CSE 3320 Operating Systems
Fall 2020, © DL, UTA, 2020

Programming Assignment 1
Introduction to OS Calls, Shell
Due: On Canvas September 15, 2020

Description:
You will develop a small "text-based shell utility" ("myshell") for Unix (or similar OS). A common complaint of the most-used UNIX (text) shells (Bourne, Korn, C) is that it is difficult to remember (long) file names, and type "long" command names. A "menu" type shell allows a user to "pick" an item (file or command)from a "menu".

You will display a simple menu, the following is a suggestion (you may change format, as long as you implement the functionality):

Your shell will look (something like) this:

```
Current Working Dir: /home/os.progs/Me
It is now: 15 September 2020, 4:00 PM

Files:         0. myshell.c
               1. a.out
               2. myshell.txt
               3. assignment1
               4. program2.c

Directories:   0. ..
               1. my_subdirectory

Operation:     D  Display
               E  Edit
               R  Run
               C  Change Directory
               S  Sort Directory listing
               M  Move to Directory
               R  Remove File
               Q  Quit
```

Your shell will read a single key "command" entered by the user,
Followed by a file or directory number
   (or [optionally]) partial file name with “name completion”
and it will output some system information on the terminal,
or run a program (by means of systems calls).
The commands you will implement:
- Edit – opens a text editor with a file.
- Run - runs an executable program. You should handle parameters.
- Change – changes working directory.
- Sort – sorts listing by either size or date (prompt user)

You will need to implement:
  Prev, and Next Operations so that the menu fits on one screen.
  Store names in an array, Only one pass through directory.

  [optional]
  Add a menu using terminal codes or curses (ncurses).

If the "myshell" program has an argument, use it as the directory starting point (working dir), like: "./myshell /home".
You should provide a reasonable user interface. (See below)
You may write this in ADA, Assembler, C, C++ or Java.(Others with permission)
You should target myshell to Unix, MacOS, (or similar), or discuss alternatives with instructor.
Constraints:
  How many files and directories can you handle (max 1024)
  How long is a file name (max 2048 characters, really: limits.h, NAME_MAX)
Bonus:
  Show additional file information (date, size, read/execute)
  Use file name completion as well as a number.
  Create/use a pull-down menu.

Please, Submit ONLY to Canvas.
All work must be your own, you may reference web sites, books, or my code but You MUST site the references.
You must submit this lab, working (or partially) by the due date.
You may be asked to demonstrate this lab, working (or partially) to the GTA after the due date.
Your program should be well commented and documented, make sure the first few lines of your program contain your name, this course number, and the lab number.
Your comments should reflect your design and issues in your implementation.
You should address error conditions: what if an illegal command is entered or misspelled, what if a path can not be reached, or an "executable file" is not executable.

Some more details:

(Most) Unix allows you to have thousands of files and subdirectories in a directory (folder).

For example you might have the following files in a directory:
a.txt
b.txt
c.txt
d.txt
e.txt
f.txt
g.txt
h.txt
i.txt
j.txt
k.txt
l.txt

but because of limited space (lines) on your screen, you can only show 5 lines of file names (there are other things to display on the screen, also.) This is a “window” into your file listing.

So you would display the first 5 file names:
0. a.txt
1. b.txt
2. c.txt
3. d.txt
4. e.txt

To show the next 5 file names, a user could type “N” and you would show
5. f.txt
6. g.txt
7. h.txt
8. i.txt
9. j.txt

Pressing “N” again would show
10. k.txt
11. l.txt

At that point, pressing “P” would show
5. f.txt
6. g.txt
7. h.txt
8. i.txt
9. j.txt

And “P” again would show
0. a.txt
1. b.txt
2. c.txt
3. d.txt
4. e.txt

Of course a user may type “N” or “P” ten times in succession, so you shouldn’t try to move the “window” of files you are showing to “before” the first, or “after” the last.

Some people like a slightly more friendly view, always showing 5 file names (if possible), so that after the window
5. f.txt
6. g.txt
7. h.txt
8. i.txt
9. j.txt

Pressing “N” again would show
7. h.txt
8. i.txt
9. j.txt
10. k.txt
11. l.txt

And some people like to have a one file “overlap” so that from
0. a.txt
1. b.txt
2. c.txt
3. d.txt
4. e.txt

Pressing “N” would show
4. e.txt
5. f.txt
6. g.txt
7. h.txt
8. i.txt

And similar overlap when pressing “P”

Note: to be able to implement this file name display (efficiently) you will need to store the file names in memory. There are several options: An array of strings, an array of pointers to strings, an array of structures, or pointers to structures. These should be standard “C” or “C++” techniques.
While it is more memory efficient to have variable length strings, in this assignment you may use fixed length strings, you may choose to have a fixed number of entries (maximum number of strings) or make this dynamic, it is your choice.

Directory (“subdirectory” or “folder”) names should be treated similarly to file names.

[optional]

For bonus points, or just for fun, you may:

Add “name completion”, that is, as someone starts typing a file name as soon as some characters match a known file name, automatically complete the name (there are many ways to do this, you can try some in Unix)

Make the display look much, much better by using “terminal codes” (that is “curses” or “ncurses”). These are not very difficult, if you use basics.
Unix was designed before “bit mapped” displays were common, and can be used on a variety of display devices (and keyboards). To handle these options in a portable manner there was a method designed that works with most “character” mapped devices (terminals), called curses (moving the cursor around, and displaying text). While there are better ways to do graphics and menus on many displays, those may not be portable, so curses (or ncurses) are still very widely used.
Basically:
initscr() – sets up software configuration for display
move(), and printw() – move cursor and display
refresh() – redisplays window
getch() – gets input from keyboard
endwin() – returns display back to previous state
(there are more functions, if you need or want them.)

Show additional file information, in addition to file name, show file size, time and date of creation, R/W/X (read/write/execute permissions), and similar information.

For additional help:

System Calls:
“google”
“help” or “man” or “doc” in Unix
https://man7.org/linux/man-pages/man2/syscalls.2.html
https://opensource.com/article/19/10/strace (to look at)

Curses:
https://www.linuxjournal.com/content/getting-started-ncurses
http://heather.cs.ucdavis.edu/~matloff/UnixAndC/CLanguage/Curses.pdf
https://invisible-island.net/ncurses/ncurses-intro.html
https://stackoverflow.com/questions/905060/non-blocking-getch-ncurses
https://web.archive.org/web/20180401093525/http://cc.byexamples.com/2007/04/08/non-blocking-user-input-in-loop-without-ncurses/

C, C++
https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html
https://en.cppreference.com/w/c