Homework #6: We All Have Our Daemons

Issued: Thursday, November 19
Due: Thursday, December 10

Purpose

This assignment asks you to unleash your previous solution on the Internet, as a character-category counting server (aka, a daemon). I provide the infrastructure, a skeletal daemon (if you will), and you stitch-in your counter, as guts (if you still will). I also provide a client, written in Bash.

A *client* requests service from a *server*, the server services the request, and the server returns the result to the client. The sequencing and content of this communication is called a *protocol*. My infrastructure implements a protocol, described below.

Our communication occurs over a (local) Unix *socket*. A server listens on a socket for a client request. A socket has a *port* number, which the server and client must agree upon. For example, the HTTP protocol uses port 80. For us, each student needs a unique port number. We will compute it from his or her *user identifier* (UID).

A server can compute the result in one of three ways:

- It can simply compute the result.
- It can start a new process, which computes the result.
- It can start a new thread, which computes the result.

Most servers are hardcoded as one of these. My infrastructure lets the client choose, via the protocol.

Protocol

After a client has connected with the server, a request is made, and the result is returned, according a textual, line-oriented, protocol:

1. The client sends a type-of-service command to the server: `single`, `thread`, or `process`.
2. The client sends a sequence of zero or more category-specification commands: "`add` *name chrs*".
3. The client sends a text-follows command: "`txt` *terminator*".
4. The client sends a sequence of zero or more text lines.
5. The client sends a line equal to *terminator*.
6. The server sends the zero or more lines of result.
7. The server closes its end of the connection, allowing the client to do the same.

Assignment

The infrastructure for your character-category counter daemon is at:

```
pub/cccd
```

It works! However, it doesn't do much in the way of counting. Begin by figuring out how to build, run, stop, and debug it. Be careful not to create multiple daemon processes:

```
$ ./cccd
$ ./cccd
$ pgrep -u $USER cccd
25951
25953
$ pkill -u $USER cccd
$ pgrep -u $USER cccd
$
```

A good way to debug a daemon is to have `gdb` attach to its process *while* it is executing. Simply run "`gdb -p` *pid*", where *pid* is the daemon's process identifier.

Don't forget: **After worshipping your daemon, kill it!**
Daemons don't die on their own — they're daemons.

The skeleton is a pile of code, but you only need to change bits of it (as noted):

- The makefile augments our usual makefile with preprocessor options for configuration, and a linker option for the thread library. `CCCD_PORT` configures the server's port number, based on value of the `UID` environment variable.
- A sample client is implemented by the Bash script `client.sh`. Another script, `nc.sh`, is an interactive way to request service.
- The daemon code is in `cccd.c` and the `fpp` module. This startup code is pretty icky.
- Three modules: `cmd_single`, `cmd_process`, and `cmd_thread` each provide an aptly named function, called by the daemon code, to service a request. You need to add to `cmd_process` and `cmd_thread`.
- The `chrcats` module is a stubbed-out version of your solution to the last assignment. You need to fill-in the stubs with your code.