COURSE Final Exam 09

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	5		
14	4		
15	4		
16	4		
17	4		
18	10		
19			
a	3		
b	3		
c	3		
d	3		
e	3		
f	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 the difference between making a copy of a file and making a link to a file?

2. When you create a new file, it has size of zero bytes. When you create a new directory, it has non-zero size. Why do directories *never* have size zero?

3. Describe one way in which terminals are like disk files and one way in which they are unlike disk files.

4. What is meant by the term *atomic operation*? Name two.

5. You can use `stat()` to see if a file with a specific name exists. How can you determine if a process with a specific PID exists?

6. If one process is writing through a pipe to another process, and the receiving process exits, how is the writing process notified that its data will never be read?

7. What is `errno` and what is its purpose?

8. Why does a server need to use `bind()` but a client does not need to use `bind()`?

Part Two

Five problems, each worth 5 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. `pause()` vs `wait()`

10. `stat()` vs `tcgetattr()`

11. `open()` vs `fopen()`

12. `interrupt-driven` vs `polling`

13. `pipe` vs `socket`

‘*Constructors and Destructors*’ The object-oriented view of the world sees all things as instances of general classes. Things come into existence by calling a constructor for the class, and things vanish by calling the destructor for that class. One can view Unix things as objects that belong to general classes. Each type of object has operations (methods) that its class supports.

For each of the following classes of Unix objects, state the method for creating one of these objects, one or more of the methods for destroying objects of that type, and state two operations you can perform on that type of thing.

14. files

15. directories

16. processes

17. file descriptor

18. An enhancement to your small shell - shell wildcards

The shell you wrote for homework does variable substitution. That is, it scans each line it reads from the user for strings of the form `$varname` and substitutes the dollar sign and variable name with the actual value of the variable.

The real shell does other sorts of substitution. One of the most useful of these is filename wildcard substitution. If the user types the line:

```
    rm *.o core /tmp/oldstuff/*
```

the shell replaces the pattern `*.o` with the names of all files in the current directory that match the pattern, and the shell replaces the pattern `/tmp/oldstuff/*` with names of files and directories that match the pattern.

The only exception is that filenames that start with a period are not matched by *.

Describe how you would modify your shell to provide the wildcard substitution feature. You do not need to include code or low-level details. You must cover at least three topics: (a) When in the sequence of parsing this step occurs (e.g. before or after splitline), (b) What system calls you would use, (c) How your logic will make correct sense of patterns like `*/*.old` or `hw*/bak/*.h` (which are supported by the shell.)

19. *Weather Grid*

Computerized Weather Stations Satellite photos provide spectacular views of storm systems and help people prepare for hurricanes, tornadoes, etc. These outer space monitors, though, cannot register temperature, rainfall, windspeed, barometric pressure. One solution would be to put low-cost computers attached to a thermometer, weather vane, wind-speed meter, and other simple machines. Position these machines spaced one mile apart on a graph-paper-like grid pattern. Use fiber optics or microwave or cell-phone links to put all these computers on the Internet.

Ok so far? Good. Each machine records a set of numbers (temp, wind direction, rainfall...) from its attached devices and sends that information to a central machine. The central machine collects the information and uses nice graphics programs to draw maps of temperatures, wind speeds, and other data. For this example assume the sensor machines and the central machine are all running Unix.

Part I: Transferring Results[6 points]

a) What method would you use for the sensors to deliver their data to the central collector? Why would you use that method? Do the sensors send data automatically? Does the collector call them and request data? Or what?

b) What information has to be delivered to the central machine? How would you format that information?

Part II: Collecting Data [3 points] A standard Unix approach to external devices is to arrange for them to appear in the `/dev` directory with names like `/dev/thermometer`, `/dev/barometer`. Simply `cat`-ing the file produces a text string with the current temperature, pressure.

c) Write an outline for the program that gathers the current measurements and gets that information to the central collector.

Part III: Receiving the Data [3 points] The central machine receives these reports from the grid of sensors. It has to store all those measurements so they can be made available to the programs that analyze the information and generate the nice pictures.

d) Write an outline for the central collector program. Explain its flow of control and any important design considerations for something that has to process input from so many sources.

Part IV: Sampling Rate[3 points] How often do results get sent in from the field?

e) If the sensors send the data themselves, how can the collector ask for more frequent or less frequent reports? If the collector polls the sensors, how does it arrange for more or less frequent queries?

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