Simple Shell Documentation

Li Shitao

May 15, 2016

1 Description

This program is a simplified shell command interpreter which is developed and tested on Ubuntu 14.04. The source codes of the program is avaliable on `https://github.com/list12356/simpleshell.git`

2 Main Function

The main function of this program is shown as following:

- resolve the internal command
- resolve the external command
- support the , including both > and >>
- support the command line pipe

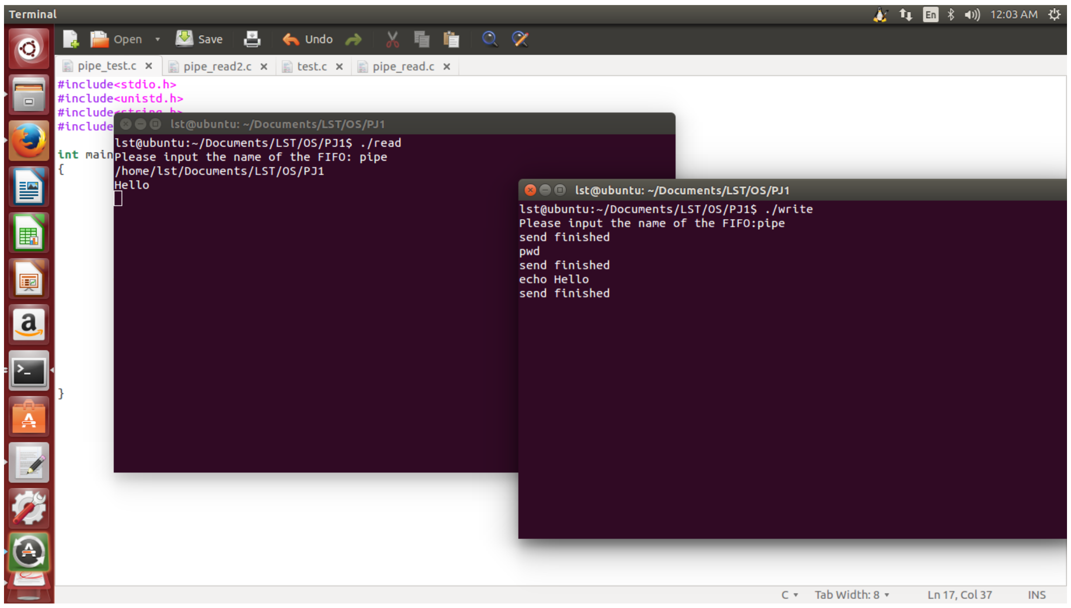
3 User Guide

3.1 Input the command

This program include two exucetable files, the write and read. Before using the program, please make sure that you have a named pipe(FIFO file) avaliable and can open the write program and read program in two different terminal respectively. First, you should type the name of the FIFO file in both program and make sure that it is valid. And then you can type the commands in the write program so that the result will show on the read program. A simple example is showed in Figure **??**

3.2 Output redirection

The program can resolve two redirection symbols, > and >>. For the symbol >, the program will first detect whether there already exists a file which match the name after the symbol. If there already exists a such file, the program will



```
lst@ubuntu:~/Documents/LST/OS/PJ1$ ./read
Please input the name of the FIFO: pipe
/home/lst/Documents/LST/OS/PJ1
Hello
```

```
lst@ubuntu:~/Documents/LST/OS/PJ1$ ./write
Please input the name of the FIFO:pipe
send finished
pwd
send finished
echo Hello
send finished
```

Figure 1: Simple Example

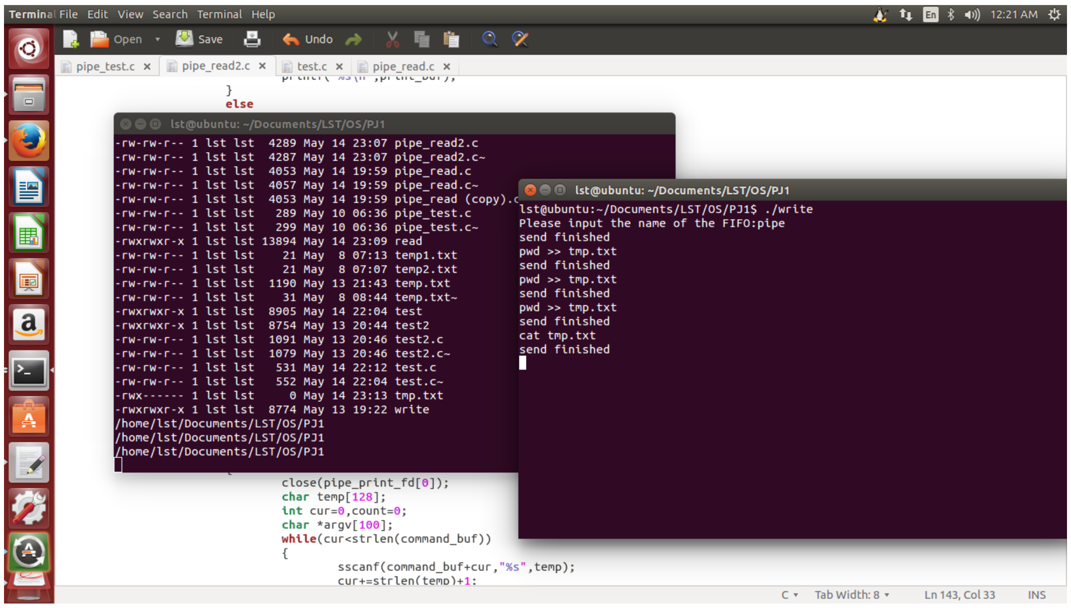
first delete it and create a new one to output the result. If there is no such file, the program wil create a new one. For the symbol >>, the program will output the result on the tail of the file, if it exists, or create a new one. An example is shown as below. We first use three “pwd >> tmp.txt” command, and use “cat tmp.txt” we can see from Figure **??**that the results are added on the tail of the file “tmp.txt”. And we use the “ls > tmp.txt”, and repeat the progress , we can see from Figure **??**that the tmp.txt file is rewritten by the program.

3.3 Command line pipe

The commandline pipe is also suported in the program. And the usage is the same as the usual case in Linux. I use several command as examples as shown in Figure**??**.

4 Structure of the Program

The basic and structure of th Program can be shown in the pesudo code of Algorithm**??**



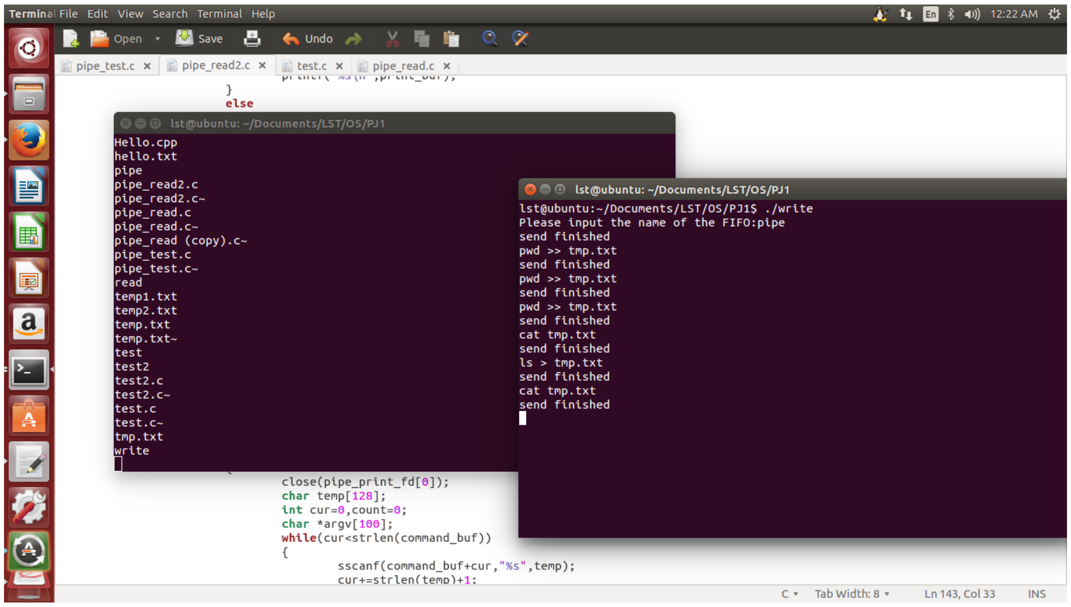
```
}
else
```

```
lst@ubuntu: ~/Documents/LST/OS/PJ1
-rw-rw-r-- 1 lst lst  4289 May 14 23:07 pipe_read2.c
-rw-rw-r-- 1 lst lst  4287 May 14 23:07 pipe_read2.c~
-rw-rw-r-- 1 lst lst  4053 May 14 19:59 pipe_read.c
-rw-rw-r-- 1 lst lst  4057 May 14 19:59 pipe_read.c~
-rw-rw-r-- 1 lst lst  4053 May 14 19:59 pipe_read (copy).c
-rw-rw-r-- 1 lst lst   289 May 10 06:36 pipe_test.c
-rw-rw-r-- 1 lst lst   299 May 10 06:36 pipe_test.c~
-rwxrwxr-x 1 lst lst 13894 May 14 23:09 read
-rw-rw-r-- 1 lst lst    21 May  8 07:13 temp1.txt
-rw-rw-r-- 1 lst lst    21 May  8 07:07 temp2.txt
-rw-rw-r-- 1 lst lst  1190 May 13 21:43 temp.txt
-rw-rw-r-- 1 lst lst    31 May  8 08:44 temp.txt~
-rwxrwxr-x 1 lst lst  8905 May 14 22:04 test
-rwxrwxr-x 1 lst lst  8754 May 13 20:44 test2
-rw-rw-r-- 1 lst lst  1091 May 13 20:46 test2.c
-rw-rw-r-- 1 lst lst  1079 May 13 20:46 test2.c~
-rw-rw-r-- 1 lst lst   531 May 14 22:12 test.c
-rw-rw-r-- 1 lst lst   552 May 14 22:04 test.c~
-rwx------ 1 lst lst     0 May 14 23:13 tmp.txt
-rwxrwxr-x 1 lst lst  8774 May 13 19:22 write
/home/lst/Documents/LST/OS/PJ1
/home/lst/Documents/LST/OS/PJ1
/home/lst/Documents/LST/OS/PJ1
```

```
lst@ubuntu: ~/Documents/LST/OS/PJ1
lst@ubuntu:~/Documents/LST/OS/PJ1$ ./write
Please input the name of the FIFO:pipe
send finished
pwd >> tmp.txt
send finished
pwd >> tmp.txt
send finished
pwd >> tmp.txt
send finished
cat tmp.txt
send finished
```

```
close(pipe_print_fd[0]);
char temp[128];
int cur=0,count=0;
char *argv[100];
while(cur<strlen(command_buf))
{
        sscanf(command_buf+cur,"%s",temp);
        cur+=strlen(temp)+1;
```

Figure 2: Output Redirection



```
}
else
```

```
lst@ubuntu: ~/Documents/LST/OS/PJ1
Hello.cpp
hello.txt
pipe
pipe_read2.c
pipe_read2.c~
pipe_read.c
pipe_read.c~
pipe_read (copy).c~
pipe_test.c
pipe_test.c~
read
temp1.txt
temp2.txt
temp.txt
temp.txt~
test
test2
test2.c
test2.c~
test.c
test.c~
tmp.txt
write
```

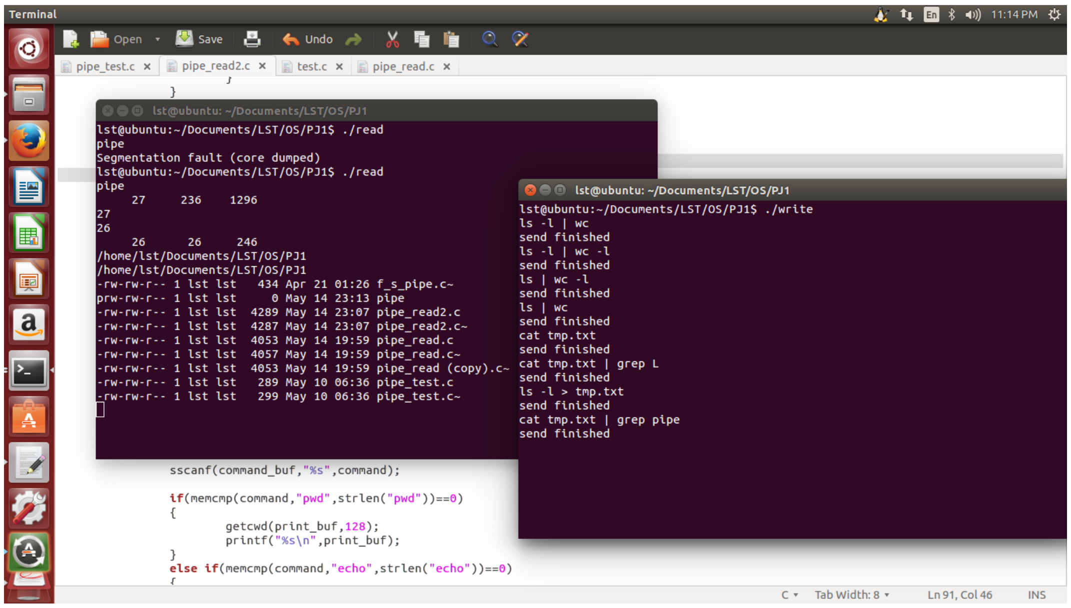
```
lst@ubuntu: ~/Documents/LST/OS/PJ1
lst@ubuntu:~/Documents/LST/OS/PJ1$ ./write
Please input the name of the FIFO:pipe
send finished
pwd >> tmp.txt
send finished
pwd >> tmp.txt
send finished
pwd >> tmp.txt
send finished
cat tmp.txt
send finished
ls > tmp.txt
send finished
cat tmp.txt
send finished
```

```
close(pipe_print_fd[0]);
char temp[128];
int cur=0,count=0;
char *argv[100];
while(cur<strlen(command_buf))
{
        sscanf(command_buf+cur,"%s",temp);
        cur+=strlen(temp)+1;
```

Figure 3: Output Redirection

Algorithm 1 Simple Shell Program

```
while true do
    read from the FIFO to fifobuf
    search for $'<', '>', '>>', '|'$
    if there is a redirection then
        dup2(outputfile,stdout)
    end if
    if there is a command line pipe then
        devide the command and make the flag
    end if
    if Internal Command then
        execute the internal command
    else
        create a child process
        if is child process then
            if there is a command line pipe then
                create a pipe
                create a grandchild process
                if is grandchild process then
                    redirect the stdout to the pipe
                    execute the command before $'|'$
                else
                    wait
                    redirect the stdin to the the pipe
                end if
            end if
            execute the command
        else
            wait
        end if
    end if
    recover the stdin and stdout
end while
```



```
lst@ubuntu:~/Documents/LST/OS/PJ1$ ./read
pipe
Segmentation fault (core dumped)
lst@ubuntu:~/Documents/LST/OS/PJ1$ ./read
pipe
     27     236    1296
27
26
     26      26     246
/home/lst/Documents/LST/OS/PJ1
/home/lst/Documents/LST/OS/PJ1
-rw-rw-r-- 1 lst lst   434 Apr 21 01:26 f_s_pipe.c~
prw-rw-r-- 1 lst lst     0 May 14 23:13 pipe
-rw-rw-r-- 1 lst lst  4289 May 14 23:07 pipe_read2.c
-rw-rw-r-- 1 lst lst  4287 May 14 23:07 pipe_read2.c~
-rw-rw-r-- 1 lst lst  4053 May 14 19:59 pipe_read.c
-rw-rw-r-- 1 lst lst  4057 May 14 19:59 pipe_read.c~
-rw-rw-r-- 1 lst lst  4053 May 14 19:59 pipe_read (copy).c~
-rw-rw-r-- 1 lst lst   289 May 10 06:36 pipe_test.c
-rw-rw-r-- 1 lst lst   299 May 10 06:36 pipe_test.c~
```

```
lst@ubuntu:~/Documents/LST/OS/PJ1$ ./write
ls -l | wc
send finished
ls -l | wc -l
send finished
ls | wc -l
send finished
ls | wc
send finished
cat tmp.txt
send finished
cat tmp.txt | grep L
send finished
ls -l > tmp.txt
send finished
cat tmp.txt | grep pipe
send finished
```

```
        sscanf(command_buf,"%s",command);

        if(memcmp(command,"pwd",strlen("pwd"))==0)
        {
                getcwd(print_buf,128);
                printf("%s\n",print_buf);
        }
        else if(memcmp(command,"echo",strlen("echo"))==0)
```

Figure 4: Command Line Pipe

5 Conclusion and Future Plan

5.1 About the External Commands

When resolving the externa commands, I used the fork() system call to create a child process, and the child process execute the command. Probably a more right way to use a pipe and redirect the stdout of the child process to the write port of the pipe and let the father process to print the result. However, this method has a fatal problem, there are some commands like "vim" that require that the output should be a terminal, not a pipe. So I aborted this method in the final version. However, you can still see it in the annotation in the source codes.

5.2 About the Redirection of the Input

The input redirection is also considered in the program but I did not do enough test.

5.3 Future Plan

To be determined.