This Really Happened

This case study is a real problem that arose during the spring of 1999 in a Perl programming course. Understanding the cause of the problem and figuring out a solution are good exercises in using the ideas and skills we have discussed in this course.

Web Servers and Standard Error

A web server, as you know by now, is a program that does three things for remote clients. It displays the contents of files, it lists the contents of a directory, and it runs programs sending the standard output of the program to the client.

What about the standard error of a program? The typical web server sends standard error to a log file on the machine the server lives on. It's not too complicated to do. The server uses `open` to open an O_APPEND connection to a log file, then `dup2`s that file descriptor to fd 2 before `exec`ing the cgi program. All errors get appended to the logfile.

Lots of Students, Lots of Errors, Lots of Disk Usage

In the Perl/CGI programming course, lots of students write lots of CGI programs on a machine used for the course. Many of those programs have syntax errors and other bugs for a while. Each time the student tests the CGI program, all the syntax errors and run-time error messages are appended to the web server error log.

That error log got really big really fast. It quickly began to use a lot of disk space.

Automatic Disk Cleaning

The initial solution was to write a program that would run periodically to delete the logfile then create a new, empty one. I think the original cleaning program would only do this if the file exceeded someone's idea of a 'large' file.

Question One: How would you set up this automatic disk cleaning?

It Sort of Didn't Work

The Unix `df` command shows the amount of free space on each disk attached to the system. The odd thing about the first solution was that even when the file was deleted and a new, empty one was created, the number reported by `df` did not go down, even though the size of the logfile would go from many megabytes to zero.

Question Two: How could it be that deleting a 4 meg, for example, file does not return 4 meg of space to the amount of free space?

tail -f Was the Problem

Students needed to see the error messages generated when they ran their CGI programs. The errors would not appear on their browser screens.

Questions Three: Why did the syntax and runtime errors in their cgi programs not appear on the browser screen?

To view the errors, the students were told to open a telnet session on their machine and run:

```
tail -f error_log
```

in that window. They would use the web browser to run their cgi script and see what happened. If the cgi program generated errors, those errors would be appended to the error log. The -f option causes tail to 'follow' along as new text was added to the end of the file.

Question Four: How would you write the -f feature of tail?

Question Five: Why did `tail -f` prevent the original cleaning method from reducing disk usage? Was the effect permanent?

This Time Fer Sure

Deleting the file did not work for reasons that you should have figured out in the previous section. A 'better' solution was devised. The first solution was to delete the file and create a new one. The next solution was to truncate the file to a size of zero bytes.

Question Six: How can you truncate a file to zero bytes? Write a program to do so when the file exceeds a certain size.

It Worked, but Something else Broke

When the big logfile was truncated, the amount of free space on the disk as reported by `df` did drop by the expected amount. This looked perfect; the disk did not fill up with error messages, the program ran automatically, so it required no supervision, the solution was clean and elegant.

The only problem was that when the file was truncated, the students running `tail -f error_log` in their telnet window stopped seeing any new messages. They ran their buggy CGI Perl scripts, got no output in the browser, or they got 'Internal Error, please see log' message, but their log file following program showed nothing.

The odd thing was if they stopped `tail -f` with Ctrl-C and started it again, everything was ok.

Question Seven: What's going on?

Question Eight: What's a solution?