OS PROJECT

UNIX SHELL AND HISTORY FEATURE

This project consists of designing a C program to serve as a shell interface that accepts user commands and then executes each command in a separate process. This project can be completed on any Linux, Unix or Mac OS X system.

-BY
Sreeya Raavi

ABSTRACT:

This program serves as a shell interface. The shell interface accepts user commands and then executes a separate process. In response to the user, the command prompt is entered as follows, then the system waits for the user to enter the command and the entered command is executed.

The waiting relationship between parent and child can be two format.
1. The parent process waits for the child process to finish.
2. Parent and child work concurrently. This situation is valid when the "&" character is wrote at the end of the command.

Another special feature of the project is that it supports a special history command. For example, the most recently used 10 commands are listed together with usage number. Eg: osh>history

BACKGROUND CONCEPTS:

You will write a C program to serve as a shell interface that accepts user commands and then executes each command in a separate process. A shell interface gives the user a prompt, after which the next command is entered.When the user enters history, the shell program will display the 10 recent commands.

PROPOSED SOLUTION:

1. Enter command in shell.

2. The command string is read into inputBuffer[] array(using read() from the command line.

3. Command string is formatted and tokenized into arguments using format Commando function and is passed to args[] array.

4. The \t' or '(blank space) in the input indicates the end of the word. Following that word will be a new argument word. Thus a single command line is separated in different argument words, When a \n' is encountered, that is treated as end of command line.

5. NULL character is added in the 'args[]' array to denote the same.The arguments are checked for correctness of the command and is stored in history[10][BUFFER_SIZE] array.

6. In the main function the child process is created using 'fork()'. 'execvp()'

loads the process with the given command if successful otherwise gives respective errors.

7. Parent process enters in to wait (calls wait() if & is not entered this is done through using the 'flag' variable.

FEATURES OF THE PROJECT:

1. Running in background → child process runs in background on adding '&'.

2. History feature → shows recently entered 10 commands.
   case 2: history_count > 10

3. Recent command entered → executes recent command on '!!'.

4. Executing Nth command in history → on entering '!N', where 'N' is the history index.

5. Error on entering invalid command in command line.

6. Error on entering '!!' as the first command.(when there is no command in history)

7. Error on entering '!N' when Nth index is vacant.

8. Error on entering '!N' when Nth index is greater than 10.

SOFTWARE AND HARDWARE REQUIRED:

We used C as our coding language. The operating system we used was MAC OS X. Here we used MAC OS X as our parent class and gedit as our child class. We used the inbuilt terminal my MAC OS.

EXPERIMENTAL RESULTS:

1. ***Executing a command***

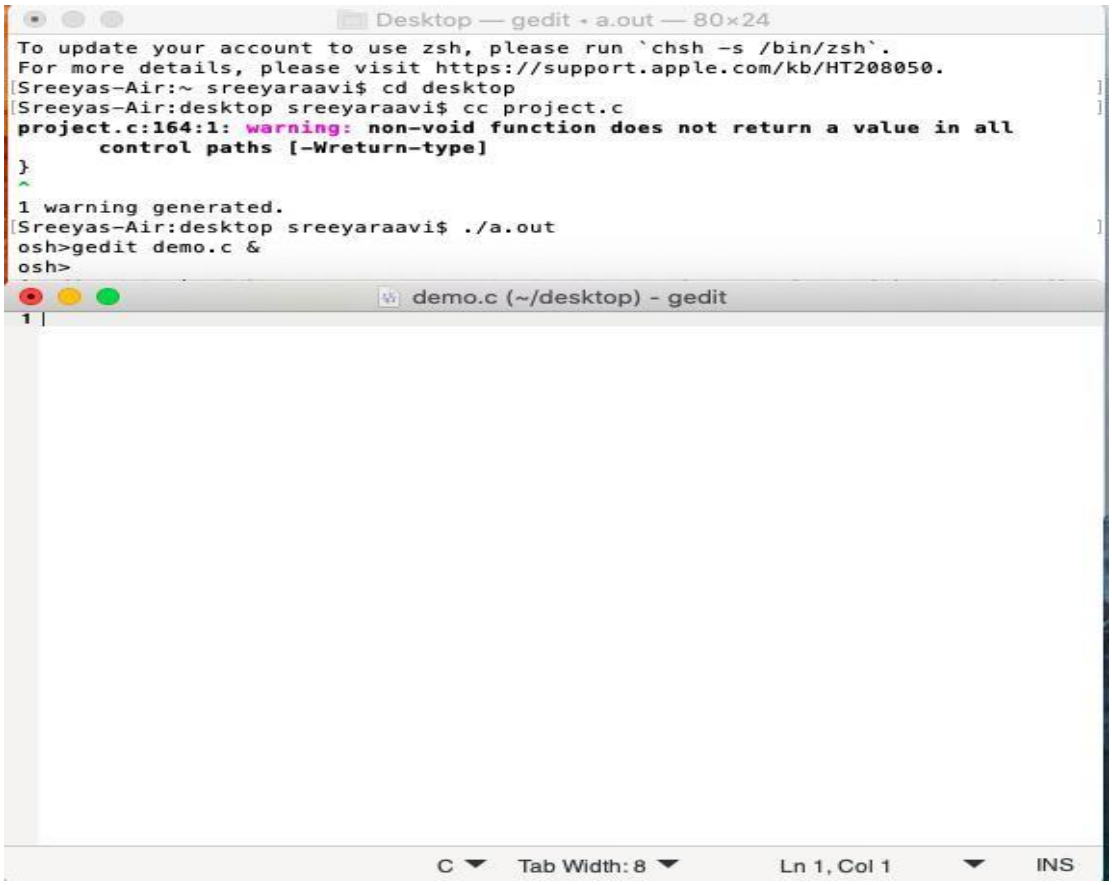
```
Desktop — a.out — 80×24
Last login: Thu Nov 26 11:08:25 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
Sreeyas-Air:~ sreeyaraavi$ cd desktop
Sreeyas-Air:desktop sreeyaraavi$ cc project.c
project.c:164:1: warning: non-void function does not return a value in all
      control paths [-Wreturn-type]
}
^
1 warning generated.
Sreeyas-Air:desktop sreeyaraavi$ ./a.out
osh>pwd
/Users/sreeyaraavi/desktop
osh>
```

2. ***Executing a command without '&'. As shown in the below image parent process will wait for the child process(gedit) to complete.***

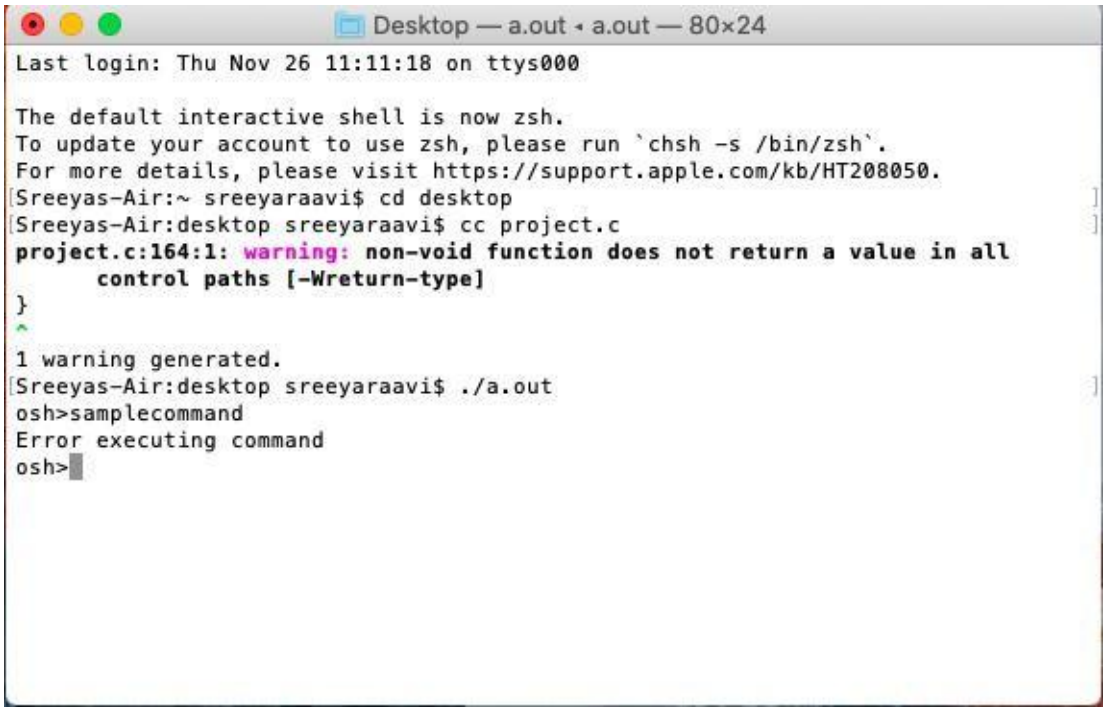
```
Desktop — gedit ◂ a.out — 80×24
~/desktop — gedit project.c        ~/desktop — gedit ◂ a.out
Last login: Thu Nov 26 09:36:31 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
Sreeyas-Air:desktop sreeyaraavi$ cc project.c
project.c:164:1: warning: non-void function does not return a value in all
      control paths [-Wreturn-type]
}
^
1 warning generated.
Sreeyas-Air:desktop sreeyaraavi$ ./a.out
osh>gedit demo.c
```

```
demo.c (~/desktop) - gedit
1
C ▾   Tab Width: 8 ▾   Ln 1, Col 1   ▾   INS
```

3. *Executing a command with '&'. As shown in the below image parent process will not wait for the child process(gedit) to complete.*



4. *Executing a invalid command*



5. *Executing history command to display previous 10 commands.*

```
Desktop — a.out ◂ a.out — 80×24

osh>cal
   November 2020
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30

osh>history
Shell command history:
10.  cal
9.  cal
8.  date
7.  ls
6.  pwd
5.  nedit
4.  ps
3.  ls
2.  mkdir
1.  ls

osh>
```

6. *Executing the recent command by using '!!' in the terminal.*

```
Desktop — a.out ◂ a.out — 80×24
4.  ps
3.  ls
2.  mkdir
1.  ls

osh>cal
   November 2020
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30

osh>!!
   November 2020
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30

osh>
```

7. ***Executing 7th command by using '!7'.***

```
Desktop — a.out ◂ a.out — 80×24
osh>history
Shell command history:
10.  clear
9.  date
8.  date
7.  cal
6.  clear
5.  wrong
4.  pwd
3.  clear
2.  pwd
1.  clear

osh>!7
   November 2020
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30

osh>
```

8. ***Executing 8th command by using '!8'.***

```
Desktop — a.out ◂ a.out — 80×24
Shell command history:
10.  clear
9.  date
8.  date
7.  cal
6.  clear
5.  wrong
4.  pwd
3.  clear
2.  pwd
1.  clear

osh>!7
   November 2020
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30

osh>!8
Wed Nov 25 17:07:35 IST 2020
osh>
```

9. ***Executing 2nd command by using '!2' in the terminal.***

```
Desktop — a.out ◂ a.out — 80×24
9.  date
8.  date
7.  cal
6.  clear
5.  wrong
4.  pwd
3.  clear
2.  pwd
1.  clear

osh>!7
   November 2020
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30

osh>!8
Wed Nov 25 17:07:35 IST 2020
osh>!2
/Users/sreeyaraavi/desktop
osh>
```

10. ***Showing the error when 11th command is accessed(history will have only 10 commands stored).***

```
Desktop — a.out ◂ a.out — 80×24
4.  pwd
3.  clear
2.  pwd
1.  clear

osh>!7
   November 2020
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30

osh>!8
Wed Nov 25 17:07:35 IST 2020
osh>!2
/Users/sreeyaraavi/desktop
osh>!11

No Such Command in the history. Enter <=!9 (buffer size is 10 along with current
 command)
Error executing command
osh>
```

11. Showing the error when user access a command not in history.

```
Desktop — a.out ◂ a.out — 80×24
osh>history
Shell command history:
4.  clear
3.  pwd
2.  ls
1.  clear

osh>!5

No Such Command in the history
Error executing command
osh>
```

Source code:

```
#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>


#define MAX_LINE 80 /* The maximum length of a command */

#define BUFFER_SIZE 50

#define buffer "\n\Shell Command History:\n"


//declarations

char history[10][BUFFER_SIZE]; //history array to store history
```

```
commands
int count = 0;



//function to display the history of commands
void displayHistory()
{

    printf("Shell command history:\n");

    int i;
    int j = 0;
    int histCount = count;

    //loop for iterating through commands
    for (i = 0; i<10;i++)
    {
        //command index
        printf("%d.    ", histCount);
        while (history[i][j] != '\n' && history[i][j] != '\0')
        {
      //printing command
```

```
                printf("%c", history[i][j]);
                j++;
            }
            printf("\n");
            j = 0;
            histCount--;
            if (histCount ==   0)
                break;
        }
        printf("\n");
}




//Fuction to get the command from shell, tokenize it and set the
args parameter

int formatCommand(char inputBuffer[], char *args[],int *flag)
{
        int length; // # of chars in command line
        int i;        // loop index for inputBuffer
        int start;    // index of beginning of next command
```

```
        int ct = 0; // index of where to place the next parameter into
args[]
        int hist;
        //read user input on command line and checking whether
the command is !! or !n

    length = read(STDIN_FILENO, inputBuffer, MAX_LINE);



     start = -1;
     if (length == 0)
            exit(0);    //end of command
     if (length < 0)
     {
            printf("Command not read\n");
            exit(-1);   //terminate
     }

    //examine each character
     for (i=0;i<length;i++)
     {
            switch (inputBuffer[i])
```

```
{
        case ' ':
        case '\t' :                    // to seperate arguments
            if(start != -1)
            {
                args[ct] = &inputBuffer[start];
                ct++;
            }
            inputBuffer[i] = '\0'; // add a null char at the end
            start = -1;
            break;

        case '\n':                     //final char
            if (start != -1)
            {
                args[ct] = &inputBuffer[start];
                ct++;
            }
            inputBuffer[i] = '\0';
            args[ct] = NULL; // no more args
            break;
```

```
                default :
                        if (start == -1)
                                start = i;
                        if (inputBuffer[i] == '&')
                        {
                                *flag    = 1; //this flag is the differentiate
whether the child process is invoked in background
                                inputBuffer[i] = '\0';
                        }
            }
        }

        args[ct] = NULL; //if the input line was > 80

if(strcmp(args[0],"history")==0)
            {
                        if(count>0)
          {

                        displayHistory();
          }
          else
```

```
        {
        printf("\nNo Commands in the history\n");
        }
        return -1;
          }

    else if (args[0][0]-'!' ==0)
    {   int x = args[0][1]- '0';
        int z = args[0][2]- '0';

        if(x>count) //second letter check
        {
        printf("\nNo Such Command in the history\n");
        strcpy(inputBuffer,"Wrong command");
        }
        else if (z!=-48) //third letter check
        {
        printf("\nNo Such Command in the history. Enter <=!9
(buffer size is 10 along with current command)\n");
        strcpy(inputBuffer,"Wrong command");
        }
        else
```

```
        {


            if(x==-15)//Checking for '!!',ascii value of '!' is 33.
            {     strcpy(inputBuffer,history[0]);   // this will be your 10
th(last) command
            }
            else if(x==0) //Checking for '!0'
            {     printf("Enter proper command");
                 strcpy(inputBuffer,"Wrong command");
            }


            else if(x>=1) //Checking for '!n', n >=1
            {
               strcpy(inputBuffer,history[count-x]);


            }


        }
    }
 for (i = 9;i>0; i--) //Moving the history elements one step higher
            strcpy(history[i], history[i-1]);
```

```
    strcpy(history[0],inputBuffer); //Updating the history array with
input buffer
    count++;
   if(count>10)
   { count=10;
   }
}

int main(void)
{
    char  inputBuffer[MAX_LINE]; /* buffer to hold the input
command */
    int flag; // equals 1 if a command is followed by "&"
    char *args[MAX_LINE/2 + 1];/* max arguments */
    int should_run =1;

    pid_t pid,tpid;
    int i;


    while (should_run) //infinite loop for shell prompt
    {
```

```
        flag = 0; //flag =0 by default
        printf("osh>");
        fflush(stdout);
        if(-1!=formatCommand(inputBuffer,args,&flag)) // get next
command
    {
        pid = fork();

          if (pid < 0)//if pid is less than 0, forking fails
          {

                printf("Fork failed.\n");
                exit (1);
          }

              else if (pid == 0)//if pid ==0
          {

                //command not executed
                if (execvp(args[0], args) == -1)
                {
```

```
                    printf("Error executing command\n");
                }
            }

            // if flag == 0, the parent will wait,
        // otherwise returns to the formatCommand() function.
        else
        {
                i++;
            if (flag == 0)
             {
                    i++;
                    wait(NULL);
             }
        }
    }
  }
}
```