

리눅스 텀 프로젝트 보고서

컴퓨터과학부

2016920004

권은진

1. cd 구현

- 추가된 헤더파일 : <pwd.h>
- 추가된 이유 : 명령어 "cd ~" 입력 시 홈 디렉토리로 가게 하기 위해 Real User ID를 문자열로 반환하는 getpwuid() 함수를 사용하기 위해서이다.
- 소스 설명

```
else if (!strcmp(arg[0], "cd")) {
    if (!strcmp(arg[1], "~")) {
        char* uid = getpwuid(getuid()->pw_name);
        if (!strcmp(uid, "root")) arg[1] = "/root";
        else {
            char tmp[100] = "/home/";
            strcat(tmp, uid);
            arg[1] = tmp;
        }
    }
    if (chdir(arg[1]) == -1) {
        fprintf(stderr, "minish [%s]: No such file or directory\n", arg[1]);
        finished = TRUE;
    }
}
```

현재 작업 디렉토리를 변경하는 chdir() 함수는 홈 디렉토리를 의미하는 "~"를 처리하지 못하므로 chdir()을 호출하기 전에 검사하여 홈 디렉토리 경로에 대한 문자열을 생성하였다. 홈 디렉토리 경로에는 항상 문자열 형태의 사용자 ID가 필요하기 때문에 getpwuid() 함수를 호출하였다.

getpwuid에 대한 함수의 원형은 아래와 같다.

```
struct passwd* getpwuid(uid_t uid);
```

문자열이 아닌 정수형 ID 형식의 Real User ID를 인자로 넘겨주면 사용자 ID의 정보를 담은 구조체 포인터를 반환한다. 포인터 변수나 uid_t 타입의 변수를 각각 선언하는 것보다는 한 줄에 처리하는 것이 깔끔한 것 같아서 getpwuid(getuid()->pw_name)의 형태로 문자열 형식의 사용자 ID를 받았다.

사용자가 root 사용자인 경우 홈 디렉토리가 루트 디렉토리 바로 아래에 있는 반면, 일반 사용자들은 /home 아래에 사용자별로 홈 디렉토리가 존재하므로 if문으로 구별하여 경로 문자열을 갱신하였다. strcat() 은 문자열을 합치는 함수로 사용자 아이디를 경로 문자열의 끝에 추가하기 위해 사용하였다. arg[1]에 바로 대입하지 않은 것은 즉, arg[1] = "/home/"의 형태가 아닌 것은 포인터가 참조하는 문자열 상수는 갱신이 불가능하기 때문에 문자 배열 tmp를 사용하였다.

chdir() 함수는 실패 시 -1 을 리턴하므로, if문으로 검사하여 오류가 날 경우 finished를 TRUE로 설정하여 while 문을 종료하고 다음 명령을 받도록 하였다.

- 실행 결과

[root 사용자]

```

root@root: ~/Documents/Linux
File Edit View Search Terminal Help
root@root:~/Documents/Linux# ./minish
msh # ls
08      '11(1)'      ipc1      ipc2_by_me.o      midterm      myls.o
08-2    12          ipc1.c    linux_course10    mini_sh.c    np
09      12-1      ipc1.o    linux_course11    mini_sh.o    vi_명령어.pdf
10      12-2      ipc2.c    linux_course12    minish
10-2    Practice  ipc2_by_me    linux_course8    myls
11      example8-12  ipc2_by_me.c    linux_course9    myls.c
msh # cd /var
msh # ls
backups  lib      lock  mail  run  tmp      www
cache    local   log   opt   spool unicornscan
msh # cd www
msh # ls
html
msh # cd ..
msh # ls
backups  lib      lock  mail  run  tmp      www
cache    local   log   opt   spool unicornscan
msh # cd ~
msh # ls
Desktop      Music      Templates      abc.txt      auto_ping.o
Documents    Pictures   Videos        auto_ping    cba.txt
Downloads    Public     'VirtualBox VMs' auto_ping.c
msh #

```

[일반 사용자]

```

kej@root: /root/Documents/Linux
File Edit View Search Terminal Help
root@root:~/Documents/Linux# su kej
kej@root:/root/Documents/Linux$ ./minish
msh # ls
08      '11(1)'      ipc1      ipc2_by_me.o      midterm      myls.o
08-2    12          ipc1.c    linux_course10    mini_sh.c    np
09      12-1      ipc1.o    linux_course11    mini_sh.o    vi_명령어.pdf
10      12-2      ipc2.c    linux_course12    minish
10-2    Practice  ipc2_by_me    linux_course8    myls
11      example8-12  ipc2_by_me.c    linux_course9    myls.c
msh # cd ~
msh # ls
Desktop Documents Downloads Music Pictures Public Templates Videos
msh # cd /root
msh # ls
Desktop      Music      Templates      abc.txt      auto_ping.o
Documents    Pictures   Videos        auto_ping    cba.txt
Downloads    Public     'VirtualBox VMs' auto_ping.c
msh # cd ~
msh # ls
Desktop Documents Downloads Music Pictures Public Templates Videos
msh #

```

2. 리다이렉션(Redirection) 구현

- 소스 설명

변수 path는 exec로 명령을 실행할 경로(디렉토리 or 파일명)를 의미한다. 사용한 이유는 아래와 같다.

- ① 리다이렉션을 했을 경우, 리다이렉션 기호가 포함된 명령의 인자 배열(comm)은 exec 계열의 함수에서 작업을 수행하지 않는다.
- ② 명령어만 입력했을 때, 현재 작업 디렉토리를 기준으로 명령을 수행하는 경우 exec 계열의 함수에 명령어

```
char *path = NULL, *symbols = "<>|";
int redir = 0, i;
int pid1, pid2;
int fd[2];

for (i = 1; comm[i] != NULL; i++) {
    if (strcspn(comm[i], symbols) == 0) {
        redir = i;
        path = (redir == 2) ? comm[redir-1] : "./";
        break;
    }
}

if (redir != 0) {
    if (strcmp(comm[redir], ">") == 0) {
        // ...
    } else if (strcmp(comm[redir], "<") == 0) {
        // ...
    } else if (strcmp(comm[redir], "|") == 0) {
        // ...
    }
} else {
    // 리다이렉션이 없는 경우
}
```

에 쓰이는 인자를 지정해주어야 한다.

변수 symbols는 리다이렉션의 포함여부를 strcspn() 함수로 검사하기 위해 사용된다.

변수 redir은 포인터 배열 comm에 리다이렉션 기호가 위치한 인덱스를 의미한다. "ls /etc | more" 또는 "cat abc.txt > cba.txt"와 같은 경우 다른 경로에서 명령을 실행했을 때의 결과를 가지고 리다이렉션한다. 또한 "ls | more" 또는 "ls > abc.txt"와 같은 경우와는 달리, 리다이렉션 기호의 앞뒤에서 명령을 실행할 경로를 확인해야 할 때 리다이렉션 기호가 comm에서의 어느 인덱스에 저장되어 있는지 알아야 한다. 기본값을 0으로 설정하여 리다이렉션이 있는지 없는지를 구분하였다. [위의 for문 아래의 if문 참고]

변수 pid1은 명령을 수행할 자식 프로세스의 반환값을 받을 변수이고, pid2와 fd는 네임드 파이프를 수행하기 위해 사용되는 변수이다.

```
for (i = 1; comm[i] != NULL; i++) {
    if (strcspn(comm[i], symbols) == 0) {
        redir = i;
        path = (redir == 2) ? comm[redir-1] : "./";
        break;
    }
}
```

뒤에 나오는 for문은 사용자가 입력한 명령문에서 리다이렉션 기호를 찾는 역할을 한다. strcspn() 함수는 첫 번째 인자의 문자열에서 두 번째 인자의 문자들의 목록 중 처음으로 일치하는 문자의 인덱스를 반환한다. 리다이렉션 기호가 있다면 첫 번째 인덱스에서 하나의 문자만 검사하면 되므로 반환 결과가 0 인지 검사한다. 만약 일치하는 문자가 없다면, 첫 번째 인자의 문자열의 길이를 반환한다.

변수 redir 에는 사용자가 입력한 명령문(comm)에서 리다이렉션 기호의 위치 i를 저장하고, path에서는 다른 경로에서 명령을 실행하였다면 기호 앞의 경로를 저장하고 아닐 경우 디폴트 값으로 현재 디렉토리의 경로를 저장한다. 즉, "ls /var | more"라는 명령이 들어올 경우, path 에는 "/var" 가 저장된다. 반면 "ls | more"이 들어올 경우 path에는 현재 작업 디렉토리 경로 "./"가 저장된다.

- 리다이렉션 소스 설명

```
if (redir != 0) {
    if (strcmp(comm[redir], ">") == 0) {
        if ((pid1 = fork()) < 0) {
            fprintf(stderr, "minish : fork error\n");
            return -1;
        } else if (pid1 == 0) {
            fd[0] = open(comm[redir+1], O_RDWR|O_CREAT|S_IROTH, 0644);
            if (fd[0] < 0) {
                perror("> open error");
                exit(-1);
            }
            dup2(fd[0], STDOUT_FILENO);
            close(fd[0]);
            execlp(*comm, *comm, path, (char*)0);
            exit(0);
        }
    }
}
```

```

    } else if (strcmp(comm[redir], "<") == 0) {
        if ((pid1 = fork()) < 0) {
            fprintf(stderr, "minish : fork error\n");
            return -1;
        } else if (pid1 == 0) {
            fd[0] = open(comm[redir+1], O_RDONLY);
            if (fd[0] < 0) {
                perror("< open error");
                exit(-1);
            }
            dup2(fd[0], STDIN_FILENO);
            close(fd[0]);
            if (strcmp(comm[0], "ls") != 0) path = comm[redir+1];
            execlp(*comm, *comm, path, (char*)0);
            exit(0);
        }
    } else if (strcmp(comm[redir], "|") == 0) {
        // ...
    }
} else {
    // 리다이렉션이 없는 경우
}

```

리다이렉션 기호가 ">" 또는 "<"인 경우 각각의 if문에서 파일을 open 할 때의 경로는 리다이렉션 기호 뒤에 파일 경로가 있는 것으로 간주하고 comm[redir+1] 을 open()의 첫 번째 인자로 준다. [파란색 글씨 참조]

리다이렉션 기호가 ">"일 때는 dup2() 함수로 open한 파일 디스크립터를 표준 출력(stdout) 파일 디스크립터가 참조하도록 해서 터미널에 출력되는 내용을 fd[0]이 참조하는 파일에 출력되도록 하고 fd[0]은 닫아준다. 이후 execlp 함수를 실행시킨다. exec계열의 함수 호출 후에는 open된 파일 디스크립터는 유지되는 속성이므로 다른 수정 없이 실행할 프로그램을 인자로 주고, path를 실행할 프로그램의 인자로 준다.

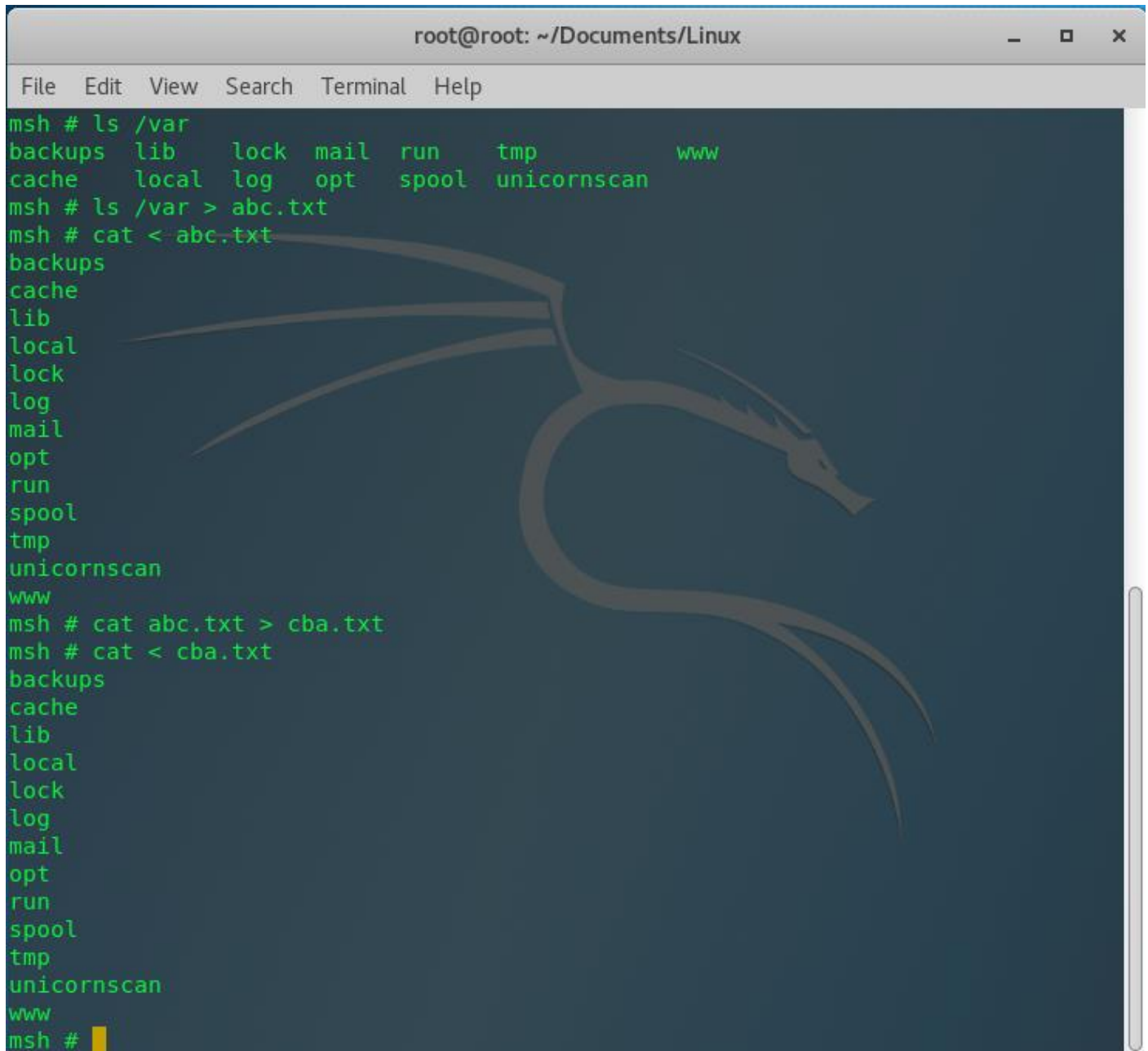
예를 들어, "ls > abc.txt"라면 ls의 결과를 터미널 창이 아닌 abc.txt 파일에 출력한다. 또는 "ls /var > abc.txt" 라면 /var라는 경로에서 실행한 ls 명령의 결과를 터미널 창이 아닌 abc.txt 파일에 출력하게 된다.

리다이렉션 기호가 "<"일 때는 dup2() 함수로 open한 파일 디스크립터를 표준 입력(stdin) 파일 디스크립터가 참조하도록 해서 파일의 출력 결과를 터미널 창에 입력되도록 하고 fd[0]은 닫아준다. 이후 path를 수정하는 if문이 있는데, [보라색 글씨 참조] 이는 명령어 cat 을 위한 if문이다. 이 명령은 리다이렉션 기호가 없을 때는 인자 배열에서 경로를 받아서 실행할 수 있는데, 리다이렉션 기호를 포함하게 되면 cat 명령을 실행할 파일의 경로를 지정해줘야 하기 때문이다. 명령어들은 세 가지 stdin, stdout, stderr 파일 디스크립터를 열고 시작하므로, 표준 입력이 참조하는 파일 디스크립터가 가리키는 파일(여기서 comm[redir+1])의 경로를 cat의 인자로 지정해준다.

왜 "<"에서는 cat을 위한 path를 설정하고, ">"에서는 path를 수정하는 if문이 없는가?

예를 들어, "cat abc.txt > cba.txt"라는 명령을 사용자가 입력했다고 가정하자. 이 경우 cat은 abc.txt의 내용을 입력받아 cba.txt 파일에 출력할 것이다. 즉, cat이 실행될 경로는 리다이렉션 기호의 뒤에 있는 "cba.txt" 파일이 아니라 리다이렉션 기호의 앞에 있는 "abc.txt"이다. 리다이렉션 기호의 뒤에 있는 "cba.txt" 파일은 표준 출력이 참조하는 파일 디스크립터의 대상이 되는 파일이므로, cat과는 아무 관련이 없다. 명령어 cat이 abc.txt의 내용을 입력 받아서 표준 출력을 할 때, 표준 출력은 "cba.txt" 파일의 파일 디스크립터를 참조하고 있으므로 "cba.txt"에 cat의 결과가 출력된다.

- 리다이렉션 실행 결과

A terminal window titled 'root@root: ~/Documents/Linux' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
msh # ls /var
backups  lib      lock    mail    run      tmp      www
cache    local   log     opt     spool    unicornscan
msh # ls /var > abc.txt
msh # cat < abc.txt
backups
cache
lib
local
lock
log
mail
opt
run
spool
tmp
unicornscan
www
msh # cat abc.txt > cba.txt
msh # cat < cba.txt
backups
cache
lib
local
lock
log
mail
opt
run
spool
tmp
unicornscan
www
msh #
```



```
root@root: ~/Documents/Linux
File Edit View Search Terminal Help
msh # ls > abc.txt
msh # ls
Desktop      Music      Templates  abc.txt    auto_ping.o
Documents    Pictures   Videos    auto_ping
Downloads     Public    'VirtualBox VMs' auto_ping.c
msh # cat < abc.txt
Desktop
Documents
Downloads
Music
Pictures
Public
Templates
Videos
VirtualBox VMs
abc.txt
auto_ping
auto_ping.c
auto_ping.o
msh # cat abc.txt
Desktop
Documents
Downloads
Music
Pictures
Public
Templates
Videos
VirtualBox VMs
abc.txt
auto_ping
auto_ping.c
auto_ping.o
msh #
```

```
root@root: ~/Documents/Linux
File Edit View Search Terminal Help
msh # ls
Desktop      Downloads  Pictures    Templates  'VirtualBox VMs'  auto_ping.c
Documents    Music      Public      Videos    auto_ping          auto_ping.o
msh # ls /var
backups  lib    lock  mail  run  tmp    www
cache    local log  opt   spool unicornscan
msh # ls /var/www
html
msh # ls > abc.txt
msh # ls
Desktop      Music      Templates  abc.txt    auto_ping.o
Documents    Pictures   Videos    auto_ping
Downloads     Public    'VirtualBox VMs' auto_ping.c
msh # cat < abc.txt
Desktop
Documents
Downloads
Music
Pictures
Public
Templates
Videos
VirtualBox VMs
abc.txt
```

- 네임드 파이프 소스 설명

```
if (redir != 0) {
    if (strcmp(comm[redir], ">") == 0) {
        // ...
    } else if (strcmp(comm[redir], "<") == 0) {
        // ...
    } else if (strcmp(comm[redir], "|") == 0) {
        pipe(fd);
        if ((pid1 = fork()) < 0) {
            fprintf(stderr, "minish : fork error\n");
            return -1;
        } else if (pid1 == 0) {
            dup2(fd[1], STDOUT_FILENO);
            close(fd[0]);
            close(fd[1]);
            execlp(*comm, *comm, path, (char*)0);
            fprintf(stderr, "minish : command not found\n");
            exit(127);
        }
        if ((pid2 = fork()) < 0) {
            fprintf(stderr, "minish : fork error\n");
            return -1;
        } else if (pid2 == 0) {
            dup2(fd[0], STDIN_FILENO);
            close(fd[0]);
            close(fd[1]);
            execvp(comm[redir+1], &comm[redir+1]);
            fprintf(stderr, "minish : command not found\n");
            exit(127);
        }
        close(fd[0]);
        close(fd[1]);
        wait();
    }
}
```


파일 디스크립터 배열 fd는 파이프의 입출력 각각을 참조하며 표준 입출력에 연결된다.

첫 번째 fork를 통해 생성된 자식 프로세스

파이프의 입력에 연결된 파일 디스크립터 fd[1]을 표준 출력(stdout) 파일 디스크립터가 참조하여, 첫 번째 자식이 실행하는 명령의 출력 결과를 파이프에 저장한다. 이후, 파이프의 입출력 파일 디스크립터들을 닫는다. 변수 path를 사용하여 경로를 지정해준다. "ls | more" 과 "ls /var | more"에서 ls가 실행될 경로를 구분하기 위한 용도이다.

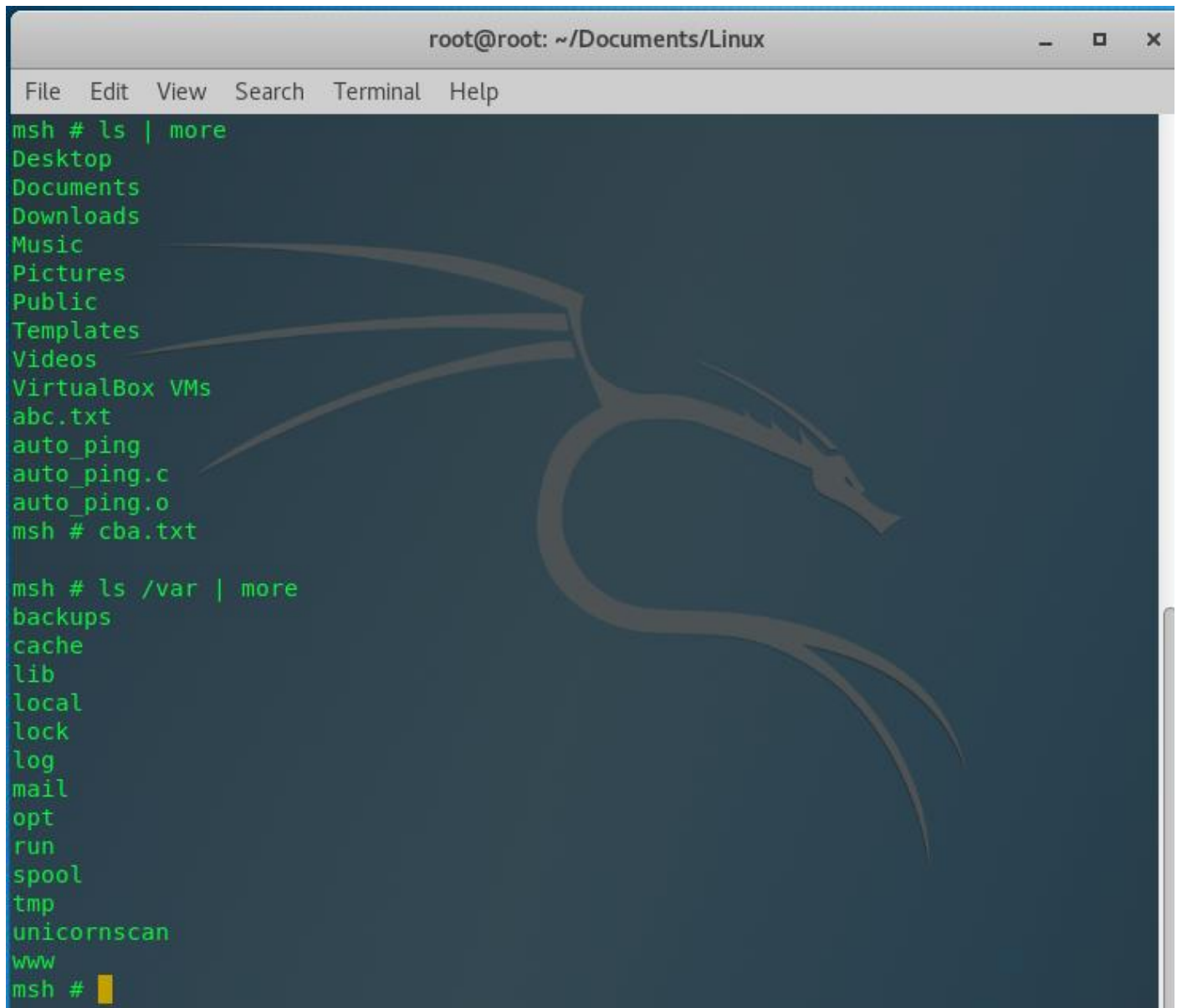
두 번째 fork를 통해 생성된 자식 프로세스

파이프의 출력에 연결된 파일 디스크립터 fd[0]를 표준 입력(stdin) 파일 디스크립터가 참조하여, 두 번째 자식이 실행될 때의 입력이 파이프의 출력과 연결되도록 한다. 결과적으로 파이프를 통해 두 자식 프로세스는 연결된다. 이후 파이프의 입출력 파일 디스크립터들을 닫는다.

두 번째 자식 프로세스에서 execlp가 아닌 execvp를 사용한 이유

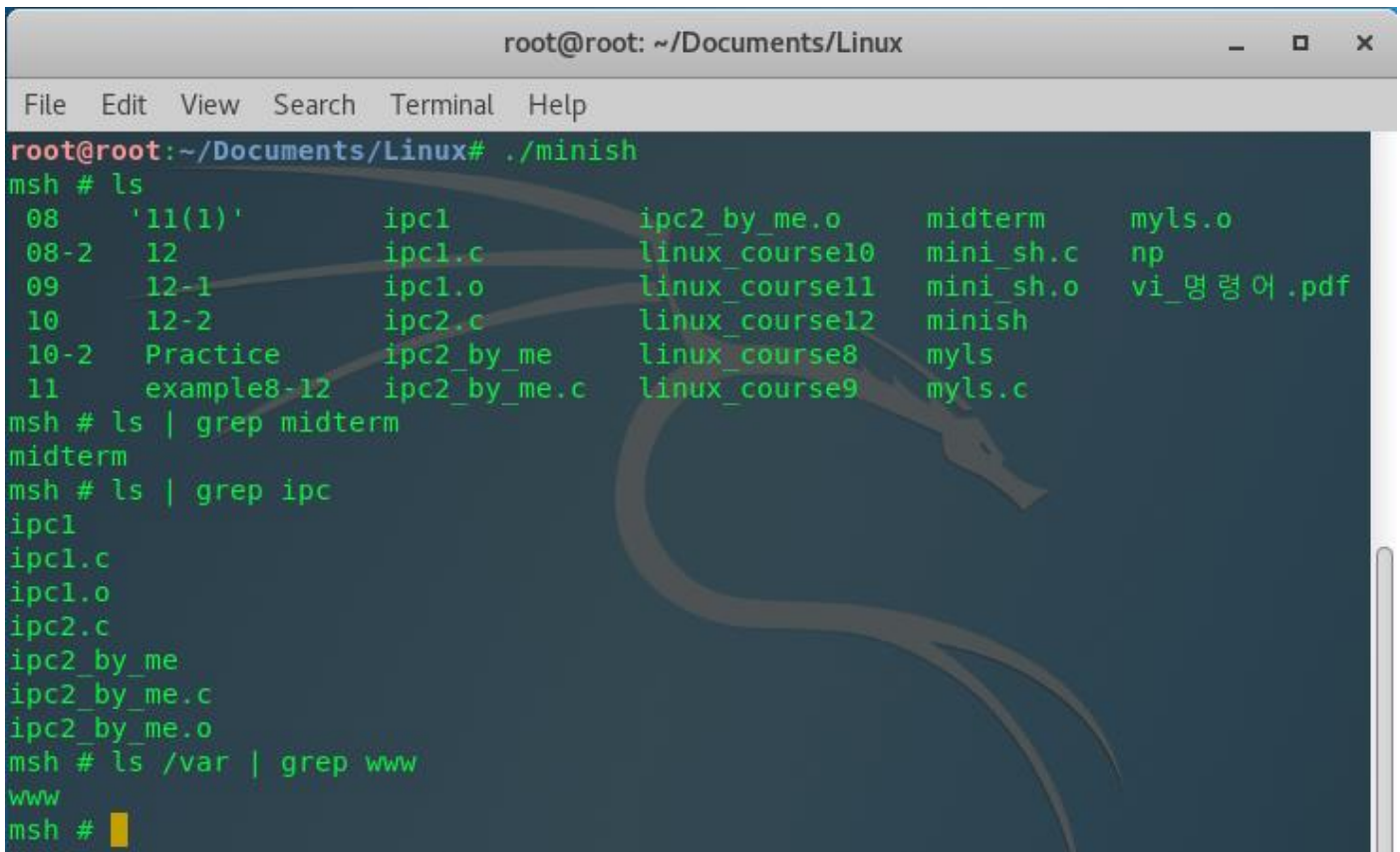
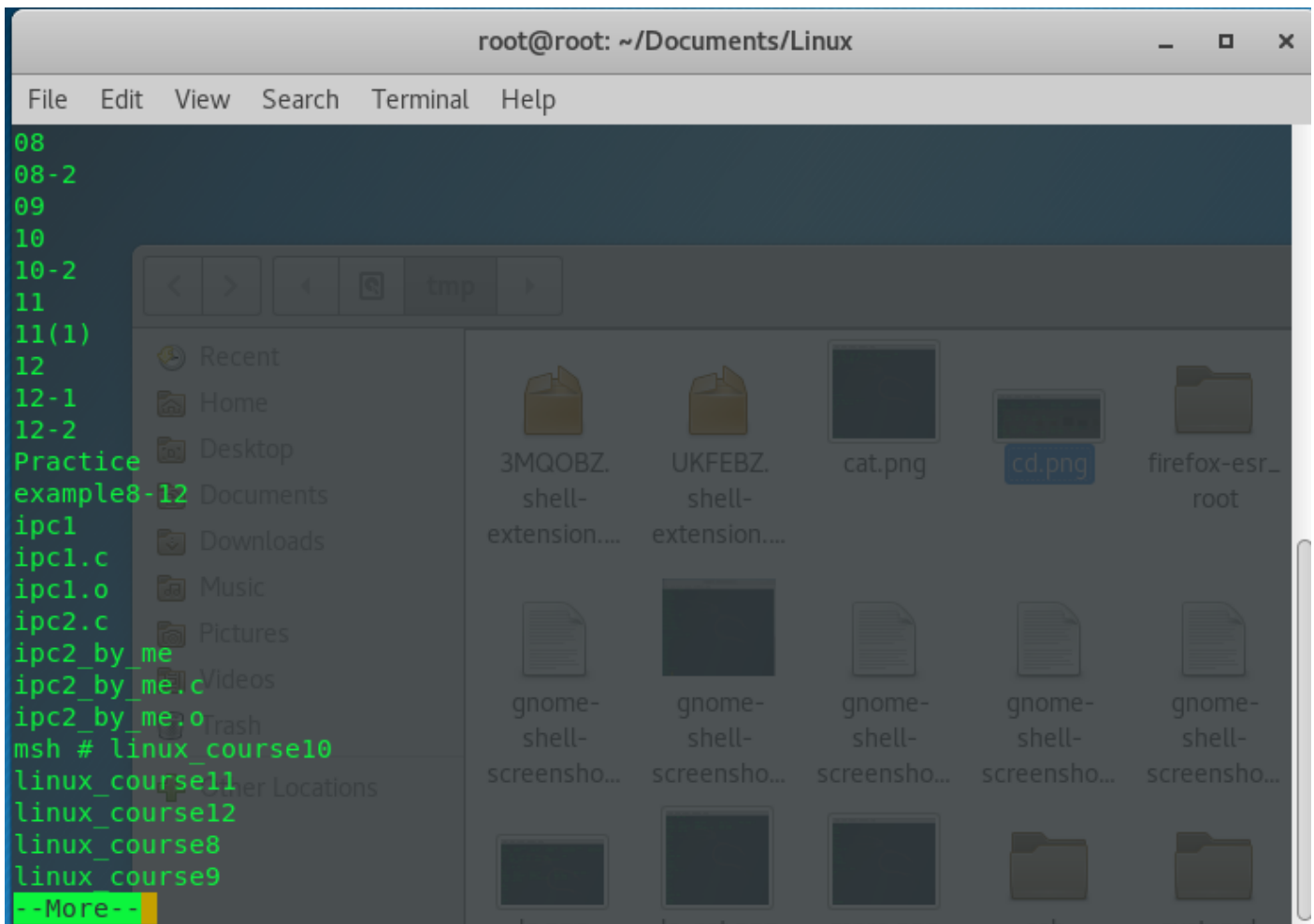
실행될 명령의 인자문제 때문이다. 파이프 기호 다음에 나오는 명령은 하나의 새로운 프로세스가 실행하는 명령어이기 때문에 execlp를 호출하게 되면 인자의 리스트를 하나하나 다 받아서 주어야 한다. 그러면 어떤 인자가 몇 개올지 알 수 없기 때문에 execvp를 통해서 벡터 형태로 파이프 기호 다음 인덱스부터 시작되는는 배열의 주소를 넘겨준다. 따라서 "ls | more" 뿐만 아니라 "ls | grep keyword" 또는 "ls /var | grep www"와 같은 인자를 요구하는 명령어도 처리할 수 있게 된다.

- 네임드 파이프 실행 결과



```
root@root: ~/Documents/Linux
File Edit View Search Terminal Help
msh # ls | more
Desktop
Documents
Downloads
Music
Pictures
Public
Templates
Videos
VirtualBox VMs
abc.txt
auto_ping
auto_ping.c
auto_ping.o
msh # cba.txt

msh # ls /var | more
backups
cache
lib
local
lock
log
mail
opt
run
spool
tmp
unicornscan
www
msh #
```



- 리다이렉션 없는 경우

```
} else {  
    if ((pid1 = fork()) < 0) {  
        fprintf(stderr, "minish : fork error\n");  
        return -1;  
    } else if (pid1 == 0) {  
        if (strcmp(comm[0], "ls") == 0 && comm[1] == NULL) {  
            comm[1] = "./";  
            comm[2] = NULL;  
        }  
        execvp(*comm, comm);  
        fprintf(stderr, "minish : command not found\n");  
        exit(127);  
    }  
}
```

리다이렉션이 없을 때는 ls 명령어를 인자 없이 입력할 경우 현재 작업 디렉토리 경로를 요구한다. "ls ./"라고 입력해야만 결과가 출력되는 것이 인자를 지정했기 때문이다. 따라서 인자가 없는 ls 명령어에 한하여 포인터 배열 comm을 위의 if문 내부처럼 수정해준다. 그리고 execvp를 호출하면, "ls"라고 입력하더라도 현재 작업 디렉토리를 기준으로 명령을 실행할 수 있다.