Assignment 3- How to Get Started

DISPLAY :

For this part of assignment, your main goal is to print a prompt in the following format: `<username@system_name:current_dir>`. To achieve this, you need to extract three pieces of information: the username, system name, and the current directory.

Let’s start with say getting username of your system. We can achieve this with the following code:

```
#include <unistd.h>

int gethostname(char *name, size_t *namelen);
```

So, you include the `<unistd.h>` header, and then you use the `gethostname` function.

Now, let's talk about extracting the system name. This is a bit similar. You'll have to refer to online resources or man pages to find the appropriate functions and headers to include.

What's the core idea here? Well, it's about including the right header files and using built-in functions.

Extracting Current Directory

Lastly, we need to handle the current directory. It's important to remember that this isn't an absolute path. To get the current path, you should consider your home directory. You can extract the portion of the path that's relative to your home directory. This will give you a concise path to display in your prompt.

By following these steps, your display function will be ready to display 😃

BUILTIN COMMANDS (ECHO & PWD)

In this part of your assignment, you'll be dealing with two essential built-in commands: `echo` and `pwd`. We'll guide you through taking user input, tokenizing it, and executing these commands effectively.

Taking User Input

Before diving into the commands, let's understand how to handle user input. In this case, we'll focus on single commands, so you won't need to tokenize on semicolons. Instead, we'll use `strtok` to tokenize on spaces and tabs. Here's how it works:

```
char *token = strtok(input, " \t");
```

here strtok function splits the input string whenever it encounters spaces or tabs.

Handling the `pwd` Command

1. **Identifying the Command**: After tokenizing the input, the first token helps you identify the command. If it's "pwd," you'll proceed with executing the `pwd` command.
2. **Displaying the Current Directory**: The `pwd` command is quite straightforward. It only requires one header file and one command to do the job (do it yourself 🙂)

Handling the `echo` Command

1. **Identifying the Command**: For the `echo` command, once again, you'll check the first token to identify it.
2. **Managing Arguments**: The `echo` command can have multiple arguments separated by spaces. To handle this, you'll loop through these arguments and print them.

By following these steps, you can efficiently handle user input, identify the command, and execute either `pwd` or `echo` as needed.

CD:

In this part of your assignment, you will implement the 'cd' command, which stands for "change directory." This command allows the user to change the current working directory within your shell. To tackle this, you should:

When dealing with the 'cd' command in your shell, follow these steps:

1. If 'cd' is used alone, set the directory to the home directory (~).
2. If 'cd' is followed by one argument, treat it as the target directory (handle the special cases)
3. Special cases:
    - If the argument is ".." navigate one level up in the directory structure.
    - If it's "-", go to the previous directory.
    - If it's "~," go to the home directory.
    - If it's ".", stay in the current directory.
4. Error handling: If 'cd' has more than one argument, consider it an error , would be good if you can handle error using perror.h .

Some Useful Commands (Read man pages)

uname, hostname, signal, waitpid, getpid, kill, execvp, strtok, fork, getopt, readdir, opendir, readdir, closedir, getcwd, sleep, fopen, chdir, getopt, pwd.h, perror.h etc.

Type: man/man 2 <command_name> to learn of all possible invocations/variants of these general commands. Pay specific attention to the various data types used within these commands and explore the various header files needed to use these commands.

Read the man pages of the following commands and use whatever you like. This is not exhaustive list. This is just the list to start looking for commands that you may require in your implementations. You may require commands other than these.

ALL THE BEST WITH THE ASSIGNMENT 👍