# 운영체제

#쉘 프로젝트


담 당 : 정준호 교수님

학 과 : 컴퓨터공학과

학 번 : 2016112177

성 명 : 서보미

dongguk UNIVERSITY

## 1. 구현한 기능

- 기본 쉘 명령어
- history 기능 (history, !!, !n)
- 재지향
- Pipe

## 2. 프로그램 구성 함수

```
void getCmd(char* args[],char* cmds[])
```
- 명령문을 입력받고, 단어 단위로 나누어서 배열 args에 저장한다. 그리고 입력받은 명령어를 cmds에 저장한다.

```
void freeArgs(char* args[])
```
- 배열 args의 메모리를 해제시켜 준다

```
void showHistory(char* cmds[])
```
- 입력했던 명령어들을 100개까지 출력한다.

```
void recentHistory(char*cmds[], char* args[])
```
- 바로 직전의 명령어를 다시 실행한다.

```
void nthHistory(char*cmds[],char* args[], int n)
```
- n번 전의 명령어를 다시 실행한다.

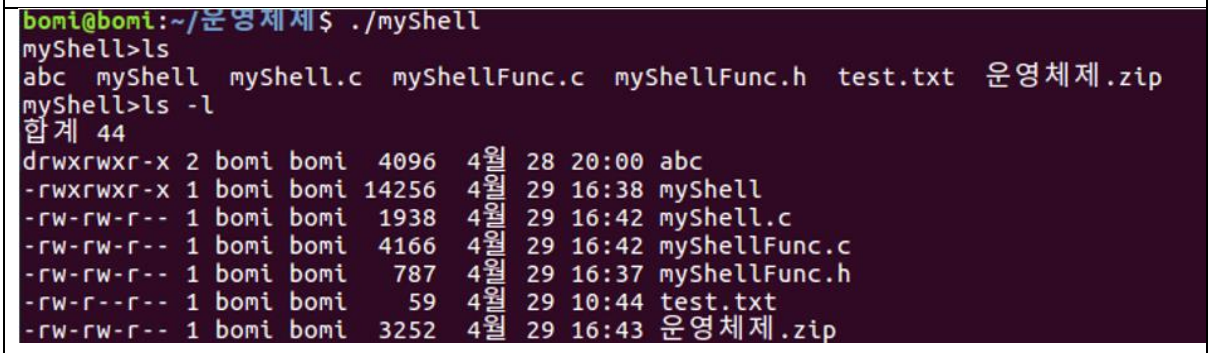
```
void exeCmd(char* cmd ,char* args[])
```
- 실행 시켰던 명령어를 다시 실행시킬 때 쓰는 함수. history기능에서 사용된다.

```
void redirectIn(char*args[])
```
- 문자 '<' 가 들어오면 실행. '<' 뒤의 파일을 앞의 명령어에 입력한다.

```
void redirectOut(char* args[])
```
- 문자 '>'가 들어오면 실행. '>' 앞의 명령어의 실행 결과를 '>'뒤의 명령어에 입력

```
void pipeFunc(char* args[])
```
- 파이프 입력이 들어오면 입력 받은 명령문을 파이프 앞부분과 뒷부분으로 나눈다. 그리고 프로세스를 하나 더 생성하여 앞부분과 뒷부분을 처리한다.

## 3. 프로그램 실행 결과화면




-ls, ls-l 명령어 실행

```
myShell>ls -l
합계 44
drwxrwxr-x 2 bomi bomi  4096  4월 28 20:00 abc
-rw-r--r-- 1 bomi bomi    83  4월 29 16:45 ls.txt
-rwxrwxr-x 1 bomi bomi 14256  4월 29 16:38 myShell
-rw-rw-r-- 1 bomi bomi  1938  4월 29 16:42 myShell.c
-rw-rw-r-- 1 bomi bomi  4166  4월 29 16:42 myShellFunc.c
-rw-rw-r-- 1 bomi bomi   787  4월 29 16:37 myShellFunc.h
-rw-r--r-- 1 bomi bomi     0  4월 29 16:57 test.txt
-rw-rw-r-- 1 bomi bomi  3252  4월 29 16:43 운영체제.zip
myShell>!!
합계 44
drwxrwxr-x 2 bomi bomi  4096  4월 28 20:00 abc
-rw-r--r-- 1 bomi bomi    83  4월 29 16:45 ls.txt
-rwxrwxr-x 1 bomi bomi 14256  4월 29 16:38 myShell
-rw-rw-r-- 1 bomi bomi  1938  4월 29 16:42 myShell.c
-rw-rw-r-- 1 bomi bomi  4166  4월 29 16:42 myShellFunc.c
-rw-rw-r-- 1 bomi bomi   787  4월 29 16:37 myShellFunc.h
-rw-r--r-- 1 bomi bomi     0  4월 29 16:57 test.txt
-rw-rw-r-- 1 bomi bomi  3252  4월 29 16:43 운영체제.zip
```

- !! 명령어 실행

```
myShell>history
3 ls
2 ls -l
1 !!
```

- history명령어 실행

```
myShell>pwd
/home/bomi/운영체제
myShell>!1
pwd
/home/bomi/운영체제
```

- !n 기능 실행 (!1이므로 바로 이전의 명령어를 수행한다.)

```
myShell>ls > ls.txt
myShell>ls
abc     myShell    myShellFunc.c  test.txt
ls.txt  myShell.c  myShellFunc.h  운영체제.zip
myShell>cat ls.txt
abc
ls.txt
myShell
myShell.c
myShellFunc.c
myShellFunc.h
test.txt
운영체제.zip
```

- 재지향 out 실행, 정상작동 확인 (ls.txt에 ls 실행 결과를 쓴다.)

```
myShell>head < ls.txt
abc
ls.txt
myShell
myShell.c
myShellFunc.c
myShellFunc.h
test.txt
운영체제_zip
```

- 재지향 in 실행. ls.txt을 head에 입력한다.

```
myShell>grep myShell ls.txt | more
myShell
myShell.c
myShellFunc.c
myShellFunc.h
```

- 파이프 실행. (ls.txt 에서 myShell이 들어간 행 출력)

```
myShell>exit
bomi@bomi:~/운영체제$
```

프로그램 종료

## 4. 소스코드

myShell.c

```c
#include "myShellFunc.h"
#include "myShellFunc.c"

int main()
{
  char *args[MAXLINE/2 + 1] = {'₩0', }, *newArgs[MAXLINE/2+1]={'₩0',};
  char *cmds[MAXCMD]={'₩0', };
  char *nth=(char*)malloc(sizeof(MAXLINE));
  int should_run = 1;
  int pid,n,i,flag,and=0;

  while(1) {
    printf("myShell>");
    fflush(stdout);

    getCmd(args,cmds);

    pid_t pid;
    pid=fork();

    if(pid==0){

      if(strcmp(args[0],"exit")==0){should_run=0;return 0;} //exit명령어 입력시 탈출
      if(cmds[ccnt-1][strlen(cmds[ccnt-1])-1]=='&')and=1;  //백그라운드로 실행

      if(strcmp(args[0],"history")==0)showHistory(cmds);  //history 명령어 입력
      if(strcmp(args[0],"!!")==0)recentHistory(cmds,args);  //!!입력
      if(args[0][0]=='!'&&args[0][1]!='!'){ //!n입력시
```

```
        strncpy(nth,args[0]+1, strlen(args[0])-1);
        n=atoi(nth);
        nthHistory(cmds,args,n);
      }

      i=0;flag=0;
      while(i<cnt){
        if(strcmp(args[i], ">")==0){ // redirect in인 경우
          flag= 1;
          redirectOut(args);
          execvp(args[0],args);
          break;
        }
        else if(strcmp(args[i], "<")==0){ //redirect out인 경우
          flag=1;
          redirectIn(args);
          execvp(args[0],args);
          break;
        }
        else if(strcmp(args[i], "|")==0){ //파이프인 경우
          flag= 1;
          pipeFunc(args);
          break;
        }
        i++;
      }

      if(flag==0) execvp(args[0],args);  //일반 명령어 처리

    }

    else if(pid>0){ //부모 프로세스일때
      if(and==0) wait(NULL);  //&가 포함되어 있으면 기다린다
      if(strcmp(args[0],"exit")==0){should_run=0;return 0;}//exit가 입력되면 부모 프로세스도
끝나야 하기에 죽인다.
    }
    else{ //fork가 정상적으로 실행 되지 않았을때 에러메세지 출력후 종료
      fprintf(stderr, "프로세스를 생성할 수 없습니다.₩n");
      should_run=0;
    }
    freeArgs(args);
  }
  return 0;
}
```

myShellFunc.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
```

```c
#define MAXLINE 80
#define MAXCMD 100

void getCmd(char* args[],char* cmds[]); //명령문 받음
void freeArgs(char* args[]); //메모리 해제

void showHistory(char* cmds[]); //100개까지의 명령어 입력내역 출력
void recentHistory(char*cmds[], char* args[]);  //바로 직전의 명령어 실행
void nthHistory(char*cmds[],char* args[], int n); //n번 전의 명령어 실행
void exeCmd(char* cmd ,char* args[]); //history기능에서 명령어 실행시 사용하는 함수

void redirectIn(char*args[]);//redirect in 함수
void redirectOut(char* args[]);//redirect out 함수
void pipeFunc(char* args[]);//파이프 함수
```

myShellFunc.c

```c
int cnt=0,ccnt=0;

void getCmd(char* args[],char* cmds[]){
  int argIndex = 0;
  char command[MAXLINE] = {'W0', };

  fgets(command, sizeof(command), stdin);//명령어 입력

  cmds[ccnt]=(char*)malloc(strlen(command));
  strncpy(cmds[ccnt],command,strlen(command));//히스토리에 저장
  ccnt++;

    //strotk함수 사용시 에러가 많아 직접 나누어준다.
  command[strlen(command) - 1] = 0;
  for(int i = 0, j = 0; i < strlen(command) + 1; i++) {
    if(command[i] == ' ' || command[i] == 'W0') {//space or null
      args[argIndex] = (char *)malloc(sizeof(char)*(j + 1));//memory allocate
      strncpy(args[argIndex], &command[i - j], j);//string copy
      args[argIndex++][j] = 0;
      cnt++;
      j = -1;
    }
    j++;
  }
}

void freeArgs(char *args[])
{
  int i = 0;
  while(i < cnt) {
    free(args[i]);
    i++;
  }
  cnt = 0;
}
```

```
void showHistory(char* cmds[]){
  //history 배열안의 값들 출력
  for(int i=0;i<ccnt-1;i++){
    printf("%d %s",ccnt-i-1, cmds[i]);
  }
}

void recentHistory(char* cmds[], char* args[]){
  freeArgs(args);  //기존에 있던 args배열을 해제해줌
  exeCmd(cmds[ccnt-2],args);  //이전의 커맨드 실행시킨다
}

void nthHistory(char* cmds[],char* args[], int n){
  freeArgs(args);  //기존에 있던 args배열을 해제해줌
    if(cmds[ccnt-n-2]==NULL){printf("NO HISTORY₩n");return;}  //예외처리
  printf("%s", cmds[ccnt-n-1]);
  exeCmd(cmds[ccnt-n-1],args);  //n번째 커맨드 실행시킨다
}

void exeCmd(char* cmd,char* args[]){
  int argIndex = 0;
  char* command;

  command=(char*)malloc(strlen(cmd));
  strncpy(command,cmd,strlen(cmd));

  //strotk함수 사용시 에러가 많아 직접 나누어준다.
  command[strlen(command) - 1] = 0;
  for(int i = 0, j = 0; i < strlen(command) + 1; i++) {
    if(command[i] == ' ' || command[i] == '₩0') { //space or NULL
      args[argIndex] = (char *)malloc(sizeof(char)*(j + 1)); //memory allocate
      strncpy(args[argIndex], &command[i - j], j); //string copy
      args[argIndex++][j] = 0;
      cnt++;
      j = -1; //문지열의 갯수를 다시 초기화 한다
    }
    j++;
  }
}

void redirectIn(char* args[]){
  int i = 1, fd;

  while(strcmp(args[i], "<") != 0)//'<' 와 같을 시 루프문 탈출
    i++;

  if(args) {
    if(!args[i + 1]) return;
    else {//파일을 연다
      if((fd = open(args[i + 1], O_RDONLY)) == -1) {
        fprintf(stderr, "%s₩n", args[i + 1]);
        return;
      }
    }
   dup2(fd, 0);
```

```
    close(fd);

    while(args[i] != NULL) {
      args[i] = args[i + 2];
      i++;
    }
    args[i] = NULL;
  }
}

void redirectOut(char* args[]){
  int i = 0, fd;

  while(strcmp(args[i], ">") != 0)//'>' 와 같을 시 루프문 탈출
    i++;

  if(args[i]) {
    if(args[i + 1] == NULL) return;//> 가 마지막으로 입력된 문자일 때
    else {
      if((fd = open(args[i + 1], O_RDWR | O_CREAT | S_IROTH, 0644)) == -1) {
        fprintf(stderr, "%s₩n", args[i + 1]);
        return;
      }
    }
    dup2(fd, 1);
    close(fd); //출력을 넣고 닫는다.
    args[i] = NULL;
    args[i + 1] = NULL;

    while(args[i] != NULL) {
      args[i] = args[i + 2];
    }
    args[i] = NULL;
  }
}

void pipeFunc(char* args[]){
  int i = 0, j = 0, fd[2];
  pid_t pid;
  char *argsPipe1[MAXLINE], *argsPipe2[MAXLINE];

  while(args[i] != NULL) {//| 문자 전의 문자열 주소 복사
    if(strcmp(args[i], "|") == 0) {
      argsPipe1[i] = NULL;
      break;
    }
    argsPipe1[i] = args[i];
    i++;
  }
  i++;
  while(args[i] != NULL) {//| 문자 이후의 문자열 주소 복사
    argsPipe2[j] = args[i];
    i++;j++;
  }
```

```
  //파이프 생성
  if(pipe(fd) == -1) {
    fprintf(stderr, "파이프 생성에 실패했습니다.");
    exit(1);
  }
  close(0);
  close(fd[1]);//입력 파이프 닫기
  dup(fd[0]);//식별자 복사
  close(fd[0]);
  execvp(argsPipe1[0], argsPipe1);//앞쪽 명령어 실행
  exit(1);

  pid = fork();

  if(pid == 0) {
    close(1);//STDOUT_FILENO 표준출력 닫기
    close(fd[0]);//출력 파이프 닫기
    dup(fd[1]);//식별자 복사
    close(fd[1]);
    execvp(argsPipe2[0], argsPipe2);//뒤쪽 명령어 실행
    exit(0);
  }
}
```

## 과제 소감

이번 과제를 하면서 쉘에서 몰랐던 기능들도 많이 알게 되었다. 사실 파이프나 재지향 이런 것들은 그저 수업시간에 이론적으로 배우기만 했었다. 추상적으로 어떤 기능을 구현 할 것이다 생각했었는데, 이번 기회에 제대로 알게 되어 기쁘다. 생각보다 자잘한 부분에서 발목이 많이 잡혀 애를 먹었다. 문자열을 분리할 때도, strtok_r 함수를 쓰니 세그먼트 오류가 나서, 함수를 쓰지 않고 일일이 다 나누어 주는 방법을 사용했다. 또 exit 명령어를 입력 받아 종료 할 때도, 자식 프로세스만 종료되고 부모 프로세스는 종료되지 않아 결국엔 둘다 처리해주는 방법을 쓰게 되었다. 프로젝트를 하면 할수록 나는 아직 부족하다는 것을 많이 느끼게 된다.