Assignment 5. System call programming and debugging

Useful pointers

- Franco Callari, Block-oriented I/O in Unix (1996)
- The Open Group Base Specifications Issue 7, IEEE Std 1003.1-2008, 2016 Edition is the official standard for commands, system calls and some higher-level library calls.
- `man strace`
- strace on Wikipedia

Laboratory: Buffered versus unbuffered I/O

As usual, keep a log in the file `lab.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 transliteration programs `tr2b` and `tr2u` that use buffered and unbuffered I/O respectively, and compare the resulting implementations and performance. Each implementation should be a main program that takes two operands *from* and *to* that are byte strings of the same length, and that copies standard input to standard output, transliterating every byte in *from* to the corresponding byte in *to*. Your implementations should report an error *from* and *to* are not the same length, or if *from* has duplicate bytes. To summarize, your implementations should like the standard utility `tr` does, except that they have no options, characters like `[`, `-` and `\` have no special meaning in the operands, operand errors must be diagnosed, and your implementations act on bytes rather than on (possibly multibyte) characters.

1. Write a C transliteration program `tr2b.c` that uses `getchar` and `putchar` to transliterate bytes as described above.
2. Write a C program `tr2u.c` that uses `read` and `write` to transliterate bytes, 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 `tr2b` and `tr2u` 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: Encrypted sort revisited

Rewrite the `sfrob` program you wrote previously 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 `sfrob`. If given the `-f` option, your program should ignore case while sorting, by using the standard `toupper` function to upper-case each byte after decrypting and before comparing the byte. You can assume that each input byte represents a separate character; that is, you need not worry about multi-byte encodings. As noted in its specification, `toupper`'s argument should be either `EOF` or a

nonnegative value that is at most `UCHAR_MAX` (as defined in `<limits.h>`); hence one cannot simply pass a `char` value to `toupper`, as `char` is in the range `CHAR_MIN`..`CHAR_MAX`.

Call the rewritten program `sfrobu`. Measure any differences in performance between `sfrob` and `sfrobu` 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.

Also, write a shell script `sfrobs` that uses standard `tr` and `sort` to sort encrypted files using a pipeline (that is, your script should not create any temporary files, and should sort based on decrypted values just as `sfrobu` does). Your shell script should also accept an `-f` option, with the same meaning as with `sfrobu`. Use the `time` command to compare the overall performance of `sfrob`, `sfrobu`, `sfrobs`, `sfrobu -f`, and `sfrobs -f`.

Submit

Submit a compressed tarball `syscall.tgz` containing the following files.

- The files `lab.txt`, `tr2b.c`, and `tr2u.c` as described in the lab.
- A single source file `sfrobu.c` as described in the homework.
- A single shell script `sfrobs` as described in the homework.
- A text file `sfrob.txt` containing the results of your sfrob 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

```
tar xf syscall.tgz
expand lab.txt sfrob.txt |
  awk '/\r/ || 200 < length'
expand tr2b.c tr2u.c sfrobu.c sfrobs |
  awk '/\r/ || 132 < length'
```

should output nothing.

$Id: assign5.html,v 1.35 2018/02/12 00:53:06 eggert Exp $