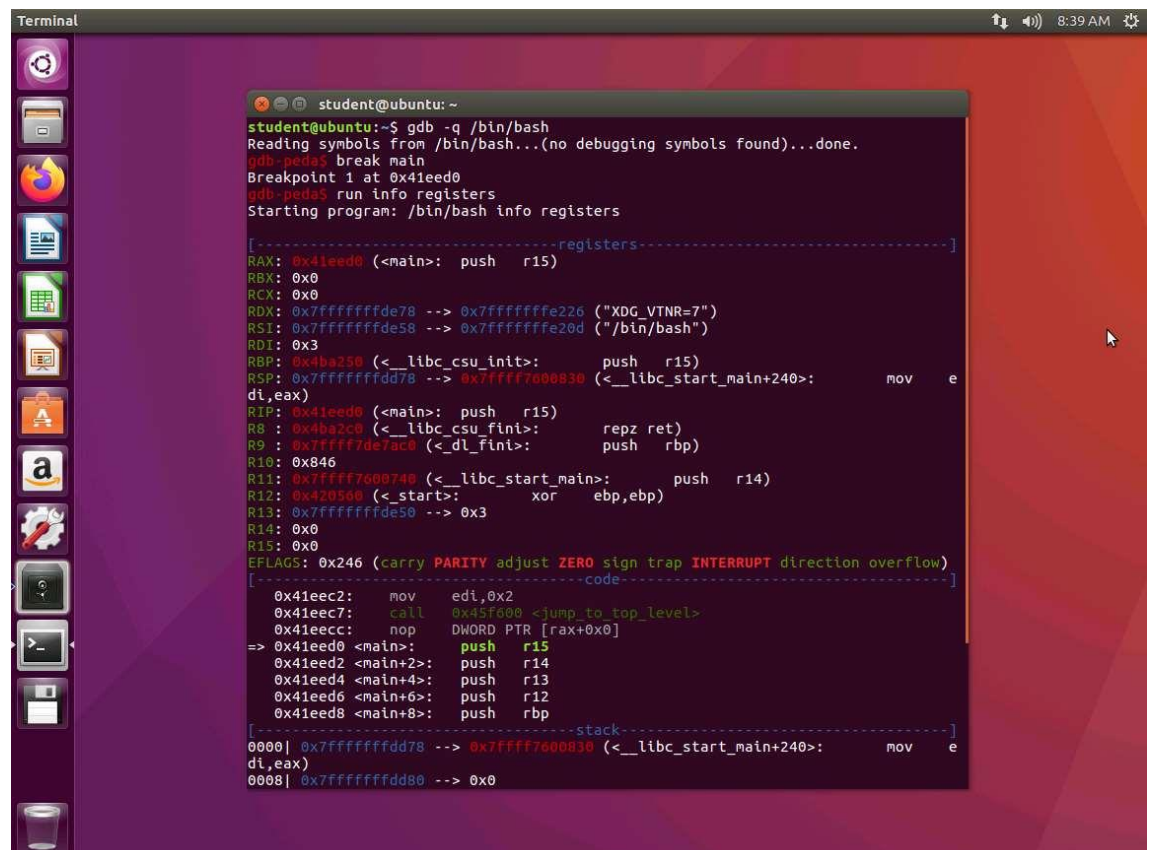
2. ☐ In the 64-bit machine, enter the following:

a. **`gdb -q /bin/bash`**

b. **`break main`**

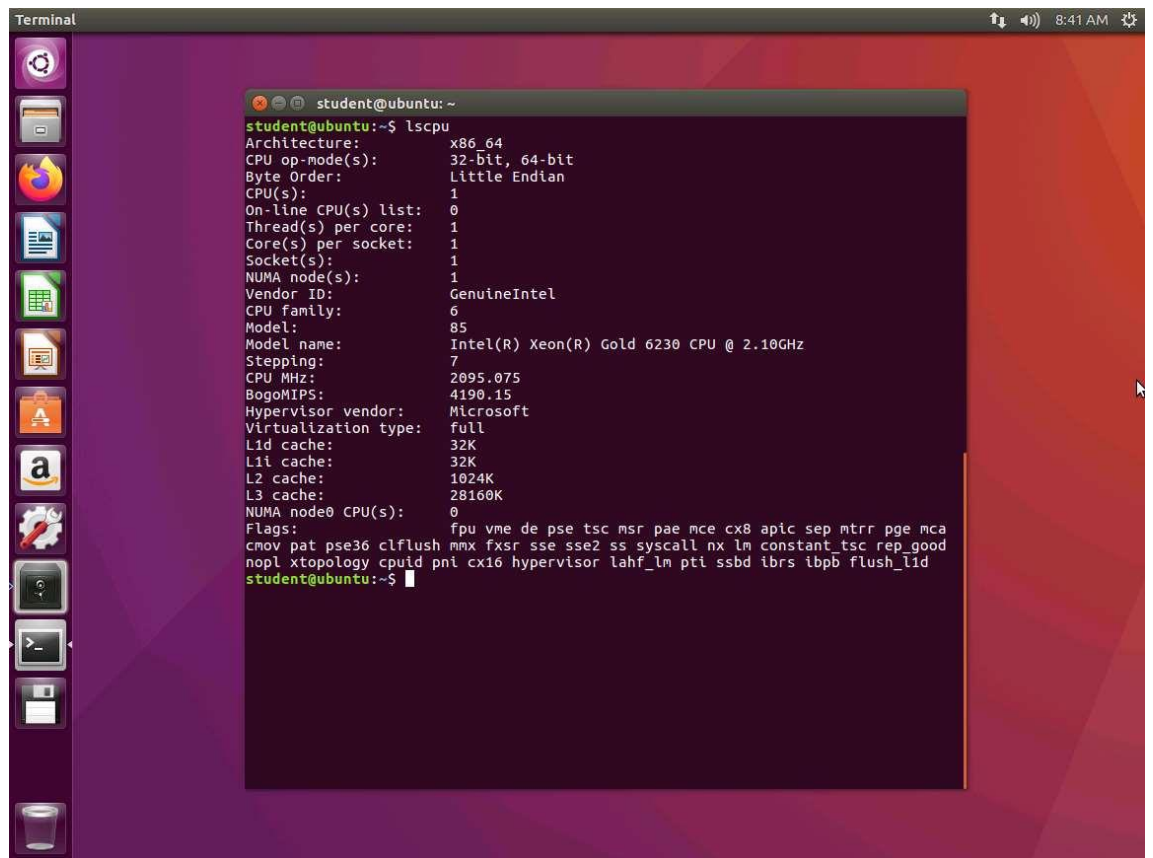
c. **`run info registers`**

The output of this command is shown in the following screenshot.



```
student@ubuntu: ~  
student@ubuntu:~$ gdb -q /bin/bash  
Reading symbols from /bin/bash...(no debugging symbols found)...done.  
gdb-peda$ break main  
Breakpoint 1 at 0x41eed0  
gdb-peda$ run info registers  
Starting program: /bin/bash info registers  
[-----registers-----]  
RAX: 0x41eed0 (<main>: push r15)  
RBX: 0x0  
RCX: 0x0  
RDX: 0x7fffffffde78 --> 0x7fffffffde226 ("XDG_VTNR=7")  
RSI: 0x7fffffffde58 --> 0x7fffffffde20d ("/bin/bash")  
RDI: 0x3  
RBP: 0x4ba250 (<_libc_csu_init>: push r15)  
RSP: 0x7fffffffdd78 --> 0x7ffff7600830 (<_libc_start_main+240>: mov e  
dl,eax)  
RIP: 0x41eed0 (<main>: push r15)  
R8 : 0x4ba2c0 (<_libc_csu_fini>: repz ret)  
R9 : 0x7ffff7de7ac0 (<_dl_fini>: push rbp)  
R10: 0x846  
R11: 0x7ffff7600740 (<_libc_start_main>: push r14)  
R12: 0x420560 (<_start>: xor ebp,ebp)  
R13: 0x7fffffffde50 --> 0x3  
R14: 0x0  
R15: 0x0  
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)  
[-----code-----]  
0x41eec2: mov edi,0x2  
0x41eec7: call 0x45f600 <jump_to_top_level>  
0x41eec8: nop DWORD PTR [rax+0x0]  
=> 0x41eed0 <main>: push r15  
0x41eed2 <main+2>: push r14  
0x41eed4 <main+4>: push r13  
0x41eed6 <main+6>: push r12  
0x41eed8 <main+8>: push rbp  
[-----stack-----]  
0000| 0x7fffffffdd78 --> 0x7ffff7600830 (<_libc_start_main+240>: mov e  
dl,eax)  
0008| 0x7fffffffdd80 --> 0x0
```

3. ☐ As the screenshot in Step 2 shows, we have additional registers along with the 64-bit version of the registers we have already discussed. Next, enter **`lscpu`**. The output of this command is shown in the following screenshot.

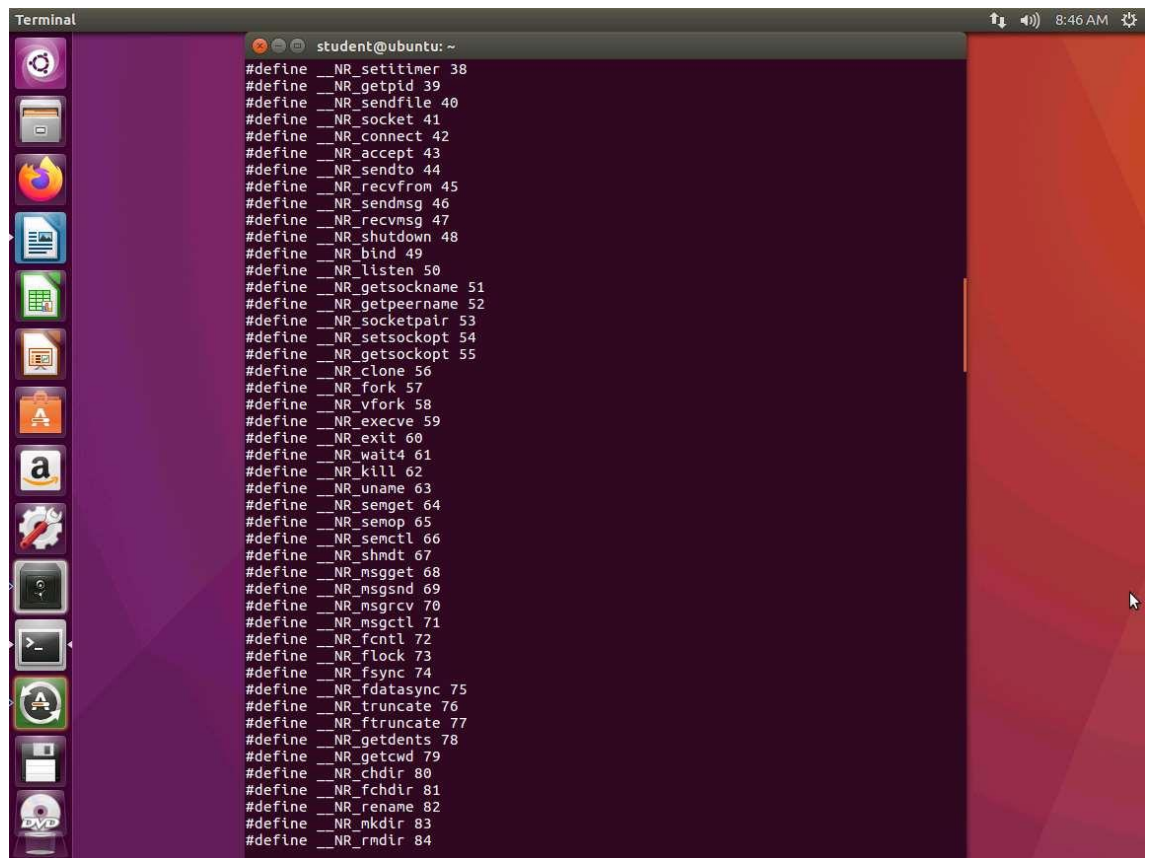


```
student@ubuntu: ~  
student@ubuntu:~$ lscpu  
Architecture:          x86_64  
CPU op-mode(s):        32-bit, 64-bit  
Byte Order:            Little Endian  
CPU(s):                1  
On-line CPU(s) list:   0  
Thread(s) per core:    1  
Core(s) per socket:    1  
Socket(s):             1  
NUMA node(s):          1  
Vendor ID:             GenuineIntel  
CPU family:            6  
Model:                 85  
Model name:            Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz  
Stepping:              7  
CPU MHz:               2095.075  
BogoMIPS:              4190.15  
Hypervisor vendor:     Microsoft  
Virtualization type:   full  
L1d cache:             32K  
L1i cache:             32K  
L2 cache:              1024K  
L3 cache:              28160K  
NUMA node0 CPU(s):    0  
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca  
cmov pat pse36 clflush mmx fxsr sse sse2 ss syscall nx lm constant_tsc rep_good  
noptl xtopology cpuid pni cx16 hypervisor lahf_lm pti ssbd ibrs ibpb flush_l1d  
student@ubuntu:~$
```

4. ☐ Next, enter **cat /usr/include/x86_64-linux-gnu/asm/unistd_64.h**.

Terminal 8:44 AM

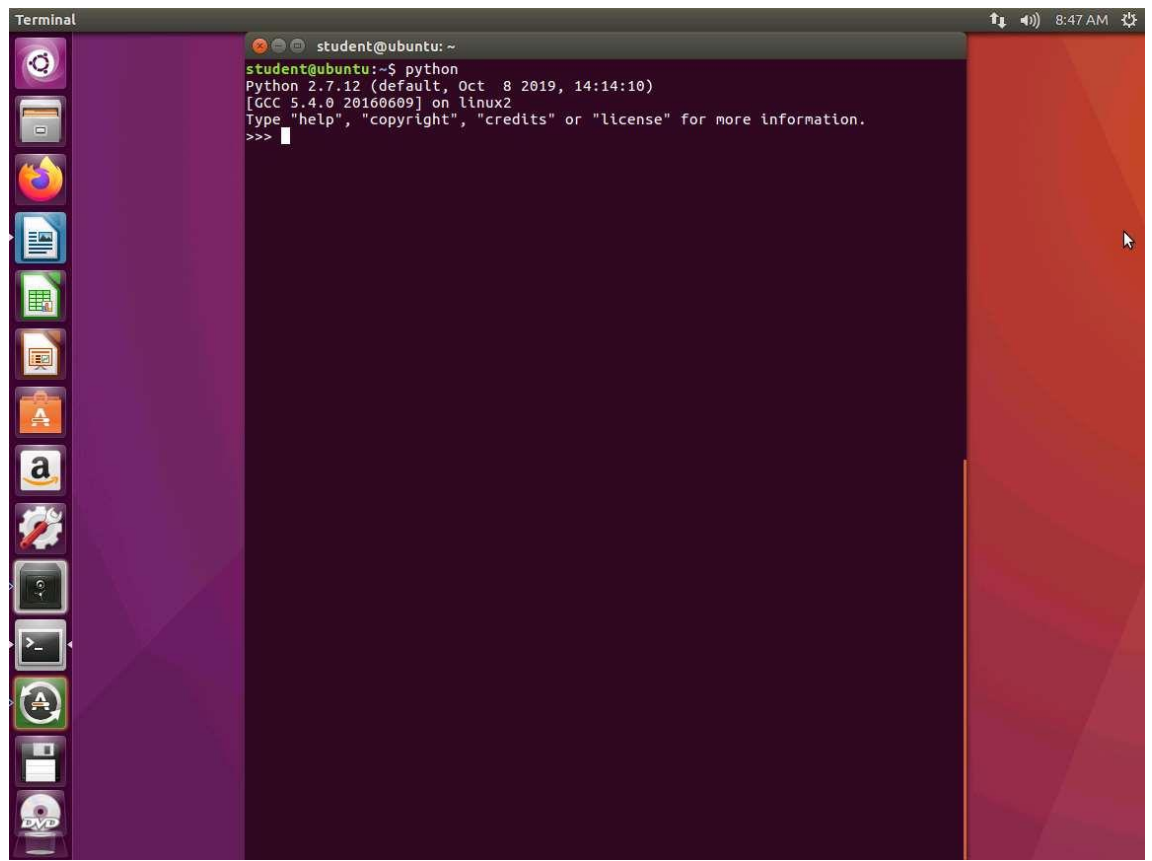
```
student@ubuntu: ~  
student@ubuntu:~$ cat /usr/include/x86_64-linux-gnu/asm/unistd_64.h  
#ifndef _ASM_X86_UNISTD_64_H  
#define _ASM_X86_UNISTD_64_H 1  
  
#define __NR_read 0  
#define __NR_write 1  
#define __NR_open 2  
#define __NR_close 3  
#define __NR_stat 4  
#define __NR_fstat 5  
#define __NR_lstat 6  
#define __NR_poll 7  
#define __NR_lseek 8  
#define __NR_mmap 9  
#define __NR_mprotect 10  
#define __NR_munmap 11  
#define __NR_brk 12  
#define __NR_rt_sigaction 13  
#define __NR_rt_sigprocmask 14  
#define __NR_rt_sigreturn 15  
#define __NR_ioctl 16  
#define __NR_pread64 17  
#define __NR_pwrite64 18  
#define __NR_readv 19  
#define __NR_writev 20  
#define __NR_access 21  
#define __NR_pipe 22  
#define __NR_select 23  
#define __NR_sched_yield 24  
#define __NR_mremap 25  
#define __NR_msync 26  
#define __NR_mincore 27  
#define __NR_madvise 28  
#define __NR_shmget 29  
#define __NR_shmat 30  
#define __NR_shmctl 31  
#define __NR_dup 32
```



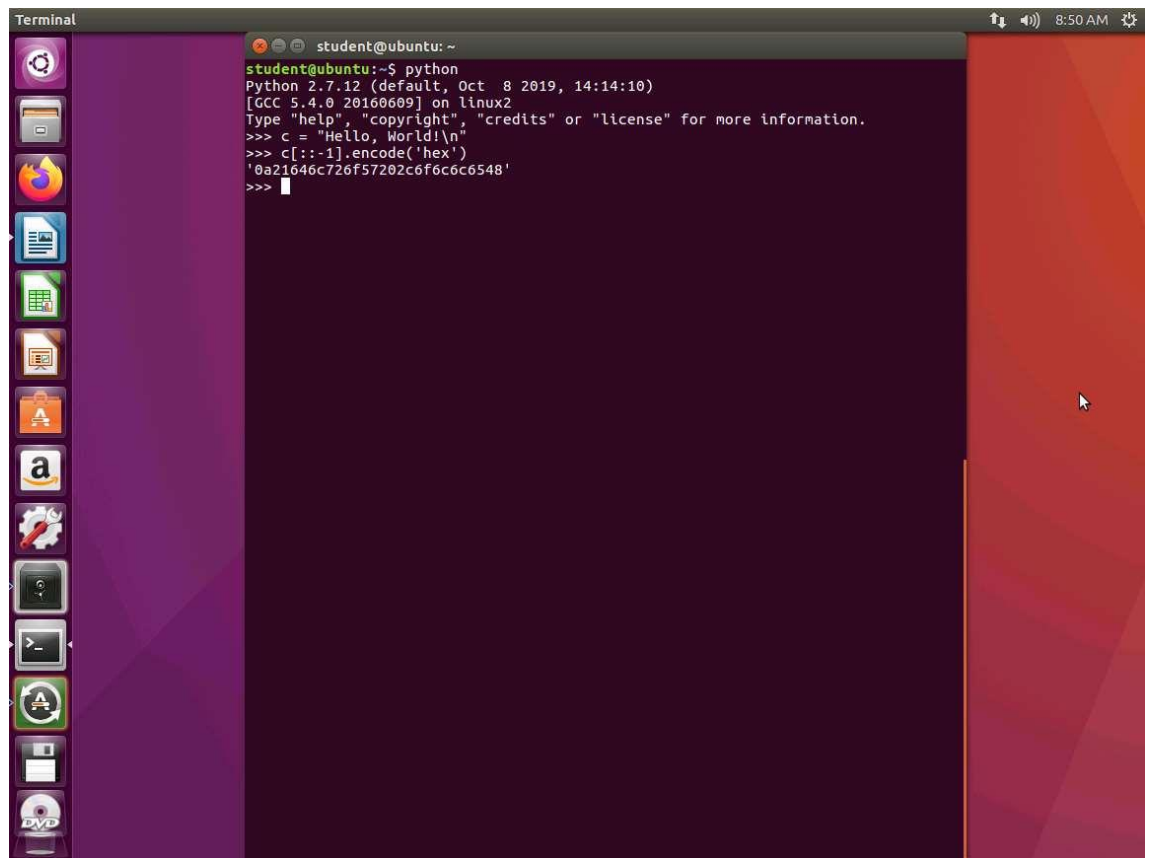
The screenshot shows a terminal window titled "Terminal" with a dark background. The prompt is "student@ubuntu: ~". The terminal displays a list of system call constants, each preceded by a "#define" and a comment. The constants are listed in ascending order of their numeric value, starting from 38 and ending at 84. The constants include: _NR_setitimer, _NR_getpid, _NR_sendfile, _NR_socket, _NR_connect, _NR_accept, _NR_sendto, _NR_recvfrom, _NR_sendmsg, _NR_recvmsg, _NR_shutdown, _NR_bind, _NR_listen, _NR_getsockname, _NR_getpeername, _NR_socketpair, _NR_setsockopt, _NR_getsockopt, _NR_clone, _NR_fork, _NR_vfork, _NR_execve, _NR_exit, _NR_wait4, _NR_kill, _NR_uname, _NR_semget, _NR_semop, _NR_semctl, _NR_shmctl, _NR_msgget, _NR_msgsnd, _NR_msgrcv, _NR_msgctl, _NR_fcntl, _NR_flock, _NR_fsync, _NR_fdatasync, _NR_truncate, _NR_ftruncate, _NR_getdents, _NR_getcwd, _NR_chdir, _NR_fchdir, _NR_rename, _NR_mkdir, and _NR_rmdir.

```
student@ubuntu: ~  
#define _NR_setitimer 38  
#define _NR_getpid 39  
#define _NR_sendfile 40  
#define _NR_socket 41  
#define _NR_connect 42  
#define _NR_accept 43  
#define _NR_sendto 44  
#define _NR_recvfrom 45  
#define _NR_sendmsg 46  
#define _NR_recvmsg 47  
#define _NR_shutdown 48  
#define _NR_bind 49  
#define _NR_listen 50  
#define _NR_getsockname 51  
#define _NR_getpeername 52  
#define _NR_socketpair 53  
#define _NR_setsockopt 54  
#define _NR_getsockopt 55  
#define _NR_clone 56  
#define _NR_fork 57  
#define _NR_vfork 58  
#define _NR_execve 59  
#define _NR_exit 60  
#define _NR_wait4 61  
#define _NR_kill 62  
#define _NR_uname 63  
#define _NR_semget 64  
#define _NR_semop 65  
#define _NR_semctl 66  
#define _NR_shmctl 67  
#define _NR_msgget 68  
#define _NR_msgsnd 69  
#define _NR_msgrcv 70  
#define _NR_msgctl 71  
#define _NR_fcntl 72  
#define _NR_flock 73  
#define _NR_fsync 74  
#define _NR_fdatasync 75  
#define _NR_truncate 76  
#define _NR_ftruncate 77  
#define _NR_getdents 78  
#define _NR_getcwd 79  
#define _NR_chdir 80  
#define _NR_fchdir 81  
#define _NR_rename 82  
#define _NR_mkdir 83  
#define _NR_rmdir 84
```

5. ☐ As you can see from the above screenshot, we have many more here with the write system call at 1. The exit is at 60.
6. ☐ Next, enter **python** to enter the python editor.



7. ☐ In the editor, enter `c = "Hello, World!\n"`.
8. ☐ Next, enter `c[::-1].encode('hex')`. The output of the command is shown in the following screenshot.

A screenshot of a terminal window on an Ubuntu desktop. The terminal title is "Terminal". The prompt is "student@ubuntu: ~". The user has entered "python", which has started the Python 2.7.12 interpreter. The prompt is now ">>>". The user has entered "c = 'Hello, World!\n'", which has been accepted. The user has then entered "c[::-1].encode('hex')", which has produced the output "'0a21646c726f57202c666c66548'". The user has entered ">>>" again, and the prompt is still ">>>". The desktop background is a purple and red geometric pattern. The left sidebar contains icons for Dash, Home, Firefox, LibreOffice, and several other applications. The top status bar shows the time as 8:50 AM and some system icons.

9. ☐ Once the Python prompt is available, we initialize a variable called **c** and store the string as **Hello, World!\n**. Next, we use Python's power of string manipulation, and in one line, reverse the order of the characters in the string with the **c[::-1]** syntax. Then, we call the encode method to encode our characters in their hexadecimal representation. This is done with the **.encode('hex')** part of the syntax. This makes our string almost usable for assembly.
10. ☐ In the terminal window, enter **readelf -a -W ~/examples/samplecode/helloworld64-s**. The output of this command is shown in the following screenshot.

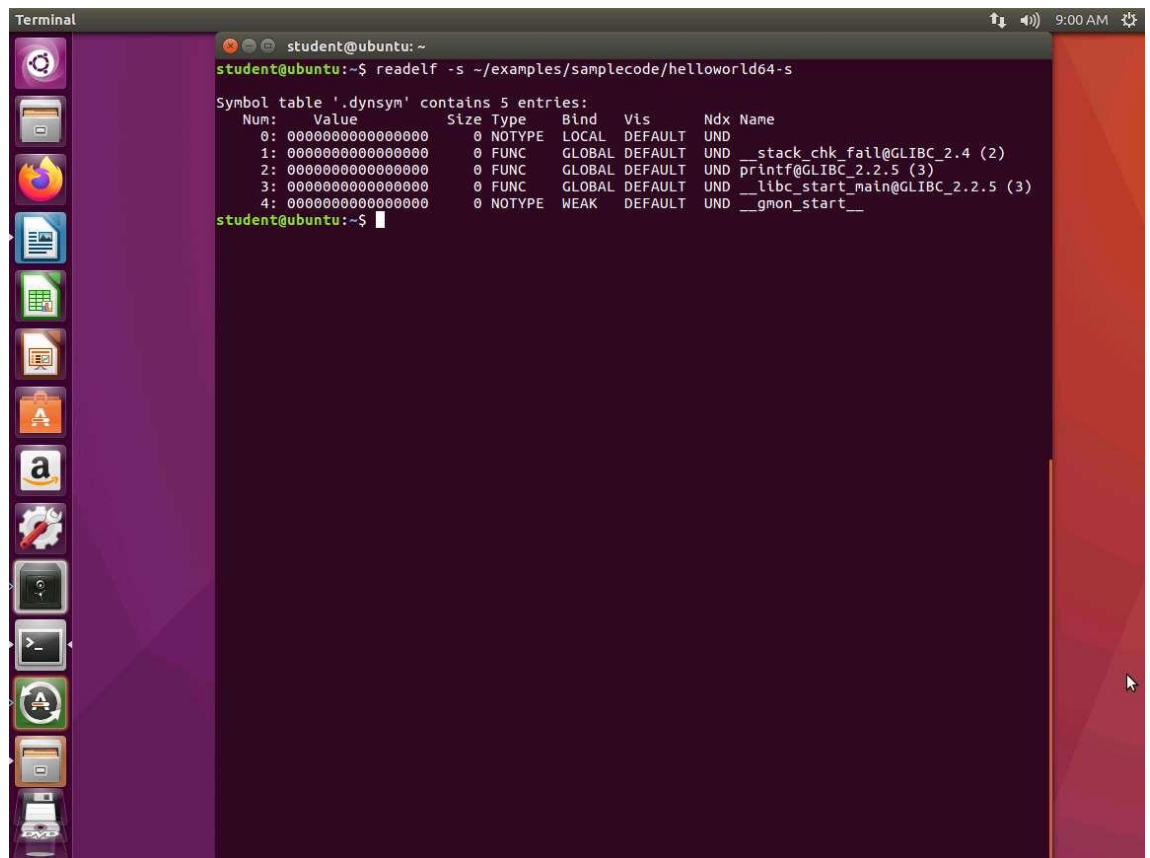

```

student@ubuntu: ~
student@ubuntu:~$ readelf -a -W ~/examples/samplecode/helloworld64-s
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                                ELF64
  Data:                                      2's complement, little endian
  Version:                               1 (current)
  OS/ABI:                                UNIX - System V
  ABI Version:                           0
  Type:                                  EXEC (Executable file)
  Machine:                               Advanced Micro Devices X86-64
  Version:                               0x1
  Entry point address:                   0x4004a0
  Start of program headers:              64 (bytes into file)
  Start of section headers:              4472 (bytes into file)
  Flags:                                  0x0
  Size of this header:                   64 (bytes)
  Size of program headers:               56 (bytes)
  Number of program headers:              9
  Size of section headers:               64 (bytes)
  Number of section headers:              29
  Section header string table index:     28

Section Headers:
[Nr] Name                               Type           Address         Off    Size  ES Flg Lk Inf AL
[ 0]                               NULL           0000000000000000 000000 000000 00  0  0  0
[ 1] .interp                          PROGBITS       0000000000400238 000238 00001c 00  A  0  0  1
[ 2] .note.ABI-tag                     NOTE           0000000000400254 000254 000020 00  A  0  0  4
[ 3] .note.gnu.build-id                 NOTE           0000000000400274 000274 000024 00  A  0  0  4
[ 4] .gnu.hash                          GNU HASH       0000000000400298 000298 00001c 00  A  5  0  8
[ 5] .dynsym                           DYNSYM        00000000004002b8 0002b8 000078 18  A  6  1  8
[ 6] .dynstr                           STRTAB        0000000000400330 000330 00005a 00  A  0  0  1
[ 7] .gnu.version                       VERSYM        000000000040038a 00038a 00000a 02  A  5  0  2
[ 8] .gnu.version_r                     VERNEED       0000000000400398 000398 000030 00  A  6  1  8
[ 9] .rela.dyn                          RELA          00000000004003c8 0003c8 000018 18  A  5  0  8
[10] .rela.plt                          RELA          00000000004003e0 0003e0 000048 18  AI  5  24  8
[11] .init                              PROGBITS       0000000000400428 000428 00001a 00  AX  0  0  4
[12] .plt                              PROGBITS       0000000000400450 000450 000040 10  AX  0  0  16
[13] .plt.got                          PROGBITS       0000000000400490 000490 000008 00  AX  0  0  8
[14] .text                              PROGBITS       00000000004004a0 0004a0 000222 00  AX  0  0  16
[15] .fini                              PROGBITS       00000000004006c4 0006c4 000009 00  AX  0  0  4
[16] .rodata                           PROGBITS       00000000004006d0 0006d0 000007 00  A  0  0  4
[17] .eh_frame_hdr                     PROGBITS       00000000004006d8 0006d8 000034 00  A  0  0  4
[18] .eh_frame                         PROGBITS       0000000000400710 000710 0000f4 00  A  0  0  8
[19] .init_array                       INIT_ARRAY     0000000000600e10 000e10 000008 00  WA  0  0  8
[20] .fini_array                       FINI_ARRAY     0000000000600e18 000e18 000008 00  WA  0  0  8
[21] .jcr                              PROGBITS       0000000000600e20 000e20 000008 00  WA  0  0  8

```

11. ☐ The **-a** argument displays the ELF header, program headers, section headers, symbols, relocations, dynamic section, version information, architecture-specific information, and a histogram of bucket list lengths. We can also see that we are dealing with an executable file as opposed to a relocatable object file and the address in memory where execution begins is **0x4004a0**. There are nine program headers, each of which is 56 bytes in size, and thirty-one section headers, each of which is 64 bytes in size.
12. ☐ Next, enter **readelf -s ~/examples/samplecode/helloworld64-s**. The output of the symbol table is shown in the following screenshot.



```
Terminal
student@ubuntu: ~
student@ubuntu:~$ readelf -s ~/examples/samplecode/helloworld64-s

Symbol table '.dynsym' contains 5 entries:
  Num:  Value              Size Type Bind Vis Ndx Name
   0:  0000000000000000      0 NOTYPE LOCAL DEFAULT UND
   1:  0000000000000000      0 FUNC GLOBAL DEFAULT UND __stack_chk_fail@GLIBC_2.4 (2)
   2:  0000000000000000      0 FUNC GLOBAL DEFAULT UND printf@GLIBC_2.2.5 (3)
   3:  0000000000000000      0 FUNC GLOBAL DEFAULT UND __libc_start_main@GLIBC_2.2.5 (3)
   4:  0000000000000000      0 NOTYPE WEAK DEFAULT UND __gmon_start__
student@ubuntu:~$
```

13. ☐ What we can gather from this part of the output is that the **printf()** function is used somewhere in the program. Larger programs with more code and those that use additional functions from shared libraries will have many more functions linked dynamically similar to this. Continuing to review the output, the .symtab section shows us all symbol references in the program, including any variables or function names, and immediately.
14. ☐ The lab objectives have been achieved. Close all windows and clean up from the exercise as required.