Assignment 6. System call programming and debugging

Useful pointers

- Franco Callari, Block-oriented I/O in Unix (1996)
- W. Richard Stevens, Advanced Programming in the Unix Environment: Unix File I/O (2003-08-01), taken from an earlier edition of W. Richard Stevens and Stephen A. Rago, Advanced Programming in the Unix Environment, 2nd ed. (2005-06-17)
- The Open Group Base Specifications Issue 7, IEEE Std 1003.1-2008 is the official standard for system calls and some higher-level library calls.
- `man strace`
- Project: strace: Summary

Laboratory: Buffered versus unbuffered I/O

As usual, keep a log in the file `lab6.txt` of what you do in the lab so that you can reproduce the results later. This should not merely be a transcript of what you typed: it should be more like a true lab notebook, in which you briefly note down what you did and what happened.

For this laboratory, you will implement the same program using both buffered and unbuffered I/O, and compare the resulting implementations and performance.

1. Write a C program `catb.c` that uses `getchar` and `putchar` to copy all the bytes in standard input to standard output. This is roughly equivalent to the `cat` command with no arguments.
2. Write a C program `catu.c` that uses `read` and `write` to read and write each byte, instead of using `getchar` and `putchar`. The *nbyte* arguments to `read` and `write` should be 1, so that the program reads and writes single bytes at a time.
3. Use the `strace` command to compare the system calls issued by your `catb` and `catu` commands (a) when copying one file to another, and (b) when copying a file to your terminal. Use a file that contains at least 5,000,000 bytes.
4. Use the `time` command to measure how much faster one program is, compared to the other, when copying the same amount of data.

Homework: Binary sort revisited

Rewrite the `binsortu` program you wrote for Homework 5 so that it uses system calls rather than `<stdio.h>` to read standard input and write standard output. If standard input is a regular file, your program should initially allocate enough memory to hold all the data in that file all at once, rather than the usual algorithm of reallocating memory as you go. However, if the regular file grows while you are reading it, your program should still work, by allocating more memory after the initial file size has been read.

Your program should do one thing in addition to the Homework 5 program. If successful, it should use the `fprintf` function to output a line of the following form to standard error before finishing:

```
Number of comparisons: 23451
```

where the integer "23451" is replaced by the actual number of comparisons done by your program, and where a "comparison" is a binary comparison between two input lines. The line should be worded exactly as above: for example, it should contain exactly three spaces. It should be terminated with a newline.

Call the rewritten program `binsortuu`. Measure any differences in performance between `binsortu` and `binsortuu` using the `time` command. Run your program on inputs of varying numbers of input lines, and estimate the number of comparisons as a function of the number of input lines.

Submit

Submit the following files.

- The files `lab6.txt`, `catb.c`, and `catu.c` as described in the lab.
- A single source file `binsortuu.c` as described in the homework.
- A text file `binsort.txt` containing the results of your binsort performance comparison as described in the homework.

All files should be ASCII text files, with no carriage returns, and with no more than 200 columns per line. The C source file should contain no more than 132 columns per line. The shell commands

```
expand lab6.txt binsort.txt |
  awk '/\r/ || 200 < length'
expand catb.c catu.c binsortuu.c |
  awk '/\r/ || 132 < length'
```

should output nothing.

$Id: assign6.html,v 1.11 2011/02/07 17:42:35 eggert Exp $