CSC 456 Homework #1 – Due 2/6/2015

Introduction to the POSIX programming environment: Diagnostic Shell

It is always good to have program for diagnostics. Your first programming assignment is to write a process identification program. This program will introduce UNIX environment, system calls, signals and the proc filesystem. Your program will be a command line program with executable name: `dash`. It should present a prompt with a prompt string: `dash>` and accept commands to be executed at the prompt.

Background:

Each process has a unique process ID, pid, associated with the process. The kernel exports a significant amount of process information in the `/proc` directory. There you can see all the running processes. Connecting into the process directory (via the process number) you can gain information about the process. You can see some examples by typing `ps aux` at the command prompt of the shell. The `ps` command is the process status command and options to this command will provide plenty of data.

Required elements:

- Typing `dash` will invoke your program.
- (35 pts) Implement the commands:
    - `cmdnm`
        - (10/35 pts) Return the command string (name) that started the process for a given process id
        - Usage: `cmdnm <pid>`
    - `pid`
        - (10/35 pts) Return the process ids for a given command string (match all substrings)
        - Usage: `pid <command>`
    - `systat`
        - (10/35 pts) Print out some process information
        - print (to stdout) Linux version information, and system uptime.
        - print memory usage information: memtotal and memfree at least.
        - print cpu information: vendor id through cache size.
        - using /proc/* files
    - `exit`
        - (5/35 pts) Exit the program
        - program termination
- (15 pts) Handle errors gracefully.
- Incorrect file or program names will not be accepted.

Solution method:

- Start the documentation and figure out how to generate the PDF: 1/23/15
- You will do the work in a subdirectory named prog1
  - `tar -czf prog1.tgz  prog1`
  - Note: **no prog1.tar.gz, no Prog1.tgz, no program1.tgz, no prog1.zip**,
- Write the parsing code: 1/26/15
- You will need to produce a Makefile to build the executable (sample below): 1/26/15
- Modify this parsing code to read from stdin (command line) and display a prompt. 1/28/15
- Create an event loop. 1/28/15
- Handle an `<enter>` without a string. 1/30/15
- Modify the event loop code to recognize the exit command. 1/30/15
- Add the `cmdnm` command. 1/31/15
- Compare to output from `ps aux`. 1/31/15
- Add the `systat` command 2/02/15
- Add the `pid` command. 2/04/15
- Add the error handling command. 2/05/15

Testing:

- Try to break the code.
- Type at least the following and hit enter:
  - `<space>`
  - `<tab>`
  - random chars
  - no chars [blank line]
  - `cmdnm abc` (should not die even if it is expecting an integer)

Code structure:

- Clean, modular, well documented code is required.
- Functions must be used to handle the user commands (a single large block is not acceptable).

Submission contents:

- Documentation - all of the following should be in: `prog1.pdf`
  - Description of the program.
  - Description of the algorithms and libraries used.
  - Description of functions and program structure.
  - How to compile and use the program.
  - Description of the testing and verification process.
  - Description of what you have submitted: `Makefile`, external functions, main, etc.
  - Format: PDF (write in any word processor and export as PDF if the option is available, or convert to PDF)
- Main program `dash.c`.
- Any required external functions `*.c`.
- Any include files you use (other than the standard ones).

- The `Makefile` to build the program `dash`.
- Tar the directory and then gzip using the correct filename.
  [And I do know that there are better compression routines than gzip.]

Notes and hints:

- You are not required to write this in C. C++ has some great string handling abilities and so would be useful in this aspect. However, this code is only a few pages of C and is not difficult in plain C.
- The point of this is to learn about system calls and some of the exported data. You will not use `fork/exec, popen(), system().`
- The kernel provides information about the processes through the `/proc` tree of the filesystem. The kernel exports the data as text files. You may read these files like any other text file. To learn about `/proc, cd /proc` and do a file list (`ls`). You can cd to a directory with a numerical name. This is all the kernel exported stats on the process with that ID number.
- If you include any of the sample code, you must completely document it. The sample code is under the Gnu Public License. Thus your code will be as well. You can find out more information about Gnu codes and the GPL at http://www.gnu.org/.
- You will need to work with command lines and strings, the following examples can give you some hints. Example of reading the arguments. If you would like more, you can find it here: Example of parsing them. You don't have to use these, they are just here to give you ideas about parsing.
- Check your code with incorrect strings or just with an empty line (plain enter). It should not seg fault.
- The cmdline in proc: `/proc/<pid>/cmdline` is a file of null terminated strings which is the parsed command line. You will need to replace the nulls with spaces to get your string compare to work correctly.
- The kernel provides information about the kernel state and kernel variables through the `/proc` tree of the filesystem. The kernel exports the data as text files. You may read these files like any other text file. Files of primary interest are `cpuinfo`, `meminfo`, `uptime`, and `version`.
- The kernel maintains data (entry into the kernel) in the /proc tree. Your program can read `/proc/cpuinfo` to get cpu speed, etc. Example of reading `/proc/cpuinfo` .

Grading approach :

- copy from the submit directory to my grading directory
- `tar -xzf prog1.tgz`
- `cd prog1`
- `make`
- `dash`
- enter commands and compare output
- assign a grade
- `cd ..`
- `rm -rf prog1`

Submission method:

You will use the submit page to submit your program.   The files need to be tarred and gzipped prior to submission. Please empty the directory prior to tar/zip. [You will tar up the directory containing the files: `tar -czf prog1.tgz prog1`]

Sample Makefile to build "crash" (your version will be slightly different):

```
#---------------------------------------------------------------------
---

# Use the GNU C/C++ compiler:
CC = gcc
CPP = g++

# COMPILER OPTIONS:

CFLAGS = -c

#OBJECT FILES
OBJS = crash.o foo.o


crash: crash.o foo.o
          ${CC} -lm ${OBJS} -o crash
crash.o: crash.c
foo.o: foo.c


#---------------------------------------------------------------------
---
```

Note: indents are tabs - NOT SPACES. Tab is a command (yes, horrible design to have whitespace commands).