**Operating System**

**Assignment 7 – Procfs**

1. **Task given:**
2. Create a node in /proc file system
3. To read/write from and into the created node.
4. **Code Description:**

The kernel module code is explained below:

1. Program header: It contains the code that adds references to the Linux kernel header files that are required for all the kernel-level functions and structures that are utilized in the program. This section also contains certain variables that are used throughout the program, such as the MAX\_buff\_SIZE and NAME.

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/init.h>

#include <linux/proc\_fs.h>

#include <linux/uaccess.h>

#define MAX\_buff\_SIZE 1024

#define NAME "buffer"

1. Objects used: Defines three globally used data objects which contains the file entry in /proc, the character buffer used by the proc file and the size of the text that is inserted into the proc file.

/\* This structure hold information about the /proc file \*/

static struct proc\_dir\_entry \*PROCFILE;

/\* The buff used to store character for this module \*/

static char procfs\_buff[MAX\_buff\_SIZE];

/\*The size of the buff \*/

static unsigned long procfs\_buff\_size = 0;

1. Module initialization and termination: These functions handle the creation of the proc file and the removal of the proc file

/\* This function is executed when the module is inserted\*/

static int \_\_init Lab7\_init(void)

{

/\* create the /proc file \*/

proc\_file = proc\_create(NAME, 0644, NULL, &fops\_struct);

if (proc\_file == NULL) {

remove\_proc\_entry(NAME, NULL);

printk(KERN\_ALERT "Error: Could not initialize /proc/%s\n",

NAME);

return -ENOMEM;

}

printk(KERN\_INFO "/proc/%s created\n", NAME);

return 0;

}

/\* This function is called when the module is unloaded \*/

static void \_\_exit Lab7\_exit(void)

{

remove\_proc\_entry(NAME, NULL);

printk(KERN\_INFO "/proc/%s removed\n", NAME);

}

module\_init(Lab7\_init);

module\_exit(Lab7\_exit);

1. The file\_operations object: It is about the file\_operations structure that is used, which refers the operations detailed in the next sections to its object.

static struct file\_operations fops\_struct = {

.read = proc\_file\_read,

.write = proc\_file\_write,

};

1. The procfile\_read function: This section explains the operation that takes place when the text is to be read from the proc file. After this, the size of the buffer is sent back as the return value.

/\* This function is called then the /proc file is read \*/

static ssize\_t proc\_file\_read(struct file \*file, char \*buff, size\_t buff\_length, loff\_t \*offset)

{

static int flag = 0;

if(flag) {

printk(KERN\_INFO "read : END\n");

flag = 0;

return 0;

}

flag =1;

printk(KERN\_INFO "reading from (/proc/%s) :\n",NAME);

return sprintf(buff, procfs\_buff);

}

1. The procfile\_write function: In this operation takes place when some text is written to the proc file. Whenever some text is written to the proc file then the procfile\_write function is called. When this happens, the text length is first checked to see if it exceeds the size of the buffer. If yes, then the text is truncated to the maximum length supported by limiting the text length to the maximum length. Then the text is copied to the character buffer using the copy\_from\_user function, using the length as the limiter. If that runs without any error, it will return the buffer size, else it will return the error code corresponding to -EFAULT

/\* This function is called with the /proc file is written \*/

static ssize\_t proc\_file\_write(struct file \*file,const char \*buff, size\_t count, loff\_t \*offset)

{

/\* get buff size \*/

procfs\_buff\_size = count;

if (procfs\_buff\_size > MAX\_buff\_SIZE ) {

procfs\_buff\_size = MAX\_buff\_SIZE;

}

/\* write data to the buff \*/

if ( copy\_from\_user(procfs\_buff, buff, procfs\_buff\_size) ) {

return -EFAULT;

}

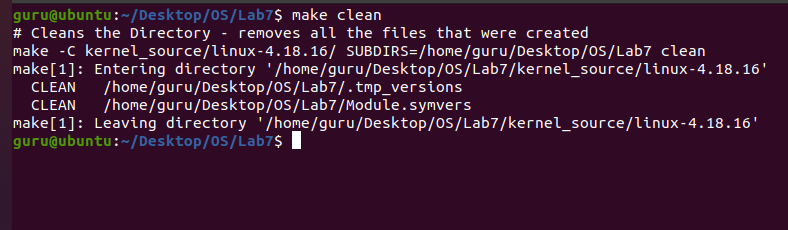
return procfs\_buff\_size;

}

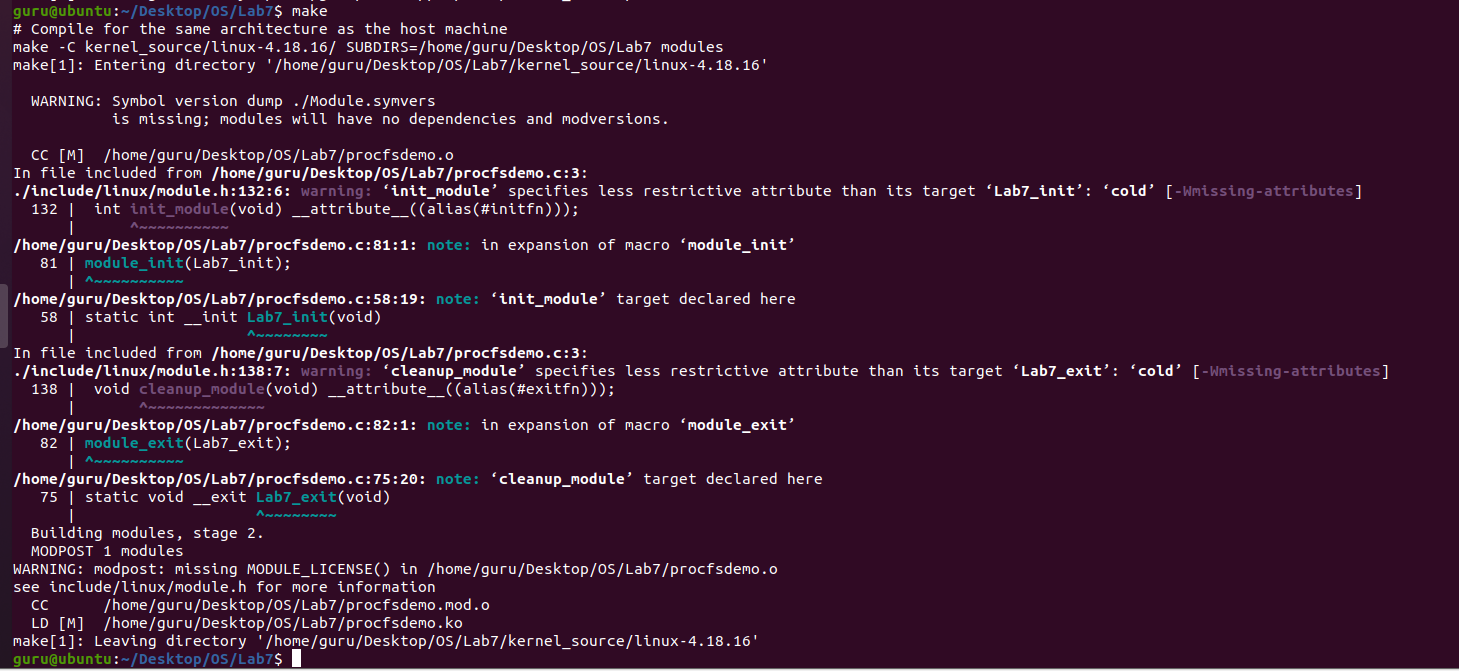
1. **Result:**

Procedure to build and run the module.

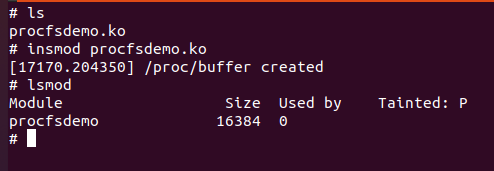
1. Create a clean build environment using ‘make clean’ command



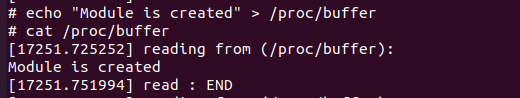
1. Build the kernel module using ‘make’ command



1. Insert the module using ‘insmod procfsdemo.ko’



1. Write and reading from proc file using command echo “Module is created” > /proc/buffer and cat /proc/buffer:



1. Removing the module from kernel using command ‘rmmod procfsdemo.ko’

