CSC 456 Homework #2 – Due 3/1/2015

The second programming assignment is extending your shell, `dash`. This program will introduce the `fork` and `exec` commands as well as `pipes` and `redirects`. You will keep the executable name: `dash`. It should present a prompt with a prompt string: `dash>` and accept commands to be executed at the prompt. The approach is to fork off a child process and run the command in the child process via an `exec`. To send a signal, you will use the kill system call. To get information on this type: `man -s2 kill`. The `-s2` option says look at the second section (programmer section) of the man pages. It gives the includes as well. To see a list of the signals, try man `-s`

Required elements:

- Typing ./`dash` will invoke your shell and display the prompt:
  - dash>
- (30 pts) Your shell should take any command (plus arguments), execute it and return any stdout, for example (these will not be the test commands however):
  - `ls -al`
  - `gcc -o foo foo.c`
  - `mv foo foop`
  - Example code
  - On initiation of the command, the (child) process pid is printed after the prompt.
  - On competition of the command, the shell should print out process stats (using the `getrusage` call):
    - Time for user & system
    - Page faults and swaps
- (10 pts) Implement the shell intrinsic functions (new):
  - `cd`
    - The standard cd command:
    - `cd <absolute path>`
    - `cd <relative path>`
- (15 pts) Implement redirection:
  - `<Unix command> > file`
  - `<Unix command> < file`
  - Example code
- (20 pts) Implement shell pipes:
  - `cmd1 | cmd2`
  - Example code
- Signal (15 pts) This function will send a signal to a process.
  - `signal <signal_num> <pid>`
- Catch and display all signals it receives.
- Implement the shell intrinsic functions (already coded):
  - `cmdnm`
  - `signal`
  - `systat`
  - `exit`
- Handle errors gracefully.
- This means that if a blank line (just a newline) is entered, the prompt is redisplayed.

Submission contents:

- Documentation (10 pts)
  - 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 (as before) `dash.c`.
- Any required external functions `*.c`.
- Any include files you use (other than the standard ones).
- The Makefile to build the program dash.
- You will need to produce a Makefile to build the executable:

Solution method:

- Read the man pages for `fork, exec, wait, getrusage, chdir.` 2/6/15
- You will do the work in a subdirectory named prog2
  - `tar -czf prog2.tgz prog2`
  - Note: no `prog2.tar.gz`
- Start with the example code that uses `fork` and `exec` to run a command. 2/6/14
- Merge with program1 code. 2/9/14
- Add the `cd` and `signals` commands. 2/9/14
- Add the process stats feature. 2/16/14
- Code pipes and redirects. 2/20/14
- Error test your code 2/23/14
- Clean up errors and write pdf. 2/28/14

Testing:

- Try to break the code.
- Type the following and hit enter:
  - `<space>`
  - `<tab>`
  - random chars
  - no chars [don't hand fork a null string]
  - `<space> <space> ..... <space>` some-cmd [don't parse based on position]

Notes and hints:

- Process statistics may be gained using the `getrusage()` system call.
- The `getrusage()` function returns a struct that has current process user time. Example of `getrusage()` call.
- Note that getrusage can return data about the current process or that of all child processes (that have terminated and been waited for): `RUSAGE_SELF` or `RUSAGE_CHILDREN`. If you have multiple child processes, then you may get cumulative data.
- 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/
- No specification was made about environment variables. Use the existing ones.
- Exit should exit out of dsh not the parent process.

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 crunch.o


      crash: ${OBJS}
            ${CC} -lm ${OBJS} -o crash

      crunch.o: crunch.c
            ${CC} -c -lm crunch.c

      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).

Grading approach:

- copy from the submit directory to my grading directory
- `tar -xzf prog2.tgz`
- `cd prog2`
- `make`
- `dash`
- enter commands and compare output
- assign a grade
- `cd ..`
- `rm -rf prog2`

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 prog2.tgz prog2` .]