COURSE

final exam 01

DATE

Your Name Here: _____

Instructions: You have TIME for this exam. Please write your answers on the pages in this exam booklet. No scrap paper or additional sheets will be accepted. Watch your time and be concise. Write clearly (illegible answers will be 'silently ignored'), and *always* check the return value of a system call. Good luck.

prob	points	got	section
1	6		
2	6		
3	6		
4	6		
5	6		
6	6		
7	6		
8	5		
9	5		
10	5		
11	5		
12	5		
13	5		
14	5		
a	5		
i	3		
ii	3		
iii	4		
iv	4		
v	4		

Problems 1-7: For each system call given, briefly describe its purpose, the arguments it takes, and the value(s) it returns.

name	**purpose**	**arguments**	**return value(s)**
write()			
lseek()			
stat()			
signal()			
fork()			
pipe()			
wait()			

Problems 8-14: For each question, write a short, accurate answer in the space provided.

8. The inode of a file contains fields for both the *type* of the file and the *mode* of the file. What is the difference between these two attributes? How are these set, changed, and what do they affect?

9. What is the difference between *execl()* and *execv()*? When do you use which? How do you use each?

10. What is the difference between *open()* and *fopen()*? Name one advantage of each and one disadvantage of each.

11. What system call do you use to determine how many characters are in a file without having to read the entire file? How do you use this system call to get the information?

12. What system call do you use to find out what key the user of a program has set to be the 'erase' key? How do you use this system call to get the information?

13. What system calls do you use to redirect the standard input of a program to come from a file? How do you use them?

14. Why does the shell need to perform the *chdir()* system call internally rather than call a separate program?

15. This problem explores the question of designing a simple communications program for Unix.

First, some background. You attach a modem to a Unix system. It appears as the file /dev/modem in the device directory. You can open it as you open any input/output device. When you write a character into the file, it goes out the wire into the modem. When you read from this file, you get the next character that comes in over the wire from the modem.

Now, for the purpose of the program. A simple communications program allows you to connect from your terminal to the modem. The operation of such a program sounds simple: **(1)** any characters you type on your keyboard get sent out over the modem line, (this includes things like Ctrl-C and backspace); **(2)** any characters that come in from the modem go to your screen; **(3)** when you type some special key combination (like NEWLINE ~ .) the program closes the connection and exits.

Part a: A Non-Unix Solution First, consider a method that works on the simplest PC's: polling. A simple implementation of such a program is one that continuously polls the two input sources. If there is a character available at the keyboard, it gets it and writes it to the modem; it then goes to the modem: if there is a character available at the modem, it gets it and writes it to the screen; it then goes to the keyboard...

Explain how you would implement such a solution under Unix. What system calls would you have to make to initialize the program? What system calls would you make in the polling loop? Then explain the drawback of this method.

Part b) A Unix Solution The Unix solution to this problem is to use two processes, one to carry characters from the modem to the screen, the other to carry characters from the keyboard to the modem. Sounds simple? Discuss this solution by answering the following questions:

i) Why does this solution eliminate the drawback of the polled method?

ii) What initialization does this program have to do to get started? What files does it need to open? How does it get the other process? Does it matter if the files are opened before the other process is created?

iii) How should this program manage signals? Why?

iv) How should this program handle tty driver settings? Why?

v) How should this program handle the 'get me out' key sequence? In particular, what wrapup operations does it need to perform and how will it do them?