OS 과제

3. 나만의 쉘 만들기

20122382

컴퓨터학부

박 준 은

목차

[I. 서론 3](#_Toc469164191)

[① 연구분야 3](#_Toc469164192)

[② 목적 및 접근방법 3](#_Toc469164193)

[II. 관련연구 3](#_Toc469164194)

[① Shell 3](#_Toc469164195)

[② 환경변수 4](#_Toc469164196)

[③ Wait() & signal() 함수 4](#_Toc469164197)

[④ fork 5](#_Toc469164198)

[⑤ 쉘 스크립트 5](#_Toc469164199)

[⑥ 명령어를 만들기 위한 함수들 6](#_Toc469164200)

[III. 설계 7](#_Toc469164201)

[① 실행환경 및 make파일과 sh파일 7](#_Toc469164202)

[② mysh.c 8](#_Toc469164203)

[③ myset.c 11](#_Toc469164204)

[④ myenv.c 13](#_Toc469164205)

[⑤ myps.c 14](#_Toc469164206)

[⑥ myjobs.c 18](#_Toc469164207)

[⑦ mycd.c 21](#_Toc469164208)

[⑧ mypwd 22](#_Toc469164209)

[⑨ myls.c 22](#_Toc469164210)

[⑩ mycat.c 26](#_Toc469164211)

[⑪ myhead.c 27](#_Toc469164212)

[⑫ mytail.c 29](#_Toc469164213)

[⑬ mysort.c 30](#_Toc469164214)

[⑭ mycp.c 33](#_Toc469164215)

[⑮ mkdir & myrm 34](#_Toc469164216)

[IV. 결론 39](#_Toc469164217)

[V. 참고자료 39](#_Toc469164218)

# 서론

## 연구분야

Shell은 운영체제상에서 시스템과 이용자가 상호작용 할 수 있도록 시스템의 명령어를 해석해주는 프로그램이다. 명령어 해석기라고도 불리는 shell은 사용자가 입력한 명령어를 해석하여 커널에서 실행 될 수 있도록 해주는 명령어 해석기 기능과 프로그래밍 기능 그리고 사용자 환경 설정 기능이 있다. 쉘의 가장 간단한 동작 방식은 명령을 표준 입력으로부터 읽은 후, 명령을 싫애시킬 자식 프로세스를 fork하는 것이다. 부모 프로세스는 다른 명령을 읽어 들이기 전에 자식 프로세스가 조욜할 때까지 기다린다. 또한 쉘은 백그라운드 프로세스, 시그널 처리 외에 파이프라인과 Redirection을 처리한다. 본 과제에서는 파이프라인과 Redirection을 제외한 기타 기능들을 부분적으로 구현하였다.

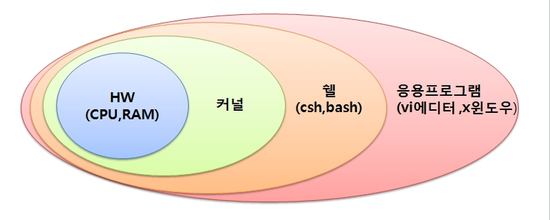
## 목적 및 접근방법

본 과제의 목적은 위에서 설명한 Shell의 기능을 부분적으로 구현하는데에 있다. 그동안 당연시하며 사용했던 ls,cd,mkdir 등의 명령어를 직접 구현 해 봄으로써, 리눅스의 전반적인 이해를 증진시키고, 백그라운드 실행을 통한 fork 사용법을 익힌다. 그럼으로써 프로스의 생성 과정과 종료. 그리고 ps 명령어를 구현해 각 프로세스의 정보를 읽어 오는 방법을 익힌다. 본 과제에서 만든 마이쉘은 하나의 프로그램으로써 동작한다. 마이쉘은 &기능을 통해 각 명령어를 백그라운드로 실행할 수 있는 기능을 포함하고 있다. 또한, SIG\_INT와 같은 인터럽트 시그널을 다루는 방법에 대해서도 익힌다.

# 관련연구

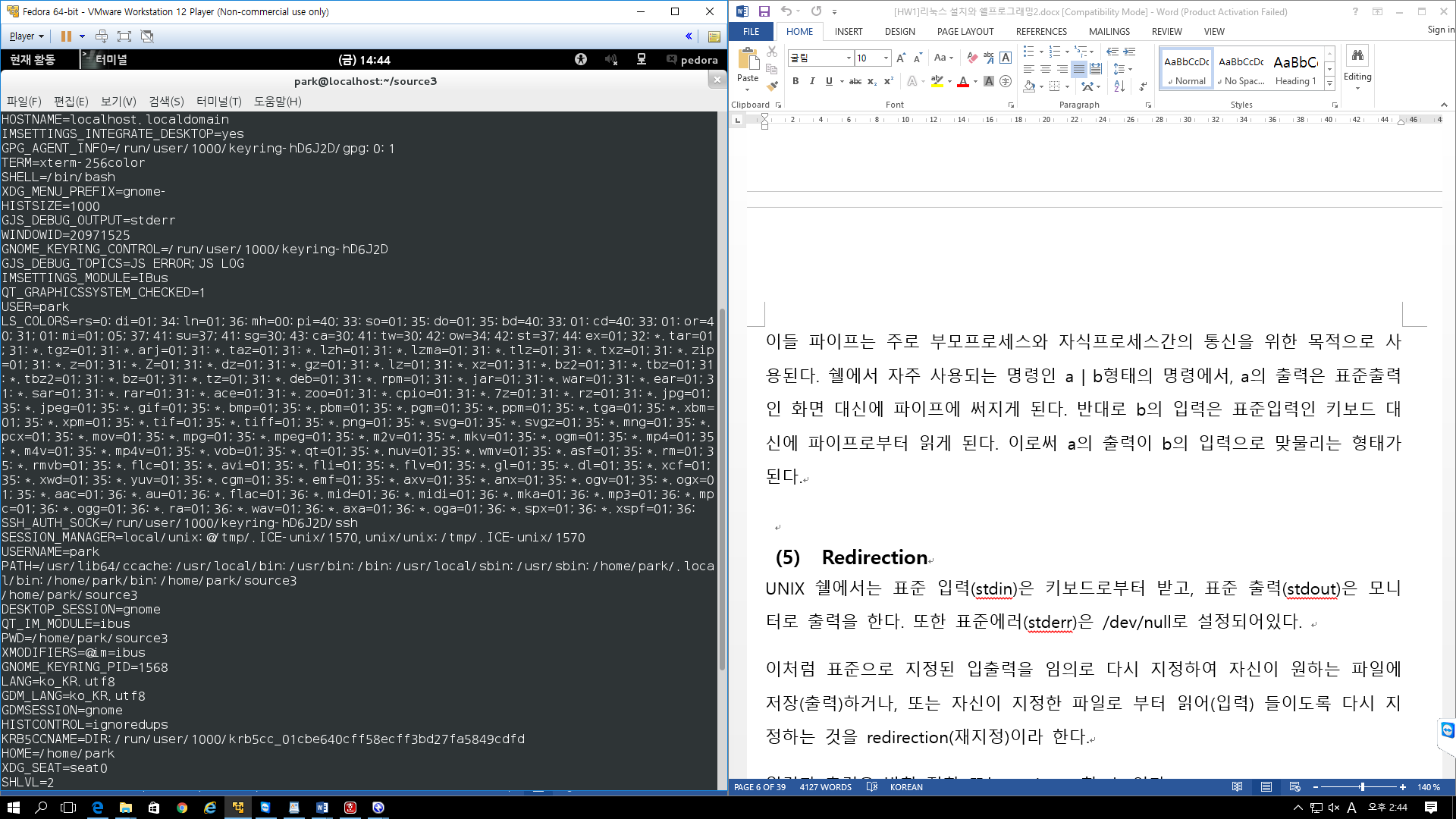
## Shell

Shell은 운영체제에서 사용자에게 다양한 운영체제 기능과 서비스를 구현하는 인터페이스를 제공하는 프로그램이다. Shell은 사용자와 운영체제의 커널 모두를 아우르는 일종의 껍데기 층이기 때문에 Shell 이라는 이름이 붙혀졌다. Shell은 커맨드라인을 읽고, 입력된 라인을 해석해 해당하는 명령을 실행시키는 기능을 가지고 있다. Shell은 명령 줄 Shell과 그래픽형 두 종류로 분류된다. 명령 줄 Shell은 운영쳊 상에서 명령 줄 인터페이스를 제공하고, 그래픽 Shell은 GUI를 제공한다.



## 환경변수

사용자들은 자신의 환경에 맞춰 환경을 최적화 시킬 수 있다. 이때 Shell 변수를 이용하게 된다. Shell 변수란, 이름을 가지면서 어떤 형태의 값을 나타내는 항목이다. Shell에서는 환경변수로 사용되는데, 이 환경변수는 실행 프로그램들에 운영체제의 환경과 실행 프로그램의 환경을 알려주기 위해 사용된다. 시스템 환경변수는 후에 우리가 구현해 볼 env명령이나 set명령어를 통해서 확인이 가능하다. 이 환경변수는 사용자가 어떤 터미널을 이용하는지, 어떤 언어를 사용하는지에 대한 정보를 가지고 있다. 아래는 본 과제를 실행한 터미널의 환경변수이다.



## Wait() & signal() 함수

fork함수를 통해서 생성된 자식 프로세스는 exit을 호출하거나 return을 통한 값을 통해 운영체제 종료를 알린다. 운영체제는 종료 내용이 운영체제에서 부모 프로세스에게 전달될 때까지 자식 프로세스를 소멸시키지 않는다. 이러한 상황의 프로세스를 좀비 프로세스라고 한다. 그리고 이런 좀비 프로세스는 종료되지 않고 리소스를 잡아먹고 있다. 따라서 자식 프로세스를 생성할 경우 좀비 프로세스가 되지 않도록 부모 프로세스가 자식 프로세스를 기다릴 필요가 있다. 이때 사용하는 함수가 wait함수와 waitpid함수이다. wait함수는 호출 시점에서 종료된 자식 프로세스가 없다면 임의의 자식 프로세스가 종료될 때까지 블로킹될 수 있다. 이러한 단점을 보완하는 함수가 waitpid이다. waitpid함수의 세번째 인자로 WNOHANG을 주게 되면 종료된 자식 프로세스가 없을 때 블로킹하지 않고 0을 리턴한다.

Wait 함수는 자식 프로세스의 종료를 알기 위해 계속해서 호출해야 한다는 단점이 있다. 이러한 단점을 보완하기 위해 signal을 사용할 수 있다. 특정 signal을 등록 해주고 시스템은 계속해서 다른 일을 하다가 기다리던 사건이 발생하면 운영체제가 특정 신호을 발생시켜 등록된 시그널 함수로 알려줘 프로세스가 끝난 것을 알게 된다. 이런 시그널을 알려주는 함수에는 signal함수와 sigaction함수가 있다.

## fork

fork 함수는 프로세스 생성 함수이다. 프로세스는 복제되는 부모 프로세스와 복제된 자식 프로세스로 나뉘게 된다. 두개의 프로세스는 같은 코드를 사용하지만 서로 다른 PID와 PCB를 가지게 된다. 복사되기 때문에 부모 프로세스가 가지고 있던 메모리 등의 시스템 자원을 공유하게 된다. 하지만 복사가 된 이후에는 서로 자원을 공유하지 않기 때문에 프로세스간 자원을 공유하기 위해서 메모리 공유나 파이프를 이용하는것과 같은 방식을 사용해야한다. 프로세스가 분리 되었을 때 부모와 자식은 pid를 통해 구분하는데 pid가 0이라면 자식 프로세스이고, 양수이면 부모 프로세스, 음수일 경우 fork() 오류를 나타낸다.

## 쉘 스크립트

쉘 스크립트란 쉘에서 사용될 수 잇는 명령어들의 조합을 모아서 만든 배치파일이다. 리눅스에서는 여러 명령어들을 파이프, 리다이렉셔느 필터 등으로 연결하여 원하는 결과를 얻어 낼 수 있다. 이런 방식으로 묶여진 명령어 조합이 반복적으로 사용된다면 이를 쉘 스크립트로, 즉 단일 명령으로 만들어 쉽게 사용할 수 있다.

스크립트라고 하는 것은 인터프리터에 의해 해석/실행되는 프로그램을 말한다. 어떤 종류의 인터프리터를 사용하는 가에 따라서 어떤 스크립트인가에 대한 이름이 정해진다. 쉘 스크립트, 펄(Perl) 스크립트 등의 이름에서 사용하는 인터프리터를 알 수 있다. 쉘 스크립트는 인터프리터로 쉘을 사용하는 스크립트를 가르킨다. 또한 어떠한 쉘을 사용하는 가에 따라서 본 쉘 스크립트, C 쉘 스크립트, 콘 쉘 스크립트 등으로 나뉜다.

## 명령어를 만들기 위한 함수들

[chdir(), mkdir(), rmdir(), access(), remove(), opendir(), getcwd(), setenv(), getenv()]

|  |  |
| --- | --- |
| int chdir( const char \*dirname );  //dirname : 변경할 디렉토리의 경로  //반환값 : 정상 일 때 0, 에러 시 -1 | 현재 작업 디렉토리를 변경한다. |
| int mkdir( const char \*dirname );  //dirname : 생성할 디렉토리 경로와 이름ㅠ  //반환값 : 정상 일 때 0, 에러 시 -1 | 디렉토리를 생성한다. 만들려는 폴더가 이미 존재하면 errono는 EEXIST, 폴더가 부정확하면 ENOENT이다. |
| int rmdir( const char \*dirname );  //dirname : 삭제할 디렉토리 경로  //반환값 : 정상 일 때 0, 에러 시 -1 | 디렉토리를 삭제한다. 단 디렉토리 내에 파일이 존재하거나 사용중이면 삭제할 수 없다. |
| int access( const char \*path, int mode );  //path : 파일이나 폴더의 전체 경로 또는 이름  //mode : 파일이나 폴더의 존재 유무 검사 또는 권한 조사  //반환값 : 정상 시 0, 에러 시 -1 | 파일이나 디렉토리의 존재 유무, 혹은 파일의 권한 및 속성을 조사할 때 사용한다. 주로 파일 및 디렉토리가 존재하는 지 조사할 때 사용한다. |
| int remove( const char \*path );  //path : 파일 경로  //반환값 : 정상 일 때 0, 에러 시 -1 | 지정한 경로의 파이릉ㄹ 삭제한다. 읽기 전용, 숨김, 시스템 속성 등을 갖는 파일은 삭제할 수 없다. 또 현재 사용중인 파일도 삭제할 수 없다. |
| DIR \*opendir(const char \*name);  //name : 열기 대상 디렉토리  //반환값 : 성공시 DIR포인터, 실패시 NULL | 지정된 디렉토리를 연다. 디렉토리를 파일처럼 열기를 한다는 것은 디렉토리 내에 파일과 서브 디렉토리 검색을 한다는 의미. |
| char \*getcwd( char \*buffer, int maxlen );  //buffer : 파일의 경로  //maxlen : 파일에 설정할 접근 모드 값  //반환값 : 정상 일 때 0, 에러 시 -1 | 현재 작업중인 디렉토리의 경로를 얻는다. 버퍼가 작으면 에러가 발생할 수 있으므로 크게 잡는 것이 좋다. |
| int setenv(const char \*envname, const char \*envval, int overwrite)  //envname : 환경변수 이름  //enval : 변수 값  //overwrite : 이미 같은 이름의 변수가 있다면 값을 변경할 지 여부를 결정  //반환값 : 성공하면 0, 실패하면 -1 | 환경 변수 목록 중에 변수값을 수정하거나 추가할 수있다. 수정된 변수 값이나 새로 추가된 환경 변수값은 실행중인 프로그램에서만 유효하며, 외부적으로는 변경되지 않는다. |
| char \*getenv(const char \*name);  //name : 구하려는 환경 변수의 이름  //반환값 : 환경 변수의 값 | 환경변수 목록 중에 원하는 변수값을 구한다. |

# 설계

## 실행환경 및 make파일과 sh파일

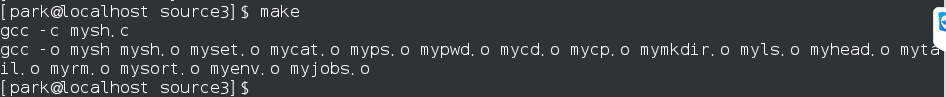
[실행환경]

* 페도라18버전 (3.6.2). 64비트 virtual machine
* CPU : intel i5-4690
* Ram : 8G

아래는 Makefile 이다. make 명령어를 이용해 한번에 컴파일 할 수있다.

OBJECTS = mysh.o myset.o mycat.o myps.o mypwd.o mycd.o mycp.o mymkdir.o myls.o myhead.o mytail.o myrm.o mysort.o myenv.o myjobs.o

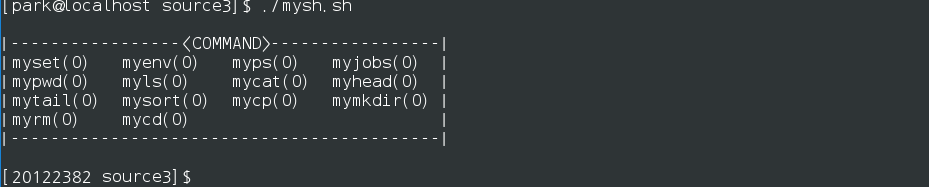
2. mysh : $(OBJECTS)
3. gcc -o mysh $(OBJECTS)
5. mysh.o : mysh.c
6. gcc -c mysh.c
7. myset.o : myset.c
8. gcc -c myset.c
9. mycat.o: mycat.c
10. gcc -c mycat.c
11. myps.o: myps.c
12. gcc -c myps.c
13. mypwd.o: mypwd.c
14. gcc -c mypwd.c
15. mycd.o: mycd.c
16. gcc -c mycd.c
17. mycp.o: mycp.c
18. gcc -c mycp.c
20. mymkdir.o: mymkdir.c
21. gcc -c mymkdir.c
23. myls.o: myls.c
24. gcc -c myls.c
26. myhead.o: myhead.c
27. gcc -c myhead.c
29. mytail.o: mytail.c
30. gcc -c mytail.c
32. myrm.o: myrm.c
33. gcc -c myrm.c
35. mysort.o: mysort.c
36. gcc -c mysort.c
38. myenv.o: myenv.c
39. gcc -c myenv.c
41. myjobs.o: myjobs.c
42. gcc -c myjobs.c



mysh.sh 파일은 다음과 같이 작성되었다.

1. #!/bin/bash
2. export PATH=$PATH:$(pwd)
3. ./mysh

쉘 스크립트를 이용해 환경변수에 내 현재 경로를 추가한 후 ./mysh 파일을 실행시킨다.



## mysh.c

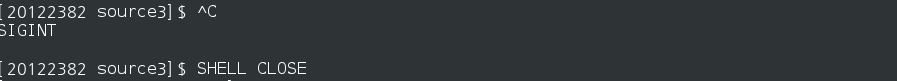
1. **void** handler(**int** signo){
2. **if**(signo==SIGINT)
3. printf("\nSIGINT\n");
4. }
6. **void** setenvironment(){
7. **char** \*pwd = (**char** \*)malloc(100);
8. **char** \*envi;
10. //get current path
11. getcwd(pwd,100);
12. envi=getenv("PATH");
13. //add path
14. sprintf(envi,"%s:%s",envi,pwd);
16. //set path
17. setenv("PATH",envi,1);
19. }
21. **void** help(**char** str[]){
23. **if**(!strcmp(str,"myset")){
24. printf("|====================================================|\n");
25. printf("|This Command is for set environment                 |\n");
26. printf("|myset PATH [option] [path]                          | \n");
27. printf("|-n option : overwrite path                          |\n");
28. printf("|-a option : add new path at the end of PATH         |\n");
29. printf("|path is string                                      |\n");
30. printf("|====================================================|\n");

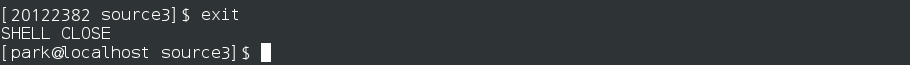
위의 코드는 처음 main을 실행할 때 SIGINT를 처리하는 부분과 환경변수에 현재 내 PATH를 추가해주는 부분이다. 아래의 help()는 명령어 옵션으로 help를 주었을 때 나오는 도움말이다. handler 함수로 시그널을 처리한다. 사용자가 ctrl+c 를 입력해도 종료되지 않는다. setevironment()에서 내 환경변수 PATH에 내 현재경로를 추가한다.

다음은 EOF, exit 처리하는 부분이다.

1. fgets(command,MAX\_COMMAND,stdin);
3. //if command is exit or EOF
4. **if**(!strcmp(command,"exit\n") || feof(stdin)){
5. printf("SHELL CLOSE\n");
6. **return** 0;
7. }

feof()로 EOF가 입력되었다면 쉘을 종료하고, 사용자가 exit을 입력했을 시에도 쉘을 종료한다. 다음은 SIGINT 핸들과 exit, EOF를 입력해본 화면이다.

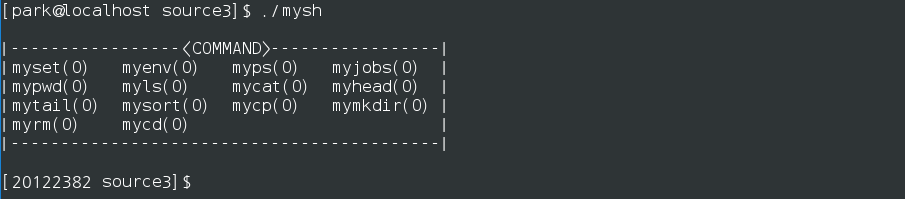




다음은 프롬프트를 띄워주는 코드이다.

1. getcwd (shell, MAX\_PATH);
2. tmp=strtok(shell,"/");
3. **while**(tmp=strtok(NULL,"/"))
4. strcpy(shell,tmp);
6. **if**(shell != 0)
7. printf("[20122382 %s]$ ",shell);

getcwd()로 현재 경로르 가져와 맨 마지막 디렉토리만 사용한다. 최종적으로 화면에 출력되는 모습은 다음과 같다.



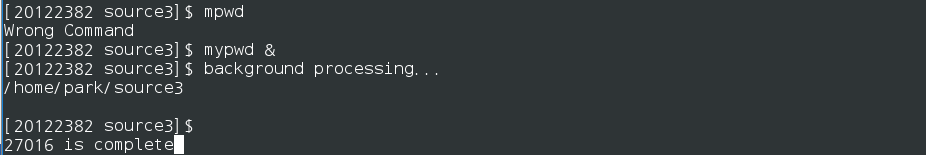
다음은 명령어를 실행하기 위해 fork()와 백그라운드(&) 실행을 하도록 하는 코드이다.

1. pid = fork();
3. **if**(pid == -1){
4. printf("ERROR : Fork Error");
5. **return** -1;
6. }
7. **else** **if**(pid ==0){
9. //background processing
10. **if**(strstr(command2,"&")){
11. pid2=fork();
13. **if**(pid2 > 0){
14. exit(0);
15. }
16. **else** **if**(pid2 == 0){
17. pid3 = fork();
19. **if**(pid3<0)
20. exit(0);
22. **else** **if**(pid3 == 0){
23. printf("background processing...\n");
24. **char** \*c = strstr(command2,"&");
25. //delete character &
26. strcpy(c,c+1);
27. }
28. **else**{
29. wait(NULL);
30. printf("\n%d is complete",pid3);
31. exit(0);
32. }
33. }
34. **else**{
35. printf("ERROR : Fork Error");
36. exit(0);
37. }
39. }
41. **if**(!strcmp(command,"mycat")){
42. **if**(strstr(command2,"--help")){
43. help(command);
44. **continue**;
45. }
46. mycat(command2);
48. }
49. **if**(pid3==0){
50. sleep(60);
51. exit(0);
52. }
53. return 0;
54. }
56. **else** **if**(pid>0){
58. // …….. //
59. myset(command2);
60. }
61. .. 명령어 생략 …

64. wait();
65. }
67. }

명령어를 실행하기위해 명령어 실행 전 main에서 fork()를 이용하는 부분과 ‘&’를 이용해 백그라운드 실행을 하는 코드이다. 기본적인 매커니즘은 다음과 같다. 먼저 명령어를 실행하기 전 fork()를 해서 main과 명령어 부분을 분리한다. 분리 후 부모 프로세스는 자식을 wait하고, 자식 프로세스는 명령어를 실행한다. 실행이 종료된 후 자식은 종료하고 부모는 wait()를 리턴받아 새로운 명령 프롬프트를 실행한다.

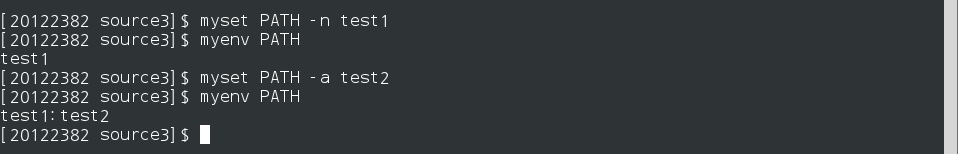
&를 입력해 명령을 실행했을 경우 백그라운드로 실행된다. 이때 백그라운드 프로세스로 만들기 위해 fork()를 두번 행한다. 처음 실행한 fork() (pid2)의 부모는 자식 프로세스를 고아로 만들기 위해 바로 종료한다. 자식 프로세스는 다시한번 fork() (pid3)를 실행해 자식프로세스에서 명령어를 실행하고, 부모 프로세스에서 자식 프로세스를 wait()한다. 자식 프로세스에서 명령어를 실행한 후 return이 되면 부모 프로세스는 wait()를 리턴받아 다시 프롬프트를 실행한다. 이때 자식프로세스가 백그라운드로 실행되는 것을 확인하기 위해 sleep(60)을 해 종료를 지연시켰다.



## myset.c

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
5. **int** myset(**char** \*command){
7. **char** \*path = {"PATH\0"};
8. **char** \*option;
9. **char** \*list;
10. **char** \*tmp;
12. //remove \n
13. command[strlen(command) -1] = '\0';
15. **if**(strstr(command," ")){
16. tmp = strtok\_r(command," ",&option);
18. //torkernize command
19. **if**(strcmp(path,tmp)){
20. printf("ERROR : arg1 is not PATH\n");
21. **return** -1;
22. }
24. **if**(!strstr(option," ")){
25. printf("ERROR : Wrong Option\n");
26. **return** -1;
27. }
28. **else**
29. option = strtok\_r(option," ",&list);
31. }
32. **else**{
33. printf("ERROR : No Path\n");
34. **return** -1;
35. }
37. //option -n
38. **if**(!strcmp(option,"-n")){
40. **if**(setenv(path,list,1)){
41. printf("ERROR : set evnvy error\n");
42. **return** -1;
43. }
44. }
45. //option -a
46. **else** **if**(!strcmp(option,"-a")){
48. //tmp for env
49. **char** \*tmp = getenv(command);
50. **char** \*addList = (**char** \*)malloc(100);
52. strcat(addList,tmp);
53. strcat(addList,":");
54. strcat(addList,list);
56. **if**(setenv(path,addList,1)){
57. printf("ERROR : set evnvy error\n");
58. **return** -1;
59. }
61. free(addList);
62. }
63. **else**{
64. printf("ERROR : Wrong Option\n");
65. **return** -1;
66. }
68. **return** 0;
70. }

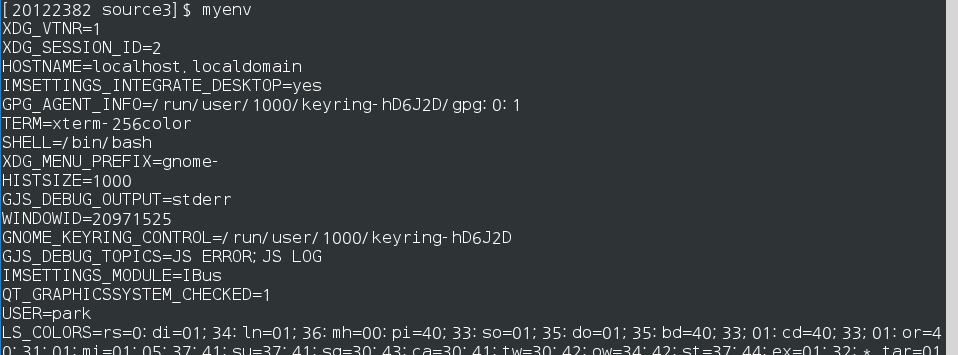
myset 명령어는 현재 설정되어있는 환경변수에 새로운 환경변수를 추가하는 명령어이다. 본 과제에서는 PATH에 새로운 경로 목록으로 덮어쓰는 기능을 가진다. 따라서 현재 환경변수를 불러온 후 PATH에 새로운 내용을 추가한다. 이때 경로 목록은 콜론으로 구분한다. –n 옵션일 경우 기존 PATH를 모두 지우고 사용자가 입력한 내용을 새로운 PATH로 가지며, -a 옵션일 경우 사용자가 입력한 내용을 새로운 경로 목록으로 추가한다. 이때 기존 shell의 환경변수에는 영향을 끼치지 않는다.



## myenv.c

1. #include<stdio.h>
2. #include<string.h>
3. #include<stdlib.h>
5. **extern** **char** \*\*environ;
7. **int** myenv(**char** \*command){
8. **int** i=0;
10. command[strlen(command) -1] = '\0';
12. **if**(!strcmp(command,"")){
14. **for**(i=0;environ[i] != NULL ; i++)
15. printf("%s\n",environ[i]);
17. }
18. **else** **if**(strstr(command," ")){
19. printf("ERROR : Wrong command\n");
20. **return** -1;
21. }
22. **else**{
23. **char** \*tmp = getenv(command);
24. **if**(tmp != NULL)
25. printf("%s\n",tmp);
26. }
28. **return** 0;
29. }

현재 설정된 환경변수 값을 가져온다. 현재 환경변수의 내용은 extern char environ 변수에 저장되어있다. 이변수의 내용을 전부 읽어오면 된다. myenv [Sring]과 같이 사용하며, String에 보고자하는 환경변수 값을 입력하면 원하는 환경변수의 값을 읽어올 수 있다.





## myps.c

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. #include<dirent.h>
5. #include<unistd.h>
6. #include<sys/stat.h>
7. #include<sys/types.h>
8. #include<time.h>
10. #define CPU\_CLOCK 100
12. **int** myps(**char** \*command){
14. command[strlen(command) -1] = '\0';
16. //check command
17. **if**(strstr(command," ")){
18. printf("ERROR : Wrong Command\n");
19. **return** -1;
20. }
21. **else** **if**(strcmp(command,"")){
22. printf("ERROR : Wrong Command\n");
23. **return** -1;
24. }
26. DIR \*dir;
27. **struct** dirent \*dirent;
28. **struct** stat fileInfo;
29. **struct** **tm** \*timeInfo;
30. **FILE** \*fp;
32. //get cpu time
33. **int** cpuSecond = uptime();
35. **if**(!(dir = opendir("/proc"))){
36. printf("ERROR : Can't open /proc\n");
37. **return** -1;
38. }

41. **int** thisid = getpid();
42. **char** ptty,ttypath[256];
43. **char** \*tmp,stat[1024];
45. sprintf(ttypath,"/proc/%d/stat",thisid);
47. fp=fopen(ttypath,"r");
48. fgets(stat,256,fp);
49. fclose(fp);
51. **int** k;
52. tmp = strtok(stat," ");
53. **for**(k=0;k<6;k++)
54. tmp=strtok(NULL," ");
55. //get this tty
56. **long** thistty = atol(tmp);
58. printf("   UID   PID   PPID   PGID   SID   STIME    TIME  CMD\n");
59. printf(" ----------------------------------------------------\n");
61. **while**(dirent = readdir(dir)){
62. **int** pid;
63. **int** ppid;
64. **int** pgid;
65. **int** sid;
66. **int** uid;
67. **long** stime;
68. **long** tty;
69. **int** ctime;
71. **char** cmdline[128];
72. **char** path[256];        //for file path
73. **char** status[1024];     //for stat
74. **struct** **tm** \*timeInfo;
76. **char** \*tmp;
77. **int** i;
79. //if hidden
80. **if**(!strcmp(dirent->d\_name,".") || !strcmp(dirent->d\_name,".."))
81. **continue**;
83. //check isDigit if pid=1, continue
84. //get pid
85. **if**((pid = atoi(dirent->d\_name)) <= 0 )
86. **continue**;
88. //open cmdline
89. //get cmd
90. sprintf(path,"/proc/%d/cmdline",pid);
91. fp = fopen(path,"r");
92. **if**(!fgets(cmdline,128,fp))
93. strcpy(cmdline,"?");
94. memset(path,0,256);
95. fclose(fp);
97. //open status file
98. //for get uid
99. sprintf(path,"/proc/%d/status",pid);
100. fp = fopen(path,"r");
102. **for**(i=0;i<7;i++)
103. fgets(status,1024,fp);
105. tmp=strtok(status,":");
106. tmp=strtok(NULL,"\t");
108. //get uid
109. uid = atoi(tmp);
111. memset(path,0,256);
112. fclose(fp);
114. //get ppid
115. sprintf(path,"/proc/%d/stat",pid);
117. fp = fopen(path,"r");
118. **if**(!fgets(status,1024,fp)){
119. printf("ERROR : Read Stat fail\n");
120. **return** -1;
121. }
122. fclose(fp);
123. memset(path,0,256);
125. //tokernize stat
126. //get ppid
127. tmp = strtok(status," ");
128. **for**(i=0;i<3;i++)
129. tmp=strtok(NULL," ");
130. ppid = atoi(tmp);
132. //get pgid
133. tmp=strtok(NULL," ");
134. pgid = atoi(tmp);
136. //get sid
137. tmp=strtok(NULL," ");
138. sid = atoi(tmp);
140. //get tty
141. tmp = strtok(NULL," ");
142. tty = atol(tmp);
144. **if**(tty != thistty)
145. **continue**;
147. //get 14th stat element for time
148. **int** e14;
149. **for**(i=0;i<7;i++)
150. tmp=strtok(NULL," ");
151. e14 = atoi(tmp);
153. //get 15th stat element for time
154. **int** e15;
155. tmp=strtok(NULL," ");
156. e15 = atoi(tmp);
158. //get stime, get 22th stat element for time
159. **int** e22;
160. **for**(i=0;i<7;i++)
161. tmp=strtok(NULL," ");
162. e22=atoi(tmp);
164. //get time
165. ctime = getcputime(e14,e15,e22,cpuSecond);
167. //current time
168. **time\_t** timer = time(NULL);
169. timeInfo = localtime(&timer);
171. //get stime  = current time - cput booting time + process running time
172. e22 = e22/CPU\_CLOCK;
173. stime = timer - cpuSecond + e22;

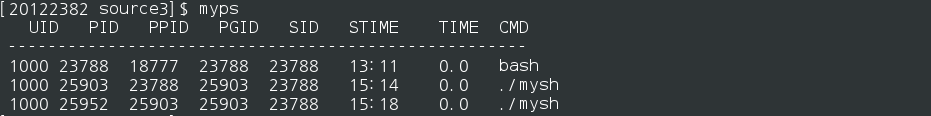
176. //current month and day
177. **int** mon = timeInfo->tm\_mon;
178. **int** day = timeInfo->tm\_mday;
180. //time\_t sstimer = stime;
181. **struct** **tm** \*stimeInfo = localtime(&stime);
183. //compare mon and stime's mon for print differ
184. **if**((stimeInfo->tm\_mon == mon) && (stimeInfo->tm\_mday == day))
185. printf("%5d %5d  %5d  %5d  %5d   %02d:%02d   %2d.%d   %s",uid,pid,ppid,pgid,sid,stimeInfo->tm\_hour,stimeInfo->tm\_min,ctime/10,ctime%10,cmdline);
187. **else**
188. printf("%5d %5d  %5d  %5d  %5d   %2d/%2d   %2d.%d   %s",uid,pid,ppid,pgid,sid,stimeInfo->tm\_mon+1,stimeInfo->tm\_mday,ctime/10,ctime%10,cmdline);
190. printf("\n");
192. }
194. closedir(dir);
196. **return** 0;
197. }
199. **int** getcputime(ulong utime, ulong stime, ulong starttime, **int** seconds)
200. {
201. unsigned **long** **long** total;
202. **int** pcpu=0;
204. total = utime + stime;
206. seconds = seconds - (**int**)(starttime/CPU\_CLOCK);
208. **if**(seconds)
209. {
210. pcpu = (**int**)(total \* 1000ULL/100.)/seconds;
211. }
213. **return** pcpu;
214. }

217. **int** uptime()
218. {
219. **FILE** \*fp;
220. **char** buf[36];
221. **double** stime;
222. **double** idletime;
224. **if** ((fp = fopen("/proc/uptime", "r")) == NULL)
225. {
226. printf("ERROR : fopen error");
227. **return** -1;
228. }
229. fgets(buf, 36, fp);
230. sscanf(buf, "%lf %lf", &stime, &idletime);
231. fclose(fp);
233. **return** (**int**)stime;
234. }

현재 실행중인 프로세스들의 상태를 보여주는 명령어이다. 보여주는 프로세스들은 tty별로 구분되어 현재 mysh 프로세스와 같은 tty를 가진 프로세스만 출력한다. getpid()를 통해 현재 프로세스의 pid를 가져오고, 그 pid를 이용해 tty를 가져왔다. 출력되는 내용은 /proc/[pid]/stat 또는 /proc/[pid]/status 에 저장되어있다. 파일입출력을 이용해 보여줄 내용에 해당하는 정보를 가지고 오면 된다. stat의 정보는 아래와 같다.

1 (systemd) S 0 1 1 0 -1 4202752 12626 1805651 47 961 13 190 5596 7344 20 0 1 0 3 52432896 1895 18446744073709551615 1 1 0 0 0 0 671173123 4096 1260 18446744073709551615 0 0 17 0 0 0 239 0 0 0 0 0 0 0 0 0 0

각각의 숫자와 문자가 해당 프로세스의 정보를 의미한다. 이 stat과 status에 나와있지 않은 내용이 STIME과 TIME 정보였다. STIME 정보를 구하기 위해 cpu uptime을 계산해 부팅된 시간을 구해 현재 시간에서 빼고, 프로세스가 실행된 시간을 더해준다. (현재시간sec – 부팅시간sec + 프로세스 실행시간clock) 여기서 프로세스 실행시간은 단위가 clock이므로 내 cpu의 clock을 가져와서 나누어주면 된다. 본 과제를 실행한 컴퓨터는 100 이었으므로 100으로 나누어주었다. TIME도 cpu uptime을 이용해 계산해주면 된다.



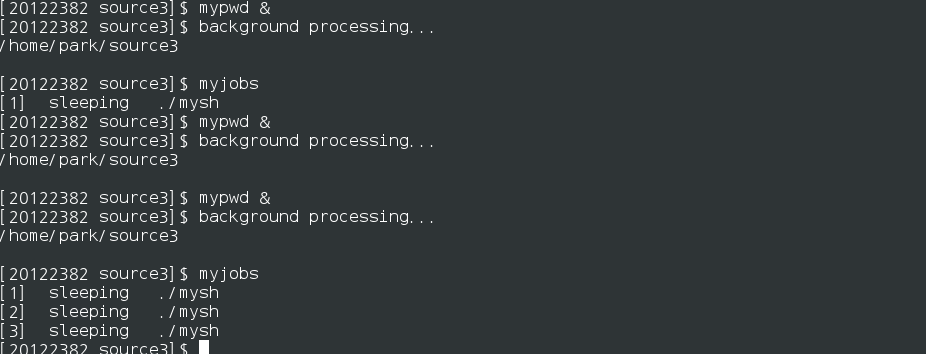
## myjobs.c

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. #include<dirent.h>
5. #include<unistd.h>
6. #include<sys/stat.h>
7. #include<sys/types.h>
8. #include<time.h>
10. #define CPU\_CLOCK 100
12. **int** myjobs(**char** \*command,**int** parentId){
14. command[strlen(command) -1] = '\0';
16. //check command
17. **if**(strstr(command," ")){
18. printf("ERROR : Wrong Command\n");
19. **return** -1;
20. }
21. **else** **if**(strcmp(command,"")){
22. printf("ERROR : Wrong Command\n");
23. **return** -1;
24. }
26. DIR \*dir;
27. **struct** dirent \*dirent;
28. **struct** stat fileInfo;
29. **struct** **tm** \*timeInfo;
30. **FILE** \*fp;
32. //get cpu time
33. **int** cpuSecond = uptime();
35. **if**(!(dir = opendir("/proc"))){
36. printf("ERROR : Can't open /proc\n");
37. **return** -1;
38. }

41. **int** thisid = getpid();
42. **char** ptty,ttypath[256];
43. **char** \*tmp,stat[1024];
45. sprintf(ttypath,"/proc/%d/stat",thisid);
47. fp=fopen(ttypath,"r");
48. fgets(stat,256,fp);
49. fclose(fp);
51. **int** k;
52. tmp = strtok(stat," ");
53. **for**(k=0;k<6;k++)
54. tmp=strtok(NULL," ");
55. //get this tty
56. **long** thistty = atol(tmp);
58. //get this pid and coutn num
59. **int** thisId = getpid();
60. **int** num=1;
62. **while**(dirent = readdir(dir)){
63. **int** pid;
64. **int** ppid;
65. **int** pgid;
66. **long** tty;
67. **int** ctime;
68. **char** pst[128];
70. **char** cmdline[128];
71. **char** path[256];        //for file path
72. **char** status[1024];     //for stat
73. **struct** **tm** \*timeInfo;
75. **char** \*tmp;
76. **int** i;
78. //if hidden
79. **if**(!strcmp(dirent->d\_name,".") || !strcmp(dirent->d\_name,".."))
80. **continue**;
82. //check isDigit if pid=1, continue
83. //get pid
84. **if**((pid = atoi(dirent->d\_name)) <= 0 )
85. **continue**;
87. //open cmdline
88. //get cmd
89. sprintf(path,"/proc/%d/cmdline",pid);
90. fp = fopen(path,"r");
91. **if**(!fgets(cmdline,128,fp))
92. strcpy(cmdline,"?");
93. memset(path,0,256);
94. fclose(fp);
96. //open status file
97. //for get uid
98. sprintf(path,"/proc/%d/status",pid);
99. fp = fopen(path,"r");
101. **for**(i=0;i<2;i++)
102. fgets(status,1024,fp);
104. tmp=strtok(status,":");
105. tmp=strtok(NULL,"(");
106. tmp=strtok(NULL,")");
107. strcpy(pst,tmp);
109. //get status
110. memset(path,0,256);
111. memset(status,0,256);
112. fclose(fp);
114. //get ppid
115. sprintf(path,"/proc/%d/stat",pid);
117. fp = fopen(path,"r");
118. **if**(!fgets(status,1024,fp)){
119. printf("ERROR : Read Stat fail\n");
120. **return** -1;
121. }
122. fclose(fp);
123. memset(path,0,256);
125. //tokernize stat
126. //get ppid
127. tmp = strtok(status," ");
128. **for**(i=0;i<3;i++)
129. tmp=strtok(NULL," ");
130. ppid = atoi(tmp);
132. //getpgid
133. tmp=strtok(NULL," ");
134. pgid=atoi(tmp);
136. //get tty
137. tmp = strtok(NULL," ");
138. tmp = strtok(NULL," ");
139. tty = atol(tmp);
141. **if**(tty != thistty)
142. **continue**;
144. **if**(pgid == parentId && pid != thisId && parentId != pid && ppid !=1)
145. printf("[%d]  %s\t%s\n",num++,pst,cmdline);

148. }
150. closedir(dir);
152. **return** 0;
153. }

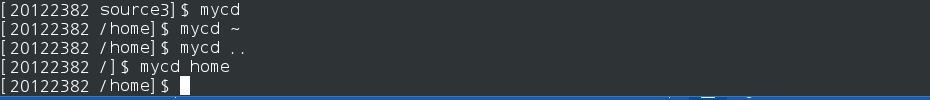
myjobs 명령어는 백그라운드로 실행중인 프로세스를 보여주는 명령어이다. 기본적인 매커니즘은 ps와 같다. /proc/[pid]/ 에 존재하는 status 파일과 stat 파일을 읽어 화면에 보여주면 된다. 백그라운드로 재생되는 프로세스임을 판별하는 방법으로 현재 실행중인 ./mysh 프로그램과 같은 PGID를 가진 프로세스를 선택하는 방법을 사용했다. 그중에서 ./mysh 프로그램과 myps 명령어는 제외했다.



## mycd.c

1. #include<stdio.h>
2. #include<string.h>
3. **int** mycd(**char** \*command){
5. command[strlen(command) -1] = '\0';
7. **if**(strstr(command," "))
8. printf("ERROR : Command Error\n");
10. //cd if no directory and ~
11. **if**(!strcmp(command,"") || !strcmp(command,"~")){
12. **if**(chdir("/home")<0)
13. printf("ERROr : Change Directory Fail!\n");
14. }
15. **else**{
16. //cd directory
17. **if**(chdir(command)<0)
18. printf("ERROr : Change Directory Fail!\n");
19. }
21. **return** 0;
22. }

디렉토리를 변경하는 명령어이다. chdir()를 사용해 디렉토리르 변경할 수 있다. ‘~’또는 아무런 입력도 하지 않으면 home으로 이동한다.



## mypwd

1. #include<stdio.h>
2. #include<stdlib.h>
4. **int** mypwd(**char** \*command){
6. //get pwd
7. **if**(!strcmp(command,"")){
8. **char** \*pwd = (**char** \*)malloc(100);
9. getcwd(pwd,100);
11. printf("%s\n",pwd);
12. free(pwd);
13. }
14. **else**
15. printf("Wrong Option\n");

18. **return** 0;
20. }

내 현제 경로를 출력해주는 명령어이다. getcwd()로 내 디렉토리의 경로를 가져올 수 있다.



## myls.c

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<unistd.h>
4. #include<string.h>
5. #include<pwd.h>
6. #include<dirent.h>
7. #include<grp.h>
8. #include<sys/stat.h>
9. #include<time.h>
10. **void** printRWX();
12. **int** myls(**char** \*command){
14. DIR \*dir;
15. **struct** passwd \*pw;
16. **struct** dirent \*dirent;
17. **struct** group \*group;
18. **struct** stat fileInfo;
19. **struct** **tm** \*timeInfo;
20. mode\_t fileMode;
22. **char** \*option1 = (**char** \*)malloc(256);
23. //filename
24. **char** \*fileName = (**char** \*)malloc(256);
26. command[strlen(command) -1] = '\0';
28. **if**(strstr(command," ")){
29. option1 = strtok\_r(command," ",&fileName);
31. **if**(strcmp(option1,"-a")){
32. printf("Wrong Option\n");
33. **return** -1;
34. }
35. }
36. **else**{
37. //option
38. **if**(!strcmp(command,"-a")){
39. strcpy(option1,"-a");
40. strcpy(fileName,"");
41. }
42. **else**{
43. strcpy(option1,"");
44. strcpy(fileName,command);
45. }
46. }
48. //check path
49. **char** path[256];
50. strcpy(path,getcwd(NULL,0));
52. **if**(strcmp(fileName,"")){
53. strcat(path,"/");
54. strcat(path,fileName);
55. }
56. **if**(fileName[0]=='/'){
57. strcpy(path,fileName);
58. }

61. //confirm fileName is file or directory
62. **int** isFile=0;
63. **if**((dir=opendir(path))==NULL){
64. //1 is fileName is File...
65. isFile=1;
66. //path=getcwd(NULL,0);
67. strcpy(path,getcwd(NULL,0));
68. //open directory
69. **if**((dir=opendir(path))==NULL){
70. printf("Fail Open Directory\n");
71. **return** -1;
72. }
73. }
75. //check exit file or directory
76. **int** check=0;
77. //read directory information
78. **while**(dirent = readdir(dir)){
80. **char** str[128];
81. sprintf(str,"%s/%s",path,dirent->d\_name);
82. stat(str, &fileInfo);
84. //skip if file is hidden mode
85. **if**(strcmp(option1,"-a") && dirent->d\_name[0] == '.') **continue**;
86. **else**{
87. **if**(isFile){
88. **if**(strcmp(fileName,dirent->d\_name)){
89. **continue**;
90. }
91. }
93. fileMode = fileInfo.st\_mode;
94. //print file mode
95. printRWX(fileMode);
96. printf(" %3d ",fileInfo.st\_nlink);
98. pw=getpwuid(fileInfo.st\_uid);
99. group=getgrgid(fileInfo.st\_gid);
100. printf("%s",pw->pw\_name);
101. printf(" %s ",group->gr\_name);
103. //print file size
104. printf(" %7d  ", fileInfo.st\_size);
106. **char** buf[100];
107. //file create time  and typqe
108. timeInfo=localtime(&(fileInfo.st\_mtime));
109. strftime(buf,**sizeof**(buf), " %m/%d %H:%M ",timeInfo);
110. printf(" %s ",buf);
112. //print name
113. **if**(dirent->d\_ino !=0)
114. printf(" %s\n",dirent->d\_name);
116. check=1;
117. }
118. }
119. closedir(dir);
121. **if**(check==0){
122. printf("There is no exit File or Directory\n");
123. **return** -1;
124. }
126. //    free(option1);
127. //    free(fileName);
129. **return** 0;
130. }

133. **void** printRWX(file\_mode){
135. //directory
136. **if**(S\_ISDIR(file\_mode))
137. printf("d");
138. **else** printf("-");
140. //user
141. **if**(file\_mode & S\_IRUSR)
142. printf("r");
143. **else** printf("-");
145. **if**(file\_mode & S\_IWUSR)
146. printf("w");
147. **else** printf("-");
149. **if**(file\_mode & S\_IXUSR)
150. printf("x");
151. **else** printf("-");
153. //group
154. **if**(file\_mode & S\_IRGRP)
155. printf("r");
156. **else** printf("-");
158. **if**(file\_mode & S\_IWGRP)
159. printf("w");
160. **else** printf("-");
162. **if**(file\_mode & S\_IXGRP)
163. printf("x");
164. **else** printf("-");
166. //ect
167. **if**(file\_mode & S\_IROTH)
168. printf("r");
169. **else** printf("-");
171. **if**(file\_mode & S\_IWOTH)
172. printf("w");
173. **else** printf("-");
175. **if**(file\_mode & S\_IXOTH)
176. printf("x");
177. **else** printf("-");
179. printf(" ");
181. }

내 현재 디렉토리 내의 파일 또는 디렉토리를 출력해주는 명령어이다. 따라서 DIR 포인터와 dirent 구조체를 이용해 해당 디렉토리의 내용을 읽어와야 한다. DIR 포인터를 이용해 해당 디렉토리를 열고 readdir로 읽어와 dirent 구조체에 넣고 각 파일의 정보들을 읽어온다. 읽었을 때 파일이름이 ‘.’이나 ‘..’일 경우 숨김파일이므로 출력하지 않는다. –a 옵션일 경우 모두 출력해준다. stat 구조체와 의미는 다음과 같다.

struct stat {  
     dev\_t           st\_dev;        /\*ID of device containing file \*/

     ino\_t            st\_ino;          /\*inode number\*/

     mode\_t       st\_mode;     /\*protection\*/

     nlink\_t         st\_nlink;       /\*number of hard links\*/

     uid\_t            st\_uid;          /\*user ID of owner\*/

     gid\_t            st\_gid;          /\*group ID of owner\*/

     dev\_t           st\_rdev;        /\*device ID (if special file)\*/

     off\_t             st\_size;         /\*total size, in byte\*/

     blksize\_t      st\_blksize;    /\*blocksize for file system I/O\*/

     blkcnt\_t       st\_blocks;     /\*number of 512B blocks allocated\*/

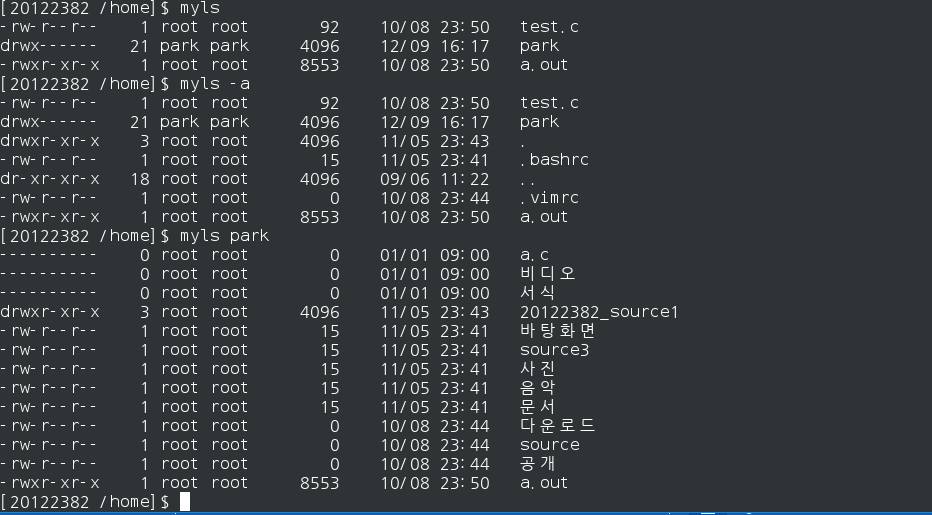
     time\_t;        st\_atime;      /\*time of last access\*/

     time\_t;        st\_mtime;     /\*time of last modification\*/

     time\_t         st\_xtime;       /\*time of last status change\*/

};

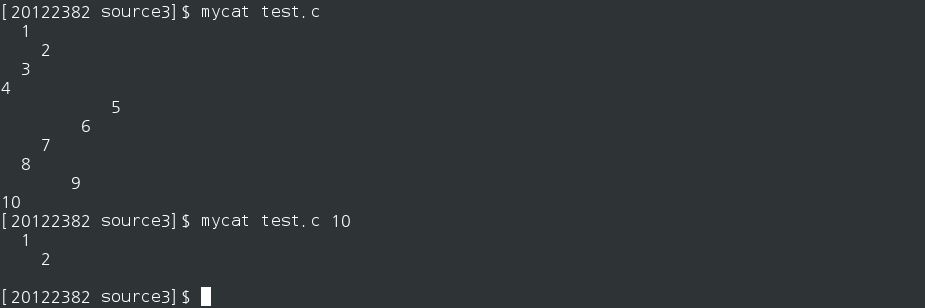
myls는 기본 ls 명령어의 –l 옵션과 같은 기능을 한다. myls [option] [filename]을 할 경우 해당 디렉토리의 목록을 보여준다. –a 옵션을 사용할 경우 숨긴파일도 출력 가능하다.



## mycat.c

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
5. **int** mycat(**char** \*command){
7. **char** \*option1;
8. **char** \*option2;
9. **char** c;
10. **int** byte;
12. //torkernize command option1 and option2
13. //option1 has file name and option2 has byte
14. option1 = strtok\_r(command," ",&option2);
16. **if**(strstr(option2," ") || !strcmp(command,"")){
17. printf("Wtong Option\n");
18. **return** -1;
19. }
21. **if**(option1[strlen(option1) -1]=='\n')
22. option1[strlen(option1) - 1] = '\0';
23. option2[strlen(option2) -1]='\0';
25. //file open
26. **FILE** \*fp=fopen(option1,"r");
28. **if**(fp==NULL){
29. printf("ERROR : no exit file\n");
30. **return** -1;
31. }
32. **else**{
33. **if**(!strcmp(option2,""))
34. **while**((c = fgetc(fp)) != EOF)
35. printf("%c",c);
37. **else**{
38. //get byte
39. byte = atoi(option2);
41. //print while byte
42. **if**(strcmp(option2,"0")&& byte==0){
43. printf("Wtong Option\n");
44. **return** -1;
45. }
47. **int** i;
48. **for**(i=0;i<byte;i++){
49. **if**((c = fgetc(fp)) != EOF)
50. printf("%c",c);
51. }
52. printf("\n");
53. }
54. }
56. fclose(fp);
58. **return** 0;
60. }

파일 내용을 출력해주는 명령어이다. mycat [filename] [num] 과같이 사용할 경우 원하는 byte 만큼 출력할 수 있다. fopen으로 파일을 연 후 fgetc로 읽어오면 된다. num이 존재하는 경우 for문을 이용해 해당 num 만큼 읽어온다.

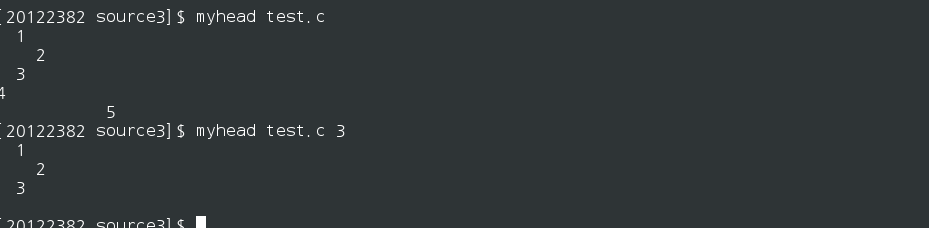


## myhead.c

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>

6. **int** myhead(**char** \*command){
8. **char** \*option1;   //filename
9. **char** \*option2;   //number
10. **char** c;
11. **int** line;
13. option1 = strtok\_r(command," ",&option2);
15. //option check
16. **if**(strstr(option2," ") || !strcmp(command,"")){
17. printf("Wtong Option\n");
18. **return** -1;
19. }
21. **if**(option1[strlen(option1) -1]=='\n')
22. option1[strlen(option1) - 1] = '\0';
23. option2[strlen(option2) -1]='\0';
25. **FILE** \*fp=fopen(option1,"r");
27. **if**(fp==NULL){
28. printf("ERROR : no exit file\n");
29. **return** -1;
30. }
31. **else**{
32. **if**(!strcmp(option2,"")){
34. //print line 5
35. **int** count=0;
36. **while**((c = fgetc(fp))!=EOF){
37. printf("%c",c);
38. **if**(c=='\n')
39. count++;
40. **if**(count==5)
41. **break**;
42. }
43. }
44. **else**{
46. //string to int
47. line = atoi(option2);
48. **if**(strcmp(option2,"0")&& line==0){
49. printf("Wtong Option\n");
50. **return** -1;
51. }
53. //print until line and EOF
54. **int** count=0;
55. **while**((c = fgetc(fp)) != EOF){
56. printf("%c",c);
57. **if**(c=='\n')
58. count++;
59. **if**((count==line))
60. **break**;
61. }
63. printf("\n");
64. }
65. }
67. fclose(fp);
69. **return** 0;
71. }

cat과 유사하게 파일을 출력하지만 첫 5줄만 출력한다. myhead [filename] [num] 과 같이 사용할 경우 첫줄부터 num 만큼의 줄을 출력한다.



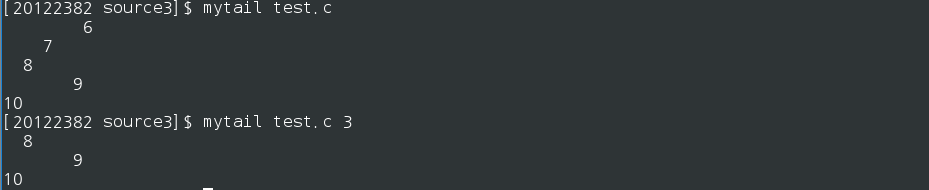
## mytail.c

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
5. **int** mytail(**char** \*command){
7. **char** \*option1;   //filename
8. **char** \*option2;   //number
9. **char** c;
10. **int** line; //print line count
11. **int** lineCount=0; //total line count
13. option1 = strtok\_r(command," ",&option2);
15. //option check
16. **if**(strstr(option2," ") || !strcmp(command,"")){
17. printf("Wtong Option\n");
18. **return** -1;
19. }
21. **if**(option1[strlen(option1) -1]=='\n')
22. option1[strlen(option1) - 1] = '\0';
23. option2[strlen(option2) -1]='\0';

26. **FILE** \*fp=fopen(option1,"r");
27. **if**(fp==NULL){
28. printf("ERROR : no exit file\n");
29. **return** -1;
30. }
32. //count total line
33. **while**((c = fgetc(fp))!=EOF){
34. **if**(c=='\n')
35. lineCount++;
36. }
37. fclose(fp);

40. fp=fopen(option1,"r");
42. **if**(!strcmp(option2,"")){
44. //print last line 5
45. **while**((c = fgetc(fp))!=EOF){
46. **if**(lineCount<6)
47. printf("%c",c);
48. **if**(c=='\n')
49. lineCount--;
50. }
51. }
52. **else**{
54. //string to int
55. line = atoi(option2);
56. **if**(strcmp(option2,"0")&& line==0){
57. printf("Wtong Option\n");
58. **return** -1;
59. }
61. //print until last line and EOF
62. **while**((c = fgetc(fp))!=EOF){
63. **if**(lineCount<line+1)
64. printf("%c",c);
65. **if**(c=='\n')
66. lineCount--;
67. }
69. }
71. fclose(fp);
72. **return** 0;
74. }

head와는 다르게 마지막 5줄을 출력한다. mytail [filmename] [num]과 같이 사용할 경우 뒤에서 num만큼의 줄을 출력한다.



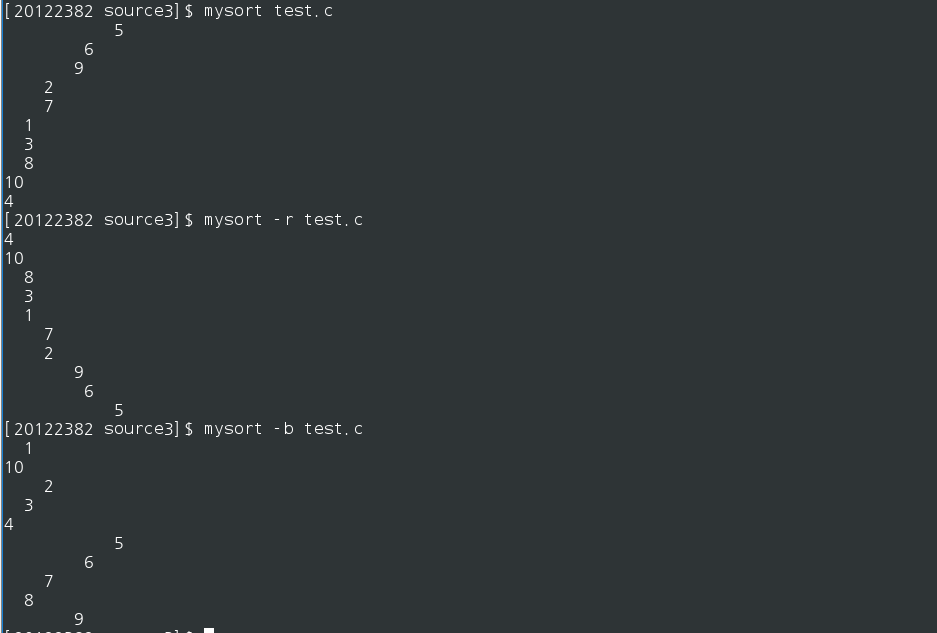
## mysort.c

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<unistd.h>
4. #include<string.h>
5. **int** counting(**char** \*);
6. **int** cmp(**const** **void** \*,**const** **void** \*);
7. **int** cmpr(**const** **void** \*,**const** **void** \*);
8. **int** cmpb(**const** **void** \*,**const** **void** \*);

11. **int** mysort(**char** \*command){
13. **char** \*option1,\*fileName;
14. **char** \*pwd;
15. **FILE** \*fp;
16. **int** count=0;
18. **if**(!strcmp(command,"")){
19. printf("ERROR : Enter File\n");
20. **return** -1;
21. }
23. command[strlen(command) -1] = '\0';
25. **if**(strstr(command," ")){
26. option1 = strtok\_r(command," ",&fileName);
27. }
28. **else**{
29. //no option
30. fileName=command;
32. //counting \n
33. **if**((count = counting(fileName)) == -1)
34. **return** -1;
36. fp = fopen(fileName,"r");
37. **if**(fp==NULL){
38. printf("ERROR : No such file\n");
39. **return** -1;
40. }
42. **char** str[count][10000];   //2D array for sort
43. **int** i=0;
45. //read file .. escape \n
46. **while**(fgets(str[i++],10000,fp));
48. //sort
49. qsort(str,count,**sizeof**(str[0]),cmp);
51. **for**(i=0;i<count;i++)
52. printf("%s",str[i]);
54. fclose(fp);
56. **return** 0;
57. }

60. **if**(!strcmp(option1,"-r")){
62. //counting \n
63. **if**((count = counting(fileName)) == -1)
64. **return** -1;
66. fp = fopen(fileName,"r");
67. **if**(fp==NULL){
68. printf("ERROR : No such file\n");
69. **return** -1;
70. }
72. **char** str[count][10000];   //2D array for sort
73. **int** i=0;
75. //read file .. escape \n
76. **while**(fgets(str[i++],10000,fp));
78. //sort
79. qsort(str,count,**sizeof**(str[0]),cmpr);
81. **for**(i=0;i<count;i++)
82. printf("%s",str[i]);
84. fclose(fp);
86. }
87. **else** **if**(!strcmp(option1,"-b")){
89. //counting \n
90. **if**((count = counting(fileName)) == -1)
91. **return** -1;
93. fp = fopen(fileName,"r");
94. **if**(fp==NULL){
95. printf("ERROR : No such file\n");
96. **return** -1;
97. }
99. **char** str[count][10000];   //2D array for sort
100. **int** i=0;
102. **while**(fgets(str[i++],10000,fp));
104. qsort(str,count,**sizeof**(str[0]),cmpb);
106. **for**(i=0;i<count;i++)
107. printf("%s",str[i]);
109. fclose(fp);
110. }
111. **else**{
112. printf("ERROR : Wrong Option\n");
113. **return** -1;
114. }
116. **return** 0;
117. }
119. **int** counting(**char** fileName[]){
121. **int** count=0;
122. **char** c;
124. **FILE** \*p = fopen(fileName,"r");
125. **if**(p==NULL){
126. printf("ERROR : No such file\n");
127. **return** -1;
128. }
130. **while**((c=fgetc(p)) != EOF){
131. **if**(c=='\n')
132. count++;
133. }
134. fclose(p);
135. **return** count;
136. }
137. //sotring without space
138. **int** cmpb(**const** **void**\*a, **const** **void**\*b){
140. //compare without ' '
141. **while**( \*(**char** \*)a == ' ') a++;
142. **while**( \*(**char** \*)b == ' ') b++;
144. **return** (strcmp( (**char** \*)a, (**char** \*)b) );
145. }
146. //sorting reverse
147. **int** cmpr(**const** **void**\*a, **const** **void**\*b){
148. **return** (strcmp( (**char** \*)b, (**char** \*)a) );
149. }
150. //sorting
151. **int** cmp(**const** **void**\*a, **const** **void**\*b){
152. **return** (strcmp( (**char** \*)a, (**char** \*)b) );
153. }

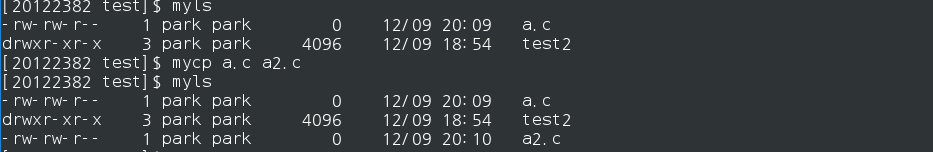
파일의 모든 내용을 읽어와 배열에 저장후 퀵소트를 이용해 오름차순 정렬한다. –r 옵션과 –b 옵션이 존재한다. mysort [option] [filename]과 같이 사용한다. –r 옵션은 내림차순으로 정렬된다. -b옵션은 공백을 제거한 후 가장 첫 문자를 비교해 오름차순으로 정렬하는 옵션이다. 파일을 읽어와 개행문자의 개수를 세어 그만큼의 크기를 갖는 2차원 배열을 생성하고, while 문으로 파일 내용을 모두 읽어와 그 배열에 내용을 저장한다. 마지막으로 퀵소트를 이용해 그 배열을 정렬한다. –b 옵션을 사용하는 경우 compare 함수에서 두개의 배열을 비교할 때 공백이라면 주소를 공백이 아닐 때 까지 증가시켜 공백이 아닌 문자를 비교해 정렬 하도록 한다.



## mycp.c

1. #include<stdio.h>
2. #include<unistd.h>
3. #include<string.h>
4. **int** mycp(**char** \*command){
6. **char** \*option1;
7. **char** \*option2;
9. **FILE** \*original, \*copy;
10. **char** c;
12. command[strlen(command) -1] = '\0';
13. option1 = strtok\_r(command," ",&option2);
15. //option1 has original file , option2 has targetfile
16. **if**(strstr(option2," ") || option1==NULL || !strcmp(option2,"")){
17. printf("Wrong Option\n");
18. **return** -1;
19. }
21. //if targetfile already exist
22. **if**(access(option2,F\_OK)==0){
23. printf("ERROR : %s is  existing file\n",option2);
24. }
26. **if**((original= fopen(option1,"r")) == NULL){
27. printf("ERROR : No Exit File\n");
28. **return** -1;
29. }
31. **if**((copy= fopen(option2,"w")) == NULL){
32. printf("ERROR : No Exit File\n");
33. **return** -1;
34. }
36. //copy file
37. **while**(!feof(original)){
38. c=(**char**)fgetc(original);
40. **if**(c!=EOF){
41. fputc((**int**)c,copy);
42. }
43. }
45. fclose(original);
46. fclose(copy);
48. **return** 0;
49. }

mycp [source] [dest] 와 같이 사용한다. source 파일을 dest의 이름의 파일로 복사한다. access()를 이용해 dest에 해당하는 파일의 유무를 점검하고, 이미 존재할 경우 복하사지 않는다. 파일을 열어 모두 읽은 후 dest 파일명으로 파일을 열어 입력해주면 된다.



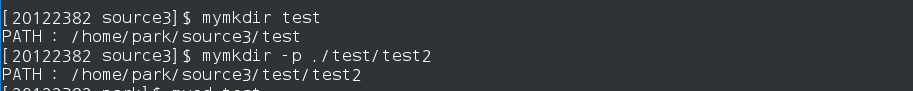
## mkdir & myrm

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<unistd.h>
4. #include<string.h>
6. **int** strcnt(**char** \*,**char**);
7. **int** mymkdir(**char** \*command){
9. **char** \*option1,\*option2;
10. **char** \*pwd = (**char** \*)malloc(100);
12. command[strlen(command) -1] = '\0';
14. //Create Directory if user didn't using -p option
15. //If using -p option, tokernizing command
16. **if**(strstr(command," ")){
17. option1 = strtok\_r(command," ",&option2);
18. }
19. **else**{
20. **if**(mkdir(command, 0755)){
21. printf("ERROR : Make Directory Faile\n");
22. **return** -1;
23. }

26. getcwd(pwd,100);
27. printf("PATH : %s/%s\n",pwd,command);
29. free(pwd);
31. **return** 0;
32. }
34. **if**(!strcmp(option1,"-p")){
36. **int** i,count=0;
37. count = strcnt(option2,'/');
39. **for**(i=0;i<count+1;i++){
40. **char** \*tmp;
42. //tokernize path
43. **if**(i==0)
44. tmp = strtok(option2,"/\n");
45. **else**
46. tmp = strtok(NULL,"/\n");
48. //file check
49. **if**(!access(tmp,F\_OK)){
51. chdir(tmp);
53. }
54. **else**{
55. mkdir(tmp, 0755);
56. chdir(tmp);
57. }
58. }

61. //print path
62. getcwd(pwd,100);
63. printf("PATH : %s\n",pwd);
65. free(pwd);
67. //back to original directory
68. **for**(i=0;i<count+1;i++)
69. chdir("..");
71. }
72. **else**{
73. printf("Wrong Option\n");
74. **return** -1;
75. }
77. **return** 0;
78. }
80. //counting '/'
81. **int** strcnt(**char** str[], **char** c){
83. **int** i,count=0;
85. **for**(i=0;str[i] != '\0' ;i++){
86. **if**(str[i]==c)
87. count++;
88. }
89. **return** count;
90. }

mkdir 함수로 현재 디렉토리에 디렉토리를 만들 수 있다. –p 옵션을 사용할 경우 경로에 해당 디렉토리가 존재하지 않을 경우 디렉토리를 만들면서 최종 디렉토리를 생성한다. 디렉토리 생성 성공 시 절대경로를 출력해준다. mkdir을 사용하기 위해 해당 경로까지 chdir로 이동한다. 해당 경로에 디렉토리를 생성한 후 다시 chdir(..)로 이동한 만큼 빠져나온다.



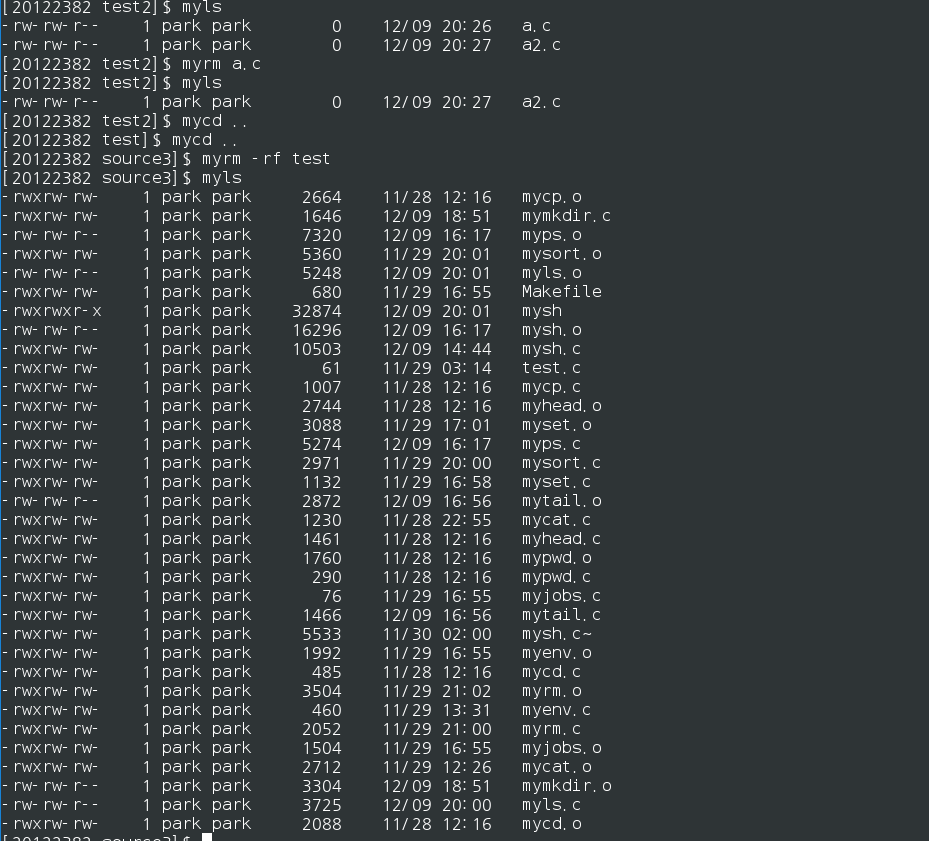


다음은 myrm이다.

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<unistd.h>
4. #include<string.h>
5. #include<sys/stat.h>
6. #include<dirent.h>
8. **int** myrm(**char** \*command){
10. **char** \*option1,\*fileName;
11. **char** \*pwd;
13. command[strlen(command) -1] = '\0';

16. //Remove file if user didn't using -p option
17. //If using -rf option, tokernizing command
18. **if**(strstr(command," ")){
19. option1 = strtok\_r(command," ",&fileName);
20. }
21. **else**{
22. //remove file
23. **if**(unlink(command)){
24. printf("ERROR : No File\n");
25. **return** -1;
26. }
28. **return** 0;
29. }
31. **if**(!strcmp(option1,"-rf")){
33. //check file or dir is real
34. **if**(access(fileName,F\_OK)==0){
35. **if**(remove(fileName)==-1){
37. //set path
38. **char** \*tmp=getcwd(NULL,0);
39. strcat(tmp,"/");
40. strcat(tmp,fileName);
42. //remove directory if it has files
43. removeAll(tmp);
44. }
45. }**else**{
46. printf("ERROR : No such File\n");
47. **return** -1;
48. }
49. }
50. **else**{
51. printf("ERROR : Wrong Option\n");
52. **return** -1;
53. }
55. **return** 0;
56. }
58. **int** removeAll(**const** **char** \*fileName){
60. DIR \*dir;
61. **struct** dirent \*dirent;
62. **char** target[2048];
64. //if not directory
65. **if**((dir=opendir(fileName)) == NULL){
66. printf("ERROR : Open Error // path : [%s]\n",fileName);
67. **return** -1;
68. }
70. **while**(1){
71. //no directory
72. **if**(!(dirent = readdir(dir))){
73. **break**;
74. }
76. **if**(dirent->d\_name[0] == '.')
77. **continue**;
79. //add path
80. sprintf(target,"%s/%s",fileName,dirent->d\_name);
82. //if counldn't remove it means it is directory
83. //recuresive removing
84. **if**(remove(target)<0){
85. **if**(removeAll(target) == -1)
86. **return** -1;
87. **continue**;
88. }
89. }
91. //close directory and remove directory
92. **if**(remove(fileName) <0){
93. closedir(dir);
94. **return** -1;
95. }
96. **else**{
97. closedir(dir);
98. **return** 1;
99. }
100. }

myrm [option] [filename] 과 같이 사용한다. remove 함수로 해당 파일을 지울 수 있다. DIR 포인터로 삭제할 파일이 있는 디렉토리를 연후 remove 함수를 사용한다. –rf 옵션을 사용할 경우 파일 뿐만 아니라 디렉토리도 삭제할 수 있다. removeAll() 함수에서는 경로에 있는 모든 것을 지우는데 이때 재귀함수를 이용해 폴더의 모든 파일을 삭제한다. remove를 시도했을 때 -1을 리턴한다면 파일을 포함하는 디렉토리라는 의미이다. 이때 재귀함수를 이용해 removeAll()을 한다.



# 결론

평소에 리눅스를 사용하며, 당연시 여겼던 명령어를 구현하면서 결국은 이러한 명령어도 C로 만든 코드에 불과하다는 생각을 했다. 이번 과제를 하며, mkdir이나 chdir과 같은 함수를 명령어를 만들기 위해 사용했는데 이러한 함수들도 한번 뜯어보고 만들어 보고 싶다는 생각이 들었다. 지금까지 프로그래밍을 할 때에는 여러 API들을 이용해 만들어져 있는 것들을 이용 했었는데, 점점 더 low level의 함수들만 이용해 코딩을 해야 할 필요성을 느꼈다.

명령어를 만들며 cat명령어나 cd,pwd와 같은 만들기 쉬운 명령어들도 있었지만, ls와 ps와 같은 명령어들은 처음보는 구조체와 파일들을 다루는 부분들이 었기 때문에 구현이 어려웠다. 특히 ps 명령어의 경우 STIME과 TIME을 구하는데에 굉장히 애를먹었다.

이번 과제를 진행하며, 전반적인 리눅스 시스템과 C에대한 이해도가 많이 올라갔다. 향후 서버나 시스템을 개발할 때 많은 도움이 될 것이다.

# 참고자료

shell : <http://freewise.tistory.com/24>

make : <http://koreaboom.tistory.com/entry/Makefile-%EB%A7%8C%EB%93%A4%EA%B8%B0>

ps 명령어 : <http://sosal.kr/115>

bash : https://ko.wikipedia.org/wiki/%EB%B0%B0%EC%8B%9C\_(%EC%9C%A0%EB%8B%89%EC%8A%A4\_%EC%85%B8