Design Document for qShell

Goal

We all know in theory that a shell an interface between operating system and user. But not all of us know how shell achieves this. The goal of developing a customized shell was to understand different aspects involved in shell design, different conditions that are required to be handled and the way in which operating system should react to every command that a user types.

Shell Supported Functions

1. Basic Command Execution which involves fork and exec.

2. Redirections which includes input, output redirection (<,>,>>)

3. Running Process in Background

4. Support alias command

5. Alarm function

6. if command; then command; else command; fi;

7. ***Additional features:*** conditional execution (,&& and ||). Customized cd command and appropriate prompt display manipulation

Shell Design Decisions and Assumptions

One of the major decisions taken around developing shell was on focusing what we really want to achieve and following design aspects specific to operating systems were taken into consideration.

1. Signal handling : The way in which shell should react when in signals like Ctrl+c are sent.

2. Exception Handling : How do we handle memory related exceptions and access related exceptions.

4. Process Management: Deal with child process creation and running process in background.

3. File Management: Manipulating file descriptors, creating duplicate file descriptors and dealing with redirection.

4. Timer Management: fetching elapsed time and accordingly take further actions.

We had to make a design decision when it comes to parsing the shell commands. In our case we made sure that we focus more on intricacies related to shell development and not focusing on lexical analysis and parsing as it completely shifts focus from operating system design to grammar.

Assumptions:

In case of parsing we are using space as a separator and tab, up-down arrow keys are not supported and may impact shell command execution.

In qshell every command is treated as character string.

Shell Command Execution Flow

1. Launch Shell and display the prompt.

2. Wait for user to enter command.

3. Once command is received simple parser(written by us) parses the command.

4. If the command systematics are incorrect display appropriate syntax error and return prompt.

5. If parser identifies command as valid command, it send it the command to execution unit.

6. Execution unit takes command and its arguments and depending on the command type it calls respective module

residing inside execution unit to execute the command.

8. If execution unit comes across any exceptions it displays appropriate message and returns shell prompt.

7. Once execution unit is finished execution it returns the prompt and shell waits for user to enter next command.

Figure below shows the architecture of qShell

qShell Components

qShell uses modular approach and consists of following components which perform specific tasks:

-Command line display – Displays prompt with current working directory.

-Parsing Unit – Takes command from user and converts it into command structures which is passed to execution unit.

-Execution Unit – As name suggests handles command execution and also looks after exception handling.

-Utility Unit – This unit facilitates configurable shell and handles cases related cd command.

-Supporting header file – Loads all the required libraries and contains definitions of data structures used in shell.

-Handler Unit – This unit takes care of signal handling associated with qShell.

Design Aspects from User Experience Perspective

1. Exception Handling using recovery mode: In this case when qShell tries to fork a process and if memory being bottleneck fork fails to create process then we try to fork the process by giving 3 attempts if all the 3 attempts fail for fork process then we show fatal error and shell terminates.

2. User Friendly: We made sure that qshell displays detailed and appropriate error messages when basic command fails to execute instead of just flashing error occurred. The reason behind detailed display messages is to pin point user what went wrong with the command user entered so that user can easily rectify it.

3. Exit Prompt: Whenever user presses Ctrl+C shell should ask user before exiting and if user confirms then and only then shell exits i.e kills all the processes and terminates itself.

4. Shell Prompt Availability: Whenever user hits enter on shell prompt instead of considering it as command new shell prompt should be displayed and its been handled accordingly so that user knows his current working directory.

5. Configurable: User can configure shell prompt sign and can customize look and feel with limited options. We also allow to configure/edit PATH variables and alarm option from profile file.

Design Aspects from Operating System Perspective

1. Exception Handling: The shell should be able handle exceptions like out of memory, file descriptor manipulation related exceptions like can not access the file etc and should accordingly educate user regarding the exception. We made sure that qShell checks for different conditions like out of memory, could not open file, invalid file name, access related issues and have handled in modules like alarm where user can not terminate process created by someone else if he doesn't have right to do so.

2. Signal Handling: In case of running processes in **background**, we have disabled interrupt signals so that child will execute in background and avoiding possibility of zombie processes.

3. Memory Management: Dynamic memory allocation for command structure and deallocation of the same once command has finished execution so that there are no dangling command structures which in turn avoids issues related to dangling pointers.

4. If then Else functionality : The 'if then else' feature required special handling in case of syntax validation as every character has to follow certain rules in order make entire command a valid command. Also return code of every process needs to be maintained and depending on the return code further execution decisions were to be made.

5. Persistent Alias: We maintain a file which records all the valid aliases and every time shell is invoked this file is loaded so that for every new session the alias persists. If user tries to create alias which resembles system command user is not allowed to do as it disrupts normal shell behavior and should be forbidden.