# 시스템SW실습2 PA3

2016310936 우승민

이번 과제는 Shell을 저만의 방식으로 구현해 내는 과제입니다. 우선 제가 사용한 함수들과 전역변수를 보면

```
void eval(char *cmdline);
void m_path(char **argv);
void my_ls(char **argv);
void my_man(char **argv);
void my_grep(char **argv);
void my_sort(char **argv);
void my_awk(char **argv);
void my_bc(char **argv);
void my_head(char **argv);
void my_tail(char **argv);
void my_cat(char **argv);
void my_cp(char **argv);
void my_mv(char **argv);
void my_rm(char **argv);
void my_cd(char **argv);
void my_pwd();
void my_exit(char **argv);
void end(int sig);
int parseline(char *buf, char **argv);
int builtin_command(char **argv);
int re_in(char **argv, int fd);
int re_out(char **argv, int fd);
void pipeline(char **argv);
void my_sig(int sig);
int pp[5];
int num=0;
int x= 0;
int in = 0;
int out =0;
```

구현해야 하는 각각의 command들을 **my_func**의 형식으로 받아서 입력한 argv를 인자로 넘겨주도록 하였습니다.

Redirection에 관한 함수로 re_in은 standard in으로 파일을 받을 때 re_out은 standard out으로 파일을 내보낼 때 사용하는 함수입니다.

Pipeline은 pipe를 처리하는 함수이고 pipe가 사용된 횟수는 num 각각의 위치를 pp array에 저장하였습니다.

전역변수 x 는 제가 입력한 argv의 단어의 개수 즉 "./wordcount 128 < a.txt" 를 입력하였으면 4가 됩니다.

In 과 out 은 redirection이 사용되었을 때 인자의 위치 즉 위와 같은 명령어에서 in은 3이 됩니다.

이제 **main함수**부터 살펴보면

```
int main()
{
    signal(SIGINT,end);
    signal(SIGTSTP,end);
    char cmdline[MAXLINE]; /* Command line */
```

```
void end(int sig){
        kill(-1,SIGCHLD);
}
```

Main 함수는 기존의 skeleton 코드에서 다른 것은 건드리지 않고 **SIGINT**와 **SIGTSTP**를 입력받을 때 swsh가 종료되지 않고 end 함수로 가게 만들어 주었습니다. 여기서 end함수는 **SIGCHLD** 신호를 보내어 fork를 사용한 함수들을 종료하게 만들어 주었습니다.

Argv를 읽는 것은 **builtin_command**에서 실행되었습니다.

```
int builtin_command(char **argv)
{
    if (!strcmp(argv[0], "quit")) /* quit command */
        exit(0);
    if (!strcmp(argv[0], "&"))    /* Ignore singleton & */
        return 1;

        int k;
        in = out = num = 0;
        for(k=0; argv[k]!=NULL; k++){
                if(strcmp(argv[k],"<")==0)
                        in = k+1;
                else if(strcmp(argv[k],">")==0)
                        out = k+1;
                else if(strcmp(argv[k], ">>")==0)
                        out = k+1;
                else if(strcmp(argv[k], "|")==0)
                        pp[num++] = k+1;
        }
        if(num!=0){
                pipeline(argv);
                return 1;
                }
```

Argv가 끝날때까지 for문을 돌면서 argv안에 redirection과 pipe 위치를 각각 전역변수에 저장하게 만들어주었습니다. 만약 pipe가 사용되었으면 다른 함수들이 아닌 **pipeline** 함수로 들어가게 됩니다.

```
int re_in(char **argv, int fd){
        int i;
        if(!argv[in]){
                errno = EACCES;
                perror("swsh");
                return -1;
        }
        else{
                if((fd=open(argv[in],O_RDONLY)) ==-1){
                        perror("swsh");
                        return -1;
                }
        }

        dup2(fd,0);
        close(fd);

        for(i = in-1; argv[i]!=NULL; i++){
                argv[i] = argv[i+2];
        }

                argv[i]=NULL;
        return 1;

}
```

먼저 **redirection input** 함수를 보면 argv 와 pipe 를 통해 standard in 을 바꾸어줄 int 형 인자 fd 를 같이 넘겨주었고,

만약 "<" 뒤에 받은 단어가 없으면 "permission denied"를 출력하고 종료하고

받았는데 없는 파일이면 "swsh : No such file or directory"를 출력하게 하였습니다.

그 후 dup2 를 사용하여 standard input 을 fd 로 바꾸어주고

Argv 에서 "< file" 부분을 지우도록 바꾸어주었습니다. Ex) "./wordcount 128 ~~< EMMA.word~~"

```
int re_out(char **argv, int fd){
        int i;
        if(!argv[out]){
                errno = EACCES;
                perror("swsh");
                return -1;
                }
        else{
                if((fd=open(argv[out],O_RDWR | O_CREAT, 0755)) ==-1){
                        perror("swsh");
                        return -1;
                }
                if(strcmp(argv[out-1],">>")==0)
                        lseek(fd,0,SEEK_END);
        }

        dup2(fd,1);
        close(fd);
        for(i = out-1; argv[i]!=NULL; i++){
                argv[i] = argv[i+2];
        }

                argv[i]=NULL;

        return 1;
}
```

**Redirection out**도 in과 동일한 방법으로 구현하였습니다. 하지만 다른 점은 input과 다르게 파일이 없어도 만들어야 하고, ">>"를 통해 redirection을 하면 **lseek** 함수를 사용하여 파일의 맨 끝으로 이동하여 작성하게 해주었습니다.

```
void pipeline(char **argv){
        int i;
        int k = 0;
        pid_t pid1, pid2;
        int fd[2];
        int status1, status2;

        char **buf = (char **)malloc(sizeof(char *)*(pp[k]));
        char **buf2 = (char **)malloc(sizeof(char *)*(x-pp[k]+2));

        for(i=0; i < pp[k]-1; i++){
                buf[i] = (char *) malloc(sizeof(char)*20);
                strcpy(buf[i], argv[i]);
        }
        buf[i]= NULL;
        for(i= 0; argv[i+pp[k]]!=NULL; i++){
                buf2[i] = (char *) malloc(sizeof(char)*20);
                strcpy(buf2[i], argv[i+pp[k]]);
        }
        buf2[i] =NULL;
```

**pipeline** 함수는 우선 argv와 같은 형태인 char형 이중 포인터 buf와 buf2를 malloc해주어

Argv를 "|" 왼쪽을 buf 오른쪽을 buf2로 나누어 주었습니다.

Ex) "./wordcount 128 < EMMA.word | grep a > a.txt" - > buf = "./wordcount 128 < EMMA.word"

Buf2 = " grep a > a.txt"

```
if(pipe(fd)<0) end(2);

in = out = num = 0;

if((pid1=(fork()==0))){
        x = pp[k]-1;
        dup2(fd[1],1);
        close(fd[0]);
        close(fd[1]);
        builtin_command(buf);
}
waitpid(pid1, &status1,  WUNTRACED);
waitpid(pid1, &status1,  WNOHANG);


if((pid2=(fork()==0))){
        x = x - pp[k]+1;
        dup2(fd[0],0);
        close(fd[0]);
        close(fd[1]);
        builtin_command(buf2);
}

waitpid(pid2, &status2,  WUNTRACED);
waitpid(pid2, &status2,  WNOHANG);
```

```
for(i=0; i<pp[k]; i++){
        free(buf[i]);
}

for(i=0; i<x-pp[k]+2; i++){
        free(buf2[i]);
}
free(buf);
free(buf2);
```

그 후 나누어진 command를 각각 pid1 과 pid2를 통해 dup2로 standard in과 out을 바꾸어 주고 기존의 **builtin_command** 함수를 실행하게 해주었습니다.

마지막으로 malloc한 buf와 buf2를 free해주었습니다.

다시 **builtin_command**로 가서 이후의 내용을 보면 전부 동일하게 if문으로 구성되어있습니다.

```
        if(!strcmp(argv[0], "exit"))
        {
                my_exit(argv);
                return 1;
        }                       //type4


        if(!strncmp(argv[0],"./",2))
        {
                my_path(argv);
                return 1;
        }


    return 0;                   /* Not a builtin command */
}
```

Strcmp를 통해 argv의 첫번째 인자의 command를 확인하고, 그에 따라 존재하는 command면 실행하고 1을 return 하게됩니다. 만약 끝까지 command가 발견되지 않으면 없는 command이므로 0을 return합니다.

**Type 1 command** 들은 전부 동일한 형태로 이루어져 있습니다.

```
void my_ls(char **argv)
{
        pid_t pid;
        int status;

        if((pid=(fork()==0))){
                int fd[2];
                if(pipe(fd)<0) end(2);

                if(in!=0){
                        if(re_in(argv,fd[0])==-1)
                                return;
                        if(out!=0)
                                out = out - 2;
                        }
                if(out!=0){
                        if(re_out(argv,fd[1])==-1)
                                return;
                }
                execv("/bin/ls", argv);
        }


        in = 0;
        out = 0;

        waitpid(pid, &status,  WUNTRACED);
        waitpid(pid, &status,  WNOHANG);

}
```

fork문을 통해 child를 생성하고 만약 redirection이 사용되었으면, re 함수로 이동하고 -1이 나온 경우에는 error가 발생한 상황이므로 return 해주어 바로 종료하게 해주었습니다.

Out = out -2 이 있는 이유는 re_in이 실행될 경우 argv의 인자를 2개씩 당겨왔기 때문에 out도 2를 줄여줘야 다음 실행에 지장이 없어서 해주었습니다. 이후에 in과 out 값을 초기화해주고 waitpid를 통해 좀비프로세스를 방지해주었습니다.

```
void my_head(char **argv)
{
        int fd[2];
        if(pipe(fd)<0) end(2);

        if(in!=0){
                pid_t pid;
                int status;

                if((pid=(fork()==0))){
                        if(re_in(argv,fd[0])==-1)
                                return;
                        if(out!=0)
                                out = out - 2;
                        if(out!=0){
                                if(re_out(argv,fd[1])==-1)
                                        return;
                        }

                        execv("/usr/bin/head", argv);
                }
                waitpid(pid, &status,  WUNTRACED);
                waitpid(pid, &status,  WNOHANG);
        }
```

**Type2 command**는 Input redirection이 있는 경우와 없는 경우 2가지로 나누어 구현하였습니다.

제가 구현한 방식으로는 redirection input을 받지 못하였기 때문에 execv함수를 사용하였습니다.

(이 경우는 type1과 동일)

Redirection input이 없는 경우는 인자를 2가 받는 경우와 아닌 경우로 나누었습니다.

```
else{
        const char *file_src = NULL;

        int fd_src;
        int a;
        char buffer[1];
        char *l = (char*)malloc(sizeof(char)*1000);


        if(x==2){
                file_src = argv[1];
                fd_src = open(file_src, O_RDONLY, 0755);

                int i=0;
                while(((a=read(fd_src, buffer,1))!=0)&&(i<1000))
                        l[i++]=buffer[0];
                l[i]='₩0';
                int k=0;
                i=0;

                for(k=0; k<10; k++){
                                while(1){
                                        printf("%c", l[i++]);
                                        if(l[i]=='₩n')
                                                break;
                                        }
                }
                printf("₩n");

        }
```

인자가 2개일 경우에는 파일을 읽어 l에 앞에서 1000자리까지 입력을 받아 줄이 끊기는 '₩n'를 10번 받을 때까지 출력하게 만들어 주었습니다. 이 경우에는 파일이 존재하지 않는 경우는 없기 때문에 에러를 넣지 않았습니다.

```
        else{
                file_src = argv[3];
                fd_src = open(file_src, O_RDONLY, 0755);

                int i=0;
                while(((a=read(fd_src, buffer,1))!=0)&&(i<1000))
                        l[i++]=buffer[0];
                l[i]='₩0';
                int k=0;
                i=0;
                int j = atoi(argv[2]);
                for(k=0; k<j; k++){
                                while(1){
                                        printf("%c", l[i++]);
                                        if(l[i]=='₩n')
                                                break;
                                        }
                }
                printf("₩n");

        }
        free(l);
        close(fd_src);

        }
in = 0;
out = 0;
```

"head -n k file"인 경우에는 k를 atoi 함수를 통해 int형으로 변형한 후 j에 저장하였고

for문에 10 대신 j를 넣어 그만큼 출력되게 하였습니다.

```
void my_tail(char **argv)
{
        int fd[2];
```

**tail command** 또한 redirection input이 사용된 경우에는 동일한 방식으로 하여 부가적인 설명은 하지 않겠습니다. 그 이외의 경우에는,

```
const char *file_src = NULL;

int fd_src;
int a;
char buffer[1];
char *l = (char*)malloc(sizeof(char)*1000);
char *ta = (char*)malloc(sizeof(char)*100);

if(x==2){
        file_src = argv[1];
        fd_src = open(file_src, O_RDONLY, 0755);
        int i=0;
        int n=-3;
        int p=0;
        while(((a=read(fd_src, buffer,1))!=0)&&(p<10)){
                lseek(fd_src,n--, SEEK_END);
                l[i++]=buffer[0];
                if(buffer[0]=='\n')
                        p++;
                }
        l[i-1]='\0';
        i=i-2;
        int k=0;
        int t=0;
        p=0;

        for(k=0; k<10; k++){
                        while(1){
                                if(l[i--]=='\n')
                                        break;
                                p++;
                                }
                        p++;
                        for(t=p; t>=0; t--)
                                ta[t] = l[i+p-t];
                        ta[p]='\0';
                        printf("%s", ta);
                        p=0;
        }
        printf("\n");
}
```

**Tail command** 의 경우에는 head 와 달리 아래에서부터 출력하는 것이기 때문에 read 를 할때에도 **lseek** 를 통해 파일의 끝에서부터 읽게 해주었습니다. 이렇게 하면 출력을 할 때에도 거꾸로 나오기 때문에 추가적으로 ta 라는 char 형을 malloc 해 주어 "\n"이 나올 때까지 거꾸로 입력받아 출력하게 해주었습니다.

인자가 2 개가 아닌 경우에는 head 와 마찬가지로 -n 옵션을 추가한 것이므로 동일하게 했습니다.

```
void my_cat(char **argv)
{
```

**cat** 의 경우에도 input redirection 은 동일하게 해주었고, 이후의 경우에서는 옵션 또한 없기 때문에 비교적 간단하게 만들었습니다.

```
else{
        if(out!=0)
                re_out(argv,fd[1]);

        const char *file_src = NULL;

        int fd_src;
        int a;
        char buffer[1];

        file_src = argv[1];
        fd_src = open(file_src, O_RDONLY, 0755);

        while((a=read(fd_src, buffer,1))!=0)
                if(write(1,buffer,1));
        close(fd_src);

}
in = 0;
out = 0;
```

**Type2** 의 마지막인 **cp** 의 경우에는 파일을 복사하는 command 입니다.

이 command 는 파일을 하나는 읽고 하나는 생성하는 것으로 받아들여 src 에서 읽은 것을 그대로 dest 에 써주게 만들어 주었습니다.

```
void my_cp(char **argv)
{
        const char *file_src = NULL;
        const char *file_dest = NULL;

        int fd_src;
        int fd_dest;
        int a;
        char buffer[1];

        file_src = argv[1];
        file_dest = argv[2];
        fd_src = open(file_src, O_RDONLY, 0755);
        fd_dest = open(file_dest, O_RDWR| O_CREAT, 0755 );

        while((a=read(fd_src, buffer,1))!=0)
                if(write(fd_dest,buffer,1));
        close(fd_src);
        close(fd_dest);

}
```

다음으로 **type3** 에서는 파일에 따라 error 를 출력해야 하므로 **stat** 을 사용하여 파일이 있는지 디렉토리 형식인지 이동이나 변경이 가능한지 파악하게 해주었습니다.

```
void my_mv(char **argv)
{
        const char *file_src = NULL;
        const char *file_dest = NULL;
        char *buf = (char*)malloc(sizeof(char)*100);

        int fd_src;
        int fd_dest;
        int a;
        char buffer[1];

        struct stat sb;
        stat(argv[1], &sb);
        switch (sb.st_mode & S_IFMT){
                case S_IFDIR:{
                        errno = EISDIR;
                        perror("");
                        free(buf);
                        return;
                        }
                }

        int mode = R_OK & W_OK & X_OK;

        if(access(argv[1], mode)!=0){
                errno  = EACCES;
                perror("");
                free(buf);
                return ;
                }
```

```
stat(argv[x-1], &sb);

int rename=0;
switch (sb.st_mode & S_IFMT){
        case S_IFDIR :{
                strcpy(buf, "./");
                strcat(buf, argv[x-1]);
                strcat(buf, "/");
                break;
        }
        default : rename=1;
}
```

또한 마지막으로 입력한 내용이 directory 일 경우에는 directory 안으로 이동하는 것이므로 판별해 주었습니다. Buf 에 **"./dir/"**을 미리 넣어 놓으면 폴더 안으로 파일을 만들 수 있습니다. 만약 directory 가 아닐 경우에는 이름을 바꾸는 목적으로 사용되었기 때문에 rename 이라는 변수를 따로 설정해 주었습니다.

```
int i;
for(i=1; i<x-1; i++){

        char *q = (char*)malloc(sizeof(char)*100);
        strcpy(q,buf);
        file_src = argv[i];
        strcat(q, argv[i]);
        if(rename== 0)
                file_dest = q;
        else
                file_dest = argv[2];

        fd_src = open(file_src, O_RDONLY, 0755);
        fd_dest = open(file_dest, O_RDWR| O_CREAT, 0755 );

        if(fd_src==-1){
                errno  = ENOENT;
                perror("");
                free(buf);
                close(fd_src);
                close(fd_dest);
                return ;
                }

        while((a=read(fd_src, buffer,1))!=0)
                if(write(fd_dest,buffer,1));

        free(q);
        close(fd_src);
        close(fd_dest);
        unlink(argv[i]);
}
free(buf);
```

그 후 파일이 여러 개 들어올 수 있기 때문에 반복문을 사용하여 기존의 파일을 옮길 파일에 read 와 write 를 사용하여 옮겨주었습니다. 마지막으로 기존의 파일은 **unlink** 함수를 사용하여 삭제해 주었습니다.

```
void my_rm(char **argv)
{

        int i;
        struct stat sb;
        const char *file_src = NULL;
        for(i=0; i<x-1; i++){
                int fd_src;
                stat(argv[i+1], &sb);
```

**rm command** 도 **mv** 과 동일한 방식으로 **stat** 을 이용하여 파일이 있는지 없는지. directory 형식인지, 변경 가능한지 확인해 주었고, 이후 **unlink** 를 사용하여 지워주었습니다.

```
void my_cd(char **argv)
{
        char *path;
        struct stat sb;
        if(x > 1){
                stat(argv[1], &sb);
                switch (sb.st_mode & S_IFMT){
                        case S_IFDIR: {
                                path = argv[1];
                                break;
                                }
                        default : {
                                errno = ENOTDIR;
                                perror("");
                                return;
                                }
                        }
                int mode = R_OK & W_OK & X_OK;

                if(access(argv[1], mode)!=0){
                        errno  = EACCES;
                        perror("");
                        return ;
                        }
        }
        else {
                path = ".";
        }
        if(chdir(path));

}
```

.

**Cd command** 는 인자를 반드시 directory 로 받아야 하므로 **stat** 으로 확인해 주었고,

path 에 인자를 넣어준 후 **chdir** 함수를 사용하여 directory 를 이동하게 해주었습니다

마지막으로 **type 4** 함수를 보면

```
void my_pwd()
{
        char buf[100];
        if(getcwd(buf,100));
        printf("%s₩n", buf);
}
void my_exit(char **argv)
{
        printf("exit₩n");
        if(x==1)
                exit(0);
        else{
                int a = atoi(argv[1]);
                exit(a);
        }
}
```

**Pwd** 함수는 **getcwd** 함수로 buf 에 현재 위치를 옮겨주어 출력하게 만들어 주었고,

**Exit** 함수는 인자를 따로 받지 않은경우에는 exit(0)을 통해 종료하게 받은 경우에는 exit(a)를 통해 종료하게 만들어 주었습니다.