COURSE Final Exam 11

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	6		
10	6		
11	6		
12	6		
13	6		
14	6		
15	6		
16	12		
17			
a	3		
b	3		
c	3		
d	3		
e	2		

.

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 kernel buffering? Name an advantage of kernel buffering, and name a disadvantage of kernel buffering.

2. What is a *bit mask*. Give two examples in Unix programming of uses of bit masks.

3. Almost all the information about a file is stored in an inode. Name one piece of information about a file that is not stored in the inode. What are the advantages of not storing this property in the inode?

4. Name three ways a Unix program can protect itself from being killed by a user pressing the Control-C key.

5. What is the role of the *execvp()* function and what does its return value represent?

6. What does the term *built-in command* mean in the shell?

.

7. How can a process determine if any of its child processes are still running?

8. Why is the *bind()* system call useful for server programs?

Part Two

Five problems, each worth 6 points

Problems 9-13: *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. `close()` vs `unlink()`

10. `sendto()` vs `write()`

11. `creat()` vs `fork()`

12. `raw mode` vs `cooked mode`

13. `socket()` vs `pipe()`

.

'*Enhancing Programs*' Many of the projects for the course have involved adding to some sample programs from class. Here are two examples of code from class and a change or two to make.

14. *timed2.c* This code is used in the time server. It redirects standard output then runs the date command. Other servers might want to run commands other than date, and might want to send both standard output and standard error over the network to the client.

Modify this function so it (a) accepts as an argument the name of the command to execute, and (b) redirects standard error as well as standard output to the connection. Make your changes on this code.

```
process_request(fd)
/*
 * send the date out to the client via fd
 */
{
        int     pid = fork();

        if ( pid == -1 ) return ;

        if ( pid != 0 ){
                wait(NULL);
                return;
        }

        dup2( fd, 1 );
        close(fd);
        execlp("date","date",NULL);
        oops("execlp date");
}
```

15. *mv1.c* This code is a version of the Unix **mv** command shown in class. mv renames a file, accepting as command line arguments the old name and the new name. This version uses link() and unlink() to move the file to a new name. This will not work, though, if the new name is on a different file system. In that case, the 'real' mv copies the file. If the link() system call fails because the target is on a different file system it sets errno to **EXDEV**.

Modify this code so it handles correctly the case of renaming a file across file systems. You may assume there is a function called *copy_file(char *src, char *dest)* that works just like the file copy program we wrote in lecture 2.

```
main(int ac, char *av[])
{
        extern int errno ;

        if ( link( av[1], av[2]) != -1 )
                unlink( av[1] );

        else if ( errno == EEXIST )
        {
                if (unlink( av[2] ) == -1 )
                {
                        perror(av[2]);
                        exit(1);
                }
                main(ac,av);
        }
}
```

.

16. An enhancement to your small shell - special variables

The shell you wrote for homework does variable substitution. That is, it scans each command line for strings of the form `$varname` and substitutes for the dollar sign and variable name the value of the variable. The variables you worked with are the 'named variables' - that is, variables with string names like TERM or HOME. These come from the environment and from assignment statements. There are other sorts of variables in the shell, these variables represent properties internal to the program.

Special Variables: Three popular ones

The three variables `$$`, `$?`, and `$0`, are used frequently in shell scripts. They mean:

variable	*meaning*
`$$`	process id of shell
`$?`	exit status of last command executed
`$0`	name of script being executed, or name of shell itself if not running from a script.

Explain how you would add these three special variables to your shell. Which parts of the program would you need to change? What changes would you need to make?

.

17. *Directory Assistance for Servers*

Some servers use well-known ports to make their services available. For example, web servers typically listen at port 80, and pop servers listen at port 110. Some servers just set up business at any port they can get. For example, several netpong games may be listening on one host, each at its own port. Napster users with files to share set up servers at any host and port they can get.

How can you find which servers are running on which hosts and which ports? In the world of telephone numbers, businesses list their services and telephone numbers in the yellow pages. Potential clients for services search the yellow pages to locate businesses that provide services they want.

On the Internet, servers come and go, change hosts and ports so quickly that a printed directory would be out of date at once.

The solution is to have a server that acts as dynamically updated yellow pages. When a service starts up, it registers itself with that central server with its type of service: netpong, netbridge, napster, netchat, etc and its hostname and port number.

When a person wants to play a game of netpong or bridge or hear music or chat, his or her client program will connect to the yellow pages and ask for listings under 'pong' or 'bridge' etc.

Ok so far? For this problem, you will discuss the general design of this server listing service (SLS).

Part I: Data Structures [3 points]

a) What sort of data should be stored for each server listed in the SLS? How would you store that data? Why?

Part II: A Protocol [3 points]

b) The SLS has to handle two sorts of transactions. It has to speak with servers who want to be listed, and it needs to speak with clients that want to look up servers providing services. List and briefly describe the major transactions the SLS provides.

Part III: SLS-Enabled Servers [3 points]

c) Say you want to make your pong game or pop server able to list itself with a SLS server. What changes do you need to make to it?

Part IV: SLS-enabled Clients[3 points]

d) sumac knows to connect to port 110, and netpong has to be told which port to connect to. A smart client could contact an SLS server, ask for a server that can play pong, pop3, or your favorite song, and then automatically connect to there. What changes would you need to make to your netpong or sumac client to make it do that?

Part V: Other Ideas[2 points] Add any other ideas or questions you have about this project.

.

.