**CS551-Operating Systems Design and Implementation**

**Project 1: Develop your own shell**

# Submitted by

**Gaurav Bharadwaja munipally (A20301453)**

**Lokesh Venkatesan (A20310836)**

**Sowjanya Potha (A20288821)**

**Objective**

The main objective of this project is to create our own shell that meets the given requirements.

# Shell Design

We designed a new shell as per the given requirements. This shell is invoked from the „Ash‟ shell provided with unix. We named this shell as “Msh”.

To run this shell we need gcc compiler as there are some functionalities that

work only gcc libraries to make Shell work

**Extra Implementation:** We have implemented Ah shell within Ash Shell. So that we can run new Shell with in present shell with all the facilities that are provided by the new Shell.

To make this execute „cd /mnt/OSDI/src‟ and enter “./main1” to execute ne shell.

# Shell Prompt

After entering into this shell, our shell will execute a PROFILE file which will displays Prompt sign and allows user to access facilities provided in /bin and /usr/bin. After entering into shell you will be directed to Home directory as we specified the path in the PROFILE file.

# Exercise Utilities

The utilities provided in /bin and /usr/bin can be exercised in the newly created shell. By using execvp() function, these facilities are made available to user at the shell prompt. The user is allowed to use the utilities whichever he wants to use on the command line. The utilities may include ls, cat, date, etc. The command line read the tokens and then parsed to generate the command and arguments. The path is generated then and execvp() function will be called and parsed tokens will be passed to it, which will execute and provide output to the user.

# Spawn a New Process

Shell by executing fork (), generates a new child process, when user enters the „spawn‟ command. The new process id will be echoed and the control is given back to the shell that we created. An alarm will be triggered after 5 minutes from the time the user entered the command. If the command takes more than 5 minutes to complete the shell prompts the user to terminate the command processing.

**Nesting of two Commands**

we can you nest the $(…) facility. The new shell is capable of implementing the nesting of two commands. When you nest the $(...) facility, the result of the nested $(...) is given to the outer $(...), and it executes normally. For instance consider the following example, if the user gives the command $wc $ (fgrep –l malloc /mnt/OSDI/src \*.c), the final result will be the value of word count.

**Terminating from New Shell**

To terminate from the new shell the user should either type the command „exit‟ ‟. The shell prompts the user to confirm the action user “Are you sure? (Yes/No). In this case entering „yes‟ will terminate the new shell and entering „no‟ will not terminate the present running shell and displays the prompt.

**Exception handling**

Errors may generate during the execution of a program. A good program always handles these errors. Therefore it is important to use some exception handling techniques to encounter the errors generated. The type of exception handling technique used will be dependent on the respective error generated. for example consider the following example, If the user enters wrong command, the exception handling function is called and the message “command not found” is displayed.

**Nesting $() Facility**

In this we implemented nesting of shell which works as using pipe system call. Two file descriptors are returned--fildes[0] and fildes[1], and they are both open for reading and writing. A read from fildes[0] accesses the data written to fildes[1] on a first-in-first-out (FIFO) basis and a read from fildes[1] accesses the data written to fildes[0] also on a FIFO basis. when we are nesting the ‘$’ Facility the last command is assumed to be writing to ‘stdo’ and the next post command is assumed to be reading from stdin. So, it common practice to assign the pipe write device descriptor to stdout in the last command and assign the pipe read device descriptor to stdin in the next command. This nesting of ‘$’ facility acts as piping of one or more commands but in counter direction as pipe. This facility is possible to implement and works only for valid commands that can give certain output to the previous command. This facility is evaluated from right to left and provides the desired output for the last command that is executed.