A Simplicified Shell For UNIX

——Fudan OS Lab

13307130255 林楚铭

Introduction

In the lab, we will design a simplified shell interface for UNIX 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.

The project has some main function:

1. Fork a child and execute user commands such as ls, ps.
2. If user commands include an ampersand (&), the parent and child processes will run concurrently.
3. Add exit and cd commands.
4. Log history commands and the process can list the history commands and execute recent commands.

Split String

If we want to execute user commands, such as “ls -l”, we should use the system calling that execvp() function.

execvp(char *command, char *params[]);

In the case for “ls -l”, we should split the string to “ls” and “-l”.

In my code, I implement splitStirng() function to process the user input.

splitString(char *str, char * args[],int& arg_num,bool& amp);

str is the user input.

args[] is the result of splitting string.

arg_num is the number of splitting string.

amp means the string is whether ampersand (&).

History Feature

In the program, it use a array to save the history commands.

The array : char * history[];

If the commands except !!, !x, cd, exit and undefined command, it will log in the history. And it use the malloc() function to store the string dynamically.

If the user input ‘!’ command such as “!!” or “!2”, it will fetch the history array and displace the string to the recent command.

If we have not the recent command, the program will output “No such command in history.”

CD Command and Others

In the program, we implement the cd command.

We use the system calling that chdir() function.

int chdir(const char *path);

If the command contains “cd” such as “cd OS/shell”, we will spilt the string to “cd” and “OS/shell” and execute the chdir function that chdir(“OS/shell”);

Others:

1. If user input exit command, the program will exit.
2. If the amp is false, the parent process will execute the function wait() and wait the child to exit. Otherwise the parent will not wait.
3. Execute fork() and create child process and the child process will invoke execvp(). It’s very normal and ordinary so we don’t need to detail it.

Code

```
#include <stdio.h>
#include <unistd.h>
#include <malloc.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#define MAX_LINE 80


void splitString(char *str, char * args[],int& arg_num,bool& amp)
{
   int len = strlen(str);
   int i,l,r;
   l = 0;
   arg_num = 0;
   for(i=0;i<len-1;i++){
      if(str[i+1] == ' '){
         args[arg_num] = (char*)malloc(MAX_LINE);
         strncpy(args[arg_num],str+l,i-l+1);
         arg_num++;
         l = i+2;
      }
   }

   args[arg_num] = (char*)malloc(MAX_LINE);
   strncpy(args[arg_num],str+l,len-l);
   arg_num++;

   //args[arg_num] = (char*)malloc(MAX_LINE);
   args[arg_num] = NULL;

   amp = false;
   int last = arg_num - 1;
   if(strcmp(args[last],"&") == 0){
```

```
        amp = true;
        free(args[last]);
        args[last] = NULL;
        arg_num--;
    }
}

int main()
{
    char *args[MAX_LINE/2 +1];
    char * history[MAX_LINE];
    int should_run = 1;
    int count_cmd = 0;
    while(should_run){

        printf("osh>");
        fflush(stdout);
        //input process
        char str[MAX_LINE];
        gets(str);

        //process !
        if(str[0] == '!' && strlen(str) > 1){
            if(str[1] == '!'){
                if(count_cmd==0){
                    printf("No such command in history.\n");
                    continue;
                }else{
                    strcpy(str,history[count_cmd-1]);
                }
            }else{
                int num = 0;
                int len = strlen(str);
                for(int i=1;i<len;i++){
                    num = num*10 + str[i] - '0';
```

```
        }
        if(num>count_cmd){
            printf("No such command in history.\n");
            continue;
        }else{
            strcpy(str,history[num-1]);
        }
    }
}
//end !

int arg_num; bool amp;
splitString(str,args,arg_num,amp);

//process 'cd'
if(strcmp("cd",args[0])==0){
    if(arg_num==1) continue;
    char path[MAX_LINE];
    strncpy(path,str+3,strlen(str));
    chdir(path);
    continue;
}
//end 'cd'

history[count_cmd] = (char*)malloc(MAX_LINE);
strcpy(history[count_cmd],str);
count_cmd++;

if(strcmp(args[0],"exit") == 0) return 0;

//fork

pid_t pid = fork();
if(pid < 0){
    printf("fork failed!\n");
```

```
        }else if(pid == 0){
            if(strcmp(args[0],"history") == 0){
                int pcount=count_cmd,i=0;
                while(pcount&&i<10){
                    pcount--;
                    i++;
                    printf("%d %s\n",pcount+1,history[pcount]);
                }
            }else{
                execvp(args[0],args);
            }
        }else{
            if(!amp) wait(NULL);
        }

    }
    return 0;
}
```