Assignment 0

CSE 130: Principles of Computer System Design, Fall 2019

Due: October 10 at 9:00PM

Goals

There are three goals for Assignment 0. The first is to get your programming environment set up properly, which will require that you install an Ubuntu 18.04 virtual machine on your personal computer. The second is to learn the lab format for CSE 130 by writing a simple program that does the same thing as the Unix `cat` command. As with all programming assignments, this lab will require that you submit a design document, README file, and lab writeup along with your code in your `git`. The third goal is to ensure that you're ready for this class by giving you a straightforward assignment that will act as a "self-test". If you have a lot of difficulty with this assignment, you may not be ready for CSE 130.

Setting up Ubuntu

Instructions on how to set up an Ubuntu 18.04 VM are available on Canvas, and will be covered in section the week of September 30th.

Programming assignment: `dog`

Design document

Before writing code for this assignment, as with every other assignment, you must write up a design document. Your design document must be called `DESIGN.pdf`, and must be in PDF (you can easily convert other document formats, including plain text and MarkDown, to PDF). Your design document should describe the design of your code in sufficient detail that a knowledgeable programmer could duplicate your work. This includes descriptions of the data structures you use, all non-trivial algorithms and formulas, and a description of each function including its purpose, inputs, outputs, and assumptions it makes about the inputs or outputs. A sample design document is available from the resources page on Canvas.
Yes, the design for `dog` will be short, but we want you to get experience writing one up on a simple program before tackling more difficult programs. **Your design document is a significant fraction of the grade for each assignment**, so get in the habit of writing a good design document *before* you start writing code. It'll make writing code a lot easier. Also, if you want help with your code, the first thing we're going to ask for is your design document. We're happy to help you with the design, but we can't debug code without a design any more than you can.

Program functionality

The only code you have to write for this assignment is to implement the basic `cat` program, *without support for any flags*. That means your code needs to copy data from each of the files specified on the command line to standard output. For example, `dog file1 file2 file3` will copy all of the data from `file1`, `file2`, and `file3` to standard output, in that order. If `-` (dash) is given as a filename, `dog` uses standard input for that file (not a file named `-`). Note that `-` may only be given as one file name, just as `cat` allows. Your program is called `dog` (rather than `cat`) to ensure that when you run it, you don't accidentally run the installed version of `cat`.

If no files are specified on the command line, `dog` should just copy standard input to standard output until it runs out of input, just like the installed version of `cat` does. Note that the data might be binary; your code must work in that case. Process files one at a time; if `dog` runs into an error with a file, the program should print an error message to standard error (not standard output!) and skip the file, handling the remainder of the files. Your error messages should be identical (except for the program name) to those printed by `cat`; use the `warn(3)` library function for this.

Your program may not use any of the C library `FILE *` functions such as `fread()` and `printf()` for user data. You may use `fprintf()`, `perror()`, and `warn()` for error messages, but you may not use `fopen()`. `sprintf()` is fine—it doesn't take a `FILE *` as an argument. Your code must use fixed-size buffers, and may allocate no more than 32 of memory for them (either via `malloc()` or as a direct variable declaration).

Your code must be in either C or C++, though it's unlikely you'll need any C++ features for this assignment. The same will be true for all future assignments: they must be written in C or C++.

Please familiarize yourself with the Coding Guide for this class, available on Canvas. Following good coding practices will be part of your grade for this assignment and all other programming assignments.

Testing your code

You should test your code on your own system. Make up commands, and try them on both `cat` and `dog`. Use `diff(1)` to see if the commands work the same.

You might also consider cloning a new copy of your repository (from GitLab@UCSC) to a clean directory to see if it builds properly, and runs as you expect. That's an easy way to tell if your repository has all of the right files in it. You can then delete the newly-cloned copy of the directory on your local machine once you're done with it.

README and Writeup

Your repository must include a README file in MarkDown format (see https://www.markdownguide.org for details) that *must* be named `README.md`.

MarkDown is a simple markup language that provides annotations for (plain ASCII) text to be shown in bold, italics, section headers, etc. A plain ASCII text file is a valid MarkDown document—you aren't *required* to use any MarkDown annotations.

The `README.md` file should be short, and contain any instructions necessary for running your code. You should also list limitations or issues in `README.md`, telling a user if there are any known issues with your code.

Your `WRITEUP.pdf` is where you'll describe the testing you did on your program and answer any short questions the assignment might ask. The testing can be unit testing (testing of individual functions or smaller pieces of the program) or whole-system testing, which involves running your code in particular scenarios. `WRITEUP.pdf` *must* have that name, and must be PDF—there are many ways to convert plain text, Word documents, and MarkDown into PDF.

For Assignment 0, your writeup must answer the following question:

- How does the code for handling a file differs from that for handling standard input? What concept is this an example of?

Submitting your assignment

All of your files for Assignment 0 must be in the `asgn0` directory in your `git`. Make sure that the assignment you push satisfy the following:

- There are no "bad" files in the `asgn0` directory (*i.e.*, object files)
- Your assignment builds properly in `asgn0` using `make` to produce a `dog` binary
- All required files (`DESIGN.pdf`, `README.md`, `WRITEUP.pdf`) are present in `asgn0`

`After you push all your assignment files and it is ready for grading,` **`submit the git commit id to canvas.`**

Hints

- This is a straightforward assignment, designed to get you familiar with the tools you'll need for the remaining assignments. Most of your time will be spent setting up the VM, which isn't difficult, but can take a few hours. Once you have the VM set up (or even while you're setting it up), do your design document *before* writing any code.
- Become familiar with the Coding Guide for this class, available on Canvas. Do this *before* writing your code, so that you don't need to go back and rewrite it.
- **[Updated on September 28]** Become familiar with how to use git and GitLab to submit your programming assignment code. Read the Git and GitLab page in Canvas.
- You'll need to use the `open(2)`, `read(2)`, `write(2)`, and `close(2)` system calls. Read the `man` pages to learn about them. These are the *only* system calls you should need, and the only calls your program should make other than to print error messages and possibly string manipulation for examining the arguments to `dog`.
- Your commit must contain the following files:
  - `README.md`
  - `DESIGN.pdf`
  - `Makefile`
  - `dog.c` or `dog.cpp`
  - `WRITEUP.pdf`
- It may *not* contain any `.o` files. You may, if you wish, include the "source" files for your `DESIGN.pdf` and/or `WRITEUP.pdf` in your repo, but you don't have to.
- You can create a zero-byte file using the `touch` command. `touch README.md` creates a zero-byte `README.md` file if none already exists. If it *does* exist, it does nothing besides accessing the file, which is harmless and non-destructive.
- If you need help, use online documentation such as `man` pages and documentation on `Makefile`s. If you still need help, ask the course staff. You should be familiar with the rules on academic integrity *before* you start the assignment.

Grading

We will grade you on *all* of the material you turn in, with the approximate distribution of points as follows: design document (35%); coding practices (20%); functionality (40%); writeup (5%).