운영체제

과제 3

: 나만의 Shell 인터프리터 구현

|  |  |
| --- | --- |
| 소속학과 | 컴퓨터학부 |
| 학번 | 20122366 |
| 성명 | 김홍석 |
| 과목명 | 운영체제 |
| 담당교수명 | 길아라 교수님 |
| 일자 | 2016-12-12 |

목차

1. 서론
   1. 연구 분야
   2. 동기
   3. 목적
   4. 연구 접근 방법
2. 관련 연구
3. 설계
4. 구현 및 모의실험
5. 결론
6. 참고 문헌 및 자료
7. 서론
   1. 연구 분야

쉘(Shell)은 [운영 체제](https://ko.wikipedia.org/wiki/%EC%9A%B4%EC%98%81_%EC%B2%B4%EC%A0%9C) 상에서 다양한 운영 체제 기능과 서비스를 구현하는 인터페이스를 제공하는 프로그램이다. 쉘(껍데기의 [영어](https://ko.wikipedia.org/wiki/%EC%98%81%EC%96%B4) 단어)은 사용자와 운영 체제의 내부([커널](https://ko.wikipedia.org/wiki/%EC%BB%A4%EB%84%90_(%EC%BB%B4%ED%93%A8%ED%8C%85))) 사이의 인터페이스를 감싸는 층이기 때문에 그러한 이름이 붙었다.

쉘은 일반적으로 명령 줄과 그래픽 형의 두 종류로 분류된다. 명령 줄 쉘은 운영 체제 상에서 [명령 줄 인터페이스](https://ko.wikipedia.org/wiki/%EB%AA%85%EB%A0%B9_%EC%A4%84_%EC%9D%B8%ED%84%B0%ED%8E%98%EC%9D%B4%EC%8A%A4)(CLI)를 제공하는 반면에, 그래픽 쉘은 [그래픽 사용자 인터페이스](https://ko.wikipedia.org/wiki/%EA%B7%B8%EB%9E%98%ED%94%BD_%EC%82%AC%EC%9A%A9%EC%9E%90_%EC%9D%B8%ED%84%B0%ED%8E%98%EC%9D%B4%EC%8A%A4)(GUI)를 제공한다.

CLI와 GUI 기반 쉘의 상대적 장점에 대해서는 논쟁이 많다. CLI 지지자는 CLI 쉘 상에서 일반적인 수행(파일 이동 등)을 훨씬 빠르게 할 수 있다고 주장한다. 반대로 GUI 지지자는 GUI 쉘의 상대적 사용성과 간편함을 내세운다. 일반적으로 성능을 중시하고 숙련된 관리자의 작업에 주로 사용되는 서버의 경우는 CLI가 많이 쓰이며 반면 일상적인 업무에 사용되는 컴퓨터엔 GUI가 많이 사용된다.

* 1. 동기

쉘을 직접 만들고 구성함으로써 각 명령어에 필요한 정보들이 어디에 저장되어 있는지 파악할 수 있고, OS 내부적인 구조의 이해도 할 수 있다.

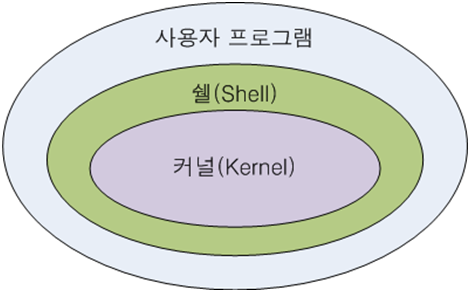
* 1. 목적

쉘을 직접 C로 구현하는 것을 목적으로 한다. 이번 과제를 통하여 OS의 내부적인 구조, 각종 프로세스 정보들이 어디에 저장되어 있는지를 파악할 수 있다. 이를 이용해 각종 명령어 ( ex. ls –l ,ps 등등)를 구현한다. 각종 명령어를 직접 구현함으로써 각 명령어가 무엇을 출력해 주는지 더 깊게 이해한다.

* 1. 연구 접근 방법

1. 기본 쉘 명령어 파악
2. 각 명령어들의 출력내용과 그에 대한 정보 출처 파악
3. 각 명령어들의 구현
4. 관련연구
   1. 쉘 (Shell)

쉘은 운영체제에서 사용자가 입력하는 명령을 읽고 해석하여 대신 실행해주는 프로그램이다. 쉘은 유닉스나 리눅스에만 있는 것이 아니라 windows같은 다른 운영체제에서도 존재한다. 쉘의 역할은 사용자가 입력한 명령을 해석하여 대신 실행해주는 것이다. 쉘의 내부 명령어라면 스스로 실행한 뒤 결과를 화면에 표시해주고 내부 명령어가 아니라면 PATH 환경변수에 지정된 경로에서 입력받은 명령과 같은 파일을 찾아 execve() 시스템 콜을 호출한다. Execve 시스템 콜이 호출 된 후 키보드와 마우스 등의 입력장치와 모니터에 해당하는 표준 출력장치의 제어권을 해당 프로그램에게 넘겨준 뒤 프로그램이 끝날 때까지 대기하는 역할을 수행한다.



* 1. 환경변수

IT([정보기술](http://terms.naver.com/entry.nhn?docId=1180012&ref=y)) 용어로, OS의 셸(shell) 등에 설정되어 있다. 변수의 이름과 의미는 미리 정해져 있기 때문에 환경변수를 읽으면 시스템의 설정을 어느 정도 알 수 있다.

OS의 환경변수는 시스템의 실행파일이 놓여 있는 디렉토리의 지정 등 OS 상에서 동작하는 응용[소프트웨어](http://terms.naver.com/entry.nhn?docId=1114228&ref=y)가 참조하기 위한 설정이 기록된다. 응용[소프트웨어](http://terms.naver.com/entry.nhn?docId=1114228&ref=y)로부터는 시스템 콜(system call:프로그래밍 언어에서 지원하지 않는 기능에 대하여 운영체계의 루틴을 호출하여 이용하는 것)이나 OS의 표준 [API](http://terms.naver.com/entry.nhn?docId=1179553&ref=y) 등을 통하여 간단히 값을 얻을 수 있도록 되어 있다.

또한 웹 브라우저의 내부 [데이터](http://terms.naver.com/entry.nhn?docId=1082441&ref=y)의 일부를 환경변수라고 하는 경우도 있는데, 이것은 [HTTP](http://terms.naver.com/entry.nhn?docId=1180001&ref=y)를 요청할 때 송신되는 것으로, 브라우저의 종류나 링크되어 있는 웹 페이지 등 웹 [서버](http://terms.naver.com/entry.nhn?docId=1180913&ref=y)가 웹 브라우저에 대하여 최적의 처리를 하기 위해 송신되는 것이다. [HTTP](http://terms.naver.com/entry.nhn?docId=1180001&ref=y)를 요청하는 응용[소프트웨어](http://terms.naver.com/entry.nhn?docId=1114228&ref=y)는 모두 환경변수를 송신하고 있다고 할 수 있다.

사용자가 의도적으로 변환할 수 있는 환경변수도 많으며, 특히 웹 브라우저에서는 자신의 정체를 숨길 목적으로 브라우저 등의 변수를 변환하는 경우도 있다.

그러나 사실과 다른 값을 환경변수에 설정하면 그 환경변수를 사용하고 있는 [서버](http://terms.naver.com/entry.nhn?docId=1180913&ref=y)나 응용[소프트웨어](http://terms.naver.com/entry.nhn?docId=1114228&ref=y)가 올바르게 작동하지 않을 수 있다. 특히, 셸의 환경변수를 변환했을 경우에는 심각한 오작동을 일으킬 수 있기 때문에 이를 취급하는 경우에는 주의가 필요하다.

* 1. Fork()



Fork 함수는 부모 프로세스를 그대로 복사하여 자식 프로세스를 생성하는 함수이다. 두 프로세스는 fork함수의 호출 다음 문장부터 수행을 하게 되는데, fork함수의 리턴값을 이용해서 분기를 함으로써 부모와 자식프로세스의 실행을 달리한다. 부모는 fork 함수의 리턴값이 자식프로세스의 pid ,자식프로세스는 fork함수의 리턴값이 0 이므로 이를 이용해 분기한다.

* 1. 사용 함수들

|  |
| --- |
| char \* getcwd(char \*buf,size\_t size);   * 작업 디렉토리의 전체 이름을 구하는 함수   Char \* buf : 작업 디렉토리 문자열을 담을 버퍼  Size\_t size : 버퍼의 크기  Return : 현재 작업 디렉토리 -> 성공  -1 -> 실패 |
| int setenv(const char \* envname, const char \*envval,int overwrite)   * 환경변수를 추가 또는 수정해주는 함수   Char \*envname : 환경변수 이름  Char \*envval : 변수 값  Int overwrite : 이미 같은 이름의 변수가 있다면 값을 변경할지의 여부  Return : 0 -> 성공  -1 -> 실패 |
| Int access(const char \* pathname, int mode)   * 파일 존재나 접근 권한 확인 함수   Char \* pathname : 파일이나 디렉토리 전체 이름  Int mode : 검사할 내용  Return : 0 -> 기능 또는 파일이 존재  -1 -> mode에 대해 거절, 에러 |
| Char \* getenv(const char \* name)   * 환경 변수의 개별값 구하는 함수   Char \* name : 구하려는 환경 변수 이름  Return : 환경 변수의 값  실패 시 NULL |
| DIR \* opendir(const char \* name)   * 디렉토리 열기 함수   Char \* name : 열기 대상 디렉토리  Return : 성공하면 디렉토리 정보 구조체 DIR 반환,  실패하면 NULL |
| int remove(const char \* pathname)   * 파일 또는 디렉토리 삭제 함수   Char \* pathname : 삭제할 파일이나 디렉토리  Return : 성공하면 0  실패하면 -1 |
| int lstat(const char \* path,struct stat \* buf)   * 파일 정보를 받아오는 함수   const char \* path : 받아오려는 파일 경로  struct stat \* buf : 성공시 반환받는 인자  Return : 성공하면 0  실패하면 -1 |
| Struct passwd \* getpwuid(uid\_t uid)   * 사용자 ID로 사용자 정보 구하는 함수   uid\_t uid : 유저 아이디  Return : 사용자 정보를 담고 있는 sturct passwd 포인터 |
| Struct group\* getgrgid(gid\_t gid)   * 그룹 ID로 그룹 정보 구하는 함수   gid\_t gid : 그룹 아이디  Return : 그룹 정보를 담고 있는 sturct group 포인터 |
| Int chdir(const char \* path)   * 현재 디렉토리를 path로 바꿈   const char \* path : 바꾸려는 경로  Return : 성공 시 0 , 실패 시 -1 |
| Int mkdir(const char \* pathname,mode\_t mode)   * pathname으로 디렉토리를 만드는 함수   const char \* pathname : 바꾸려는 경로  mode\_t mode : 디렉토리 모드  Return : 성공 시 0 , 실패 시 -1 |

* 1. 사용 구조체, 변수들

passwd 구조체

struct passwd{

char \*pw\_name; /\* username \*/

char \*pw\_passwd; /\* password \*/

\_\_uid\_t pw\_uid; /\* user id \*/

\_\_gid\_t pw\_gid; /\* group id \*/

char \*pw\_gecos; /\* real name \*/

char \*pw\_dir; /\* home directory \*/

char \*pw\_shell; /\* shell program \*/

};

group 구조체

struct group {

char \*gr\_name; /\* group name \*/

char \*gr\_passwd; /\* group password \*/

gid\_t gr\_gid; /\* group id \*/

char \*\*gr\_mem; /\* list of group member \*/

};

extern char \*\* environ;

stdlib.h 아래에 있는 environ 변수를 사용한다. 여기에는 환경변수의 모든 목록이 문자열의 형태로 되어있다. 이를 사용하여 환경변수를 가져와서 출력한다.

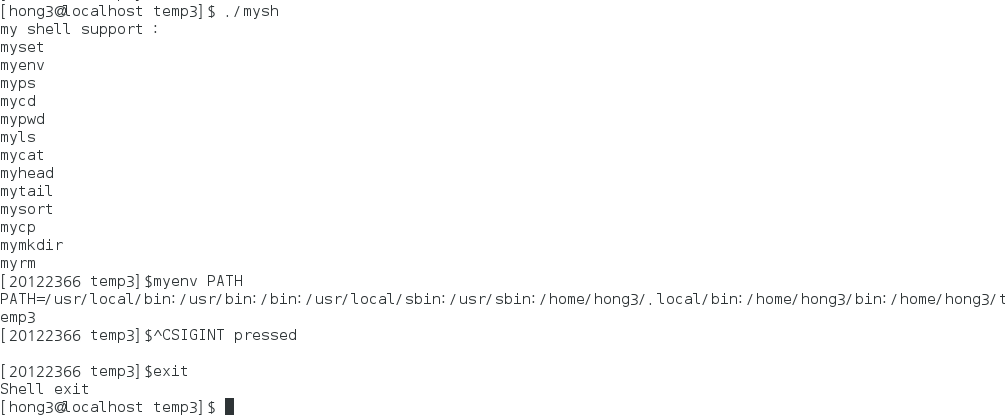
1. 설계

개발환경 : Fedora 18 (64bit)

커널 버전 : 3.5.5

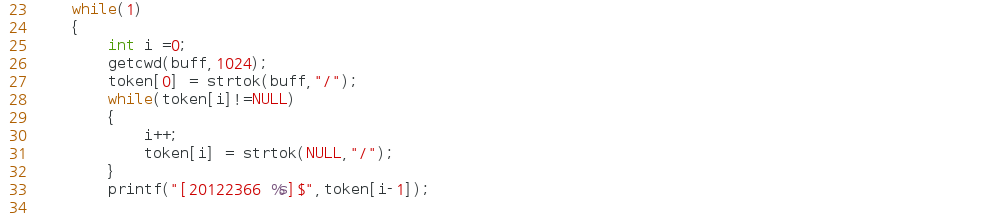
* 1. 기본 Shell 기능 구현
  2. 명령어 구현
     + 1. Myset
       2. Myenv
       3. Myps
       4. Myjobs
       5. Mycd
       6. Mypwd
       7. Myls
       8. Mycat
       9. Myhead
       10. Mytail
       11. Mysort
       12. Mycp
       13. Mymkdir
       14. Myrm

1. 구현 및 모의실험
   1. 기본 Shell 기능 구현



* Mysh 의 실행 화면

기본 shell 기능 구현을 최선을 다했지만 background process trace 부분을 구현하지 못하였다. 추가로 myjobs 명령어 또한 구현하지 못하였다. 그러나 다른 명령어들은 최선을 다해서 구현을 완료하였다.



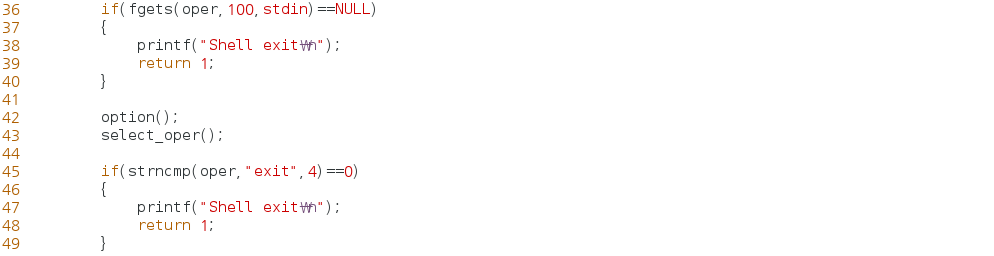
* 기본 프롬프트 변경

While 문 안에 getcwd함수를 이용해 구현하였다. 받은 현재디렉토리의 절대경로를 strtoknizer 함수를 이용해서 현재 디렉토리만 출력하게 끔 처리해주었다.



* 환경 변수의 PATH에 현재 디렉토리 추가.

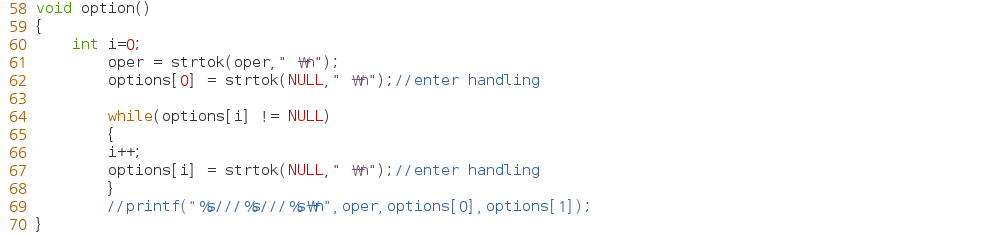
우선 기존 환경변수의 PATH 설정에 현재 디렉토리를 추가하는 부분이다. Getenv(“PATH”)를 통해서 현재 PATH설정을 받아오고 getcwd() 함수를 통해서 현재 디렉토리를 문자열로 받아온 후 현재 PATH + 현재 디렉토리를 setenv()함수로 다시 PATH에 추가 해준다.





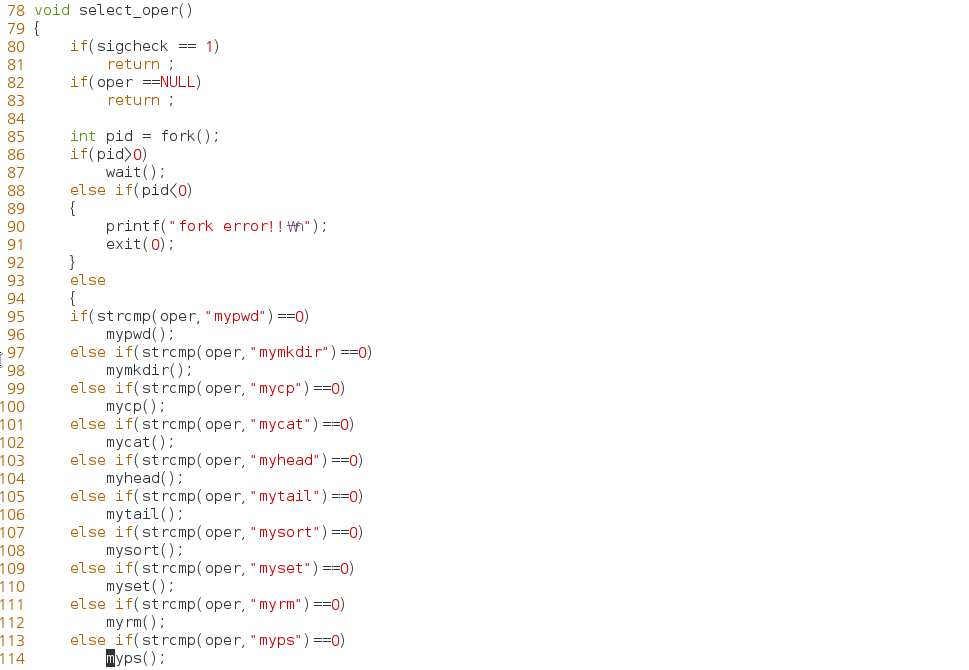
* ctrl-d, EOF, exit, sigint 처리

ctrl-d는 EOF를 입력으로 주는데 맨 위 if문에서 처리를 해 줬다. 그리고 exit은 문자열 비교를 통해, sigint는 signal에 등록해서 처리를 하였다.



* 입력받은 문자열을 strtoknizer 함수를 통해서 나눠주는 부분

명령어 [옵션] [타겟] 등의 포맷을 처리하기 위해 입력받은 문자열들을 처리해준다. 명령어는 oper로, 옵션 및 타겟은 options[]로 받았다.



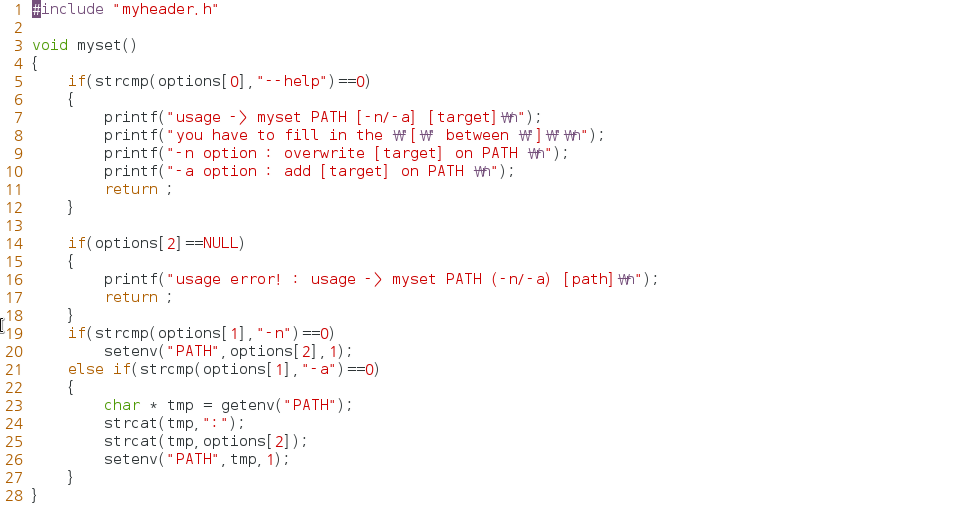
* 각 함수들 실행하는 부분

Fork 를 이용해 자식프로세스에서 각 명령어들을 실행시켜준다. 이 동안 부모프로세스는 wait 함수로 잠들어있고 자식프로세스가 실행된다. 자식프로세스가 명령어를 다 수행하면 자식프로세스는 죽이고 다시 부모프로세스가 실행되면서 쉘이 계속 실행되게 된다.

* 1. 명령어 구현
     + 1. Myset

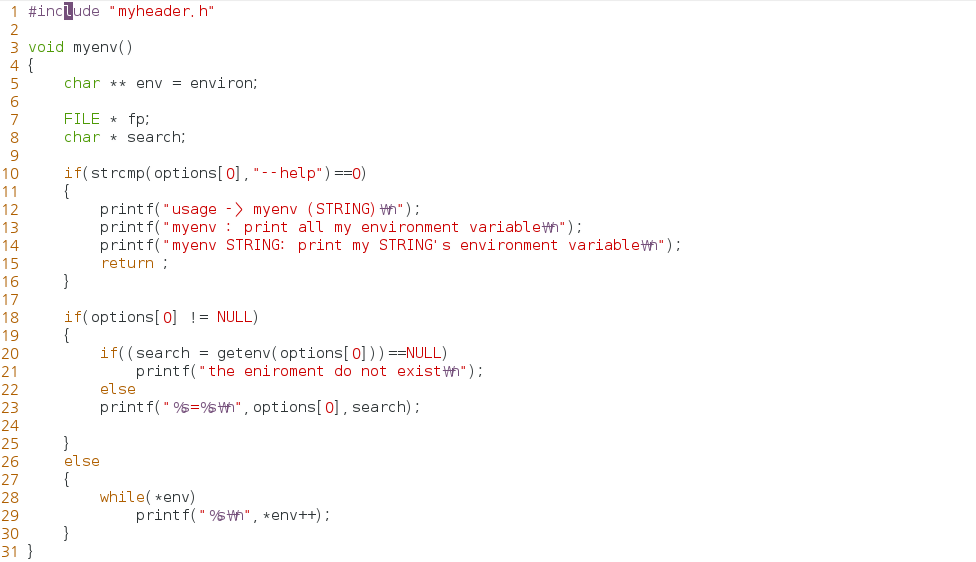


* -n옵션, -a옵션 실행화면



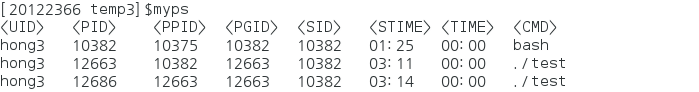
환경변수를 추가할 때 쓰는 set 명령어를 구현하였다. –n 옵션은 덮어쓰기이므로 setenv함수를 이용해 바로 PATH항목에 덮어쓰기 하였다. –a 옵션은 붙여쓰기이므로 getenv함수를 이용해서 환경변수를 받아온 후 : 콜론을 문자열에 붙인 후 새로 추가하려는 옵션을 추가해주었다.

* + - 1. Myenv
* Myenv 명령어의 실행화면은 myset 명령어를 참고.

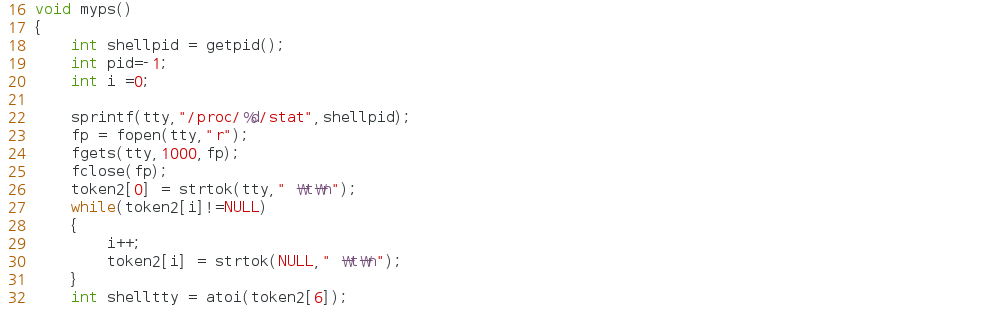


환경변수 전체를 출력하는 명령어이다. Myheader.h 에 extern char \*\* environ;라고 선언되어 있고 이 environ변수를 가져와서 쓴다. Environ 변수는 환경변수를 저장하고 있는 변수이며 이를 이용해 환경변수를 출력해준다. 그리고 string 입력이 들어오면 이 string을 이용해 getenv로 환경변수를 받아와서 출력해준다.

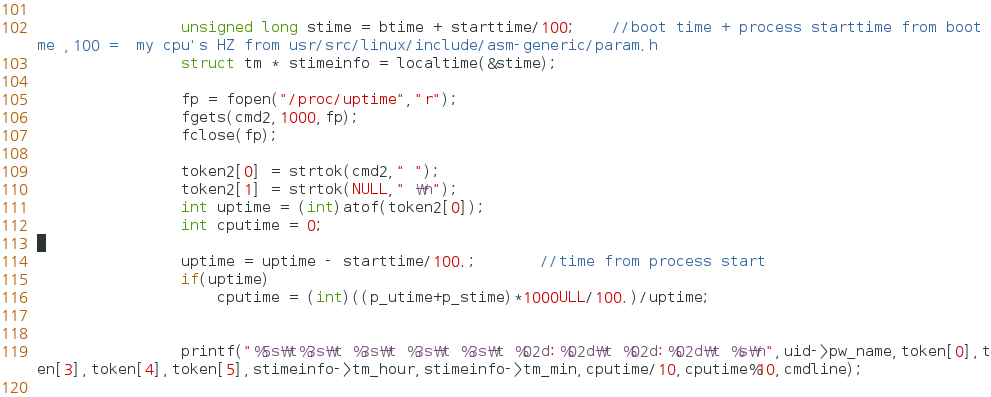
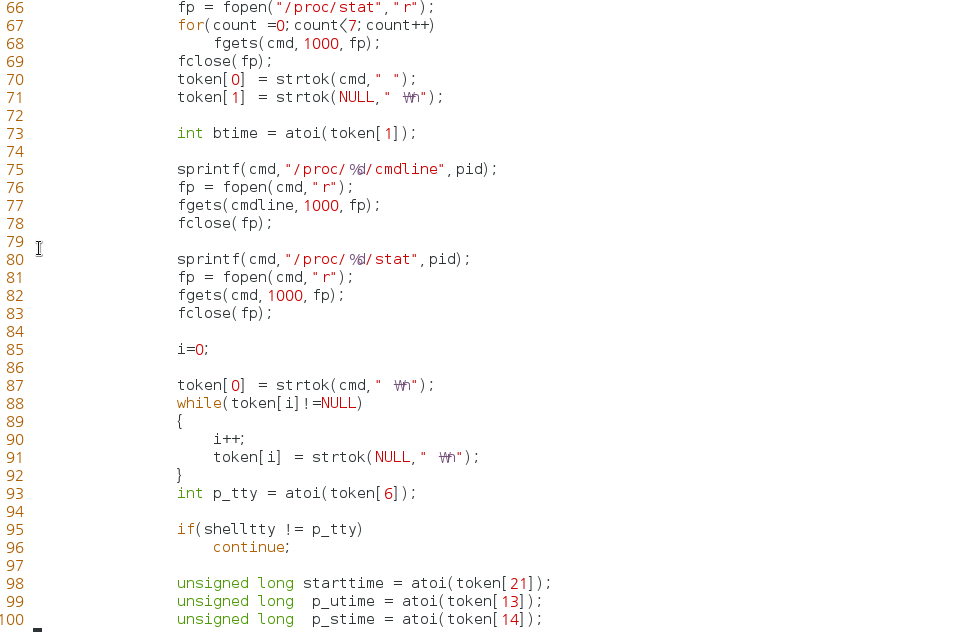
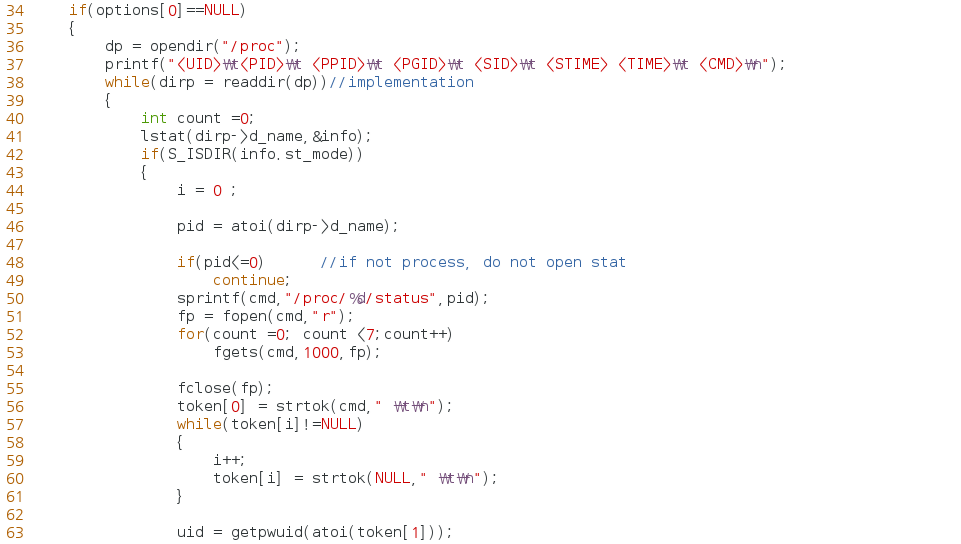
* + - 1. Myps



* Myps 실행화면 fork를 통해 구현함으로써 ./test 라는 프로세스가 두 번 실행됨을 알 수 있고 ./test와 같은 tty의 프로세스만 출력해준다.



Myps 명령어는 ps 명령어를 구현한다. Ps 명령어를 구현하는데 tty기준으로 구현하기 위해선 myshell의 tty를 구해서 비교 후 shell tty기준으로 ps명령어의 내용을 출력해야한다. 그러므로 실행시킨 나만의 쉘의 pid를 구한 후 이 pid를 이용해 현재 프로세스의 tty를 구하고 이를 이용해 같은 tty인 프로세스를 출력해준다.

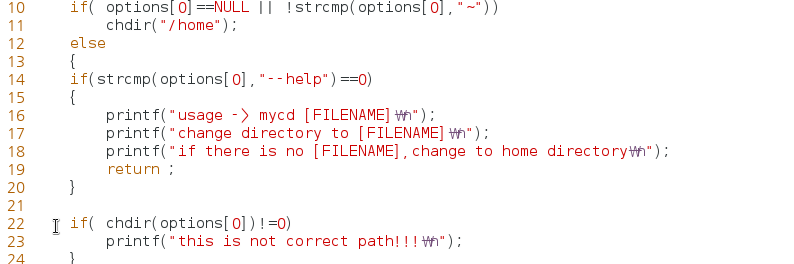


그리고 ps의 각 칼럼들에 들어갈 정보는 여러곳에서 가져올 필요가 있다. Cmd의 정보는 /proc/[pid]/cmdline 에서, pid,ppid,pgid,sid 등은 /proc/[pid]/stat 등에서 가져와서 적절히 정보를 가공한 후 출력해주는 식으로 구현했다. 특히 Stime과 time 칼럼이 매우 어려웠는데, 이들은 여러 곳에서 정보를 가져와 가공해야만 하기 때문에 난이도가 있었다. Stime의 경우 /proc/status 에서 btime 을 가져오고 /proc/[pid]/stat에서 starttime 을 가져와서 starttime을 cpu의 HZ값인 100으로 나눈 후 더해서 이를 localtime 함수를 통해서 나타냈다. Starttime 은 단위가 sec가 아니라서 한번 더 가공을 해줘야하므로 HZ값으로 나누어줬다.

* + - 1. Mycd



* Mycd명령어의 실행화면



Mycd의 경우 chdir이라는 함수로 쉽게 구현하였다. 명령어의 인자가 없거나 ~인 경우 홈디렉토리로 이동하게 하였고, 경로에 맞지않으면 chdir이 -1을 리턴하므로 이를 이용해 에러 메시지도 출력하게 하였다.

* + - 1. Mypwd

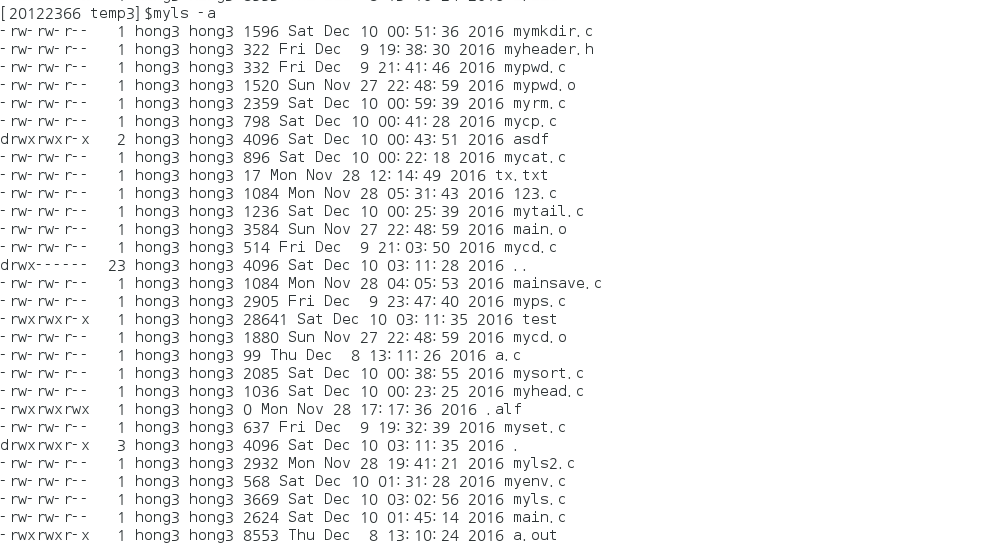


* Mypwd 실행화면

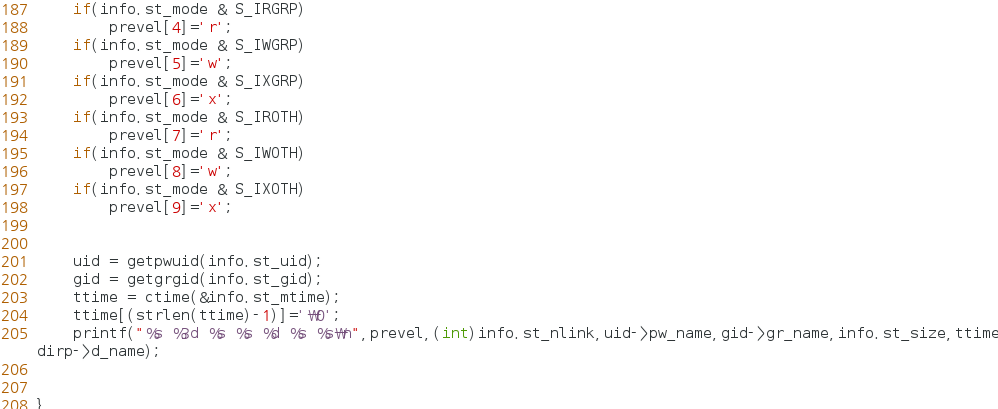
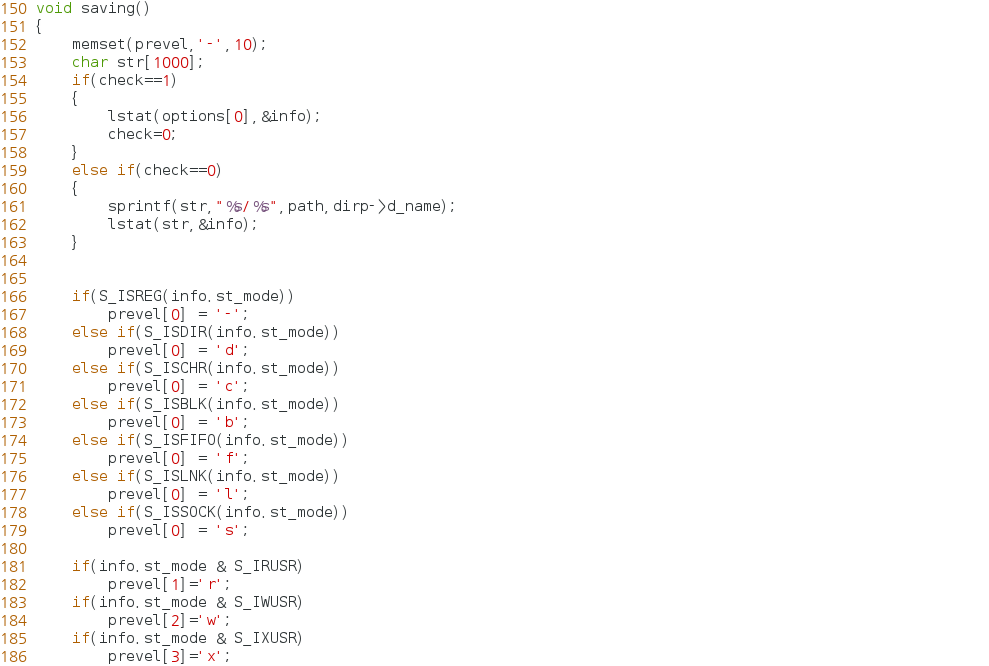


Getcwd함수를 이용해서 현재 디렉토리를 바로 받아온 후 출력해주었다.

* + - 1. Myls



* Myls –a 실행화면, 현재 디렉토리의 숨김파일까지 보여준다.



Myls의 한 터플을 출력하는 함수인 saving()함수이다. stat구조체를 참조하여 각 권한을 조사하고 stat구조체의 내용을 이용해 uid,gid 등의 정보를 구해왔다. 이 uid gid를 이용해서 또다른 구조체인 passwd ,group 등의 구조체를 getpwuid, getgrgid함수로 반환받았고, 여기에 있는 pw\_name , gr\_name 정보를 출력해주었다. 그 후 나머지 칼럼들은 stat구조체의 값들을 바로 출력해주는 식으로 나타내었다.

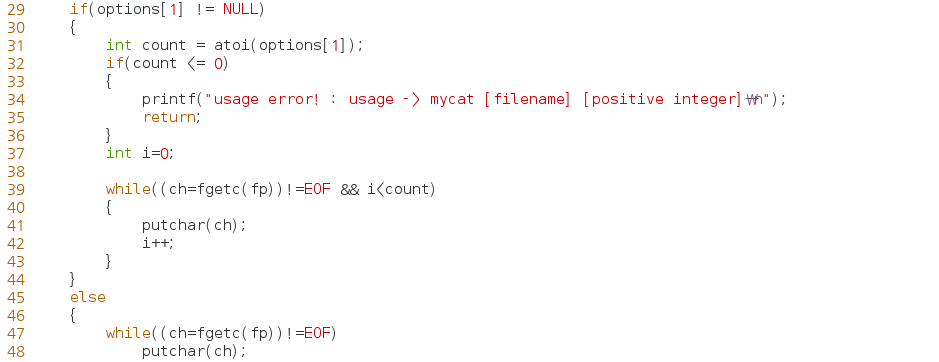


각 터플들을 구하는 방법은 saving()함수를 부름으로써 구현하였고 위의 스크린샷은 받은 스트링이 절대경로로 작성됐는지 상대경로로 작성했는지를 먼저 검사한다. 그 후 알맞게 opendir함수를 부르고 이를 통해 파일인지 디렉토리인지 구분한다. Dp==NULL이면 디렉토리가 열리지 않는다는 뜻이므로 파일임을 짐작할 수 있다. 그러므로 현재 디렉토리의 파일정보를 출력해준다. Dp!=null이라면 디렉토리임으로 그 디렉토리 내의 파일들의 정보를 출력해준다.

* + - 1. Mycat



* Mycat 명령어의 실행화면



한 바이트씩 다 읽어와서 출력해주는 명령어이다. Fgetc로 한 바이트씩 읽어서 출력해주고, 만약 number 옵션을 줬을 경우 카운트를 해가면서 number 바이트까지만 출력한다.

* + - 1. Myhead



* Myhead 명령어의 실행화면

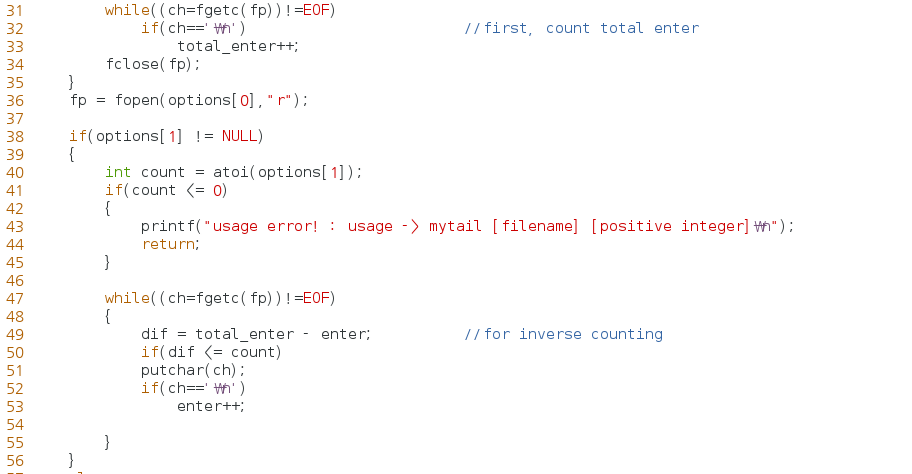


Mycat과 비슷하게 구현하였다. 차이점은 개행문자 마다 카운트를 해 주어서 개행문자를 기준으로 출력해준다는 것이다.

* + - 1. Mytail

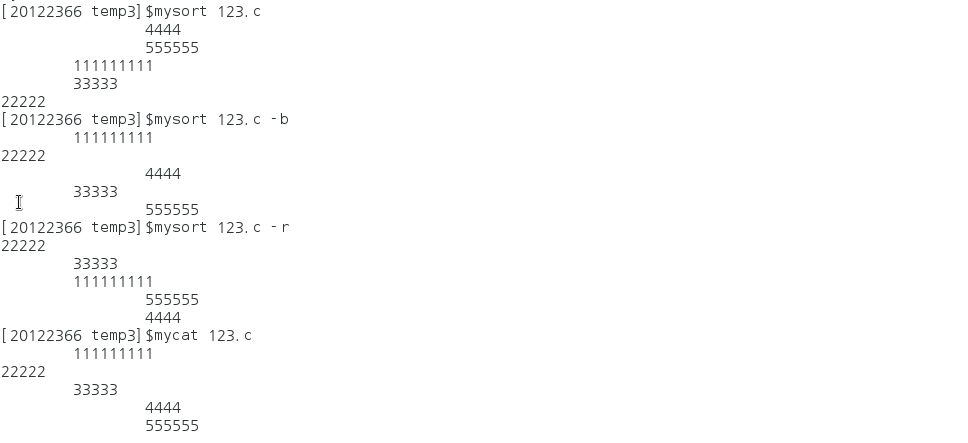


* Mytail 명령어의 실행화면

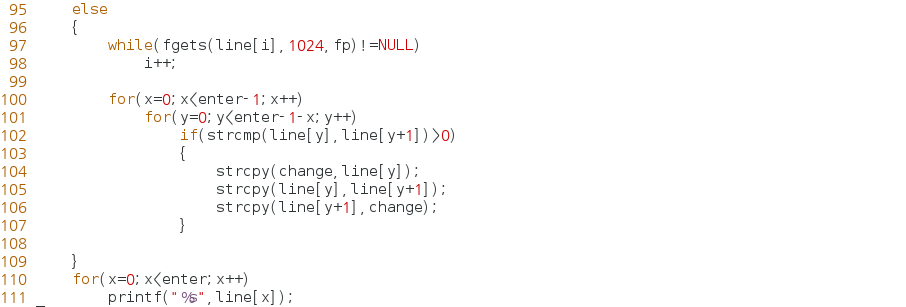


파일을 끝에서부터 몇 줄을 출력해주는 명령어이다. 명령어를 실행하면 파일을 한번 읽어서 개행문자의 수를 읽고, 이를 이용해 끝에서부터 몇 줄을 읽을지 계산한다. 그 후 비교문을 통해서 출력해준다.

* + - 1. Mysort



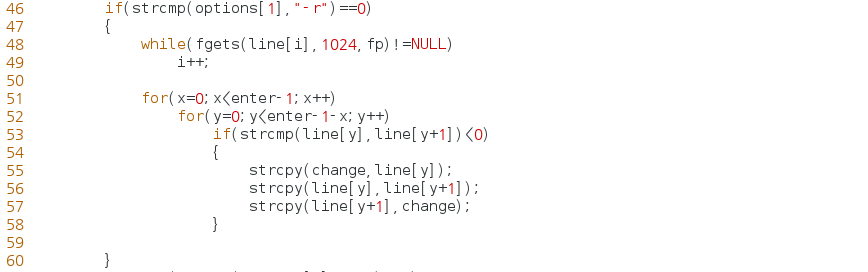
* Mysort 명령어의 실행화면



버블 정렬을 이용해서 라인별로 정렬을 한 후 출력해준다.



-b 옵션의 구현방법이다. Line2라는 배열을 하나 더 만들어서 모든 공백을 없애준다. 그렇게 되면 공백이 없어진 상태로 line2에 저장되므로 이를 이용해 비교를 한다. 비교문에서만 line2를 이용해 정렬을 하고 실제로 출력하는 건 line으로 한다.



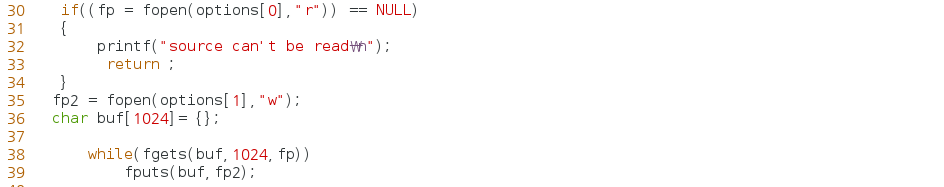
-r 옵션의 구현방법이다.

기존 정렬하는 방식에서 비교문만 수정해서 쉽게 구현했다.

* + - 1. Mycp



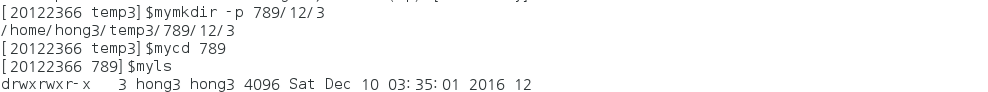
* Mycp 명령어의 실행화면



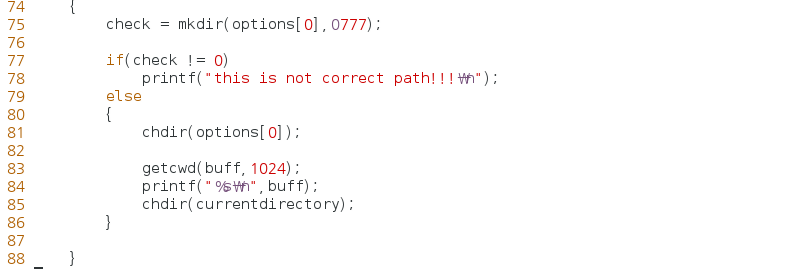
Source로 읽어올 것이 있으면 읽어 온 후 dest로 source의 내용 전부를 복사해온다. 반복문안의 fgets, fputs로 쉽게 구현하였다.

* + - 1. Mymkdir

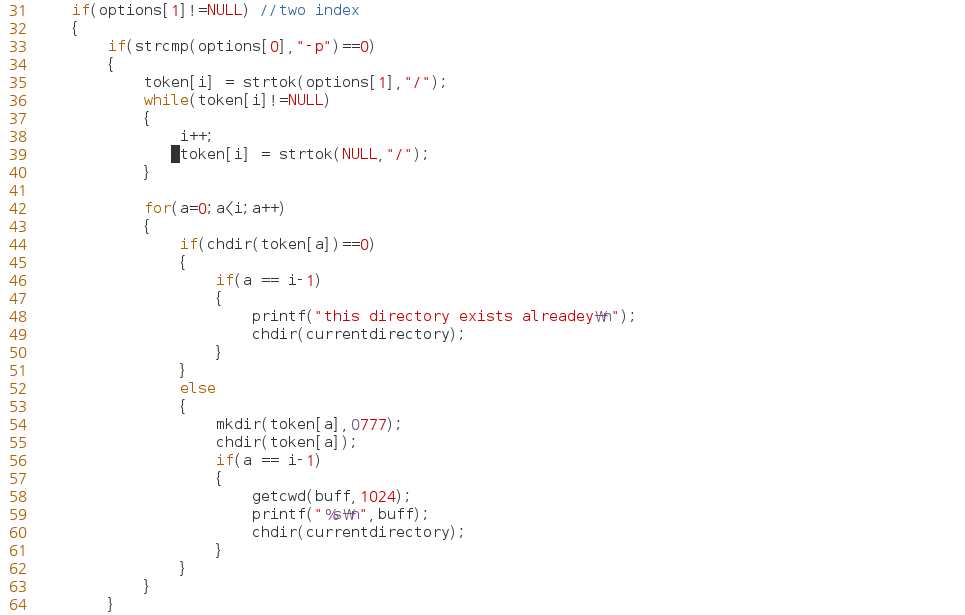




* Mymkdir 명령어의 실행화면



이 명령어 역시 mkdir이란 라이브러리 함수가 있으므로 쉽게 구현하였다. 명령어 뒤의 문자열로 디렉토리를 만들며, 만약 만드려는 디렉토리가 이미 존재할 시 check가 -1을 리턴하므로 에러메세지를 출력한다.



-p 옵션의 구현방법이다. -p옵션이 들어오면 / 토큰으로 하나씩 자르고 경로를 하나씩 만들어간다. 그 과정에서 chdir함수가 필요하며 다 만들고 난 후 만든 디렉토리의 절대경로를 출력하고 다시 원래의 디렉토리로 돌아간다.

* + - 1. Myrm



* Myrm 명령어의 실행화면. Mymkdir 명령어로 만든 디렉토리를 –rf 옵션을 사용해 삭제했다.



File 일 때만 삭제하는 알고리즘이다. 들어온 스트링을 비교해서 파일이면 삭제하고 디렉토리면 에러메세지를 출력하게 했다.



-rf 옵션의 구현방법이다. –rf 옵션이 들어왔을 때는 deleting이라는 함수를 구현해서 재귀적으로 삭제하도록 구현했다. 삭제하려는 디렉토리 안에 또 다른 디렉토리가 있을 수 있으므로 이렇게 구현해야만 했다. 디렉토리 안에 내용물이 있으면 삭제가 안되므로 내용물을 다 삭제하고 rmdir함수를 사용해서 디렉토리를 삭제해준다.

1. 결론

리눅스의 쉘을 직접 구현해봄으로써 리눅스의 내부 구조에 대해서 좀 더 심도있는 이해가 가능하게 되었다. 실제로 /proc 디렉토리에 많은 파일, 디렉토리들이 있었는데 이 과제를 수행하기 전엔 각 파일들이 무엇을 의미하고 나타내는 지 알지 못했는데, 이제는 숫자로 생기는 디렉토리들이 무엇을 의미하는지, 그 안의 파일들이 무엇을 의미하는지 조금이나마 알게 되었다. 또한 각종 라이브러리 함수를 쓰면서 그 사용법과 동작원리에 대해서 알게 되었으며 이에 따른 각종 구조체에 대한 지식도 습득하게 되었다.

1. 참고 문헌 및 자료

[https://ko.wikipedia.org/wiki/%EC%85%B8](https://ko.wikipedia.org/wiki/%EC%BB%A4%EB%84%90_(%EC%BB%B4%ED%93%A8%ED%8C%85))

* 쉘의 기본 개념.

<http://gongdoo.tistory.com/241>

* 리눅스 쉘 스크립트 개념.

<http://forum.falinux.com/zbxe/index.php?document_srl=405822&mid=gcc>

* Makefile 만들기

<http://forum.falinux.com/zbxe/>

* 각종 라이브러리 함수에 대한 헤더, 인자 등의 정보 참고