CSC458 - Operating Systems

Operating systems with Dr. Karlsson

Exam 1 - A Diagnostic Shell

This is a shell called `dash` that is used for system process diagnostic and experimentation. The shell is executed using the `./dash` command after which you can do your usual day to day shell operations at the `dash>` prompt. Certain system process info is printed before and after issuing commands for your understanding benefit.

Algorithms/Libraries

I decided to use C++ so I could have access to more modern lanuage features.

For command parsing, I used the C++ string class and a string stream for tokenization.

I used standard system calls for all other operations to keep dependencies to a minimum.

Functions/Program Structure

main.cpp

This file contains the main loop, user input, various helpful constants, and a function useful for splitting a string into its tokens.

dash.cpp

The main bread and butter of the program, this contains a class `Dash` that contains two functions.

`run(vector<string> tokens)` - this commmand accepts tokenized input from `main.cpp`, and runs the command. It verifies if the command is unique to `dash`, in which case it runs the desired commands for desired output, or if it is a general system command. It could use some cleanup, perhaps another day... 🙃

`getInfo(string value)` - this is a PRIVATE function that runs informational commands how all my commands should have been run but I wasn't smart enough/was too lazy until the end of writing the program. Essentially takes in a destination and cats it to the output stream. This is used for getting cpuinfo, meminfo, kernel version, and uptime.

makefile

Just your good old, classic makefile that got made, scaled to really cool modular structure, didn't work, messed with it for a day or two, didn't get anywhere, and converted back to just your regular good ol' classic makefile again. Tradition!

Compilation/Usage

Build

As stated above, the good ol' regular make file should build the whole thing just fine with:

```
make
```

and should clean the whole thing just fine with:

```
make clean
```

I'm not a C++/make wizard on purpose cause I have better things to do with my time so that's as complicated as that build system gets.

Usage

To start the program, build and then run `./dash`.

There are a couple of custom commands for your own personal benefit/enjoyment. Here are their details:

`cmdnm <process_id>` - prints the command used to initiate the process_id from the arguments `pid <command>` - prints the process id or ids associated with a command name, supplied from the arguments `systat` - prints various interesting system stat stuff

Everything else about the shell should work similar to what you're familiar to, minus all the good parts that make shell like ZSH such a pleasure to use.

Testing

This was tested by running a bunch of commands against it during development as well as general 'poking around' on the following systems:

- Raspbian Linux on Pi 2
- Ubuntu 20.10 on my home server
- MacOS Big Sur
- Lubuntu 20.04 in VM
- Ubuntu 20.04 on Windows Subsystem for Linux

Various levels of functionality were supported in each environment. Here's a summary:

- getrusage() was suspect in all of the environments but worked best in MacOS (other systems struggled to give me system time)
- Windows Subsystem for Linux did all the normal shell stuff fine but the custom commands didn't work as you don't have access to the same file structure at root
- Raspbian and native ubuntus did well, supporting most things pretty well. Did notice that the arm processor reports `cpuinfo` in a much more verbose manner than the Intel x86 procs do
- There is a bug (?) that happens sometimes where you enter in a command and it pretends like it runs but then locks the whole shell from executing any more commands, even though input is still accepted. I wasn't able to trace down a definite cause so it's just gonna be a known thing that happens. A quick `ctrl-c` and `./dash` will fix the problem. Not ideal, but what can you do. 🤷