BAHIRDAR UNIVERSITY

Faculty of computing

Department of software engineering

Operating systems and systems programming

System calls

prtcl

PREPARED BY KIRUBEL MAMUYE
ID NO:-1309016
SUBMITTED TO :LEC. WENDMU

Table of contents

INTRODUCTION

This assignment is a brief description of a system call known as prtcl which has 5 parameters and flags total in general i.e., int prctl (int option, unsigned long arg2, unsigned long arg3, unsigned long arg4, unsigned long arg5) so in this assignment I am going to describe the what , why and how this system call works and then what is the meaning and functions of each parameters and finally develop a source code to implement the prtcl system call so first of all let's talk about what a system call is:-(commonly abbreviated to **syscall**) is the programmatic way in which a computer program requests a service from the kernel of the operating system on which it is executed. This may include hardware-related services (for example, accessing a hard disk drive or accessing the device's camera), creation and execution of new processes, and communication with integral kernel services such as process scheduling. System calls provide an essential interface between a process and the operating system. and then what prtcl is in general **prctl**() manipulates various aspects of the behaviour of the calling thread or process.

Note that careless use of some **prctl** () operations can confuse the user-space run-time environment, so these operations should be used with care.

prctl() is called with a first argument describing what to do(with values defined in *<linux/prctl.h>*), and further arguments with a significance depending on the first one.

1.1 what ,why and how prtcl?

In this section I am going to describe what prtcl is and then why it is used and then how we use prtcl in a general sense

What is prtcl?

first of all prtcl is a system call **prctl**() manipulates various aspects of the behaviour of the calling thread or process

Note that careless use of some **prctl**() operations can confuse the user-space run-time environment, so these operations should be used with care.

prctl() is called with a first argument describing what to do(with values defined in *<linux/prctl.h>*), and further arguments with a significance depending on the first one. And also The **prctl**() system call was introduced in Linux 2.1.57 where This call is Linux-specific and IRIX has a **prctl**() system call (also introduced in Linux 2.1.44 as irix_prctl on the MIPS architecture), with prototype.

Why prtcl?

In this section I am going to describe the question why prtcl is used in short so as we know prtcl is used for programming AS mentioned in the above **prctl**() manipulates various aspects of the behaviour of the calling thread or process. So the rest of the applications of using prtcl system call is clearly described under using the arguments parameters and the the rest.

How partcl?

- ✓ **int prctl(int** *option***, unsigned long** *arg2***, unsigned long** *arg3***, unsigned long** *arg4***, unsigned long** *arg5***);**

as shown in the above prctl has 5 parameters and lots of commands to be implemented in this section I am going to show you how to use prtcl system call. So here how the success and error identified

On success, PR_GET_DUMPABLE, PR_GET_KEEPCAPS, PR_GET_NO_NEW_PRIVS, PR_GET_THP_DISABLE, PR_CAPBSET_READ, PR_GET_TIMING, PR_GET_TIMERSLACK, PR_GET_SECUREBITS, PR_MCE_KILL_GET, PR_CAP_AMBIENT+PR_CAP_AMBIENT_IS_SET, and (if it returns) PR_GET_SECCOMP return the nonnegative values described above. All other option values return 0 on success. On error, -1 is returned, and errno is set appropriately.

So below the option and errors are clearly put so you have to implement

1.2 Description of parameters and flags

The first parameter of the "prctl" system call defines what has to be done with the initialized values in header. All the other arguments or parameters would be used as per the first argument and its worth. here are a set of options we use in prtcl:- and options to get the maximum number of processes per user, get the maximum number of processors the calling process can use, find out whether a specified process is currently blocked, get orset the maximum stack size, and so on.

✓ **int prctl(int *option*, unsigned long *arg2*, unsigned long *arg3*, unsigned long *arg4*, unsigned long *arg5*);**

✓ *option*

- PR_CAP_AMBIENT – read/change ambient capability of calling thread referencing value in arg2, in regards to:
    - PR_CAP_AMBIENT_RAISE – capability in arg3 is added to ambient set
    - PR_CAP_AMBIENT_LOWER – capability in arg3 is removed from ambient set
    - PR_CAP_AMBIENT_IS_SET – returns 1 if capability in arg3 is in the ambient set, 0 if not
    - PR_CAP_AMBIENT_CLEAR_ALL – remove all capabilities from ambient set, set arg3 to 0
- PR_CAPBSET_READ – return 1 if capability specified in arg2 is in calling thread's capability bounding set, 0 if not
- PR_CAPBSET_DROP – if calling thread has CAP_SETPCAP capability in user namespace, drop capability in arg2 from capability bounding set for calling process
- PR_SET_CHILD_SUBREAPER – if arg2 is not zero, set "child subreaper" attribute for calling process, if arg2 is zero, unset
- PR_GET_CHILD_SUBREAPER – return "child subreaper" setting of calling process in location pointed to by arg2
- PR_SET_DUMPABLE – set state of dumpable flag via arg2
- PR_GET_DUMPABLE – return current dumpable flag for calling process
- PR_SET_ENDIAN – set endian-ness of calling process to arg2 via PR_ENDIAN_BIG, PR_ENDIAN_LITTLE, or PR_ENDIAN_PPC_LITTLE
- PR_GET_ENDIAN – return endian-ness of calling process to location pointed by arg2
- PR_SET_KEEPCAPS – set state of calling process's "keep capabilities" flag via arg2
- PR_GET_KEEPCAPS – return current state of calling process's "keep capabilities" flag
- PR_MCE_KILL – set machine check memory corruption kill policy for calling process via arg2
- PR_MCE_KILL_GET – return current per-process machine check kill policy
- PR_SET_MM – modify kernel memory map descriptor fields of calling process, where arg2 is one of the following options and arg3 is the new value to set
    - PR_SET_MM_START_CODE – set address above which program text can run

    - PR_SET_MM_END_CODE – set address below which program text can run
    - PR_SET_MM_START_DATA – set address above which initialized and uninitialized data are placed
    - PR_SET_MM_END_DATA – set address below which initialized and uninitialized data are placed
    - PR_SET_MM_START_STACK – set start address of stack
    - PR_SET_MM_START_BRK – set address above which program heap can be expanded with brk
    - PR_SET_MM_BRK – set current brk value
    - PR_SET_MM_ARG_START – set address above which command line is placed
    - PR_SET_MM_ARG_END – set address below which command line is placed
    - PR_SET_MM_ENV_START – set address above which environment is placed
    - PR_SET_MM_ENV_END – set address below which environment is placed
    - PR_SET_MM_AUXV – set new aux vector, with arg3 providing new address and arg4 containing size of vector
    - PR_SET_MM_EXE_FILE – Supersede /proc/pid/exe symlink with a new one pointing to file descriptor in arg3
    - PR_SET_MM_MAP – provide one-shot access to all addresses by passing struct prctl_mm_map pointer in arg3 with size in arg4
    - PR_SET_MM_MAP_SIZE – returns size of prctl_mm_map structure, where arg4 is pointer to unsigned int
- PR_MPX_ENABLE_MANAGEMENT – enable kernel management of memory protection extensions
- PR_MPX_DISABLE_MANAGEMENT – disable kernel management of memory protection extensions
- PR_SET_NAME – set name of calling process to null-terminated string pointed to by arg2
- PR_GET_NAME – get name of calling process in null-terminated string into buffer sized to 16 bytes referenced by pointer in arg2
- PR_SET_NO_NEW_PRIVS – set calling process no_new_privs attribute to value in arg2
- PR_GET_NO_NEW_PRIVS – return value of no_new_privs for calling process
- PR_SET_PDEATHSIG – set parent-death signal of calling process to arg2
- PR_GET_PDEATHSIG – return value of parent-death signal into arg2
- PR_SET_SECCOMP – set "seccomp" mode for calling process via arg2
- PR_GET_SECCOMP – get "seccomp" mode of calling process
- PR_SET_SECUREBITS – set "securebits" flags of calling thread to value in arg2
- PR_GET_SECUREBITS – return "securebits" flags of calling process
- PR_GET_SPECULATION_CTRL – return state of speculation misfeature specified in arg2

- PR_SET_SPECULATION_CTRL – set state of speculation misfeature specified in arg2
- PR_SET_THP_DISABLE – set state of "THP disable" flag for calling process
- PR_TASK_PERF_EVENTS_DISABLE – disable all performance counters for calling process
- PR_TASK_PERF_EVENTS_ENABLE – enable performance counters for calling process
- PR_GET_THP_DISABLE – return current setting of "THP disable" flag
- PR_GET_TID_ADDRESS – return clear_child_tid address set by set_tid_address
- PR_SET_TIMERSLACK – sets current timer slack value for calling process
- PR_GET_TIMERSLACK – return current timer slack value for calling process
- PR_SET_TIMING – set statistical process timing or accurate timestamp-based process timing by flag in arg2 (PR_TIMING_STATISTICAL or PR_TIMING_TIMESTAMP)
- PR_GET_TIMING – return process timing method in use
- PR_SET_TSC – set state of flag determining if timestamp counter can be read by process in arg2 (PR_TSC_ENABLE or PR_TSC_SIGSEGV)
- PR_GET_TSC – return state of flag determining whether timestamp counter can be read in location pointed by arg2

Returns zero on success or value specified in option flag.

Errors

- ✓ ***EACCES*** *option is* ***PR_SET_SECCOMP*** *and arg2 is* ***SECCOMP_MODE_FILTER****, but the process does not have the* ***CAP_SYS_ADMIN*** *capability or has not set the no_new_privs attribute (see the discussion of* ***PR_SET_NO_NEW_PRIVS*** *above).*
- ✓ ***EACCES*** *option is* ***PR_SET_MM****, and arg3 is* ***PR_SET_MM_EXE_FILE****, the file is not executable.*
- ✓ ***EBADF*** *option is* ***PR_SET_MM****, arg3 is* ***PR_SET_MM_EXE_FILE****, and the file descriptor passed in arg4 is not valid.*
- ✓ ***EBUSY*** *option is* ***PR_SET_MM****, arg3 is* ***PR_SET_MM_EXE_FILE****, and this the second attempt to change the /proc/pid/exe symbolic link, which is prohibited.*
- ✓ ***EFAULT*** *arg2 is an invalid address.*
- ✓ ***EFAULT*** *option is* ***PR_SET_SECCOMP****, arg2 is* ***SECCOMP_MODE_FILTER****, the system was built with* ***CONFIG_SECCOMP_FILTER****, and arg3 is an invalid address.*
- ✓ ***EINVAL*** *The value of option is not recognized.*
- ✓ ***EINVAL*** *option is* ***PR_MCE_KILL*** *or* ***PR_MCE_KILL_GET*** *or* ***PR_SET_MM****, and unused* ***prctl()*** *arguments were not specified as zero.*
- ✓ ***EINVAL*** *arg2 is not valid value for this option.*
- ✓ ***EINVAL*** *option is* ***PR_SET_SECCOMP*** *or* ***PR_GET_SECCOMP****, and the kernel was not configured with* ***CONFIG_SECCOMP****.*
- ✓ ***EINVAL*** *option is* ***PR_SET_SECCOMP****, arg2 is* ***SECCOMP_MODE_FILTER****, and the kernel was not configured with* ***CONFIG_SECCOMP_FILTER****.*
- ✓ ***EINVAL*** *option is* ***PR_SET_MM****, and one of the following is true*

** arg4 or arg5 is nonzero;*

** arg3 is greater than* ***TASK_SIZE*** *(the limit on the size of the user address space for this architecture);*

** arg is **PR_SET_MM_START_CODE**, **PR_SET_MM_END_CODE**, **PR_SET_MM_START_DATA**,**PR_SET_MM_END_DATA** or **PR_SET_MM_START_STACK**, and the permissions of the corresponding memory area are not as required;*

** arg2 is **PR_SET_MM_START_BRK** or **PR_SET_MM_BRK**, and arg3 is less than or equal to the end of the data segment or specifies a value that would cause the **RLIMIT_DATA** resource limit to be exceeded.*

- ❖ ***EINVAL** option is **PR_SET_PTRACER** and arg2 is not 0, **PR_SET_PTRACER_ANY**, or the PID of an existing process.*
- ❖ ***EINVAL** option is **PR_SET_PDEATHSIG** and arg2 is not a valid signal number.*

- ✓ ***EINVAL** option is **PR_SET_DUMPABLE** and arg2 is neither **SUID_DUMP_DISABLE** nor **SUID_DUMP_USER**.*
- ✓ ***EINVAL** option is **PR_SET_TIMING** and arg2 is not **PR_TIMING_STATISTICAL**.*
- ✓ ***EINVAL** option is **PR_SET_NO_NEW_PRIVS** and arg2 is not equal to 1 or arg3, arg4, or arg5 is nonzero.*
- ✓ ***EINVAL** option is **PR_GET_NO_NEW_PRIVS** and arg2, arg3, arg4, or arg5 is nonzero.*
- ✓ ***EINVAL** option is **PR_SET_THP_DISABLE** and arg3, arg4, or arg5 is nonzero.*
- ✓ ***EINVAL** option is **PR_GET_THP_DISABLE** and arg2, arg3, arg4, or arg5 is nonzero.*

***EINVAL** option is **PR_CAP_AMBIENT** and an unused argument (arg4, arg5, or, in the case of **PR_CAP_AMBIENT_CLEAR_ALL**, arg3) is nonzero; or arg2 has an invalid value; or arg2 Is **PR_CAP_AMBIENT_LOWER**, **PR_CAP_AMBIENT_RAISE**,or and arg3 does not specify a valid capability.*

- ❖ ***ENXIO** option was **PR_MPX_ENABLE_MANAGEMENT** and the kernel or the CPU does not support MPX management. Check that the kernel and processor have MPX support.*
- ❖ ***EOPNOTSUPP** option is **PR_SET_FP_MODE** and arg2 has an invalid or unsupported value.*
- ❖ ***EPERM** option is **PR_SET_SECUREBITS**, and the caller does not have the **CAP_SETPCAP** capability, or tried to unset a "locked" flag, or tried to set a flag whose corresponding locked flag was set (see **capabilities**(7)).*
- ❖ ***EPERM** option is **PR_SET_KEEPCAPS**, and caller's **SECBIT_KEEP_CAPS_LOCKED** flag is set (see **capabilities**(7)).*
- ❖ ***EPERM** option is **PR_CAPBSET_DROP**, and the caller does not have the **CAP_SETPCAP** capability.*
- ❖ ***EPERM** option is **PR_SET_MM**, and the caller does not have the **CAP_SYS_RESOURCE** capability.*
- ❖ ***EPERM** option is **PR_CAP_AMBIENT** and arg2 is **PR_CAP_AMBIENT_RAISE**, but either the capability specified in arg3 is not present in the process's permitted and inheritable capability sets, or the **PR_CAP_AMBIENT_LOWER** securebit has been set.*

1.3 list the flags and their use with code implementation

The prctl system call has been used in the C language to manipulate diverse characteristics of the calling function or process activities. The first parameter of the "prctl" system call defines what has to be done with the initialised values in header. All the other arguments or parameters would be used as per the first argument and its worth. Let's take a deep glance at the "prctl" system call in C while we have been working on the Ubuntu 20.04 at the time of implementing this article.

Example 1:- Open and log in from Ubuntu 20.04 and launch the application named "terminal" from the activity area. This could be done by utilizing a simple key shortcut "Ctrl+Alt+T" on your desktop. Create a C-type file to implement the prctl() system call, perform the command shown in the snap underneath.

```
$ touch prtcl.c
```


✓ After creation, let's open the file with a GNU Nano editor as per the shown instruction.

```
$ nano prtcl.c
```

Add the code shown in the snap image underneath within the GNU file. The code contains necessary header files for the working of a prctl() code. Then we have created and defined 4 threads named process1, process2, process3, and process4. All 4 processes or functions contain the void as a general or signature parameter, but it could be something else. As we have elaborated before, the first parameter of the "prctl()" system call will show what we have to do with the calling function. So, We have called prctl() in all 4 methods to set the name of a process by using the "PR_SET_NAME" argument. After the 2 second sleep, the puts function will be executed to set the name of a process.

```
 GNU nano 4.8                                    prctl.c
#include <stdio.h>
#include <stdlib.h>
#include <sys/prctl.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
void process1(void) { // Void is signature parameter, can be of any other type
  prctl(PR_SET_NAME,"process1");
  sleep(2);
  puts("process1");
  while(1){};                          }
void process2(void) {
  prctl(PR_SET_NAME,"process2");
  sleep(2);
  puts("process2");
  while(1){};                           }
void process3(void) {
  prctl(PR_SET_NAME,"process3");
  sleep(2);
  puts("process3");
  while(1){};                           }
void process4(void) {
  prctl(PR_SET_NAME,"process4");
  sleep(2);
  puts("process4");
  while(1){};                            }
```

Then we have declared an array type pointer named “fp” and its elements contain the names of 4 methods or processes. The main method declared a variable “id” here indicates processes. The “for” loop has been used here to create a child process for every parent process using the “fork()” method and save that to variable “int”. The “if” statement has been used to check if the “id” is 0. If the condition meets, it will print the child process number, and the “fp” array will be used as a method to fetch the first element, process 1, and so on until the loop ends. The calling of methods in this way would make it execute all the methods defined above.

```
void (*fp[4])(void) = { process1,process2,process3,process4 };

main ( int argc, char *argv[] )
{
  int id; //indicates process id
  for(int i=1; i<=4; i++)
  {
   id = fork(); // fork is used to create child process of parent process
   if (id == 0)
   {
     printf ("Child Process = %d\n",i);
     fp[i]();
     exit(1);
   }
  }
  return EXIT_SUCCESS;
}
```

vo

Compile the file first.

```
$ gcc prctl.c
```

```
kalsoom@virtualbox:~$ gcc prctl.c
```

The execution of the file shows the below output. The name has been set for each process.

```
$ ./a.out
```

```
kalsoom@virtualbox:~$ ./a.out
Child Process = 1
Child Process = 2
kalsoom@virtualbox:~$ Child Process = 3
Child Process = 4
process2
process3
process4
```

Example 2: Let's have another illustration of prctl. Let's open the prctl.c file.

```
$ nano prctl.c
```

```
kalsoom@virtualbox:~$ nano prctl.c
```

After the headers have been included, the method "cap_1" has been initialized. The file descriptor "f" has been defined, and a variable "res" has been initialized with a value "-1". Now the file descriptor will be used to get the maximum capability from the kernel. The file descriptor will open the file as read-only from the kernel folder. If the file descriptor gets more than 0 characters, the "buf" array will be defined with size 32. Two integers have been defined, and the read method has been used to get the data from the buffer using file descriptor and saved to the variable "num". If the variable "num" value is greater than 0, the index-matched value of variable "num" will be initialized as Null. The "sscanf" method will bind the "res" pointer with the "buf" array and store it within variable "r". That's how maximum capability could be got from the kernel. If the value for variable "r" does not equal 1, it will update the value of "res" with "-1" again. In the end, the descript has been closed.

```
 GNU nano 4.8                          prctl.c                          Modifi
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/prctl.h>
int cap_1(void) {
    int f;
    int res = -1;
    // trying to get the max capability over the kernel interface
    f = open("/proc/sys/kernel/cap_last_cap", O_RDONLY);
    if (f >= 0) {
        char buf[32];
        int num, r;

        if ((num=read(f, buf, 31)) >= 0) {
            buf[num] = '\0';
            r = sscanf(buf, "%d", &res);
            if (r != 1)
                res = -1;       }
        close(f);
    }
    return res;  }
```

The second method, “cap_2” has been used to initialize the capability variable equals 0. The prctl() method use “PR_CAPBSET_READ” to read the maximum capability. If the capability’s value is greater than 0, it will be incremented. When the capability gets to 0, it will stop incrementing and return the “cp” value with a decrement of 1.

```
 GNU nano 4.8                          prctl.c                          Modified
int cap_2(void) {
    // try to get maximum capability by trying get the status of
    // each capability indeviually from the kernel.
    int cp = 0;
    while (prctl(PR_CAPBSET_READ, cp) >= 0)
        cp++;
    return cp - 1;
}
```

The main method is getting the capability from the “cap_1” and cap_2 and print it upon the condition is met.

```
int main(int argc, char **argv) {
    int cp;

    cp = cap_1();
    if (cp != -1)
        printf(" The maximum capability for [cap_1] is : %d\n", cp);
    else
        printf("The [cap_1] couldn't get max capability over kernel interface\>

    cp = cap_2();
    printf("The maximum capability for [cap_2] is : %d\n", cp);

    exit(0);
}
```

The compilation and running of this file show that the maximum capacity value is 40.

```
$ gcc prctl.c

$ ./a.out
```

```
kalsoom@virtualbox:~$ gcc prctl.c
kalsoom@virtualbox:~$ ./a.out
 The maximum capability for [cap_1] is : 40
The maximum capability for [cap_2] is : 40
```

Conclusion: In this guide, we have discussed two examples to elaborate on the prctl() system call in C. It will help you a lot as we have demonstrated it with two different arguments.

"" **Function prototype** ""

```
#include <sys/prctl.h>



int prctl(int option, unsigned long arg2, unsigned long arg3, unsigned long arg4, unsigned long ar
g5);
```

2 "" **function** ""

```
prctl(PR_SET_NAME, “process_name”)
```

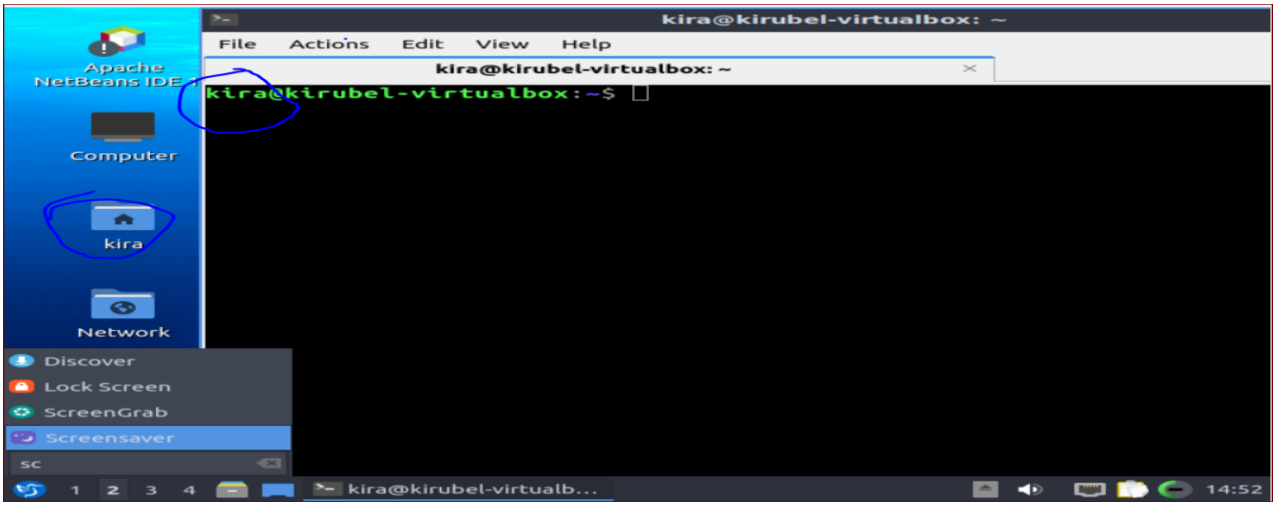
The first parameter is the operation type specified PR_SET_NAME, i.e., the process name is provided a second process parameter is the name string, a length up to 16 bytes of the OK, it is very simple!

3 "" Examples ""

```
void setPthreadName(char *name)

{

        if(name != NULL)

        {

                (void)prct(15, (unsigned long)name);//lname up to 16 characters

        }

}
```

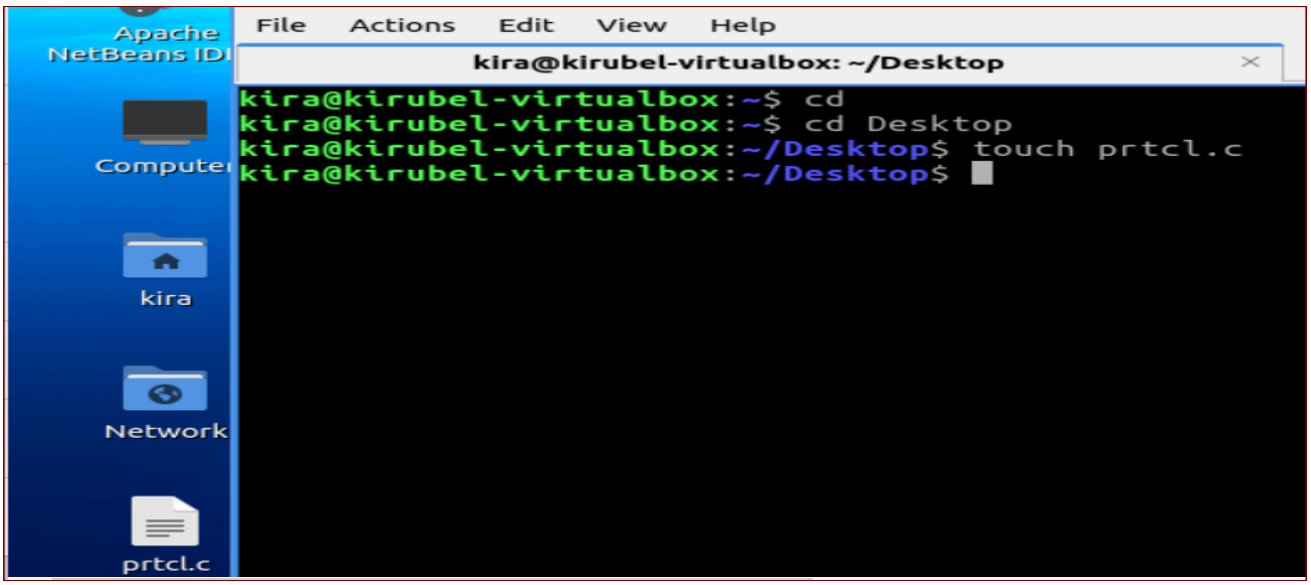
Implementation of the code on my own Linux system i.e., LUBUNTU

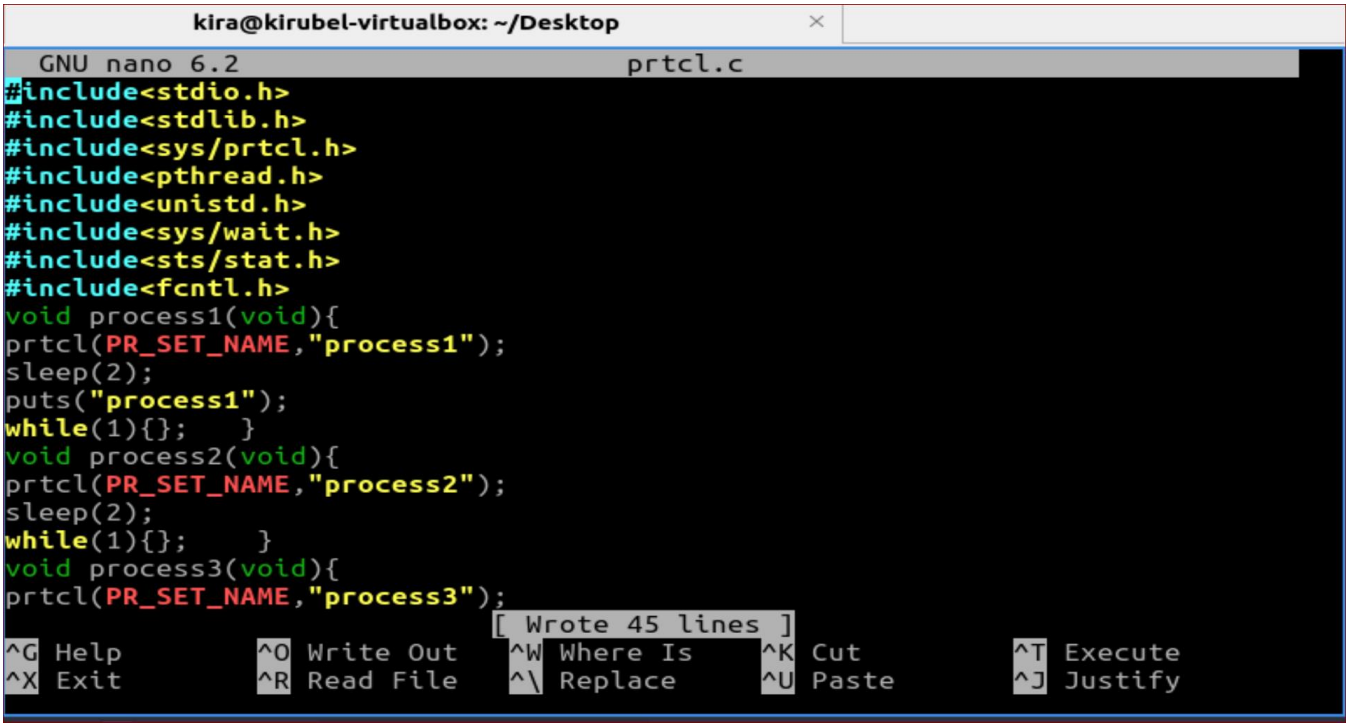
Below I pur the implementation of my prtcl system call on my own Linux system from the previous assignment which is created using user name kira and the above sample examples are going to be implemented on Linux system as all. As shown below the linux system is lununtu and created using by my own name kira/kirubel in oracle vm virtual box and now I am going to implement the system calls in this linux terminal.



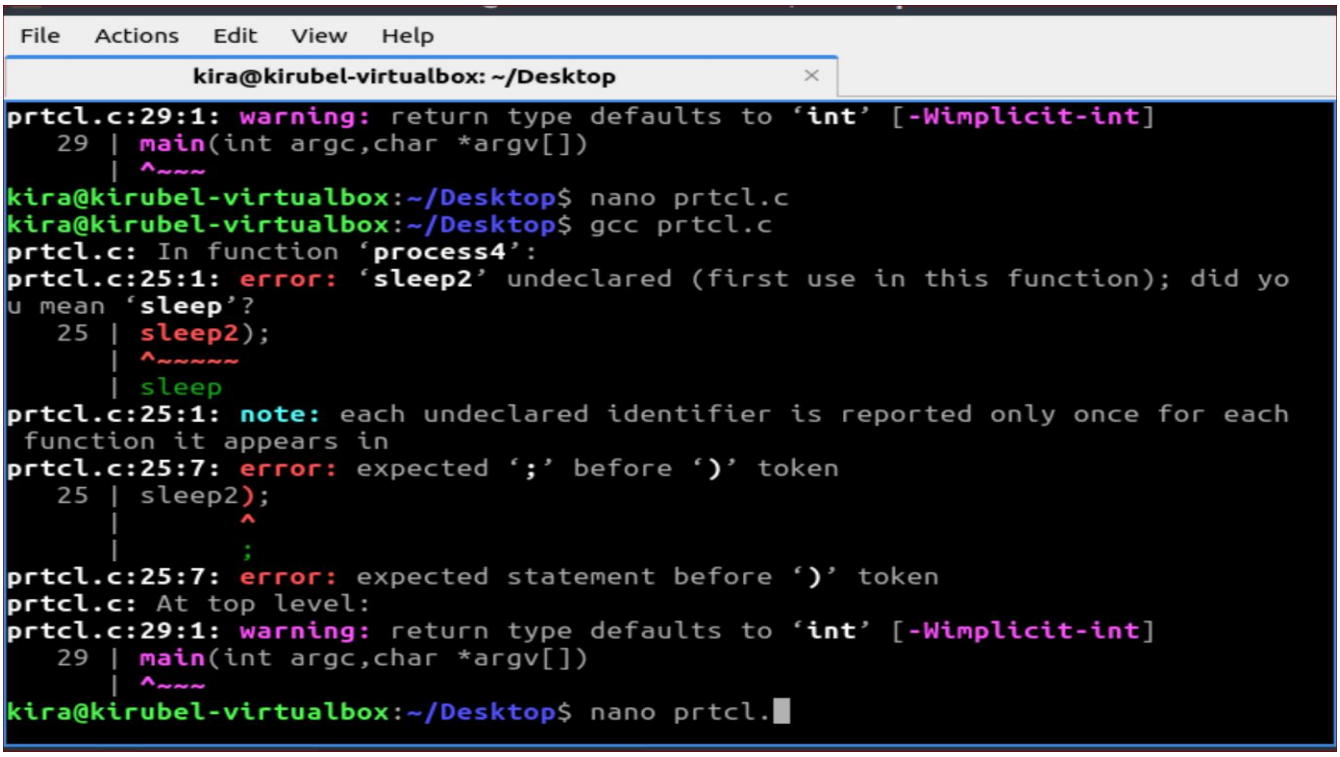
Example 1: implementation using PR_SET_NAME. to set a name for different process in this case 4 processes as mentioned in the above PR_SET_NAME – set name of calling process to null-terminated string pointed to by arg2

Step 1: creating prtcl.c file on desktop

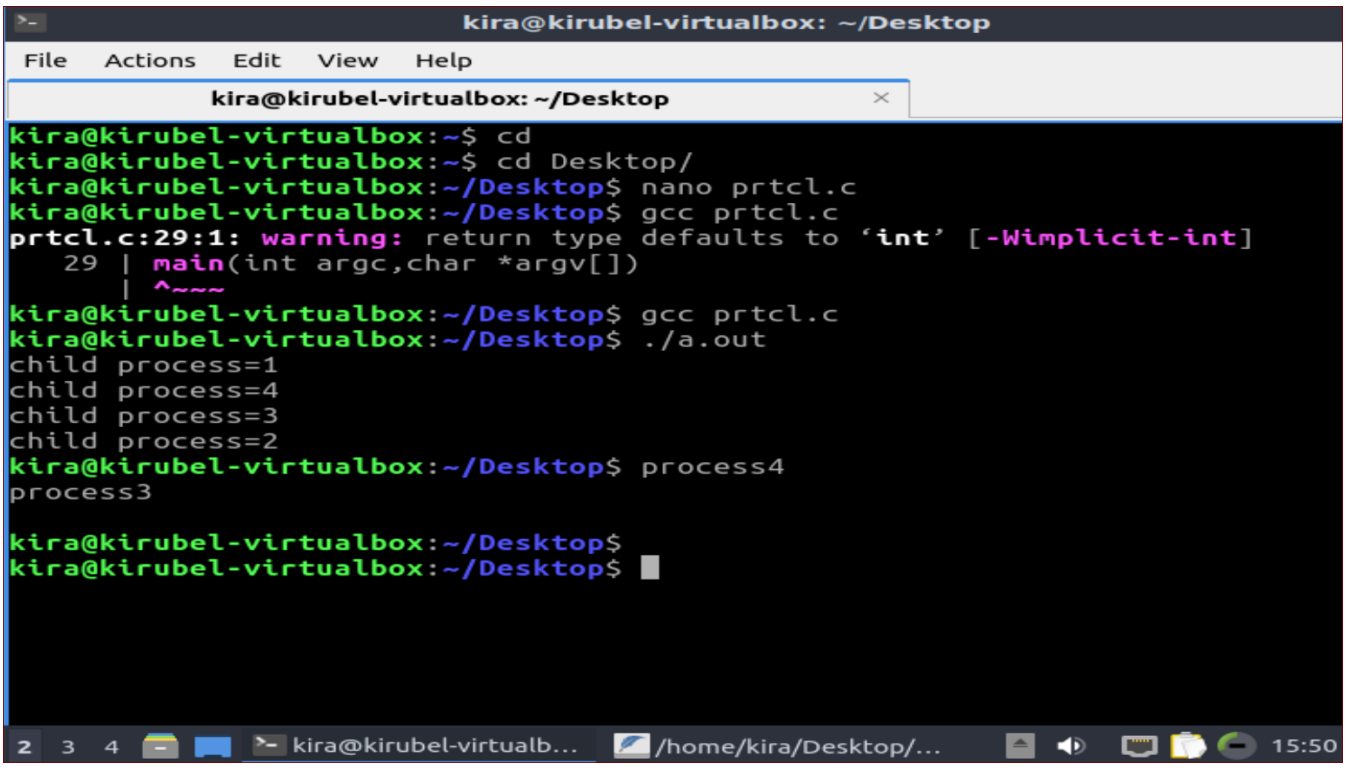




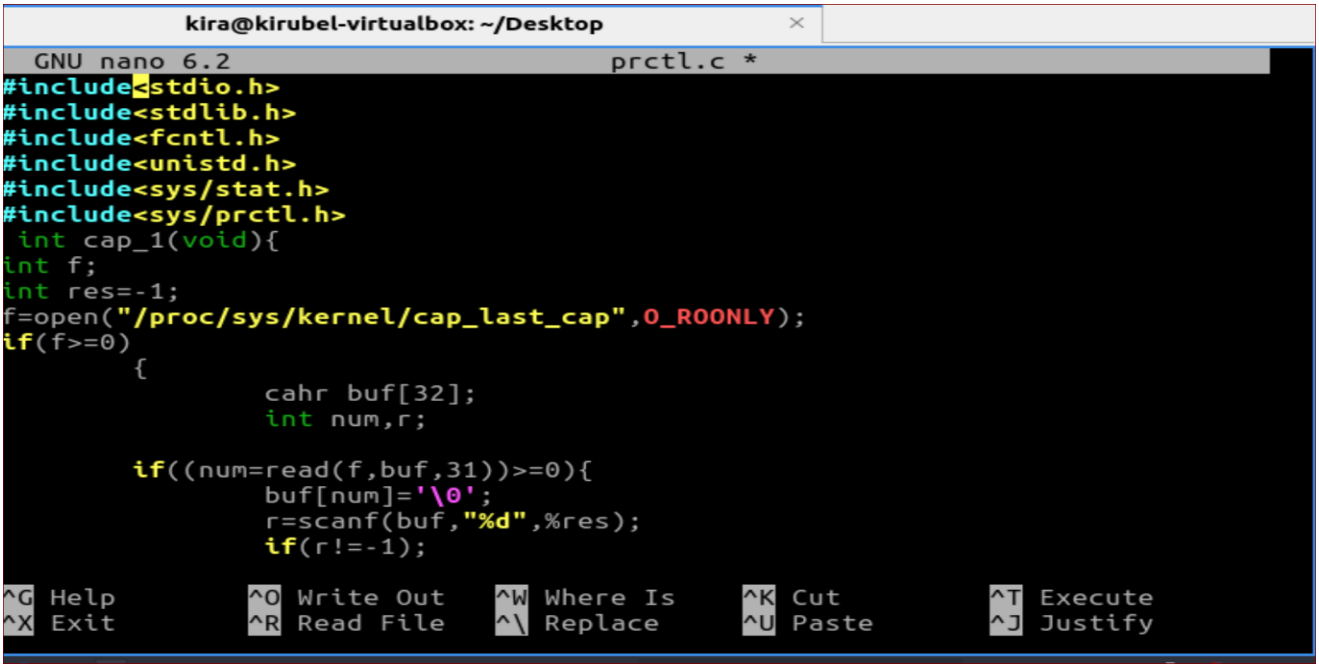
I faced a lot of of errors at first but within time I debug out them using the gcc compiler guide on lubuntu terminal as shown below.

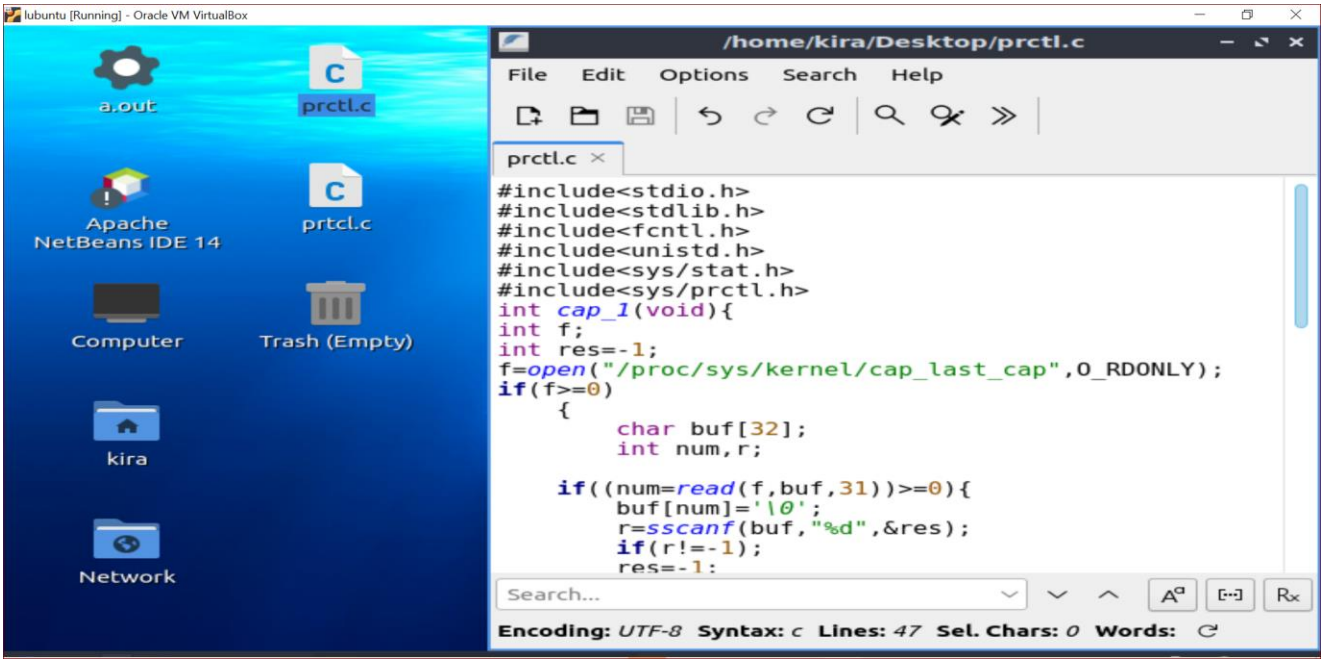


as shown below the error is debugged and the final output is compiled and executed and as shown below the name has set for each processes.

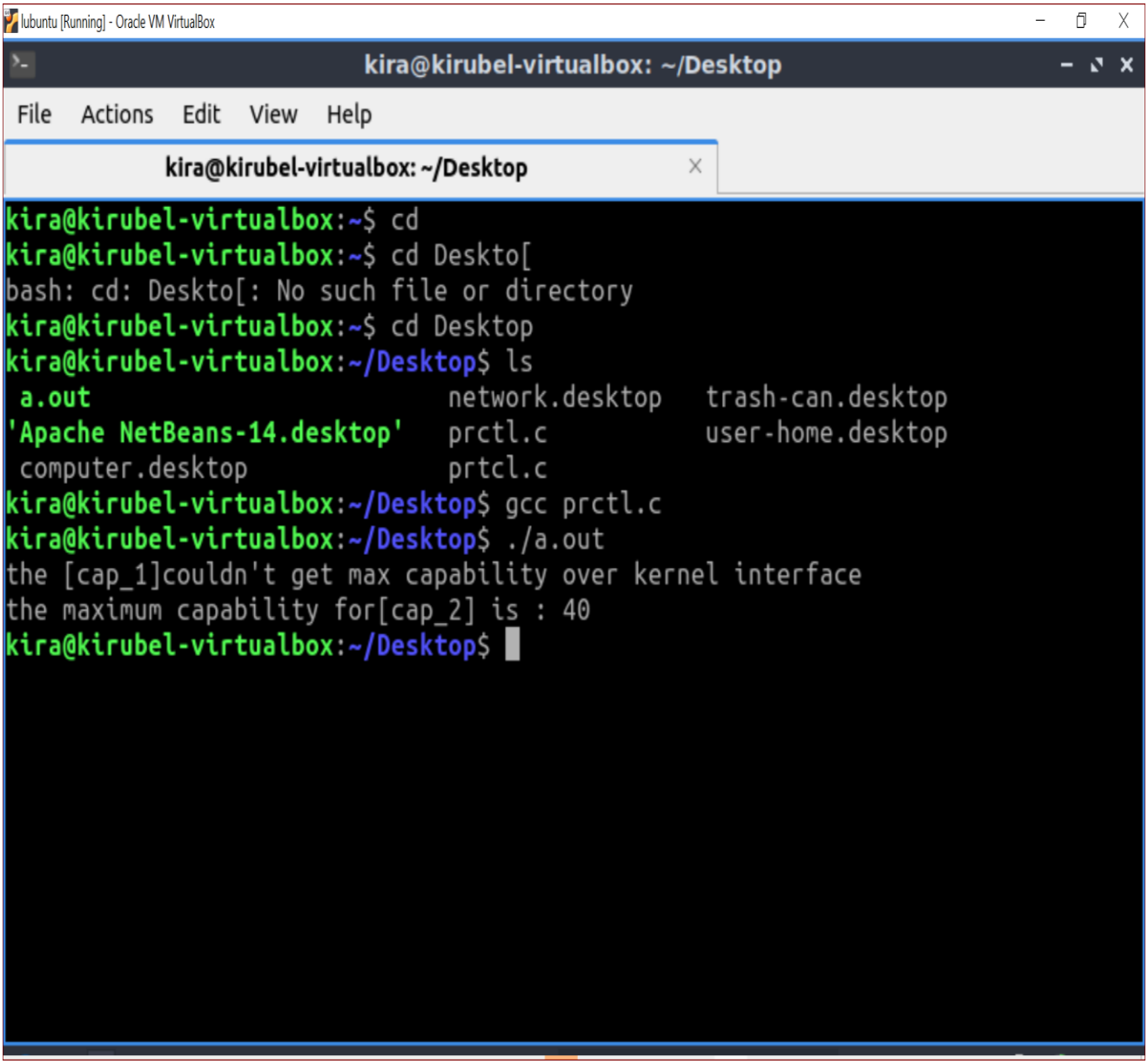


EXAMPLE 2: here is implementation of another example on my own Linux system using PRCTL system calls so here I am going to use is PR_CAPBSET_READ.





As shown in the above the PRCTL example 2 is working on my own linux system after debugging the errors so now I am going to compile the it using gcc compiler on lubuntu. So the out put be like the following below.



References

- ✓ http://www.kernel.org/doc/man-pages/online/pages/man2/prctl.2.html
- ✓ https://www.kernel.org/doc/man-pages/.
- ✓ https://manpages.ubuntu.com/manpages/bionic/man2/prctl.2.html