IMPLEMENTING A SHELL

USP LAB PROJECT-2

(UE15ECS352)

SUSHMA REDDY T (01FB15ECS318)
SWETHA K S N (01FB15ECS321)
TANYA MEHROTRA (01FB15ECS323)

Functions implemented:

Phase 1:

Implement a basic shell that takes input from command line and executes them.

S. No	Function	Description
1	void sst_loop(void);	sst_loop() will be executed until the exit command is entered. It calls sst_read_line function which read input from the shell terminal .
2	char * sst_read_line(void);	Reads input from the terminal. Reads character by character from the terminal until it encounter \n or EOF. The characters read are stored in buffer and it is returned to sst_loop();
3	char ** sst_split_line (char *line, char *s);	It splits the line according to token passed in variable s and it returns array of tokens . While splitting it also checks if the command has | (piping), < or > (redirection) and sets their respective flags accordingly which will be useful further.
4	int sst_execute (char ** args);	It checks if the entered command is cd or help or exit . If the command is one among these it calls their respective functions otherwise calls sst_launch to execute the command.
5	int sst_cd (char ** args);	It calls chdir function to change the current working directory. It passes args[1] as parameter to chdir .
6	int sst_help (char ** args);	It displays some of the built in functions that would help the user.
7	int sst_exit (char **args);	It is called when user enters exit command . It returns 0 to sst_loop which will stop the process.
8	int sst_launch (char **args);	It executes the commands entered by user . Piping and redirection commands are not executed. It forks and calls execvp in child process to execute the command .

		Command name and arguments of the command are passed as parameter to execvp. Parent process waits until child exits . It checks if the command has to be executed in background . If it has to be executed in background then parent process does not wait for the child's exit status.
9	int checkForCommands (char *line);	sst_loop calls this function . Command entered is passed as argument for this function. checkForCommands function checks if the command entered has piping, redirection, history ,alias ..etc. Based on the command entered it calls appropriate function.
10	void cat2Function (char *line);	Redirects stdout to test file. Tokenizes the input to get output file name and opens the file in w+ mode. Stops reading the input as soon as \q is encountered, which marks the end of the text.
11	void checkIFSyntax (char *line)	Function to check basic syntax for IF in Bash. IF in bash starts with “if” and terminates with “fi” having one or none “else” in between. If the input satisfies this syntax, “Valid Input” is printed, else its a “Syntax Error”.

Phase 2:

- Implement history and alias command .
- Implement a shell editor.
- Support piping and redirection.

History command:

History structure :

```
struct node
{
  char* data;
```

```
 char* timestamp;
 struct node *next;
 struct node *prev;
};
```

The command name and the time at which it was executed is stored in the structure.

In sst_loop function each time a command is executed the command line is stored in history structure and current time is given by the function time(). Total 25 commands are stored . If it exceeds 25 commands then first node is removed and a new node is at the end of list.

S. No	Function	Description
1	void historyPrint();	History linked list is traversed . Name of the command and time of execution is printed for atmost 25 command.

Alias command :

Alias Structure :

```
struct alias
{
 char *originalValue;
 char *newValue;
};
```

Alias structure stores original command name and new alias name of the command.

S. No	Function	Description
1	void aliasFunc (char * line);	checkForCommands function checks if the command has alias in it. If the command has alias then aliasFunc is called. Here the line is split with = and ".We get two tokens . The second token contains the original name of the command . The first token is again split with respect to space . Now the second token will

		have the alias name of the command. Original command name and its alias is stored in alias list.
2	char* checkAlias (char *line);	Every time a command is entered this function is called to check if the command is an alias of an another command. If it is an alias of an another command then the original value of it is returned to checkForCommands function.

Pipe and I/O redirection :

S. No	Function	Description
1	void pipeInputOutput (char *line);	Takes care of the input having pipe along with redirection operations for both input and output. It consists of two cases : a) if '>' appears before '<', get the name of the input and output file by tokenizing the input. Fork a new process and create the output file. After duping it, pass the command in parsePipedInput function. b) if '<' appears before '>', get the name of the output and input file by tokenizing the input. Fork a new process and create the output file. After duping it, pass the command in parsePipedInput function.
2	void pipeAndOutput (char * line);	Takes care of the input having pipe and redirection to an output file ('>'). Get the output file name, create it, dup it, and pass the command in parsePipedInput function.
3	void pipeAndInput (char * line);	Takes care of the input having pipe and an redirection from an input file('<'). Get the input file name, open it, dup it, and pass the command in parsePipedInput function.
4	void onlyRedirection (char *line);	Takes care of the input having only redirection operations for both input and output. It consists of two cases : a) if '>' appears before '<', get the name of the input and output file by tokenizing the

		input. Fork a new process and create the output file. After duping it, pass the command in execvp. b) if '<' appears before '>', get the name of the output and input file by tokenizing the input. Fork a new process and create the output file. After duping it, pass the command in execvp.
5	void parsePipedInput (char **token);	Input tokens are taken two at a time and then piped. On the first fork, close the write end and open the read end in the child process. In the parent, fork again and close the read end and open the write end in its child. This way, output of one command becomes input to the other. Repeat the same for all tokens.

Editor(Shell editor):

S. No	Function	Description
1	void editor();	Takes a long command in separate lines, each line terminated by a backslash, and executes it. The end of command is indicated by \q. The command is passed into checkForCommands function for its execution

Phase 3:

Implement two custom functions

1. **ls -z** : Prints files of zero size in the current directory.
2. **ls -itime** : Prints the files and directories in descending order of their inode modification time.

```
struct fileInfo
{
  char *name;
```

```
    time_t mtime;
};
```

It contains filename and its inode modification time.

S. No	Function	Description
1	void printZeroSizeFiles();	Prints files of zero size in the current directory. getcwd() gives current working directory. The current directory is passed to opendir() which returns pointer of DIR type. This pointer is used to traverse the directory. It is passed as argument to readdir() which returns pointer of struct dirent type. Struct dirent contains directory or file name and inode number. The directory name is passed to stat() function which returns inode structure for that file or directory. One of the member of stat structure is st_size which tells the size of file or directory. If the file is regular file and its size is zero then it is printed.
2	void sortWithINodeTime();	Prints the files and directories in descending order of their inode modification time. getcwd() gives current working directory. The current directory is passed to opendir() which returns pointer of DIR type. This pointer is used to traverse the directory. It is passed as argument to readdir() which returns pointer of struct dirent type. Struct dirent contains directory or file name and inode number. The directory name is passed to stat() function which returns inode structure for that file or directory. Stat structure contains a member called st_ctime which contains inode status change time. File name and its inode status change time is stored in fileInfo structure. FileInfo list is sorted according to inode status change time in descending order and list is printed.