EMERXTE TRAINING PROJECT DOCUMENTAION FRAMEWORK

REQUIREMENTS & DESIGN DOCUMENT

Emertxe Information Technologies (P) Ltd

Minishell

VERSION: 0.1 REVISION DATE: 09-06-2016

Version	Date	Changed By	Modifications
0.1	09-06-2016	Biju MK	Initial Draft

Contents

1 Introduction

Implement a minimalistic shell, mini-shell(msh) as part of the Linux Internal module.The objective is to understand and use the system calls w.r.t process creation, signal handling, process synchronization, exit status, text parsing etc..

2 Requirement details

2.1 Provide a prompt for the user to enter commands

1. Display the default prompt as **msh>**
2. Prompt should be customizable using environmental variable **PS1**
    - To change the prompt user will do **PS1=NEW_PROMPT**
    - Make sure that you do not allow whitespaces between =
      i.e., do not allow **PS1 = NEW_PROMPT**
    - In the above case, it should be treated like a normal command

2.2 Execute the command entered by the user

1. User will enter a command to execute
2. If it is an external command
    - Create a child process and execute the command
    - Parent should wait for the child to complete
    - Only on completion, msh prompt should be displayed
    - If user entering without a command should show the prompt again

2.3 Special Variables:

1. Exit status of the last command (echo $?)
   - After executing a command the exit status should be available
   - echo $? should print the exit status of the last command executed
2. PID of msh (echo $$)
   - echo $$: should print msh's PID
3. Shell name (echo $SHELL)
   - echo $SHELL: should print msh executable path

2.4 Signal handling

Provide short cuts to send signals to running program

1. Ctrl-C (Send SIGINT)

   On pressing Ctrl-C
   - If a programming is running in foreground, send SIGINT to the program (child process)
   - If no foreground program exists, re-display the msh prompt
2. Ctrl+z (Send SIGSTP)

   On pressing Ctrl+z
   - The program running in foreground, should stop the program and parent will display pid of child

2.5 Built-in commands

1. exit

   exit: This command will terminate the msh program
2. cd

   cd: Change directory
3. pwd:

   show the current working directory

2.6 Background Process / Job control

1. Allow a program to run in background

   To run a program in background use ampersand (&) after the command. For eg: **sleep 50 &**
2. Implement fg, bg and jobs commands
   - **bg** will will move a stopped process to background sleep 10 & is equallent to sleep 10 then ctrl + z and bg.

After this the msh prompt should be displayed indicating it is readyto accept further commands. After a bg process ends, cleanup the process using wait.

NOTE: You may have to use SIGCHLD signal handler for this

On termination of bg process, display its exit status.User should be able to run any number of background processes.

- **fg** will bring a background process to foreground. Only fg bring last background process, or fg <pid> will bring given pid to foreground.
- **jobs** will print all background process details.

2.7 Pipe functionality

1. Allow multiple processes communication by using pipes.
2. Create pipes and childs dynamically as per pipes passed from command-line

   Eg: ls | wc, ls -l /dev | grep tty | wc -l

3 Design details

You should write down the how your are going to implement the project.

You have to think through the design, then write down a detailed description of the design.

Use diagram like flowchart to give clarity. Write down the algorithm/pseudo-code as appropriate.

To implement job control create a datastucture (linked list) of jobs. sample code is given.

4 Coding guidelines

1. Use proper naming conventions , variables, functions, data types, file names and directories.
2. Do not hard code values.
3. Arrange class definition into separate headers an functions in separate .h and .c files.
4. Add block comments for files, and functions. Apart from this add comments wherever applicable

5 References

Provide a list of all documents and other sources of information referenced