

Lab Exam 2006

Name :

Student Number :

Mark :

Time : 80 minutes

Permitted Material : You are allowed to bring any written/printed notes. However, you are not allowed to access any electronic materials. Also you are not allowed to access the web(except the COMP3300 web pages) or your class accounts.

Raise your hand and the lecturer will mark people off as they complete questions(or parts of questions). Have one window which shows a terminal of your script working and another window which shows your code. I will initial your script as you complete sections.

Question 1. [12 marks]

a) [6] The file /dev/null acts as a data sink. That is data can be written to this file and it basically vanishes! In proc write your own version of 'null'. That is when data is written to it the data should just vanish. When you cat '/proc/null' simply return the string 'This file is a black hole!'. An interaction with this proc entry should work as follows:

```
# echo -n "hello" > /proc/null
# cat /proc/null
This file is a black hole!
```

b) [6] Add to your implementation of /proc/null such that it returns the total number of bytes that has been written to it. Now it should work as follows:

```
# echo -n "hello" > /proc/null
# cat /proc/null
5 Bytes Written.
# echo -n "done" > /proc/null
# cat /proc/null
9 Bytes Written.
```

Question 2. [6 marks]

a) [2] Locate the system call within the kernel that mounts a new file system. (With this one just show me the file opened at the location of this routine. Point to it don't say anything.)

b) [2] Modify the kernel such that you add a printk that says 'doing mount' every time this system call routine is called.

c) [2] Add either a proc entry or a system call that enables you to turn off mounting. You should get the mount system call routine to return an appropriate error when you turn it off. When it is turned on the routine should just work as normal. Note that by default mounting should be turned on. If you added a system call then create a user space program to test your new system call.

<please turn over for question 3>

Question 3. [2 marks]



a) [2] Write two separate programs that when run will share some common memory. (use `mmap` and `shm_open`) Set up a test that shows this sharing of data. The first process should be able to set some data in the common memory. If `sharedmem` is the common memory then the first program should be able to set it (`sharedmem[0] = 21`) and the second program should be able to print this data (`printf("%d",sharedmem[0]);`). Note that, you will need to run the process in two different windows. Also add to the first program an infinite loop or a large sleep to stop it dropping out.