P1_Shell

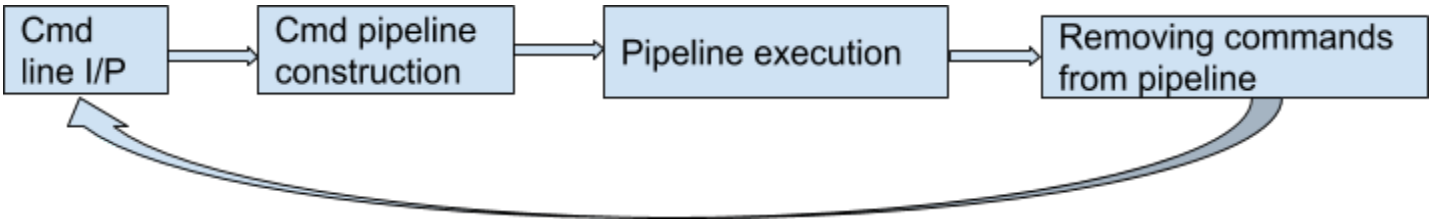
Introduction-

- An interactive shell-like interface that lets the user execute commands as on a regular bash shell.
- P1_Shell supports multiple piping operators i.e. '|' operator. All the commands support input redirection(<), output redirection(>) and output append(>>).
  In addition to this, the shell supports 2 new pipe operators '||' and '|||'.
    - || :-
        - **Usage-**
            - cmd1<space>||<space>cmd2,cmd3
                - Commands after the || operator should be ',' separated with no space between ',' and commands.
        - **Use case-**
            - The output from execution of cmd1 is passed as input to both cmd2 and cmd3. cmd2 and cmd3 execute sequentially.
    - ||| :-
        - **Usage-**
            - cmd1<space>|||<space>cmd2,cmd3,cmd4
                - Commands after the ||| operator should be ',' separated with no space between ',' and commands.
        - **Use case-**
            - The output from execution of cmd1 is passed as input to cmd2, cmd3 and cmd4. cmd2 , cmd3 and cmd4 execute sequentially.

Running the shell-

- Run the command 'make' from the terminal in the directory shell.
- This will compile shell.c, cmd2.c and cmd3.c into an executable shell and execute ./shell, thereby starting the pseudo-shell.
- 'CTRL + \' will prompt to exit the shell.

Command Execution-



1. Command line I/P:-

a. Commands are read till a '\n' is encountered.
b. After successful read, the input is passed to create_pipeline() function to parse the input.

2. Pipeline creation:-

a. create_pipeline()-
   i. This method is used in the absence of '||' and '|||'operators.
   ii. Command pipeline (structure defined as pipeline) is created by parsing the commands separated by '|' in the input.
   iii. Each command is stored in a command structure storing information about command arguments, I/O redirection and next command in the pipeline.

b. create_pipeline2()-
   i. Similar to create_pipeline(), but used when '||' operator is present in the input command.
   ii. Each command is stored in the same command structure as in **a.**

c. create_pipeline3()-
   i. Similar to create_pipeline(), but used when '|||' operator is present in the input command.
   ii. Each command is stored in the same command structure as in a.

3. Pipeline execution:-

All commands are executed using execvp() function.

a. exec_commands()-

i. It is called for executing the command pipelines that do not contain '||' or '|||' operators.
ii. Takes the command pipeline as input and executes each command sequentially.
iii. In the presence of '|', the output of the command is passed as input to the next command in the pipeline. It has been implemented using n-1 pipes where n is the number of commands in the pipeline.

b. exec_commands2()-

i. This is invoked in the presence of '||' operator.
ii. Executes cmd1 and stores its output in a buffer.
iii. This output is passed as input to the cmd2 and cmd3 which are executed one after the other.

c. exec_commands3()-

i. This is invoked in the presence of '|||' operator.
ii. Executes cmd1 and stores its output in a buffer.
iii. This output is passed as input to the cmd2, cmd3 and cmd4 which are executed one after the other.

4. Command removal:-

a. remove_commands()-

i. Takes as input the commands pipeline and resets it before taking the next command from the user.

Command Structure:-

```
struct command{
   int argc;
   char *argv[NUM_OF_ARGS];
   bool ip_redirect;
   bool op_redirect;
   bool op_append;
   char ip_file[LENGTH_OF_ARG];
   char op_file[LENGTH_OF_ARG];
   struct command * next;
};
```

Pipeline structure:-

```
typedef struct{
   struct command * begin;
   struct command * end;
   int cmd_cnt;
}pipeline;
```

Important function signatures:-

```
void create_pipeline(char *input, pipeline *pipeline, int flag);
void exec_commands(pipeline *pipeline);

void create_pipeline2(char *input, pipeline *pipeline, int flag);
void exec_commands2(pipeline *pipeline);

void create_pipeline3(char *input, pipeline *pipeline, int flag);
void exec_commands3(pipeline *pipeline);

void remove_commands(pipeline *pipeline);
```