CSE 231: Operating Systems

ASSIGNMENT 3

SHELL

Shell is a basic program run by the OS to help user perform many functions like reading a file, creating a file, listing all files in directory, changing directory, install packages, update them, and much more. The shell has certain inbuilt commands like cd, history etc. and a lot more external commands whose code is written and compiled as separate files. Some external commands are ls, sort, uniq, which, pwd, etc. The binaries for these external commands is mostly found in /bin, but if not, then can be found using which command.

These commands can become even more powerful if combined with some operators like pipe, IO redirection, etc. For eg: pipe helps in inter-process communication.

These operations can mostly be implemented using pipe, fork, read, write, open, close, dup, exec and wait APIs provided by the OS.

In this assignment, the objective is to implement a basic shell, which given an external command executes it by invoking its binary. The command can have pipes, IO, etc. The commands can be nested as well. Following are the features implemented:

Syntax	Meaning
command	Executes the command and waits for the command to finish, prints error message if the command is invalid
command > filename	Redirects stdout to file filename. If the file does not exist creates one, otherwise, overwrites the existing file
command >> filename	If the filename already exists appends the stdout output, otherwise, creates a new file
1>filename	Redirects stdout to filename
2>filename	Redirects stderr to filename

Submitted by: Vibhu Agrawal, 2018116

2>&1	Redirects stderr to stdout
command<filename	Reads from the given file. If the file does not exist then throws an error.
|	Pipes output of left command to input of right command.
exit	Exist from the shell program

Sample input explained:

Input: /bin/ls | /bin/sort | /bin/uniq

Code Unboxed:

Initially, the infinite while loop waits for some input. It does not hog onto the CPU, but just waits until an input is received. The input is read by readline function implemented from STDIN. After reading the input, pipe_splits method breaks down the input in arrays of strings on the basis of pipes and counts the total number of commands to be executed (3 in this case.)

Now a call to fork creates a new child which executes the actual command. While the child is executing, the parent just waits for the child to return, and again goes through the while loop on returning.

In child, a call to executeCommand() is made. Execute command directly moves onto execution of command if just a single command remains to be executed, noted using value of pipeCount. (Explained later).

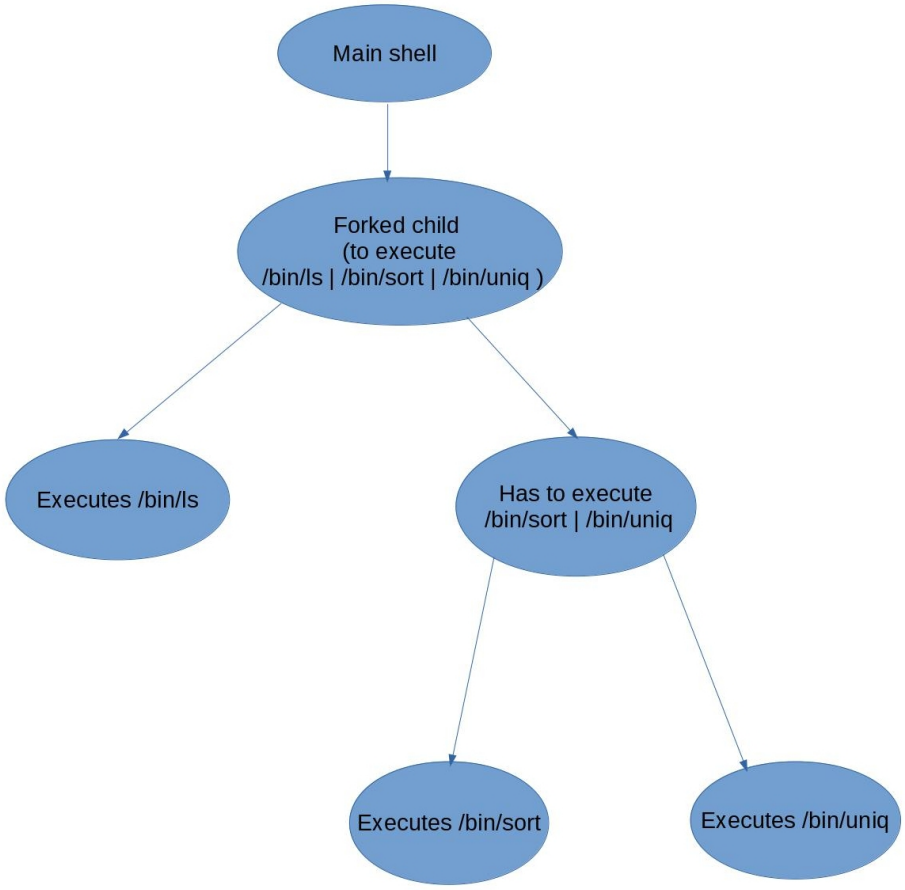
If it is a piped command, as in this case, two childen are created, and the first child calls executeCommand() after closing STDOUT and setting the read end of pipe to file descriptor 1, for the command on the left part of the pipe. Now since this command cannot be broken down further, therefore, the command is executed using a call to execvp(). To set the file descriptors, the command is parsed, and if any IO operation is found, file descriptors are changed according to it. The command is broken down into NULL

terminated arguments on the basis of space and the command name. These are sent to execvp as parameters.

For the second child, executeCommand is again called, and if it still contains pipes (checked using the value of index in global list containing all commands, and pipeCount), then the process of creating further 2 children is continued in the same recursive manner.

The parent process makes two calls to wait() in order to wait for its two children to return.

The process tree in this case is as follows:



Corresponding Pseudocode:

```
int setSymbols(char* input, char* command, char** com_args){
        Traverse the input to find spaces and special symbols like <, >>, >, or
space{

        if (character of input is space){
                write '\0' to last used index in com_args
                start writing to next index of com_args
        }else{
                com_args[present_index] = character read from input
        }
}


        After traversal, last index of com_args = NULL
        command = com_args[0];

        return 0;
}


void executeCommand(command){

        if (it is only remaining command){
                char* com;
                char **com_args;
                int status = setSymbols(command, com, com_args)       //to set the
file descriptors on the basis of IO redirection symbols and to split command
into command name and appropriate arguments

                if (status == -1){
                        return
                }
```

```
        int result = execvp(com, com_args);

        if (result == -1){
            return ERROR;
        }

    }

    else{

        set up the pipe fd;

        if (fork() == 0){
            //for command on left
            close(1)
            dup(fd[1])
            close(fd[1])
            close(fd[0])
            executeCommand(command->only left portion) //behaves as only remaining command
        }

        if (fork() == 0){
            //for right command
            close(0)
            dup(fd[0])
            close(fd[1])
            close(fd[0])
            executeCommand(command->only right portion) //For the given command, behaves like only remaining command after second call
        }

        close both ends of pipe to avoid blocking
        Wait for both the children to return
```

```
        }
}
int main(){

        while (1){
                command = readline()        //to read command from STDIN

                if (command == NULL){
                        continue
                }

                if (!strcmp(command,exit)){
                        return 0
                }

                int pid = fork()

                if (pid == 0){
                        executeCommand(command)
                        return 0
                }
                else if (pid > 0){
                        wait()
                }
                else{
                        print("ERROR")
                }
        }
}
```

Submitted by: Vibhu Agrawal, 2018116