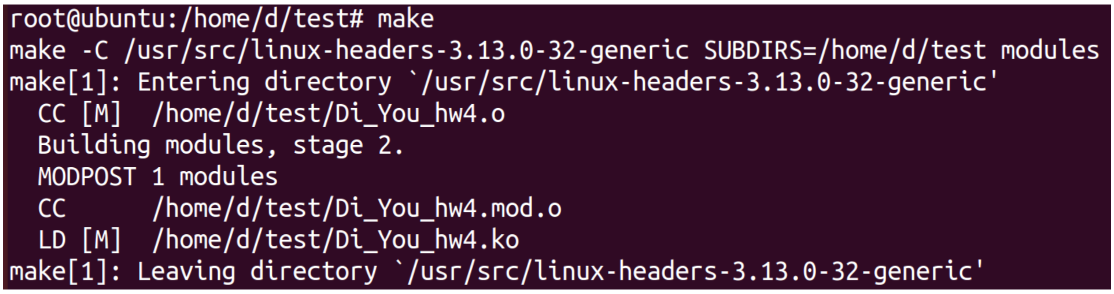
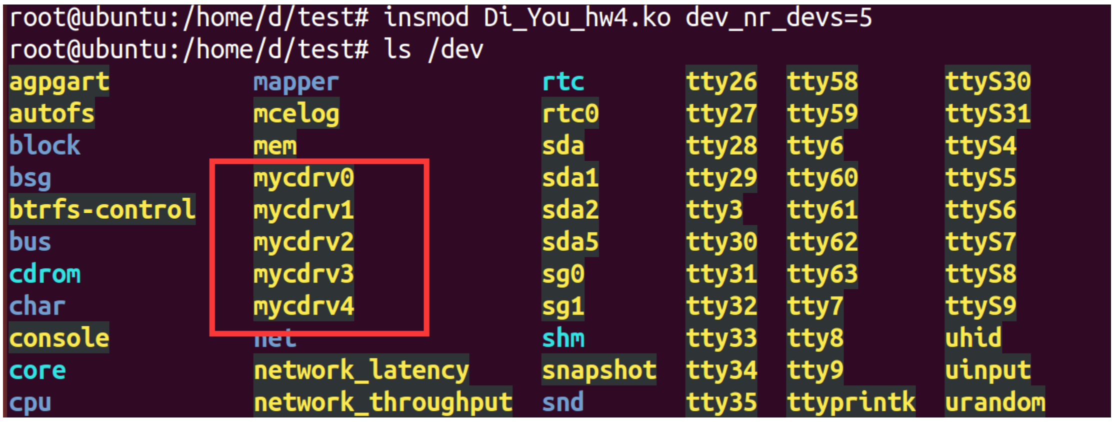
1. Put "Makefile" ,"Di_You_hw4.h", and "Di_You_hw4.c" in the same directory. Then use "make" command to compile.



```
root@ubuntu:/home/d/test# make
make -C /usr/src/linux-headers-3.13.0-32-generic SUBDIRS=/home/d/test modules
make[1]: Entering directory `/usr/src/linux-headers-3.13.0-32-generic'
  CC [M]  /home/d/test/Di_You_hw4.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/d/test/Di_You_hw4.mod.o
  LD [M]  /home/d/test/Di_You_hw4.ko
make[1]: Leaving directory `/usr/src/linux-headers-3.13.0-32-generic'
```

2. Insert module "Di_You_hw4.ko".

Use command "dev_nr_devs=5" to create 5 device nodes. Without this command the default device node number is 3.

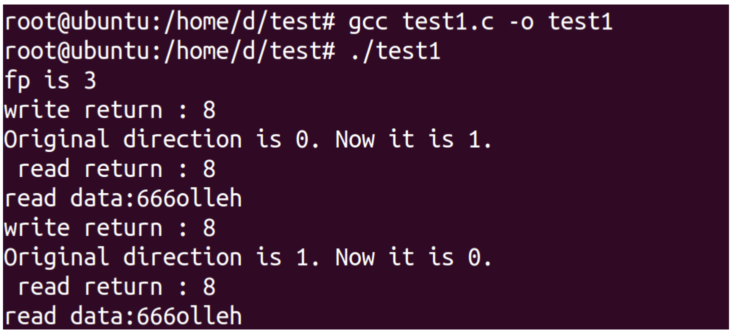


```
root@ubuntu:/home/d/test# insmod Di_You_hw4.ko dev_nr_devs=5
root@ubuntu:/home/d/test# ls /dev
agpgart          mapper              rtc       tty26  tty58      ttyS30
autofs           mcelog              rtc0      tty27  tty59      ttyS31
block            mem                 sda       tty28  tty6       ttyS4
bsg              mycdrv0             sda1      tty29  tty60      ttyS5
btrfs-control    mycdrv1             sda2      tty3   tty61      ttyS6
bus              mycdrv2             sda5      tty30  tty62      ttyS7
cdrom            mycdrv3             sg0       tty31  tty63      ttyS8
char             mycdrv4             sg1       tty32  tty7       ttyS9
console          net                 shm       tty33  tty8       uhid
core             network_latency     snapshot  tty34  tty9       uinput
cpu              network_throughput  snd       tty35  ttyprintk  urandom
```

3. To test the function of "lseek" and "ASP_CHGACCDIR", the following test is done:

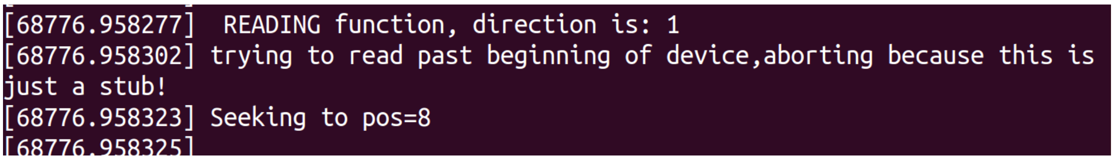
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
//Di_You_hw4.h needs to be included to use the ASP_CHGACCDIR function.
#include "Di_You_hw4.h"

int main()
{
    int fp, ret, ori;
    unsigned char ch[]="hello666";
    unsigned char *buf1 = (unsigned char *)malloc(sizeof(ch)+1);
    unsigned char *buf2 = (unsigned char *)malloc(sizeof(ch)+1);
    //init buff
    memset(buf1, 0, sizeof(ch)+1);
    memset(buf2, 0, sizeof(ch)+1);
    strcpy(buf1, ch);//ch to buf1
    //open device
    fp = open("/dev/mycdrv0", O_RDWR);
    printf("fp is %d\n", fp);

    ret = write(fp, buf1, sizeof(ch)-1);     //write buf1 to fp
    printf("write return : %d\n", ret);

    ori=ioctl(fp, ASP_CHGACCDIR, 1);     //change the direction from 0 to 1
    printf("Original direction is %d. Now it is 1. \n ", ori);

    ret = read(fp, buf2, sizeof(ch)-1);//read fp to buf2
    printf("read return : %d\n", ret);
    printf("read data:%s\n", buf2);

    ret = read(fp,buf2, sizeof(ch)-1);    //test the out of bound checking function
    lseek(fp,8,0);     //test the lseek function

    ret = write(fp, buf1, sizeof(ch)-1);//write buf1 to fp
    printf("write return : %d\n", ret);
    ori=ioctl(fp, ASP_CHGACCDIR, 0); //change the direction from1 to 0
    printf("Original direction is %d. Now it is 0. \n ", ori);
    ret = read(fp, buf2, sizeof(ch)-1);//read fp to buf2
    printf("read return : %d\n", ret);
    printf("read data:%s\n", buf2);

    close(fp);
    return 0;
}
```

This test application first writes "hello666" into mycdrv0 in regular direction(0).
Then it reads from that file pointer in the reverse(1) manner. So it should read "666olleh".
Then we use lseek to set the file pointer to the end of the string (at the last "6" in "hello666").
Then we write "hello666" in a reverse manner, which is "666olleh".
At last we read in regular manner. So it should read "666olleh"
The output is:



```
root@ubuntu:/home/d/test# gcc test1.c -o test1
root@ubuntu:/home/d/test# ./test1
fp is 3
write return : 8
Original direction is 0. Now it is 1.
 read return : 8
read data:666olleh
write return : 8
Original direction is 1. Now it is 0.
 read return : 8
read data:666olleh
```

When you type in "dmesg", the out of bound warning is shown:



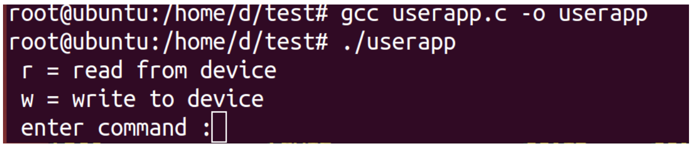
```
[68776.958277]  READING function, direction is: 1
[68776.958302] trying to read past beginning of device,aborting because this is
just a stub!
[68776.958323] Seeking to pos=8
[68776.958325]
```

The result is the same as expected.

4. To test if the device can be opened at the same time, the following experiment it done with "userapp.c" provided on Canvas.

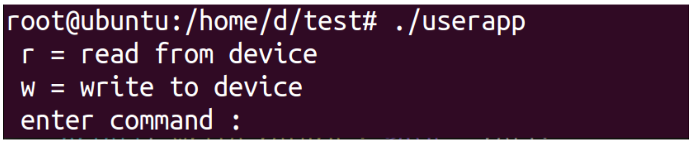
We first open two "userapp" at the same time in different terminals:

terminal1:



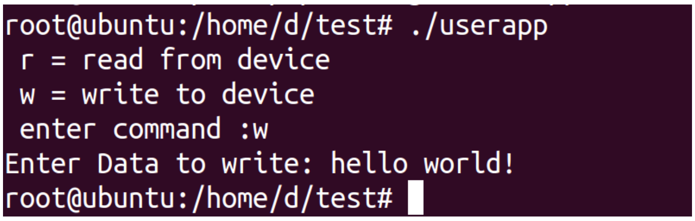
```
root@ubuntu:/home/d/test# gcc userapp.c -o userapp
root@ubuntu:/home/d/test# ./userapp
 r = read from device
 w = write to device
 enter command :
```

terminal2:



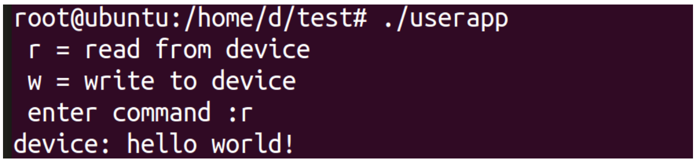
```
root@ubuntu:/home/d/test# ./userapp
 r = read from device
 w = write to device
 enter command :
```

Then we write "hello world!"in terminal1:



```
root@ubuntu:/home/d/test# ./userapp
 r = read from device
 w = write to device
 enter command :w
Enter Data to write: hello world!
root@ubuntu:/home/d/test#
```

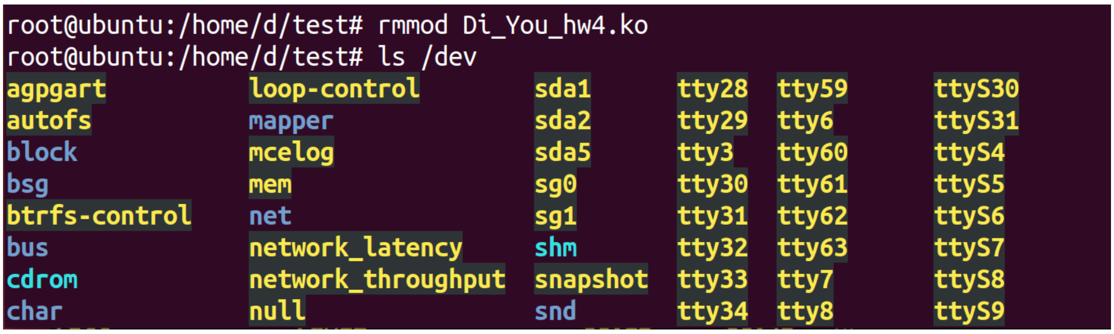
Then we read in terminal2:



```
root@ubuntu:/home/d/test# ./userapp
 r = read from device
 w = write to device
 enter command :r
device: hello world!
```

"hello world!" is read as expected.

5. Remove Module. The device nodes are gone in /dev.



```
root@ubuntu:/home/d/test# rmmod Di_You_hw4.ko
root@ubuntu:/home/d/test# ls /dev
agpgart          loop-control        sda1      tty28  tty59      ttyS30
autofs           mapper              sda2      tty29  tty6       ttyS31
block            mcelog              sda5      tty3   tty60      ttyS4
bsg              mem                 sg0       tty30  tty61      ttyS5
btrfs-control    net                 sg1       tty31  tty62      ttyS6
bus              network_latency     shm       tty32  tty63      ttyS7
cdrom            network_throughput  snapshot  tty33  tty7       ttyS8
char             null                snd       tty34  tty8       ttyS9
```