Lab 3: Experiments with Low-Level I/O

This lab will show you how to use the raw I/O routines to investigate the effect of buffer size on the speed of execution. It will answer the question: "When copying files, is there an optimal value for the buffer size?" This is an example of "Computer Science as a lab science."

This page will be available as

> http://www.tricity.wsu.edu/~bobl/cpts360/lab03_raw_io/writeup.html

It's also linked via lab link on the course web page.

Part 1: Building `raw_copy`

The first step is to create a program `raw_copy` that will copy one file to another much like `cp(1)`, but will do so using raw I/O with a user-specified buffer size for each `read(2)` call. Its command line syntax is

> `$ ./raw_copy` *bufferSize inputFilename outputFilename*

Here is the pseudocode:

```
compute the buffer size from argv[] (hint: atoi(3))
allocate a buffer of the given size bytes (hint: malloc(3))
open the input file read-only (hint: open(2))
open the output file write-only and with "truncate" and "create"
 flags set, making sure that permissions permit writing (hint: open(2))
loop:
    attempt to read the given size bytes from the input file into
     the buffer (hint: read(2))
    if zero bytes were read,
        exit the loop (we're done reading)
    write however many bytes were read from the buffer to the output file (hint: write(2))
free the buffer (hint: free(3))
close the input file (hint: close(2))
close the output file (hint: close(2))
```

Write a makefile to compile and link `raw_copy`.

Notes:

- You don't need to create a separate function: `raw_copy` can do everything in `main()`.
- All system calls should check for errors. Feel free to use the `SYSCALL_CHECK()` macro by downloading and `#include`-ing `syscall_check.h` from the lab web page. Instructions are at the beginning of the file.
- Also feel free to use `ALLOC_ARRAY()` and the related macros in `allocarray.h`.
- The result of `read(2)` is the number of bytes read, which may be smaller than the buffer size.
- Remember to use `valgrind(1)` to check for leaks. The grader will.

Run tests to make sure your program works. A good, large, test file is one of the `/boot/initrd.img*` files, which is big (about 40MB), readable by everybody, and present on every Linux system. A good test command should be:

> `$ ./raw_copy` *bufferSize initrdFile* `tmp_file`

To check that `raw_copy` is working properly, run:

```
$ cmp initrdFile tmp_file
```

If the files are identical, there will be no output.

Part 2: Experiments

Working on the command line, use the `time` shell builtin (run "`$ help time`" for details) to measure elapsed ("real"), user, and system ("sys") time. (This program does mostly I/O, so most of its CPU time will be spent in system mode.) Collect data by varying the buffer size from 1 to 16384 by powers of 2 (i.e., 1, 2, 4, 8, 16, ...). How does varying the size of the buffer affect the amount of elapsed, user, and system time your program requires to copy the same data?

There's a script `plot_log_log` downloadable from the lab directory that takes data from an ASCII file (argument) with *x* (buffer size) and *y* (time) values on each line and plots them with a log-log plot. (The file `example.txt` is an example.) You need the `gnuplot` package installed and an X11 server. (`MobaXTerm` to `elec` *might* work for this.)

If these experiments are done properly, there are trends in the data you will be expected to notice and describe. Put that description in a `README.txt` file included in your submission tarball.

Submission

Put the source file, `Makefile`, and `README.txt` in a tarball and submit it via Blackboard.