Shell

Assignment 3

Suchet Aggarwal

2018105

Design Documentation:

The given implementation of shell is capable of handling simple commands present in any UNIX/Linux Distribution such as ls, cat, etc.
The following table shows the capabilities of the shell designed…

Syntax	Meaning
command	execute the command and wait for the command to finish, print error message if the command is invalid
command > filename	redirect stdout to file filename. If the file does not exist create one, otherwise, overwrite the existing file
command >> filename	If the filename already exists append the stdout output, otherwise, create a new file
1>filename	redirect stdout to filename
2>filename	redirect stderr to filename
2>&1	redirect stderr to stdout
command < filename	use file descriptor 0 (stdin) for filename. If command tries to read from stdin, effectively it will read from filename.
|	pipe command (as discussed in class)
exit	exit from the shell program

By default, the shell program waits for user input from the stdin. After the user enters some command, the shell program parses the input to interpret I/O redirection, pipe, etc.) and then:

If there is a single command (i.e. no pipes):

Then a child process is forked, and with appropriate output/input redirection and the command is executed.

Else:

The No. of pipes (and hence the no. of commands) are determined, and those many no. of pipes are created and at each iteration, and a child process is forked and this executes the corresponding command, after appropriately adjusting the output/input file descriptors according to the pipe in the original command.

When the command **"/bin/ls | /usr/bin/sort | /usr/bin/uniq"** is executed, the input string is passed to the read_command() input, the input is tokenised into three commands

1. /usr/bin/ls
2. /usr/bin/sort
3. /usr/bin/uniq

And correspondingly line 90 of shell.c is executed (as cnt = 3), 2 pipes are created and these pipes are set accordingly by process

pid[0]:

redirects output to fd[0][1]

pid[1]:

redirects input to fd[0][0]
redirects output to fd[1][1]

pid[2]:

redirects input to fd[1][0]

and each process calls exec(), while the parent waits for all child process finish their execution.

Pseudocode corresponding to the Main shell:

```
While the input is not “exit” do:
        Read input from stdiin
        Fork a new child process and call read_command(input)
        While the parent waits for the child to finish
        The child process returns and calls exit(0)
```

Pseudocode corresponding to read_command:

```
Read_command(input):
        If the command is empty string:
                Return to main

        Count the no. of occurrences of “|” (pipe), and tokenise the string input separated by the
        delimiter “|”, and store the i^th token in cmd[i]

        For i in {0…cnt-1]:
                Tokenize cmd[i] separated by delimiter “ “ (white space) into args[i]
                i.e.: args[i][j] = j^th Token in cmd[i]
                i.e.: args[i] = { command , first argument , second argument … }

        If “<” is in args[0] :
                Then create a file descriptor fdi and open a file of the specified name and set rest of
                the args[i][k] to NULL (where args[i][k] was equal to “<”)

        If “>” or “>>” is in args[cnt-1] :
                Then create a file descriptor fdf and open a file of the specified name (in append
                mode if “>>” is present) and set rest of the args[i][k] to NULL (where args[i][k] was
                equal to “>” or “>>”)

        If cnt>1:
                Create pid[] , fd[][2] of size cnt
                Fork a child process and then set the pipe accordingly and call exec, while the
                parent waits
        Else:
                Create a single child process, set input/output redirection and then call exec, while
                the parent waits.
```