CMPT-300
Assignment 3
Assignment 3 NYRC

Please Indicate what components you have completed(Yes,Partial,No):
*(For any partial Completion Please give description in report)
**(Mark penalties will be assessed for claiming a part is present in code when it is not!)

Part A(25%):

Setup Environment(Yes,Partial,No):: Yes
Compile Kernel: Yes
Modified: Kernel: Yes

Part B(75%):

Kernel Space:

Error Handling: Yes
Basic Process Info(Info on input PID): Yes
Linked List Traversal(Relative PID Info): Yes

User-Space:
Testing Code: Yes

General Question:
Briefly Describe What you have learned in this assignment:

In this assignment, we learned how to set up a virtual machine using QEMU and install a custom kernel. Through this kernel, we were able to implement and test our own system calls. Implementing a system call requires adding the system call pointer to the system calls table and defining how to reference the system call in unistd_32.h.
In regards to creating the actual system call, we learned how to access information about a process given its **task_struct**. Particularly, by finding the **task_struct** of a process with **find_task_by_pid**, we were able to access all the information in its struct to find the values for each of the prinfo fields. Furthermore, in this assignment, we learned the structure of processes being doubly linked lists and how to traverse them and how to access their children and siblings.

Part A: Briefly answer these questions(You only need a few sentences each).

1) What is QEMU and how did you use it in this assignment?

QEMU is a process emulator which can emulate many different CPUs and is a full system emulator capable of accessing the network and to emulate peripheral devices. Using QEMU as a system emulator, we installed the latest Linux kernel within in order to test system calls otherwise unable to since that would require constructing the system each time we edited the system call. Now, with a test kernel, we are able to continuously build it to create a system call.

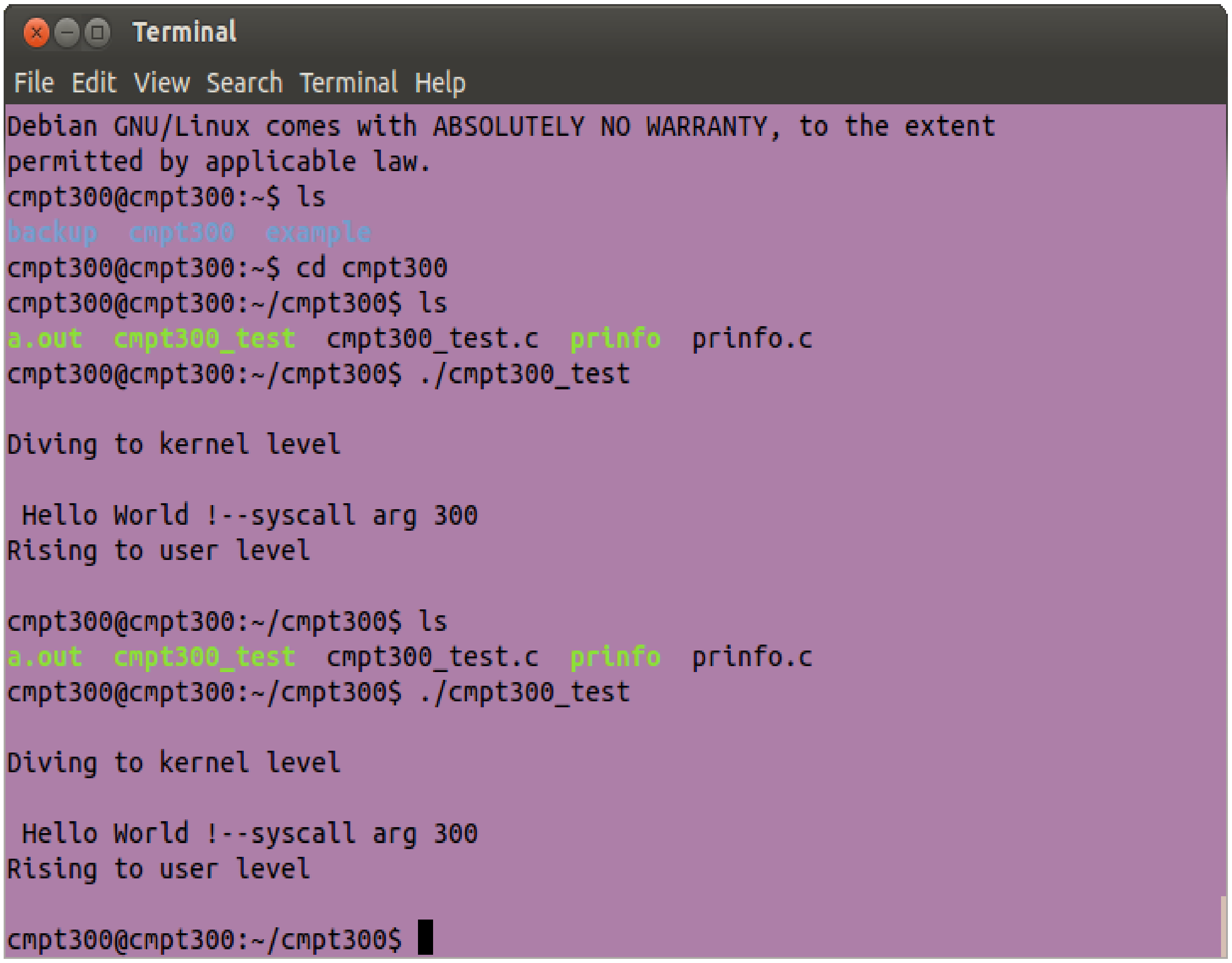
2) What is contained in arch/x86/kernel/syscall_table_32.S. How does the operating system use the syscall table?

Contained in **arch/x86/kernel/syscall_table_32.S** is the system call table which has a list of all system calls in the system. Any new system call added to the system must also be added to this table. The operating system uses the syscall table to translate a system call number to the actual system call referenced by it.

3) How do you directly call a syscall from user-space?

To directly call a syscall from user-space, one must run the function syscall and provide it as an argument the system call number and arguments that that particular system call needs. For example, the system call created for Part A of the assignment, cmpt300_test, can be called as follows: syscall(_cmpt300_TEST_, 300). **_cmpt300_TEST_** is defined as 327, the system call number which is passed into syscall, and 300 is the argument for the system call.

4) Run your VM with the modified kernel and run your user-space testing code. Attach a screen shot to this report(The part where we see the diving to kernel level and the rising to user level).



```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
cmpt300@cmpt300:~$ ls
backup  cmpt300  example
cmpt300@cmpt300:~$ cd cmpt300
cmpt300@cmpt300:~/cmpt300$ ls
a.out  cmpt300_test  cmpt300_test.c  prinfo  prinfo.c
cmpt300@cmpt300:~/cmpt300$ ./cmpt300_test

Diving to kernel level

 Hello World !--syscall arg 300
Rising to user level

cmpt300@cmpt300:~/cmpt300$ ls
a.out  cmpt300_test  cmpt300_test.c  prinfo  prinfo.c
cmpt300@cmpt300:~/cmpt300$ ./cmpt300_test

Diving to kernel level

 Hello World !--syscall arg 300
Rising to user level

cmpt300@cmpt300:~/cmpt300$ 
```

Part B: Briefly answer these questions(You only need a few sentences each).

1) What is the purpose of a process "nice" value, and how did you find it?

A process "nice" value is used to determine CPU scheduling priority, with a high of 19 and a low of -20. A higher "nice" value means a lower priority and a lower "nice" value means a higher priority. To find the "nice" value, we took the **static_prio** integer found in **task_struct** of the process in **sched.h** and subtracted 120.

2)
a) What data-structure is used in kernel space to traverse between related PIDs information?

A doubly linked list structure was used to traverse between related PIDs information. The struct for each node in the linked list is of a **list_head** type.

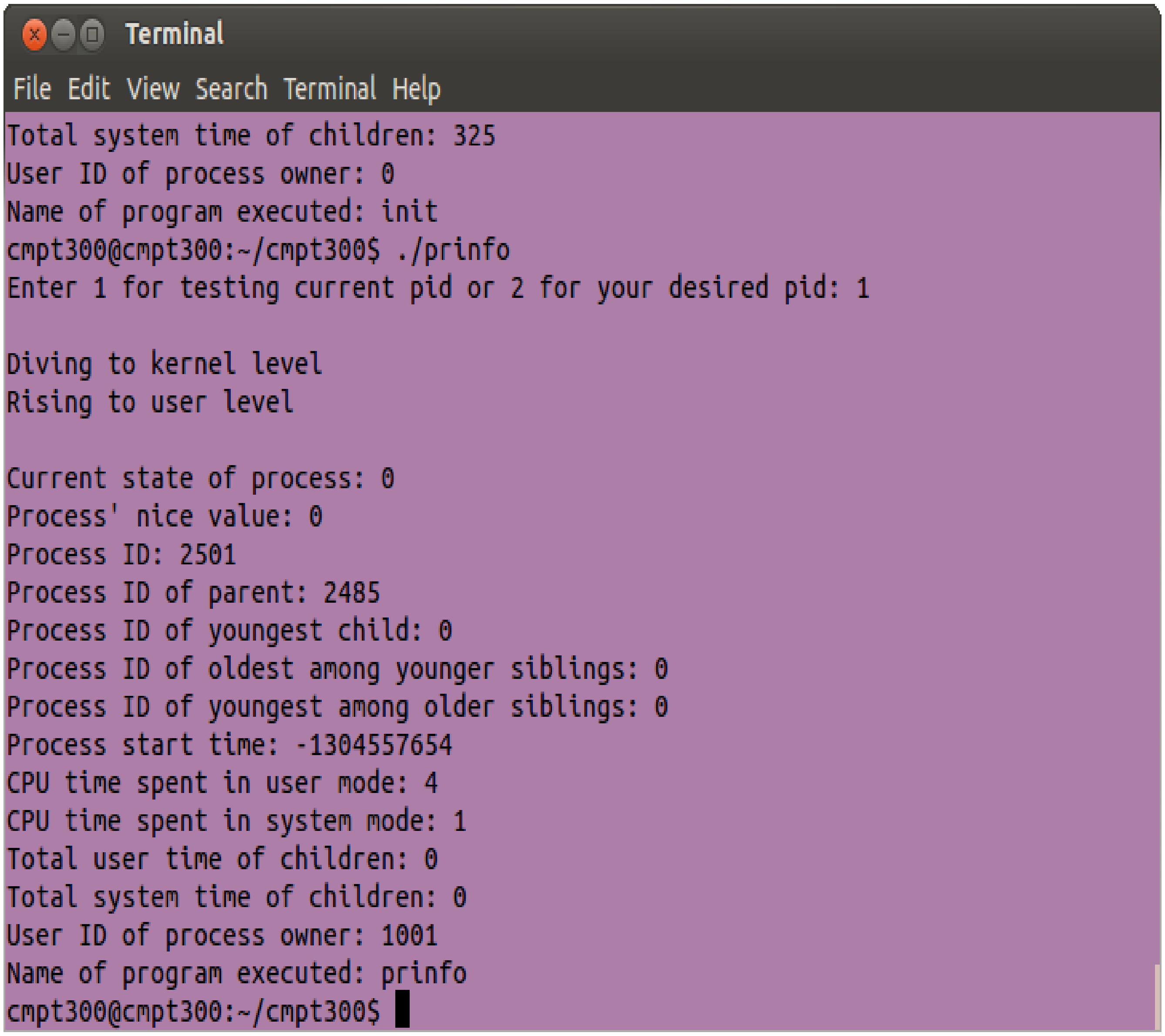
b) How did you traverse the data-structure?

To traverse the data-structure, we called the macro **list_for_each()** found in <include/linux/list.h> which requires 2 arguments to be passed in: a pointer of the data structure used to traverse the list, and a **list_head** struct indicating which list to traverse.

3) How do you pass the target PID from user-space to kernel space?

To pass the target PID from user-space to kernel space, a PID is stored inside the **prinfo** struct where the **prinfo** struct is then passed into kernel space through syscall() as an argument.

4) Attach three screen shots of your user-space testing code being run.
a) One screen shot of the testing coding running with the PID of your program(ex. use getpid()).

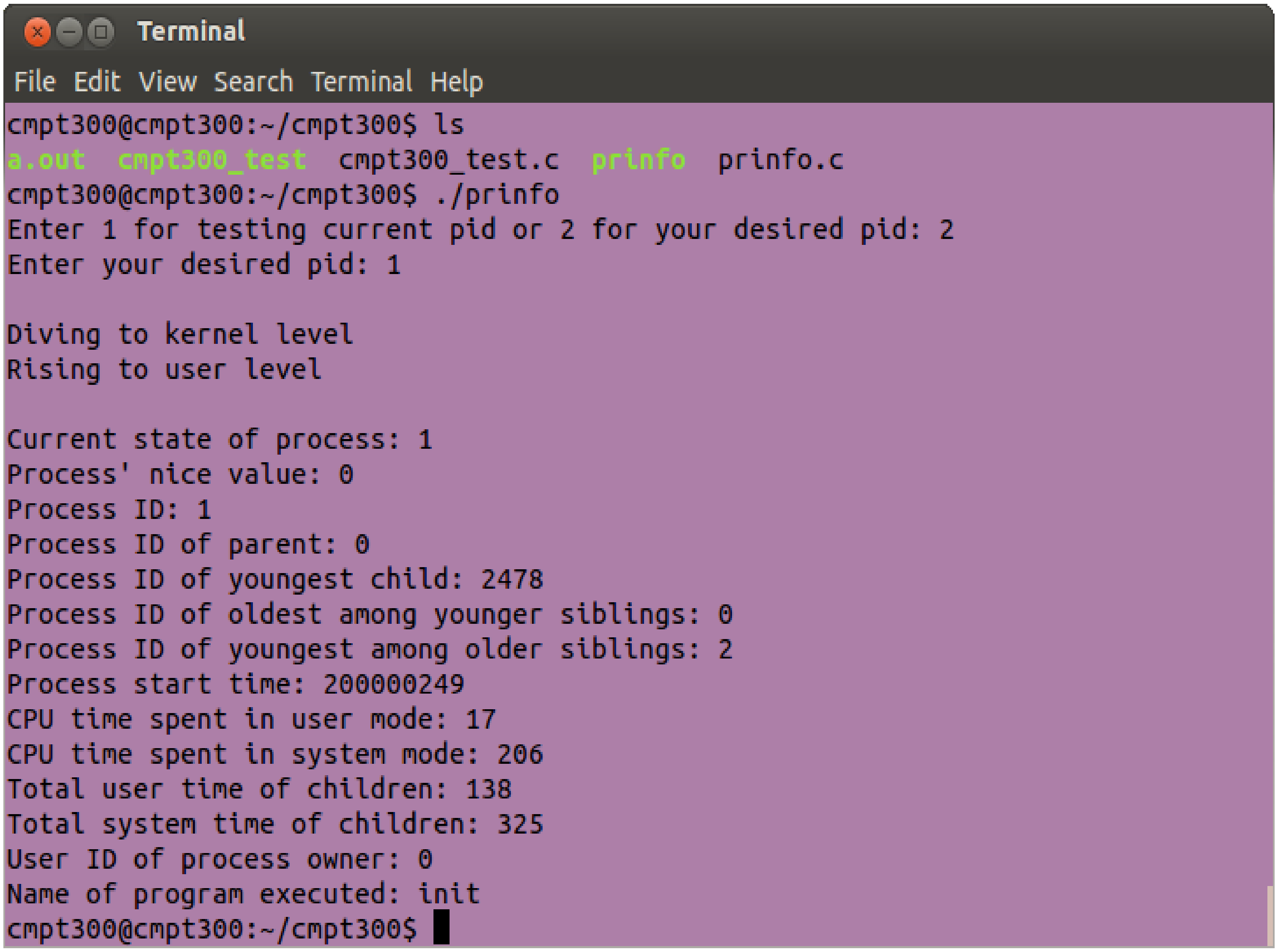


```
Total system time of children: 325
User ID of process owner: 0
Name of program executed: init
cmpt300@cmpt300:~/cmpt300$ ./prinfo
Enter 1 for testing current pid or 2 for your desired pid: 1

Diving to kernel level
Rising to user level

Current state of process: 0
Process' nice value: 0
Process ID: 2501
Process ID of parent: 2485
Process ID of youngest child: 0
Process ID of oldest among younger siblings: 0
Process ID of youngest among older siblings: 0
Process start time: -1304557654
CPU time spent in user mode: 4
CPU time spent in system mode: 1
Total user time of children: 0
Total system time of children: 0
User ID of process owner: 1001
Name of program executed: prinfo
cmpt300@cmpt300:~/cmpt300$
```

b) One screen shot of the testing coding using the PID of init (pid = 1).

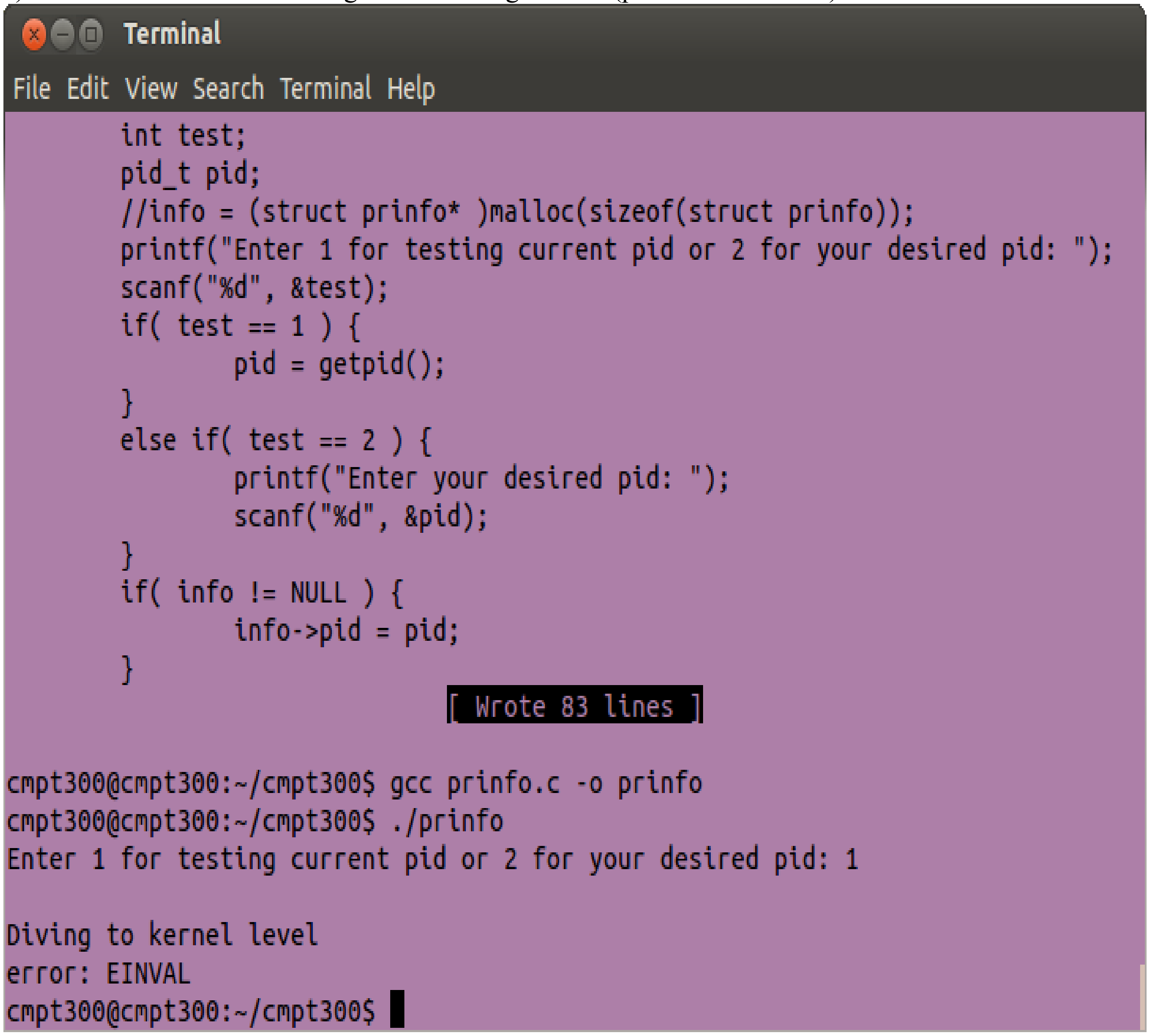


```
cmpt300@cmpt300:~/cmpt300$ ls
a.out  cmpt300_test  cmpt300_test.c  prinfo  prinfo.c
cmpt300@cmpt300:~/cmpt300$ ./prinfo
Enter 1 for testing current pid or 2 for your desired pid: 2
Enter your desired pid: 1

Diving to kernel level
Rising to user level

Current state of process: 1
Process' nice value: 0
Process ID: 1
Process ID of parent: 0
Process ID of youngest child: 2478
Process ID of oldest among younger siblings: 0
Process ID of youngest among older siblings: 2
Process start time: 200000249
CPU time spent in user mode: 17
CPU time spent in system mode: 206
Total user time of children: 138
Total system time of children: 325
User ID of process owner: 0
Name of program executed: init
cmpt300@cmpt300:~/cmpt300$
```

c) One screen shot of the testing code handling an error(prinfo struct = null).



```
        int test;
        pid_t pid;
        //info = (struct prinfo* )malloc(sizeof(struct prinfo));
        printf("Enter 1 for testing current pid or 2 for your desired pid: ");
        scanf("%d", &test);
        if( test == 1 ) {
                pid = getpid();
        }
        else if( test == 2 ) {
                printf("Enter your desired pid: ");
                scanf("%d", &pid);
        }
        if( info != NULL ) {
                info->pid = pid;
        }
                                [ Wrote 83 lines ]

cmpt300@cmpt300:~/cmpt300$ gcc prinfo.c -o prinfo
cmpt300@cmpt300:~/cmpt300$ ./prinfo
Enter 1 for testing current pid or 2 for your desired pid: 1

Diving to kernel level
error: EINVAL
cmpt300@cmpt300:~/cmpt300$ 
```