*COURSE Final Exam 07*

*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 | 4 | | |
| 2 | 4 | | |
| 3 | 4 | | |
| 4 | 4 | | |
| 5 | 4 | | |
| 6 | 4 | | |
| 7 | 4 | | |
| 8 | 4 | | |
| | | | |
| 9 | 5 | | |
| 10 | 5 | | |
| 11 | 5 | | |
| 12 | 5 | | |
| | | | |
| 13 | 4 | | |
| 14 | 4 | | |
| 15 | 4 | | |
| 16 | 4 | | |
| 17 | 4 | | |
| 18 | 4 | | |
| | | | |
| 19 | 7 | | |
| | | | |
| a | 4 | | |
| b | 4 | | |
| c | 4 | | |
| d | 5 | | |
| | | | |
| | | | |

**Problems 1-8: Short answer questions. Answer each question clearly, precisely, and refer to specific system calls when appropriate. Write your answer in the space provided.**

1. What is the difference between a *system call* and a *library function*? Give an example of each.

2. If you know the inode number of a file, how can you find the name of the file?

3. Some filenames refer to disk files (e.g. /etc/passwd) and some names refer to input/output devices (e.g. /dev/modem). Name one way in which a disk file is like a device. Name one way in which a disk file is unlike a device.

4. What is the relationship between the systems calls *kill()* and *signal()*?

5. What is the purpose of the `execvp()` system call? What value does it return, and what does that value indicate?

6. What is the purpose of environment variables? Give one example.

7. What is the meaning of `icrnl` mode, and why is it useful?

8. What is a *datagram socket*, and in what situations it is useful?

## Part Two — Four problems, each worth 5 points

Problems 9-12: Compare and contrast. Each of these problems mentions two related concepts, system calls, or operations. For each pair, explain briefly and clearly (a) what they have in common, (b) when you would use the first item, and (c) when you would use the second item.

9. `sleep()` vs `pause()`

10. `ln` vs `cp`

11. `pipe()` vs `socket()`

12. `dup()` vs `open()`

'*Sending a Message*' That phrase is common in the popular press. It usually describes an act, or word intended to warn another person, country, or group about some matter of critical importance to the sender. Computer systems consist of lots of objects that send messages to one another. The Unix system contains many situations in which something sends a message to another thing. For each of the following situations, explain (a) who sends the message, (b) who receives the message, (c) how the message is sent.

13. When a process tries to read data past the end of a file.

14. When the user wants to stop a program.

15. When a process tries to open a file to which is has no access rights.

16. When a child process terminates successfully.

17. A remote server has no more data to transmit to a client.

18. A program wants the curses library to update the user's screen

19. An enhancement to your small shell - the `while` loop

The only control structure required for your small shell assignment was `if .. then .. else .. fi`. The real shell includes the `while` loop. An example of its use is:

```
while grep float program.c
do
    echo "There are still references to floats in your program."
    echo "Please edit the file and make them doubles."
    vi program.c
done
```

The shell executes the command after the word `while` and if the command is successful, the shell executes all the commands between the `do` and `done` lines. The shell then returns to the `while` line, executes the command again, and if the command succeeds, etc..

In this question, you will explore two methods for implementing loops in shell scripts or interactive shells.

a) One method for implementing loops in a shell script is to remember the location of the start of the loop and use lseek() ( or its buffered version: fseek() ) to jump back to that location when the shell sees the 'done' line. This method works, but it has two problems.

i) Why does it not work for shells that read commands from a user at a tty?

ii) Why is it inefficient?

b) Describe a way to implement `while` that overcomes these two deficiencies. Your description does not need to be detailed; describe the general algorithm and the associated data structures.

20. *Extending I/O Redirection over a Network*

The Elegance of I/O Redirection Unix shells provide simple mechanisms for redirecting input and output of commands. The command:

```
who | sort > output
```

runs two programs and connects the output of the first command to the input to the second command, and it also sends the output of the second command to a file called 'output'. What if you wanted to enhance this feature so users could connect commands running on different machines?

History of Distributed Computing Back in the earliest days of Unix, the `uux` command allowed users to combine commands and files on different machines. A command such as

```
scws23!who | fas!sort > mymachine!/usr/spool/uucppublic/output
```

could be passed to `uux`, and it would run 'who' on scws23, send the output of that command to the input of 'sort' on fas, and would put the result of that command into a file on a third machine. Any commands or filenames without a machine prefix were assumed to be on the local machine.

The `uux` command transferred the data from one machine to another using the same modem connections used for email. The `uux` command used the notation `machine!command` and `machine!file` to refer to commands and files on other machines.

A command called `rsh` (BSD) or `remsh` (ATT) is the current tool for running commands on remote machines, but the syntax is less elegant than that of `uux`.

*Problem* For this part of the exam, answer the following questions that explore the details of adding `uux`-style syntax to a Unix shell.

First, consider how the shell would handle `who > fas!userlist`.
Sketching some diagrams may help with your explanations.

a) Why must the remote machine have a server process running? If you were implementing this system, what would the server process do? How would your server process know where to put the output? [4]

b) On the client end, the shell has to redirect output before it `execvp()`s the `who` command. What steps must the shell take to perform this cross-machine redirection? Be specific and explain your ideas. [4]

c) What errors may arise in the course of performing this operation, where do they occur, how are they identified, and how is the user notified? [4]

Second, consider what is involved if the command is remote. For example, `scws23!who > userlist`

d) What server is needed on the remote machine, and what sequence of system calls are required to attach the output of the remote command to a local file? [5]