

## [README]

真正的執行過程請見影片

執行結果中的 new.c 是舊檔名，後來才把它改成 HW1\_4108056007.c

## [前置作業]

先安裝 readline

```
cs4108056007@cs4108056007-VirtualBox:~/HW1$ sudo apt-get install libreadline-dev
[sudo] password for cs4108056007:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-hwe-5.4-headers-5.4.0-42 linux-hwe-5.4-headers-5.4.0-66 linux-hwe-5.4-headers-5.4.0-67 linux-hwe-5.4-headers-5.4.0-70
  linux-hwe-5.4-headers-5.4.0-72 linux-hwe-5.4-headers-5.4.0-73 linux-hwe-5.4-headers-5.4.0-74 linux-hwe-5.4-headers-5.4.0-84
  linux-hwe-5.4-headers-5.4.0-86 linux-hwe-5.4-headers-5.4.0-87
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  libtinfo-dev
Suggested packages:
  readline-doc
The following NEW packages will be installed:
  libreadline-dev libtinfo-dev
0 upgraded, 2 newly installed, 0 to remove and 15 not upgraded.
Need to get 214 kB of archives.
After this operation, 1134 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

## [基本功能]

1. 印出 prompt，其中 BEGIN 和 CLOSE 是定義 prompt 的顏色(我是設綠色)

```
////////////////////////////////prompt////////////////////////////////
getcwd(curr_path, sizeof(curr_path));
printf(BEGIN(49, 32)"%s$ "CLOSE, curr_path);

#define BEGIN(x, y) "\033["#x";"#y"m" //x: string background, y: string font color (for prompt)
#define CLOSE "\033[0m"
```

結果:

```
cs4108056007@cs4108056007-VirtualBox:~/HW1$ ./new
/home/cs4108056007/HW1$
```

2. 讀第一個字元，此時使用 getch()，讓輸入的字先不顯示在鍵盤上，若:

- a. 第一個字元為 ESC，代表是輸入上/下鍵，所以要再把剩下兩個字元吃掉，並呼叫 display\_up()/display\_down() 去印出，history 的 command

```
if(buff[0]=='\033'){ //arrow key pressed
    getch();
    switch(getch()){
        case 'A': //up arrow
            display_up();
            pre_is_arrow=true;
            continue;
            break;
        case 'B': //down arrow
            display_down();
            pre_is_arrow=true;
            continue;
            break;
    }
}
```

- b. 第一個字元為 ENTER，去執行剛剛按上下鍵得到的 history command

```

else if(buff[0]==10){ //only press enter key: execute the command after select from up/down key
    if(pre_is_arrow==true) {
        strcpy(buff, history[curr_index]);
    }
    else {
        printf("\n");
        continue; //if previous command is not from arrow, do nothing
    }
}

```

- c. 第一個字元非前兩者(代表是一般的指令)，用 `getline` 讀入剩下的字元，並合併到 `buff` string

```

else{ //normal command input(not enter nor arrow)
    temp_input=(char*)malloc(MAX_BUFFER*sizeof(char));
    printf("%c", buff[0]); //print the char that haven't been printed bc of getch()
    temp_input= getline(ps); //read the remaining command
    strcat(buff, temp_input); //strcat the first character and the remaining command

    pre_is_arrow=false;
}

```

3. 把要執行的指令加到 `history` stack (因為是記錄歷史，所以 `stack` 只會 `push`)

```

//////////add buff to history stack//////////
push(buff);

void push(char* new_command){
    if(stack_top==STACK_SIZE-1) printf("Stack is full!\n");
    else{
        int i;
        stack_top++;
        for (i=0; i<MAX_BUFFER && new_command[i]!='\0'; i++){
            history[stack_top][i]=new_command[i];
        }
        curr_index=stack_top+1;
    }
}

```

4. 將 `buff` 裡的指令和參數分開

```

//////////separate the command and the connected parameters(par1, par2, par3)//////////
char command[MAX_BUFFER]="\0", par1[MAX_BUFFER]="\0", par2[50], par3[50];
int index=0, outfd, outfd_back;

for(i=0; i<buff_size && buff[i]!='\0' && buff[i]!=' '; i++){
    command[i]=buff[i]; //get command
}
command[i]='\0'; i++;

for (; i<buff_size && buff[i]!=' ' && buff[i]!='\0'; i++){
    par1[index]=buff[i]; //get parameter1
    index++;
}
par1[index]='\0';
par2[0]='\0';

if(buff[i]!='\0'){
    i++;
    for (index=0; i<buff_size && buff[i]!=' ' && buff[i]!='\0'; i++){
        par2[index]=buff[i]; //get parameter2
        index++;
    }
    par2[index]='\0';
}

par3[0]='\0';
if(buff[i]!='\0'){
    i++;
    for (index=0; i<buff_size && buff[i]!=' ' && buff[i]!='\0'; i++){
        par3[index]=buff[i]; //get parameter3
        index++;
    }
    par3[index]='\0'; i++;
}
}

```

5. 接著就是用 `strcmp/strncmp` 去判斷是哪個指令(程式碼有點長，我就直接放結果，可以去看 `code` 的註解)

- a. `pwd`

```

/home/cs4108056007/HW1$ pwd
/home/cs4108056007/HW1

```

- b. echo "string" 或是 echo string 或是 echo \$環境變數

```
/home/cs4108056007/HW1$ echo "Hello"
Hello
```

```
/home/cs4108056007/HW1$ echo 12345
12345
```

```
/home/cs4108056007/HW1$ echo $USER
cs4108056007
```

- c. cd 或是 cd 絕對路徑

```
/home/cs4108056007/HW1$ cd
/home/cs4108056007$ cd /home
/home$
```

- d. export 環境變數=要設定的東西

```
/home/cs4108056007/HW1$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
/home/cs4108056007/HW1$ export PATH=/home
/home/cs4108056007/HW1$ echo $PATH
/home
```

- e. 多一個 exit 指令去離開這個 shell

```
/home/cs4108056007$ exit
cs4108056007@cs4108056007-VirtualBox:~/HW1$
```

- f. external command: 用 child process 去執行 execvp()

```
////////////////////////////////external command////////////////////////////////
else{
    pid_t ex_pid;
    ex_pid= fork();
    if(ex_pid<0){
        printf("fork process error\n");
        return 0;
    }
    else if(ex_pid==0){ //child process
        if(redirect==true){
            dup2(outfd, 1); //redirect output to file par2
            execvp(command, par2, NULL);
            fflush(stdout);
            dup2(outfd_back, 1);
            close(outfd);
        }
        else{
            if(par1[0]!='\0' && par2[0]!='\0') execvp(command, command, par1, par2, NULL); //with 2 par
            else if(par1[0]!='\0') execvp(command, command, par1, NULL); //with 1 par
            else execvp(command, command, NULL); //with no par
        }
        return 0;
    }
    else { //parent process
        wait(NULL);
    }
}
```

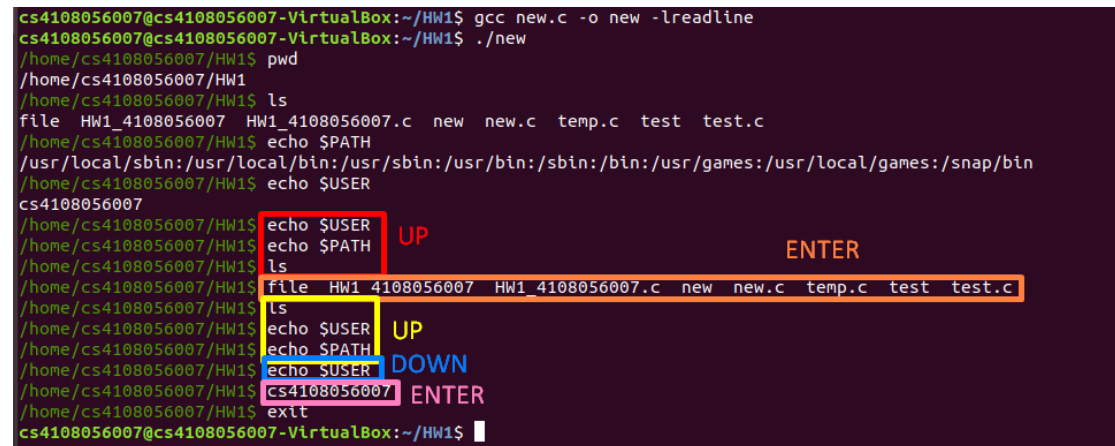
6. 執行過程的影片: <https://youtu.be/BmtSahRHvwY>

## [查詢歷史]

如基本功能那裡介紹的，會把歷史指令放到一個 `stack array`，然後按上下鍵時就會去讀 `stack` 裡面的內容

執行過程的影片：<https://youtu.be/snf7EfJDUEE>

影片錄不到上下鍵，所以我把影片裡面 `key` 的東西標出上下鍵，如下：



```
cs4108056007@cs4108056007-VirtualBox:~/HW1$ gcc new.c -o new -lreadline
cs4108056007@cs4108056007-VirtualBox:~/HW1$ ./new
/home/cs4108056007/HW1$ pwd
/home/cs4108056007/HW1$ ls
file HW1_4108056007 HW1_4108056007.c new new.c temp.c test test.c
/home/cs4108056007/HW1$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
/home/cs4108056007/HW1$ echo $USER
cs4108056007
/home/cs4108056007/HW1$ echo $USER UP
/home/cs4108056007/HW1$ echo $PATH ENTER
/home/cs4108056007/HW1$ ls
file HW1_4108056007 HW1_4108056007.c new new.c temp.c test test.c
/home/cs4108056007/HW1$ ls
/home/cs4108056007/HW1$ echo $USER UP
/home/cs4108056007/HW1$ echo $PATH DOWN
/home/cs4108056007/HW1$ echo $USER ENTER
/home/cs4108056007/HW1$ cs4108056007
/home/cs4108056007/HW1$ exit
cs4108056007@cs4108056007-VirtualBox:~/HW1$
```

如果第一個影片還是不夠清楚：第二個影片是有含鍵盤的：

<https://youtu.be/XFC5p-mMrjl>

## [background execution]

1. 如果是 `'&'` 結尾的，就 `fork child process` 去執行我的 `background`，`parent process` 則是去重新 `input` 一個新指令

```
////////////////////////////////background execution////////////////////////////////
if(buff[buf_size-1]=='&'){
    bg_finish=false; //background is not finished

    pid= fork();
    child_pid= getpid();
    if(pid<0){
        printf("fork process error\n");
        return 0;
    }
    else if(pid==0){ //background(child)
        buff[buf_size-2]='\0'; buff[buf_size-1]='\0'; //erase the ending '&'
    }
    else { //foreground(parent)
        printf("[%d] %d\n", ++process_num, getpid());
        buff[buf_size-2]='\0'; buff[buf_size-1]='\0';
        free(buff);
        continue;
    }
}
```

2. `bg command` 會先用 `waitpid` 檢查 `background` 是否結束 (參數 `WNOHANG` 會立即回傳 `result`，而不是像一般的 `wait` 會等 `child`)  
再依照結果去印出 已結束或還在 `background`

```

//////////bg command//////////
if(pid>0 && strcmp(buff, "bg")==0){
    int status;

    pid_t result= waitpid(pid, &status, WNOHANG); //check if background(child) process still exist
    if(result<0) {
        bg_finish=true; //background has finished
    }

    if(bg_finish==true){
        printf("bash: bg: job has terminated\n");
        //printf("Done %s\n", buff);
        process_num--;
        free(buff);
        continue;
    }
    else{ //background not finished
        printf("bash: bg: job %d already in background\n", process_num);
    }
}
}

```

3. 每個創立的 background(child) 都會在該輪 while 迴圈的最後結束掉

```

//////////background(child) process exits//////////
if(pid==0) {
    bg_finish=true;
    exit(0); //exit curr background(child) process
}

```

4. 執行影片: <https://youtu.be/fDtYW5c8Wy4>

## [Output redirection]

1. 會先判斷輸入的指令中，有沒有>>或是>

> : redirection==true, append==false

>> : redirection==true, append==true

```

for (i=0; i<MAX_BUFFER && buff[i]!='\0'; i++){ //caculate buff size
    buff_size++;
    if(buff[i]=='>') redirect=true; //detect the redirection sign '>'
}

if(strcmp(par1, ">>")==0 || strcmp(par2, ">>")==0) append=true;

```

2. 用剛剛解讀出的 target file 去 create/append file

```

//////////if it is redirection, open target file//////////
if(redirect==true) {
    fflush(stdout);
    outfd_back=dup(1);
    //create/append output file
    if(par3[0]!='\0' && append==false) outfd=open(par2, O_CREAT|O_WRONLY|O_TRUNC, 0666);
    else if(par3[0]!='\0' && append==false) outfd=open(par3, O_CREAT|O_WRONLY|O_TRUNC, 0666);
    else if (par3[0]!='\0' && append==true) outfd=open(par2, O_WRONLY|O_APPEND, 0666);
    else if (par3[0]!='\0' && append==true) outfd=open(par3, O_WRONLY|O_APPEND, 0666);

    if(!outfd){
        perror("error creating file\n");
        return EXIT_FAILURE;
    }
}

```

3. 將那些 stdout 是 screen 的指令，都用 dup2()改成寫入 file，寫完後再把 stdout 改回預設的 screen

(參考: [https://blog.csdn.net/weixin\\_44718794/article/details/106620448](https://blog.csdn.net/weixin_44718794/article/details/106620448))

```

if(redirect==true){
    dup2(outfd, 1); //redirect output to file par2

    if(par3[0]!='\0') execlp(command, par3, par1, NULL); //with 2 par: ls -l >> file
    else execlp(command, par2, NULL); //with 1 par: ls > file

    fflush(stdout);
    dup2(outfd_back, 1);
    close(outfd);
}

```

#### 4. 執行結果

```
/home/cs4108056007/HW1$ ls
HW1_4108056007 HW1_4108056007.c new new.c temp.c test test.c
/home/cs4108056007/HW1$ ls > file
/home/cs4108056007/HW1$ cat file
file
HW1_4108056007
HW1_4108056007.c
new
new.c
temp.c
test
test.c
/home/cs4108056007/HW1$ ls >> file
/home/cs4108056007/HW1$ cat file
file
HW1_4108056007
HW1_4108056007.c
new
new.c
temp.c
test
test.c
file
HW1_4108056007
HW1_4108056007.c
new
new.c
temp.c
test
test.c
```