Networking Education with Knowledge & Technology

LINUX SHELL DEVELOPMENT

PRODUCT DESIGN SPECIFICATION

Version <1.0>

<04/07/2013>

NEK Tech

Corporate Office: NEKTech Educational Consulting Pvt. Ltd., 178, M-9, Chitra Complex, MP Nagar Zone I, Bhopal – 462011

Email: service@nektech.in Web: www.nektech.in Contact: +91-77710 43826, +91-755 4224605

VERSION HISTORY

Version #	Implemented By	Revision Date	Approved By	Approval Date	Reason
1.0	*Deepika Pandey*	*06/07/2013*	*Pankaj Saraf*	*09/07/2013*	Approved

TABLE OF CONTENTS

1 INTRODUCTION

1.1 PURPOSE OF THE PRODUCT DESIGN SPECIFICATION DOCUMENT

This Document is intended to provide lower level design and implementation details of “NEK Tech Shell”. Firstly we design main(),it takes a user input from “Command Line Interfaces”. In this main(), parses the input and handles modifies And arranges command/arguments according to “Linux System Call Interfaces” and provide to kernel to get the required services from Linux kernel. If user input is cd (which is not actually command) then it calls chdir(). // shubhangi Design Doc

If user input is any command then, the command execution in Traditional Unix Shell is achieved by two Unix Interfaces (System calls) *fork()* and *exec()*: (Briefly described below) // Pallavi Design Doc

fork():

When you come to metaphorical "fork in the road" you generally have two options to take, and your decision affects your future. Computer programs reach this fork in the road when they hit the `fork()` system call.

At this point, the operating system will create a new process that is exactly the same as the parent process. This means all the state that was talked about previously is copied, including open files; register state and all memory allocations, which includes the program code.

The return value from the system call is the only way the process can determine if it was the existing process or a new one. The return value to the parent process will be the Process ID (PID) of the child, whilst the child will get a return value of 0.

At this point, we say the process has `forked` and we have the parent-child relationship as described above.

exec():

Forking provides a way for an existing process to start a new one, but what about the case where the new process is not part of the same program as parent process? This is the case in the shell; when a user starts a command it needs to run in a new process, but it is unrelated to the shell.

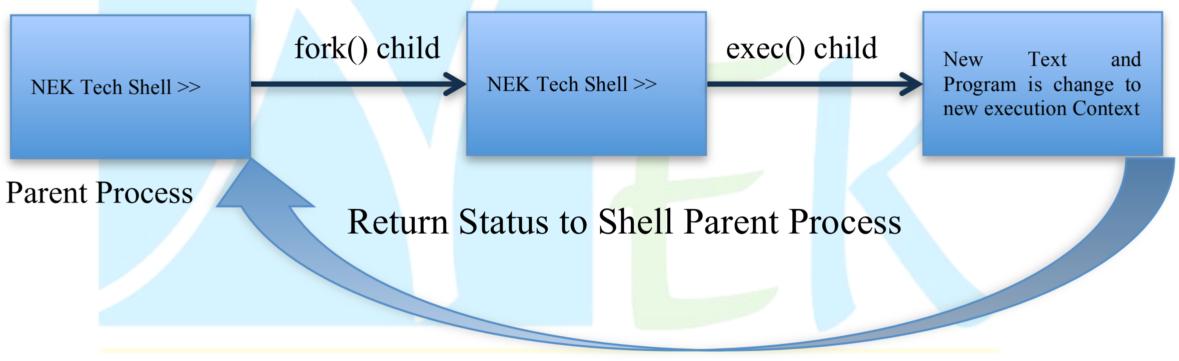
This is where the `exec` system call comes into play. `exec` will *replace* the contents of the currently running process with the information from a program binary.

Thus the process the shell follows when launching a new program is to firstly `fork`, creating a new process, and then `exec` (i.e. load into memory and execute) the program binary it is supposed to run.

Example- ls -l

2 GENERAL OVERVIEW AND DESIGN GUIDELINES/APPROACH

"NEK Tech Shell" waits for user input and once it receives command (except cd command a special case). It forks a new shell process and exec() it with the binary provided. The outer line of the architecture is as follows:


NEK Tech Shell >>
fork() child
NEK Tech Shell >>
exec() child
New Text and Program is change to new execution Context
Parent Process
Return Status to Shell Parent Process

2.1 ASSUMPTIONS / CONSTRAINTS / STANDARDS

"NEK Tech Shell" First version will provide the whole idea about he operating systems work model. It is expected to handle Change directory command, which is special case as, it will be achieved by *chdir ()* system call rather than fork/exec model. This version of shell will not cover the implementation for shell special features, logical operator, shell special characters and other shell functionalities.

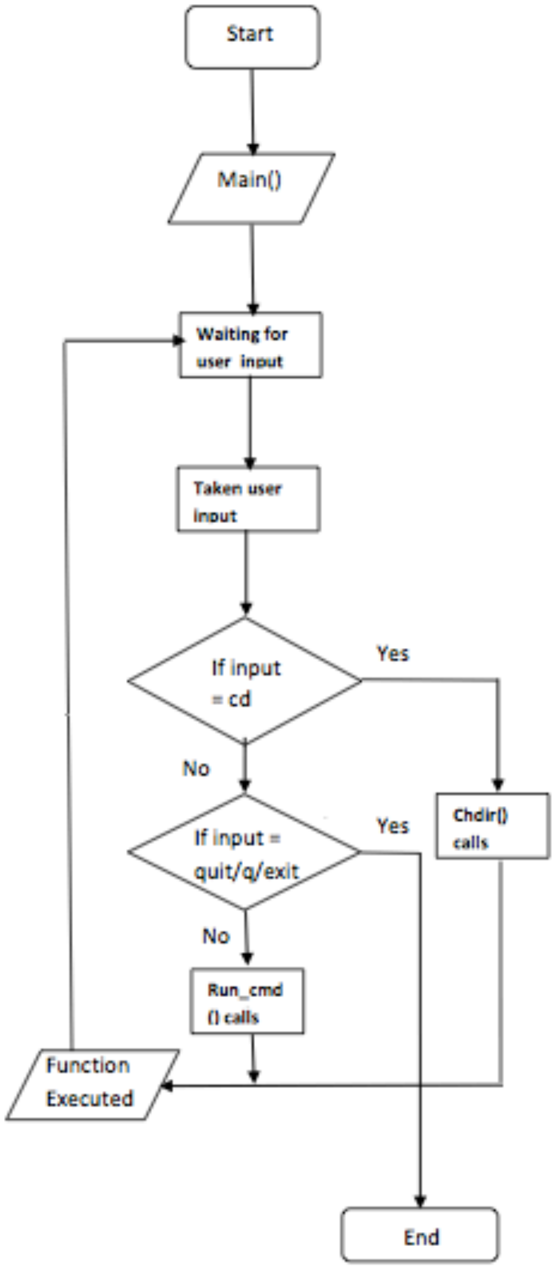
3 ARCHITECTURE DESIGN

This module will handle the call and executions of command in "NEK Tech Linux shell".

3.1 LOGICAL VIEW

This module will cover the functionality of taking user input and deciding the calls depending on the input. Following will be the flow chart for it.

Flow Chart-


Start
Main()
Waiting for user input
Taken user input
If input = cd
Yes
No
Chdir() calls
If input = quit/q/exit
Yes
No
Run_cmd () calls
Function Executed
End

3.2 HARDWARE ARCHITECTURE

N/A

3.3 SOFTWARE ARCHITECTURE

This module will be called by "NEK Tech shell" main function to create new process and perform an exec operation on the same.

3.4 SECURITY ARCHITECTURE

N/A. Not covered in this cycle of development.

3.5 COMMUNICATION ARCHITECTURE

N/A. No Client-Server connectivity for "NEK Tech shell" in this version.

3.6 PERFORMANCE

It will impose generic overhead on performance to create a process in the system.

4 SYSTEM DESIGN

4.1 USE-CASES

Shell Provides command line interface, which allows advance user to use operating systems services to perform computational, needs.

4.2 APPLICATION PROGRAM INTERFACES

Following interfaces will be used in this module:

fork():

When you come to metaphorical "fork in the road" you generally have two options to take, and your decision affects your future. Computer programs reach this fork in the road when they hit the `fork()` system call.

At this point, the operating system will create a new process that is exactly the same as the parent process. This means all the state that was talked about previously is copied, including open files; register state and all memory allocations, which includes the program code.

The return value from the system call is the only way the process can determine if it was the existing process or a new one. The return value to the parent process will be the Process ID (PID) of the child, whilst the child will get a return value of 0.

At this point, we say the process has `forked` and we have the parent-child relationship as described above.

exec():

Forking provides a way for an existing process to start a new one, but what about the case where the new process is not part of the same program as parent process? This is the case in the shell; when a user starts a command it needs to run in a new process, but it is unrelated to the shell.

This is where the `exec` system call comes into play. `exec` will *replace* the contents of the currently running process with the information from a program binary.

Thus the process the shell follows when launching a new program is to firstly `fork`, creating a new process, and then `exec` (i.e. load into memory and execute) the program binary it is supposed to run.

4.3 USER INTERFACE DESIGN

N/A

4.4 PERFORMANCE

N/A

4.5 COMPLIANCE

N/A

5 PRODUCT DESIGN SPECIFICATIONAPPROVAL

The undersigned acknowledge they have reviewed the "NEK Tech Shell" Specification document and agree with the approach it presents. Any changes to this Requirements Definition will be coordinated with and approved by the undersigned or their designated representatives.

Signature:	______________________	Date:	____________
Name:	______________________		
Title:	______________________		
Role:	______________________		

Appendix A: References

The following table summarizes the documents referenced in this document.

Document Name and Version	Description	Location
Fork() and exec()	Linux Shell internals	http://www.bottomupcs.com/fork_and_exec.html
Processes	Linux Shell internals	http://www.ucblueash.edu/thomas/Intro_Unix_Text/Process.html

Appendix B: Key Terms

The following table provides definitions for terms relevant to this document.

Term	Definition
fork()	Linux system call to create a process
exec()	Linux system call to start new program with in a process