Atishay Jain
2018026
4 March 2020

Shell

Assignment 3

Execution of /bin/ls | /usr/bin/sort | /usr/bin/uniq

Working:

Check the current working directory using getcwd( ) and print it at each iteration of while loop. Call read_command( ) to read the command (or wait till command is entered) and then execute the command.

In read_command( ), initialise a char * pointer and allocate it space. Read the input and store it in the char * pointer (called input) and then call remove_space( ) to remove any extra spaces. Next count the number of '|' symbols to determine the number of forks we need to do and check if the command entered is exit and incase it is we return value zero, which causes the while loop to break and thereby terminate the program. Also check if the command is cd as it needs to be explicitly handled using chdir as well as check if the input is an empty string.

Fork the process and ask the parent thread to wait until the child thread finishes execution upon which it returns from the function. If there is a pipe present, make changes in the string that input points to so that it contains only the first command. Create a pipe(fd).

Run a while loop until all the commands have been executed. In each iteration, create a new pipe and change the file descriptors accordingly. Also update input so that it now points to the next command we need to execute. Break out of the loop anytime we enter the if condition, i.e. any of the child threads.

Split the command on the basis of spaces and put it in char ** args. If we encounter exit or cd in any of intermediate commands, then that thread exits. Next we check for any explicit I/O redirection, in the above example there are none so we continue. Call execvp(cmd, args) to execute the command.

Execution

1. Input is read and stored is char * input
2. Count number of pipes, i.e. 2 in this case
3. Create a copy of input and store it in true_input
4. Check if the command is exit, cd or an empty string, it is neither in this case so the relevant code is skipped.
5. Fork the current thread, and the parent thread waits for the child thread to finish execution.
6. The child processes, continues now. It changes input such that input now only contains "/bin/ls" , i.e. the first command.
7. Enter the while loop and create a pipe. It then forks the process. The parent then goes to the else condition, relevant changes in the file descriptors are done, and input is updated to contain the next command, "/usr/bin/sort" in our case. And it proceeds on to the next iteration. The next time input is set to "/usr/bin/sort" and breaks out of the while loop.
8. For each of the the threads, input contains the separate command only and are all out of the while loop now.
9. 'cmd' and 'arg' are initialised and now arg contains all the arguments(including the command) itself in it. Arg is 2-D array and arguments are recognised using spaces. For example arg contains ['/bin/ls', NULL] for the first command.
10. As none of the files require any I/O redirection, all the if conditions to check the relevant I/O redirection are skipped.
11. 'execvp' function call is made with parameters cmd and arg and the command thus executes.
12. As all the commands successfully execute, the printf statements below are never executed as execvp kills the thread.

Pseudocode

```
Main:
        flag = 1;
        while(flag == 1):
                print(current directory + " $ ");
                flag = read_command( );
read_command( ):
        initialise *input <- allocate memory
        input <- Read from console until end of line.
        cnt <- number of pipes in input;
        true_input = deep_copy(input);
        if (true_input == exit):
                return 0;
        if (true_input == cd):
                handle cd command explicitly and return 1;
        if(true_input == empty string):
                return 1;
        int process = fork( );
        if(process != 1): //process is parent
                wait( );
                return 1;
        if(counter > 1): //atleast one pipe
                input <- first command from true_input;
        int fd[2];
        while(cnt - - > 1) :
                pipe(fd);
                process == fork( );
                if( process == 0): //is a child
                        if(counter):
                                close(fd[0])
                                close(1)
                                dup(fd[1])
                                close(fd[1])
                        break;
                else: // not a child
                        close(fd[1])
                        close(0)
                        dup(fd[0])
```

```
                close(fd[0])
                input <- next command from true_input;
        initialise char * cmd and char ** arg;
        arg = input( ) . split( ) // split input on the basis of spaces and store it in arg.
        check if command in exit:
                return 0
        check if command is cd:
                terminate this thread
        check if any I/O redirection is required:
                make the relevant changes
        call execvp(cmd, args); // this executes the command and also kills the thread
        print("Command not fount") //in red colour in case execvp fails.
        exit(0); // kill any thread that reaches this line
```

Assumptions

- No tabs are used while giving input.
- Tab completion isn't required.
- Arrow keys don't work to show prep commands.
- cd doesn't work in pipe