Objective 1: Implementing System call using write () and read() method.

1. **Using write () system call:**

```
#include<unistd.h>

int main(){
write(1,"utsab singh",15);
}
~
~
~
```

Output:

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# gcc test.c

┌──(root㉿kali)-[/home/kali/Desktop]
└─# ./a.out
utsab singh

┌──(root㉿kali)-[/home/kali/Desktop]
└─#
```

2. **Using read() system call:**

```
#include<unistd.h>

int main(){
char buff[20];
write(1,"what is your name\n",18);
read(0,buff,10);
write(1,"your name is:",13);
write(1,buff,10);

}
~
```

Output:

```
root@kali: /home/kali/Desktop
File  Actions  Edit  View  Help

┌──(root㉿kali)-[/home/kali/Desktop]
└─# gcc read.c

┌──(root㉿kali)-[/home/kali/Desktop]
└─# ./a.out
what is your name
utsab
your name is:utsab

┌──(root㉿kali)-[/home/kali/Desktop]
└─#
```

Objective 2: Process management in Linux.

1. **Using top command for managing Linux processes:**

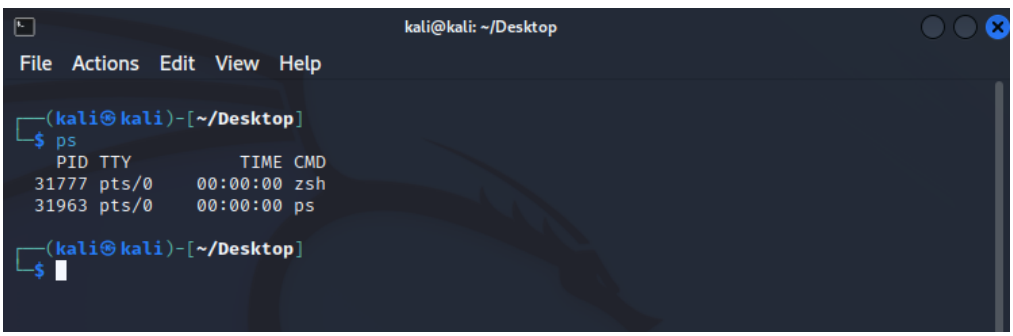
```
root@kali: /home/kali/Desktop
File  Actions  Edit  View  Help
top - 05:39:56 up  1:06,  2 users,  load average: 0.04, 0.07, 0.01
Tasks: 196 total,   1 running, 195 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.7 us,  0.7 sy,  0.0 ni, 98.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
MiB Mem :   1972.9 total,    761.9 free,    741.5 used,    469.5 buff/cache
MiB Swap:   1024.0 total,   1024.0 free,      0.0 used.   1072.3 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+
    682 root      20   0  469928 149296  64264 S   8.3   7.4   2:02.61
   1176 kali      20   0  607096 121736  44184 S   1.3   6.0   0:05.46
   1178 kali      20   0  358600  40160  20188 S   1.0   2.0   0:32.61
   1128 kali      20   0 1226652 106516  77056 S   0.7   5.3   0:24.69
  15034 kali      20   0  465788 105376  86000 S   0.7   5.2   0:06.75
   1180 kali      20   0  358416  30752  21000 S   0.3   1.5   0:16.41
   1293 kali      20   0  290452  41308  30316 S   0.3   2.0   0:18.42
  19234 root      20   0   10184   3600   2988 R   0.3   0.2   0:00.06
      1 root      20   0  167484  12068   8936 S   0.0   0.6   0:02.03
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.02
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
      5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
      8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
     10 root       0 -20       0      0      0 I   0.0   0.0   0:00.00
     11 root      20   0       0      0      0 I   0.0   0.0   0:00.00
     12 root      20   0       0      0      0 I   0.0   0.0   0:00.00
     13 root      20   0       0      0      0 I   0.0   0.0   0:00.00
     14 root      20   0       0      0      0 S   0.0   0.0   0:00.07
```

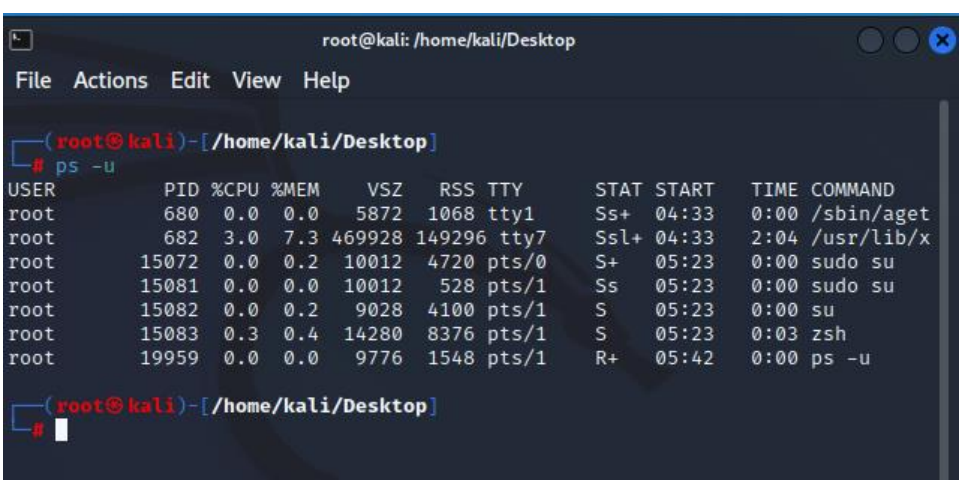
Terminologies used:

PID: Unique Process ID given to each process.
User: Username of the process owner.
PR: Priority given to a process while scheduling.
NI: 'nice' value of a process.
VIRT: Amount of virtual memory used by a process.
RES: Amount of physical memory used by a process.
SHR: Amount of memory shared with other processes.
S: state of the process
'**D**' = uninterruptible sleep
'**R**' = running
'**S**' = sleeping
'**T**' = traced or stopped
'**Z**' = zombie
%CPU: Percentage of CPU used by the process.
%MEM; Percentage of RAM used by the process.
TIME+: Total CPU time consumed by the process.
Command: Command used to activate the process.

2. Using the 'ps' command:



To get more info we use 'ps -u':



3. Stop a process using Kill command:

```
kali@kali: ~
File  Actions  Edit  View  Help

┌──(kali㉿kali)-[~]
└─$ ps -u
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
kali       22033  0.4  0.3  13484  7176 pts/0    Ss   05:47   0:00 /usr/bin/z
kali       23424  0.0  0.0   9776  1568 pts/0    R+   05:50   0:00 ps -u

┌──(kali㉿kali)-[~]
└─$ kill 22033
```

The syntax for killing a process is:
$ kill [pid]
Copy
Alternatively you can also use :
$ kill -9 [pid]
Copy

4. Changing the priority of a process:

In Linux, you can prioritize between processes. The priority value for a process is called the 'Niceness' value.
Niceness value can range from -20 to 19. 0 is the default value.
The fourth column in the output of top command is the column for niceness value.

```
File  Actions  Edit  View  H
top - 05:53:38 up  1:20,
Tasks: 201 total,   1 run
%Cpu(s):  0.8 us,  1.3 sy
MiB Mem :   1972.9 total,
MiB Swap:   1024.0 total,

    PID USER      PR  NI
  21479 root      20   0
    468 root      20   0
  21682 kali      20   0
     15 root      20   0
   8307 root      20   0
  21651 kali      20   0
  21675 kali      20   0
  21684 kali      20   0
  21685 kali      20   0
```

To start a process and give it a nice value other than the default one, use:

$ nice -n [value] [process name]

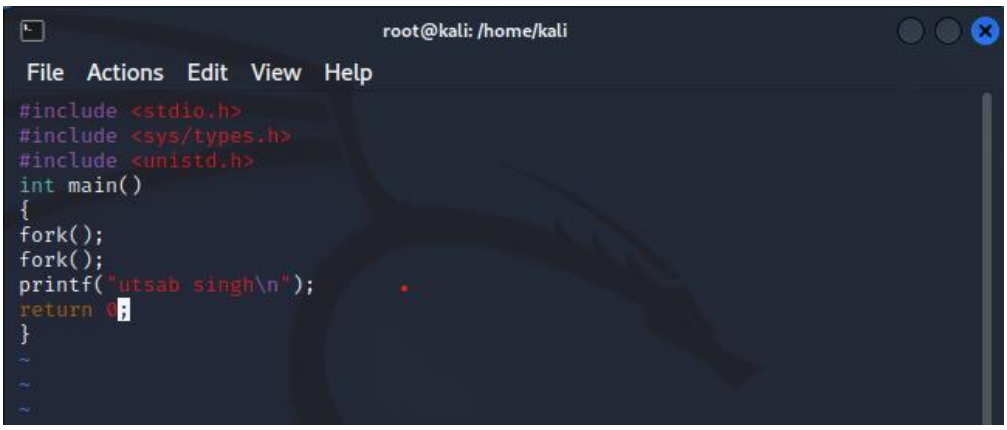
Copy

To change nice value of a process that is already running use:

renice [value] -p 'PID'

Objective 3: Create a process using fork() method:

Program:



Output:

Fork command works on basis of 2^n. Since there are 2 fork() in program, the output is printed 4 times.

Objective 4: Thread simulation using inbuild function:

Program 1: C program to demonstrate use of pthread basic functions

```
root@kali: /home/kali/Desktop
File  Actions  Edit  View  Help
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

void *myThreadFun(void *vargp)
{
sleep(1);
printf("Printing GeeksQuiz from Thread \n");
return NULL;
}
int main()
{
pthread_t thread_id;
printf("Before Thread\n");
pthread_create(&thread_id, NULL, myThreadFun, NULL);
pthread_join(thread_id, NULL);
printf("After Thread\n");
exit(0);
}
~
```

Output:

```
root@kali: /home/kali/Desktop
File  Actions  Edit  View  Help
┌──(root㉿kali)-[/home/kali/Desktop]
└─# gcc thread.c -lpthread

┌──(root㉿kali)-[/home/kali/Desktop]
└─# ./a.out
Before Thread
Printing GeeksQuiz from Thread
After Thread

┌──(root㉿kali)-[/home/kali/Desktop]
└─#
```

Program 2: C program to show multiple threads with global and static variables.

```
root@kali: /home/kali/Desktop
File  Actions  Edit  View  Help
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
// Let us create a global variable to change it in threads
int g = 0;
// The function to be executed by all threads
void *myThreadFun(void *vargp)
{
// Store the value argument passed to this thread
int *myid = (int *)vargp;
// Let us create a static variable to observe its changes
static int s = 0;
// Change static and global variables
++s; ++g;
// Print the argument, static and global variables
printf("Thread ID: %d, Static: %d, Global: %d\n", *myid, ++s, ++g);
}
int main()
{
int i;
pthread_t tid;
// Let us create three threads
for (i = 0; i < 3; i++)
pthread_create(&tid, NULL, myThreadFun, (void *)&tid);
pthread_exit(NULL);
return 0;
}
~
```

Output:

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# gcc multithreading.c -lpthread

┌──(root㉿kali)-[/home/kali/Desktop]
└─# ./a.out
Thread ID: 905422528, Static: 2, Global: 2
Thread ID: 905422528, Static: 4, Global: 4
Thread ID: 905422528, Static: 6, Global: 6

┌──(root㉿kali)-[/home/kali/Desktop]
└─#
```

Conclusion:

Hence, using kali Linux, the objectives of the lab were successfully completed.