OS Project 2 - UNIX Shell and History Feature

Project for Computer Architecture & Operating Systems by Chentao Wu, 2016 Autumn Semester

1. Project Introduction:

This project consists of modifying a C program which serves as a shell interface that accepts user commands and then executes each command in a separate process. This program is composed of three functions: main(), handle-SIGINT() and setup().

The setup() function reads in the user's next command (which can be add to 80 characters), and then parses it into separate tokens that are used to fill the argument vector for the command to be executed.

The handle-SIGINT() function handles the SIGINT signal. This function prints out the message"Caught Control C" and then invokes the e x i t () function to terminate the program.

The main() function presents the prompt COMMAND-> and then invokes setup() , which waits for the user to enter a command. The contents of the command entered by the user is loaded into the args array.

2. Project Environment:

VirtualBox with Linux Ubuntu 16.04.

3. Project Realization:

There're four functions: main(), setup(), handle_SIGINT(), ProcessRCommand().

Part1 main() function

The main() function is the entry of this program. There're three steps taken by main() function.

(1) Allocate the momory to history array which saves the latest command.

(2) Set up the signal handler

(3) loop the following step until we receive <Contorl + D>

1) Use setup() function to read the command.

2) Make judgement of which if the command is "r" or "r+x"

3) Fork a child process using fork()

4) If the background == 1, the parent will wait,otherwise it will invoke the setup() function again.

```
1  int main(void){
2      int i, j;
3      int * res = mmap(NULL, sizeof(i),PROT_READ|PROT_WRITE,
           MAP_SHARED|MAP_ANONYMOUS,-1,0);
4      char inputBuffer[MAX_LINE]; // buffer to hold the command
           entered
5      int background;             // equals 1 if a command is
           followed by '&'
6      char *args[MAX_LINE/2 + 1];    // command line arguments
7      int count;
8      *res = 1;
9
10     for(i = 0; i < 10; ++i)
```

```
11          for(j = 0; j < 10; ++j)
12              history[i][j] = (char*)malloc(80*sizeof(char));    //
                    Allocate the memory
13
14      strcpy(buffer,"Caught Control C\n");
15      if (signal(SIGINT, handle_SIGINT) == SIG_ERR) //Error
16          printf("ERROR!\n");
17
18      while (1){                  // Program terminates normally inside
            setup
19          *res = 1;
20          background = 0;
21          printf("COMMAND-> ");
22          fflush(0);
23          setup(inputBuffer, args, &background);        // get next
                command
24          i = 0;
25          if (args[0] == NULL) continue;
26          if (args[0] != NULL && strcmp(args[0],"r") != 0){
27              while(args[i] != NULL){
28                  strcpy(history[nextPosition][i], args[i]);
29                  ++i;
30              }
31              CommandLenth[nextPosition] = i;
32              nextPosition = (nextPosition + 1) % 10;
33          }
34
35          //Deal with the "r" command
36          if (strcmp(args[0],"r") == 0){
37              if (args[1]==NULL){       //"r" command only
38                  i = (nextPosition + 9) % 10;
39                  for(j=0; j<CommandLenth[i]; ++j)
40                      strcpy(history[nextPosition][j], history[i][j
                            ]);
41
42                  CommandLenth[nextPosition] = j;
43                  nextPosition = (nextPosition + 1) % 10;
44                  flag =1;
45              }
46              else{                       //"r x" command
47                  i = nextPosition;
48                  count = 10;
49                  while(count--){
50                      i = (i + 9) % 10;
51                      if (strncmp(args[1], history[i][0], 1) == 0){
52                          for(j=0; j<CommandLenth[i]; ++j)
53                              strcpy(history[nextPosition][j],
                                    history[i][j]);
54
55                          CommandLenth[nextPosition] = j;
```

```
                        nextPosition = (nextPosition + 1) % 10;
                        flag=1;
                        break;
                    }
                }
                if(count == -1){
                    printf("No such instruction!\n");
                       continue;
                }
            }
        }
        pid_t pid = fork();
    //pid_t pid;
        if(pid < 0){
            printf("Fork failed.\n");
        }
        else if(pid == 0){                      //Child Process
            if(strcmp(args[0],"r") == 0){
                ProcessRCommand(args);
                exit(0);
            }
            else{
                execvp(args[0],args);
                int s = (nextPosition -1) % 10;
                *res = 2;
                printf("Wrong Instruction!!!\n");
                exit(0);
            }
        }
        else{                                   //Parent Process
            if(background == 1)
                continue;
            pid_t rpid;
            do{
                rpid = wait(NULL);
                if(*res == 2){
                    CommandLenth[nextPosition-1] = 0;
                    nextPosition = nextPosition - 1;
                    if (nextPosition == -1) nextPosition = 9;
                    int idx;
                    for (idx = 0; idx < 10; ++idx)
                        strcpy(history[nextPosition][idx], "\0");

                    *res = 1;
                }
            } while (rpid != pid);
        }
    }
}
```

Part2 setup() function

The setup() function reads in the next command line, separating it into distinct tokens using whitespace as delimiters.

```
void setup(char inputBuffer[], char *args[], int *background){
    int length, i, start, ct;
    ct = 0;
    length = read(STDIN_FILENO, inputBuffer, MAX_LINE);
    start = -1;
    if (length == 0)          // If Ctrl+D is entered, end the user
        command stream
        exit(0);
    if (inputBuffer[0]=='e' && inputBuffer[1]=='x' && inputBuffer
       [2]=='i' && inputBuffer[3]=='t' &&
    inputBuffer[4]=='(' && inputBuffer[5]==')' && inputBuffer
       [6]=='\n')
    exit(0);
    if (length < 0){
        perror("Failed in reading the command");
        exit(-1);
    }

    // examine every character in the inputBuffer
    for (i = 0; i < length; i++){
        switch (inputBuffer[i]){
            case ' ':
            case '\t':                  // argument separators
                if(start != -1){
                    args[ct] = &inputBuffer[start];     // set up
                       pointer
                    ct++;
                }
                inputBuffer[i] = '\0'; // add a null char; make a
                    C string
                start = -1;
                break;

            case '\n':                  // should be the final
               char examined
                if (start != -1){
                    args[ct] = &inputBuffer[start];
                    ct++;
                }
                inputBuffer[i] = '\0';
                args[ct] = NULL; //no more arguments to this
                   command
                break;

            case '&':
```

```
                    *background = 1;
                    inputBuffer[i] = '\0';
                    break;

                default:                // some other character
                    if (start == -1)
                        start = i;
            }
        }
        args[ct] = NULL; // just in case the input line was > 80
}
```

Part 3 handle_SIGINT() function

The handle_SIGINT() function lets the program catch the "control + C" to print the latest 10 command on the shell.

```
void handle_SIGINT()
{
    write(STDOUT_FILENO, buffer, strlen(buffer));
    printf("Command history:\n");
    int i = nextPosition;
    int j;
    int counter = 10;

    while(counter--){
        printf("%d\t",counter+1);
        for(j = 0; j < CommandLenth[i]; ++j)
            printf("%s ", history[i][j]);

        printf("\n");
        i = (i + 1) % 10;
    }
    printf("COMMAND->");
    fflush(0);
    return;
}
```

Part 4 ProcessRCommand() function

The ProcessRCommand() function deal with the command "r" and "r+x". The "r" command recalls the latest command. The "r+x" command runs any of the previous 10 commands where 'x' is the first letter of that command. And they will be echoed on the user's screen and the command is also placed in the history buffer as the next command.

```
void ProcessRCommand(char *args[])
{
    int i, j, count = 10;
    char *newargs[MAX_LINE/2 + 1];      //A copy of array history
    for(i = 0; i < MAX_LINE/2 + 1; ++i)
        newargs[i] = (char*)malloc((MAX_LINE/2+1)*sizeof(char));
```

```
            //Allocate the memory
    if (flag == 1)
        nextPosition = (nextPosition - 1) % 10;

    history[nextPosition][0] = '\0';
    if (args[1] == NULL){                                   //command "r
       "
        i = (nextPosition + 9) % 10;                    //points to the
            command to be executed
        for(j = 0; j < CommandLenth[i]; ++j)
            strcpy(newargs[j], history[i][j]);

        newargs[j] = NULL;
        execvp(newargs[0], newargs);
        printf("Wrong Instruction!\n");
        exit(0);
    }
    else{                   //Command "r x"
        i = nextPosition;
        while(count--){
            //Find the nearest command begin with "x"
            i = (i + 9) % 10;
            if (strncmp(args[1], history[i][0], 1) == 0){
                printf("%s\t%s\n", args[1], history[i][0]);
                for(j = 0; j < CommandLenth[i]; ++j)
                    strcpy(newargs[j], history[i][j]);

                newargs[j] = NULL;
                execvp(newargs[0], newargs);
                printf("Wrong Instruction!\n");
                exit(0);
            }
        }
        printf("No such instruction!\n"); //There is no command
           start with "x"
    }
}
```

4. Project Result:

1. Some simple command

```
xiaolanchong@xiaolanchong-VirtualBox:~$ ./shell.out
COMMAND-> ls
Desktop    Downloads         Music     Public   shell.out  Videos
Documents  examples.desktop  Pictures  shell.c  Templates
COMMAND-> cal
      十一月 2016
日 一 二 三 四 五 六
       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

COMMAND-> date
2016年 11月 10日 星期四 19:47:28 CST
COMMAND->
```

图 1: Result: simple command

2. Control + C

```
COMMAND-> cal
      十一月 2016
日 一 二 三 四 五 六
       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

COMMAND-> date
2016年 11月 10日 星期四 19:47:28 CST
COMMAND-> ^CCaught Control C
LibreOffice Calc :
10
9
8
7
6
5
4
3       ls
2       cal
1       date
COMMAND->
```

图 2: Result: Control + C

3. r Type

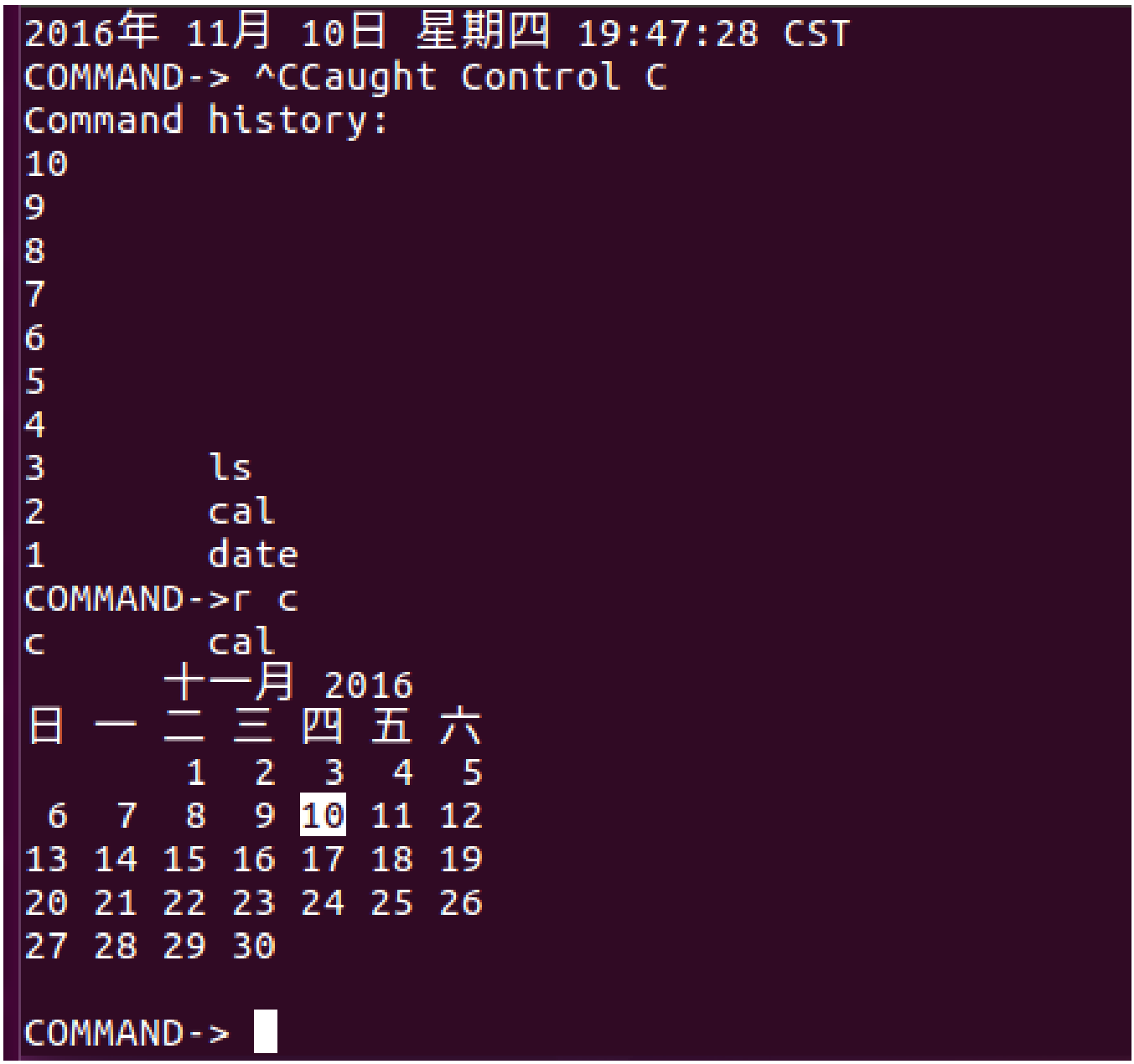

图 3: Result: r + "x"

```
 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

COMMAND-> ls
Desktop    Downloads         Music     Public   shell.out  Videos
Documents  examples.desktop  Pictures  shell.c  Templates
COMMAND-> r
Desktop    Downloads         Music     Public   shell.out  Videos
Documents  examples.desktop  Pictures  shell.c  Templates
COMMAND-> ^CCaught Control C
Command history:
10
9
8
7
6       ls
5       cal
4       date
3       cal
2       ls
1       ls
COMMAND->
```

图 4: Result: r

4. & run concurrently

```
27 28 29 30

COMMAND-> ls
Desktop    Downloads         Music     Public   shell.out  Videos
Documents  examples.desktop  Pictures  shell.c  Templates
COMMAND-> r
Desktop    Downloads         Music     Public   shell.out  Videos
Documents  examples.desktop  Pictures  shell.c  Templates
COMMAND-> ^CCaught Control C
Command history:
10
9
8
7
6       ls
5       cal
4       date
3       cal
2       ls
1       ls
COMMAND->ls &
COMMAND-> Desktop    Downloads       Music     Public   shell.out  Videos
Documents  examples.desktop  Pictures  shell.c  Templates
```

图 5: Result: & run concurrently

5. exit()

```
Documents  examples.desktop  Pictures  shell.c  Templates
COMMAND-> r
Desktop    Downloads         Music     Public   shell.out  Videos
Documents  examples.desktop  Pictures  shell.c  Templates
COMMAND-> ^CCaught Control C
Command history:
10
9
8
7
6       ls
5       cal
4       date
3       cal
2       ls
1       ls
COMMAND->ls &
COMMAND-> Desktop    Downloads       Music     Public   shell.out  Videos
Documents  examples.desktop  Pictures  shell.c  Templates

COMMAND-> exit
Wrong Instruction!!!
COMMAND-> exit()
xiaolanchong@xiaolanchong-VirtualBox:~$
```

图 6: Result: exit()

6. Wrong Instruction

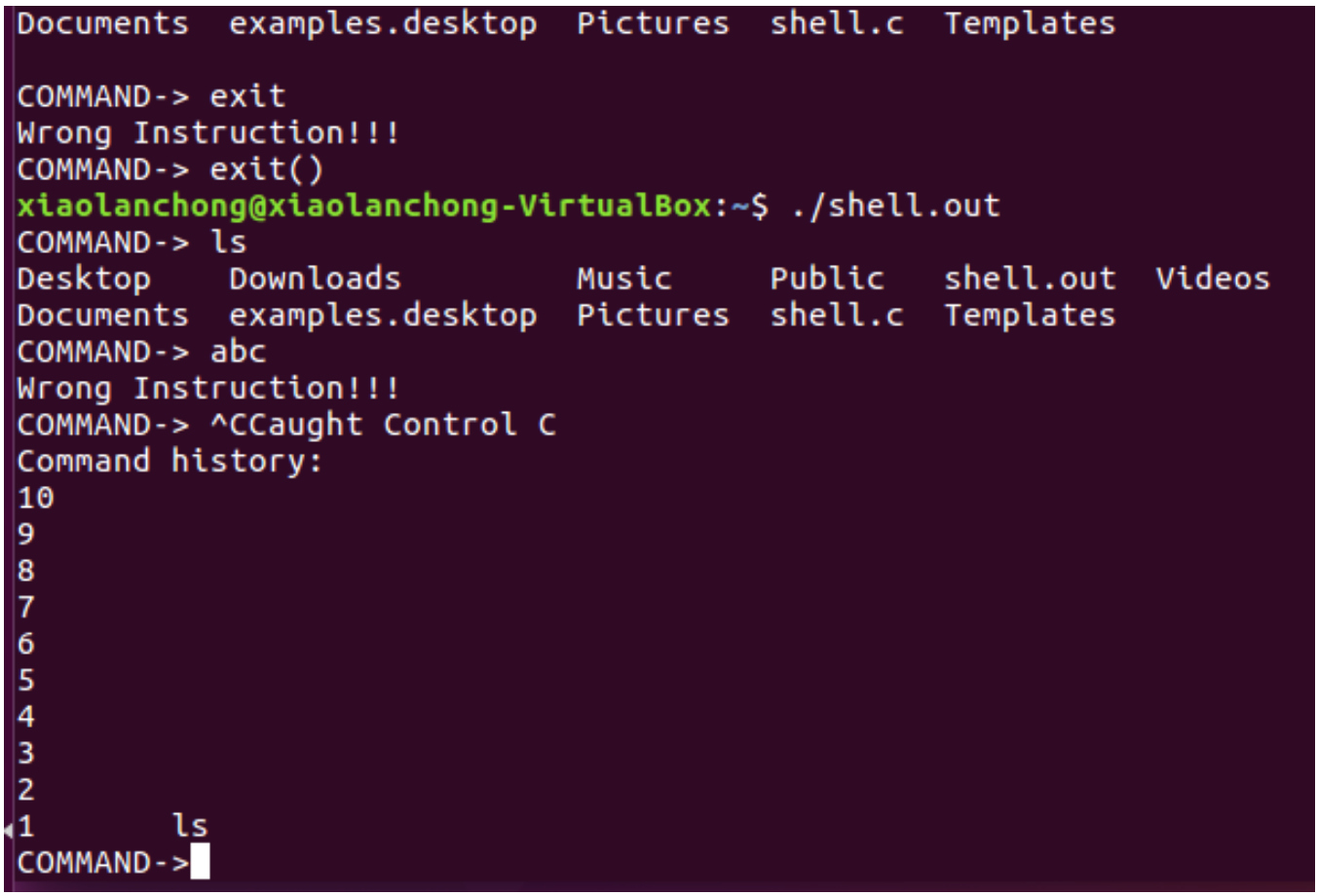

图 7: Result: Wrong Instruction

The problem I have met

I have met many problems,such as:

(1)When I entered Control + C , there will be two history feature on the screen, which means the handle_SIGINT() function was run twice, after I checked the code I found that it was the child process and parent process both catching the signal. And if the process is failed, the child process should exist, the problem was solved quickly.

(2)At first I did't achieve the function of communication between the child process and parent process, and parent process can't know if the command of child is true or false, and I used shared memory to solve it.

(3)The cd and exit command can not be executed, after google I found it is beacuse that the execvp() function don't have these two commands, and if we want to solve it we could achive it by adding the extra discrimination by strcpy() at the outside, but it will add to extra expense.

Harvest

Through this project I learned a lot knowledge about shell and how to executed a command through the shell. And the communication between the child process and parent process was really a problem when doing this project. Fortunately at last I solved it. The operating system is a very fun thing to manipulate, and solving problems makes me very proudable.