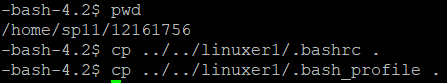
**System Programming Lect8 HW**

**001분반 / 12161756 / 윤성호**

**2. Homework**

**0) Set up bash startup file: copy .bashrc and .bash\_profile into your login directory**

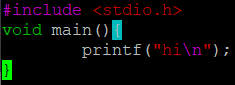
****

: 현재 디렉토리(로그인 디렉토리)에 /home/linuxer1에 있는 .bashrc와 bash\_profile을 복사한다. 그 후 putty창을 닫고 다시 실행한다.

****

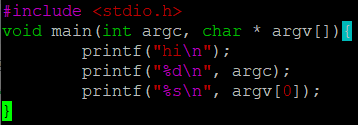
: 프롬프트가 [12161756@linuxer2 ~]로 변경되었다. 이제 프로그램명 앞에 [./]를 붙이지 않아도 프로그램을 실행할 수 있게 되었다. 현재 디렉토리 정보가 포함된 .bashrc와 bash\_profile을 불러왔기 때문이다.

**1) Try following program which doesn't receive command line arguments.**

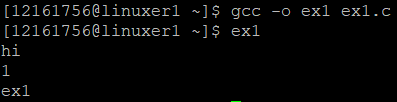


: command line argument를 하나도 받지 않는 프로그램이다. void main의 소괄호 안에 아무 것도 입력하지 않는다.

**2) Try following program that receives one command line argument.**

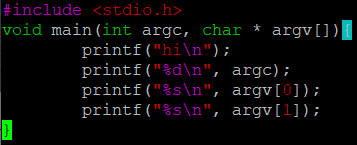


: argc 인자에는 command line argument의 개수, 포인터 배열인 argv[]에는 각 argument가 저장된 주소값들이 넘겨진다.

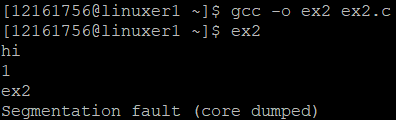


: argc값을 통해 이 프로그램에 넘겨준 command line argument는 1개이고, 이 argument값은 argv[0]를 통해 실행파일명인 ex1임을 알 수 있다.

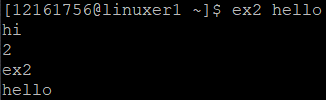
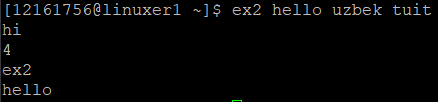
**3) Try following program that receives two command line arguments.**



: 2)에서 [printf(“%s\n”, argv[1]);] 한 줄이 추가되어 입력이 필요한 argument수는 2개가 된다.



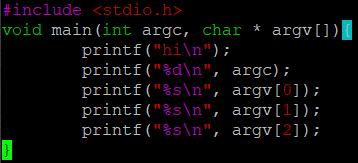
: ex2로만 프로그램을 실행한다. 실행 결과를 통해 이 프로그램에는 ex2라는 argument 하나만 넘겨진 것을 알 수 있다. 마지막 줄에서 오류가 발생한 이유는 넘기지 않은 두 번째 argument가 저장된 argv[1]를 출력하려했기 때문이다.

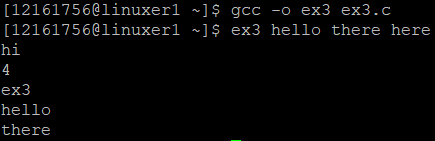
: [ex2 hello]로 argument 2개를 넘기자 오류없이 잘 실행되었다.

: 프로그램에 필요 이상으로 argument를 전달하면 프로그램은 필요한 것을 제외한 나머지는 무시한다. (3번째, 4번째 argument인 uzbek과 tuit가 입력되었음에도 실행 오류는 발생하지 않는다.)

**4) A program that receives three command line arguments.**

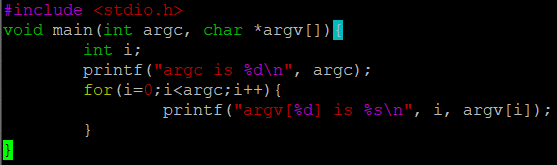


: command line argument 3개를 필요하는 프로그램이다.

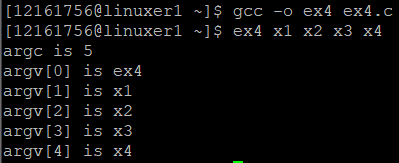


: 4개의 argument를 넘겨줬으나 프로그램은 3번째 argument까지만 출력하여 here은 출력되지 않는다. here을 출력하려면 argv[3]을 printf하는 코드가 추가되어야 한다.

**5) A program that receives any number of arguments.**

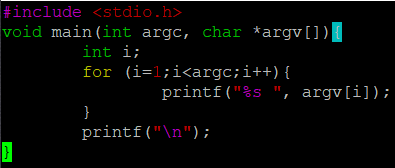


: command line argument의 개수(argc)와 for문으로 각 argument(argv[])를 출력하는 프로그램이다.

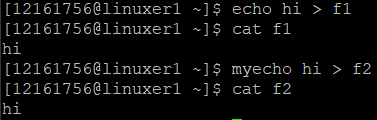
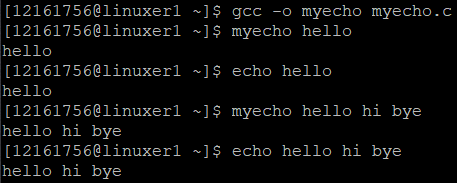


: [ex4 x1 x2 x3 x4]로 실행할 경우 argument의 수는 5이고 각 argument의 주소값은 입력 순서대로 argv[0]부터 차례대로 저장되는 것을 알 수 있다.

**6) Try following and explain the difference from echo command.**

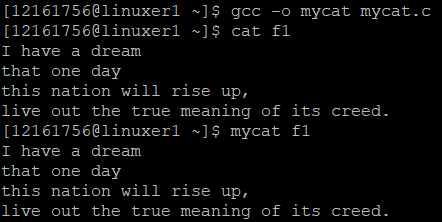
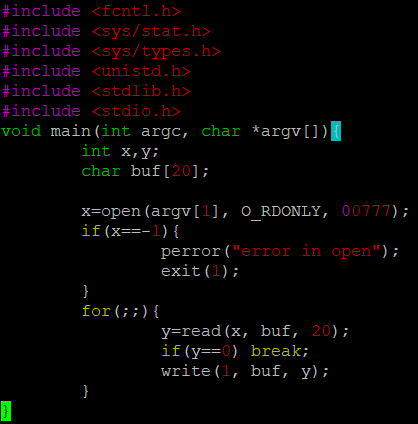
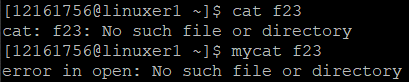


: for문으로 실행파일명(argv[0])을 제외한 argument를 모두 출력하는 프로그램이다.



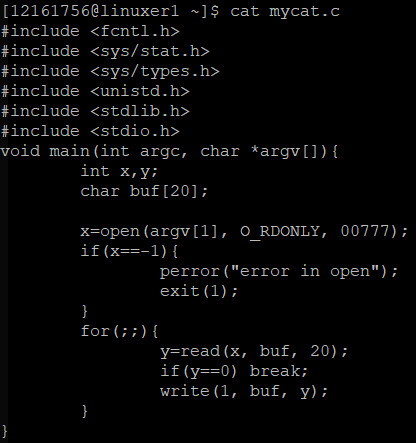
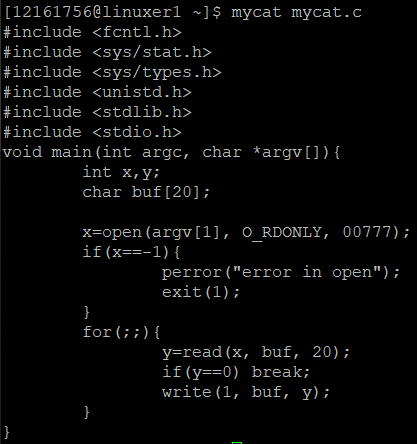
1. 작성한 프로그램과 echo 명령어가 동일한 기능을 하는 모습이다.

2. echo와 같이 활용하는 redirection(>)도 똑같이 활용 가능했다.

**7) Try following and explain the difference from cat command.**

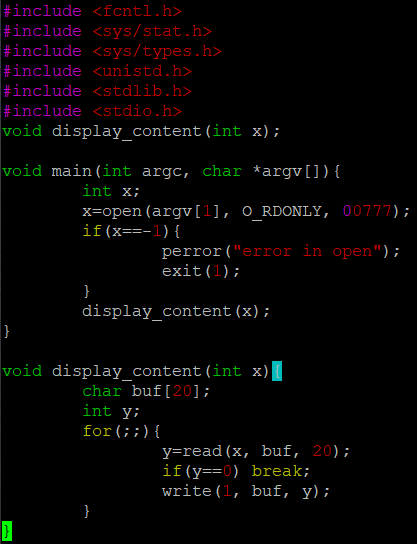
1. 프로그램명(mycat) 뒤에 적을 argument인 argv[1]를 읽기전용으로 open하여 file 번호를 변수 x에 return한다. 오류 발생시 x에 -1이 return되는 점을 이용하여 if문과 perror로 에러메시지를 출력하고 exit(1)로 프로그램을 종료하는 예외 처리문을 작성한다. 예외 처리문 이후 무한루프에서 argv[1] 파일을 20bytes씩 read하여 buf에 저장한 후 write(1,…,…)로 y bytes만큼 화면에 출력한다. 파일 내용을 모두 읽었을 때 (read한 bytes를 담는 y값이 0이면) 루프를 탈출한다.

2. [cat f1]과 [mycat f1]의 결과가 동일했다.  
3. 현재 디렉토리에 없는 f23파일을 출력 시도시 직접 입력한 오류 메시지만 다를뿐 오류 내용은 동일하게 출력되었다.

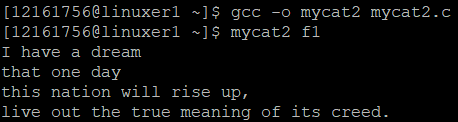
 

: mycat.c 파일도 cat과 mycat에서 모두 동일하게 출력했다.

**8) Try following: mycat2.c. Use functions for your program.**

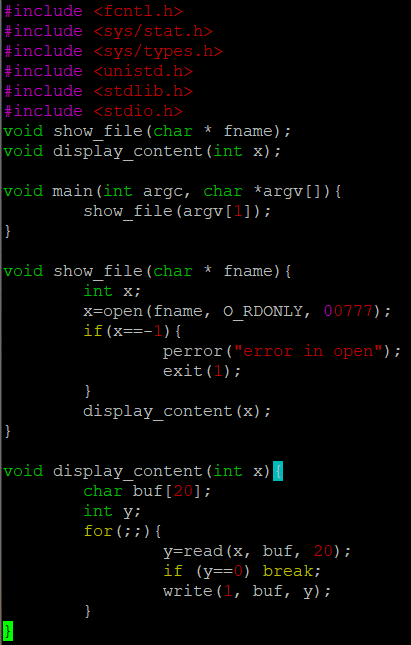


: 7)에서 작성한 mycat.c와 동일하나 예외처리 후 내용을 출력하는 부분을 display\_content라는 void 함수로 분리하였다. 해당 함수 내용을 main문 뒤에서 기술하기 때문에 main문 위에서 먼저 함수를 선언한다. display\_content함수는 file 번호가 담긴 int 변수 x를 인자로 받아 해당 파일의 모든 내용을 출력하는 함수이다.

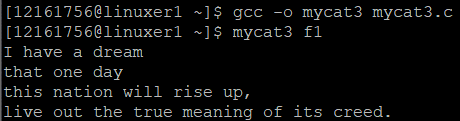


: [mycat2 f1]를 입력하자 f1파일의 모든 내용이 출력되었다.

**9) Try following: mycat3.c.**



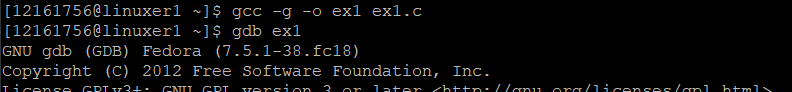
: 이번에는 8)에서 작성한 main문의 모든 내용을 show\_file이라는 함수에서 작성한다. 해당 함수는 char형 포인터 변수 fname을 인수로 한다. main문에서 show\_file함수로 argv[1](내용을 출력할 파일)를 넘긴다. 이후 8)과 동일한 내용이 기술되나, 8)의 open 함수 뒤에 argv[1]이 입력된 자리에 이를 넘겨받은 변수 fname을 입력하는 것이 다르다. 또한, 8)에서 작성했던 display\_content 함수를 show\_file함수 안 마지막 줄에서 불러와 파일 내용을 출력하는 기능을 구현한다.

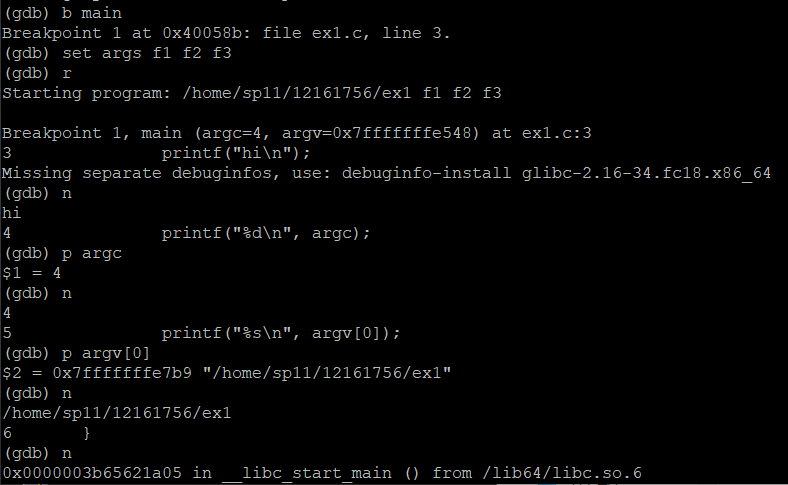


: f1의 내용이 모두 출력되었다. 7) -> 8) -> 9) 순으로 기능별로 함수를 분리했고 실행 결과는 모두 같았다.

**10) You can debug programs with command line arguments as follows. Debug mycat3.c with gdb. To pass command line arguments to gdb, do "set args arg1 arg2 ...".**

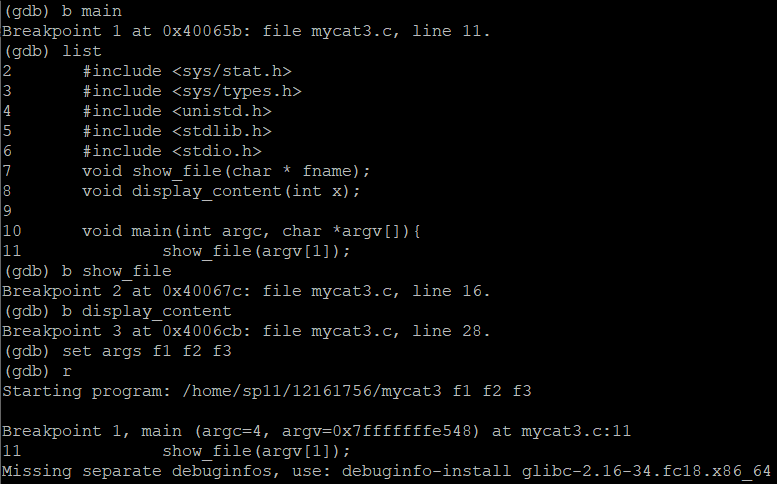
**< Debugging ex1.c >**



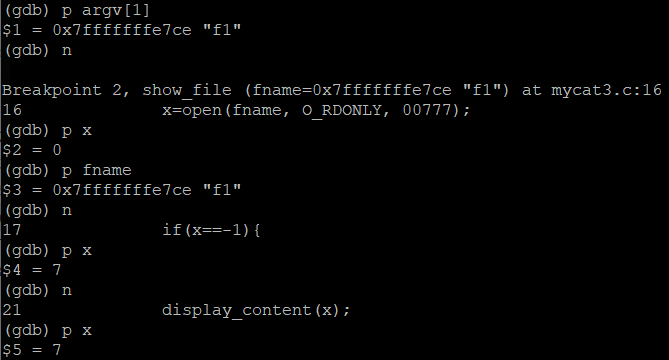


= [gcc -g -o]로 컴파일 후 gdb 디버깅을 하였다. main에 break point를 설정하고 [set args f1 f2 f3]으로 2~4번째 argument로 지정한다. r로 프로그램을 실행한 후 n으로 한 줄씩 실행한다. p로 argc값을 출력하자 4(ex1, f1, f2, f3)가 나왔다. argv[0] 값은 ex1의 주소 값이 출력되었다.

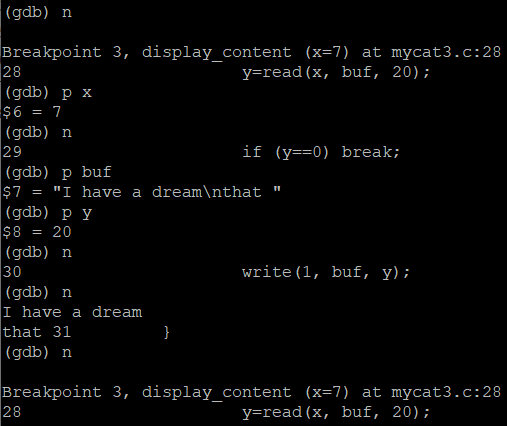
**< debugging mycat3.c >**



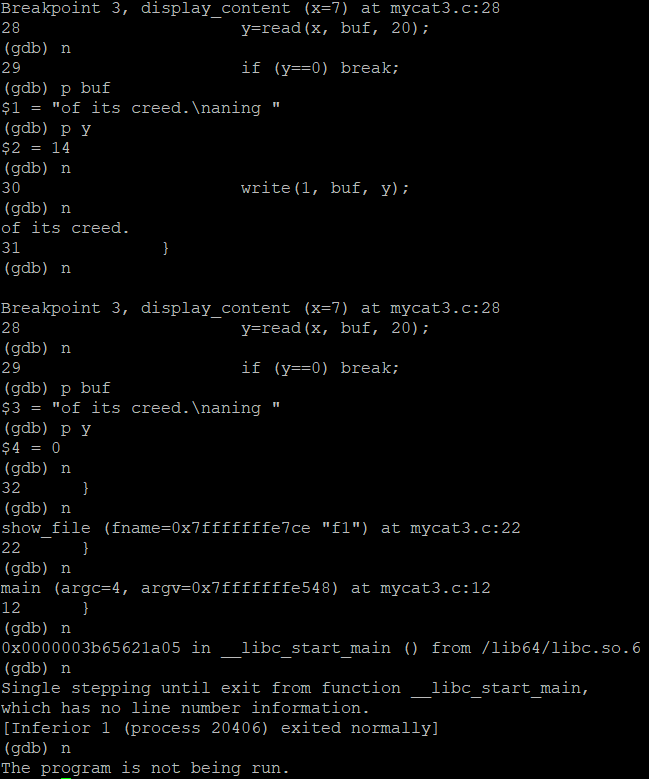
: breakpoint 1은 main, 2는 show\_file, 3은 display\_content 함수로 설정한 후 argument로 f1, f2, f3을 지정한다. 그리고 r로 프로그램을 실행한다.



: argv[1]을 print하자 파일 f1의 주소 값이 출력된다. 다음 줄을 실행하자 지정했던 breakpoint 2(show\_file)로 넘어간다. 16번째 줄에서 x를 print하니 0이 출력된다. 아직 16번째 줄이 실행되지 않았기 때문에 파일 번호가 return 되지 않았다. fname는 show\_file 함수가 main으로부터 넘겨받은 argv[1]값이다. f1의 주소 값이 출력되는 것을 통해 show\_file 함수로 잘 넘어간 것을 알 수 있다. 17번째 줄에서 x값을 출력하자 7이 나왔다. f1을 open하여 생성된 파일번호는 7이다. 21번째 줄에서도 마찬가지로 x값은 7이다.

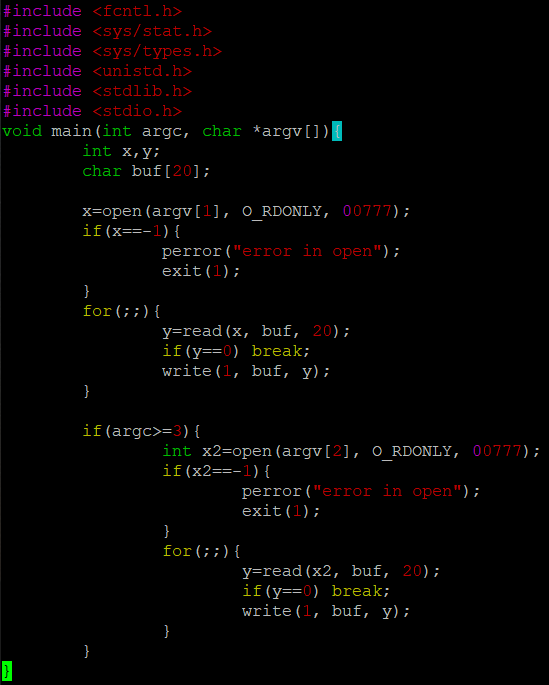


: 이후 n으로 다음 줄을 실행했더니 breakpoint 3(display\_content)로 넘어간다. 이 함수는 show\_file 함수로부터 파일번호 x를 넘겨받는다. x가 7로 출력되어 잘 넘어온 것을 확인하였다. 이후 buf를 print하자 f1의 앞에서부터 20bytes 내용이 출력되었다. read에서 읽은 bytes를 저장하는 변수 y에는 20이 입력되었다. 다음 줄을 계속 실행하자 for루프로 인해 breakpoint로 다시 돌아가는 모습이다.

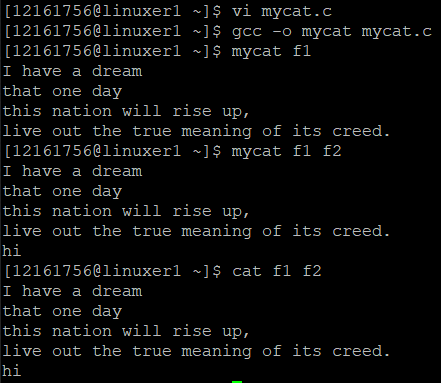


= n으로 계속 다음 줄을 실행하다 buf에 f1의 끝부분인 ‘of its creed.’가 저장될 때 y값은 14로 나타났다. 이후 다시 돌아온 read에서는 f1의 내용을 모두 읽었기 때문에 y=0이 되었고 break로 loop를 빠져나온다. 그리고 이 display\_content 함수를 품은 show\_file과 show\_file를 사용한 main 함수를 거쳐 모든 코드가 종료된다.

**11) Modify mycat.c such that it can handle two input files.**



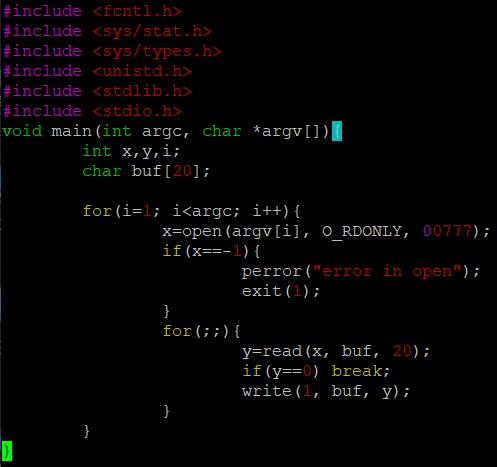
: 기존의 mycat.c의 for문 밑에 argc가 3이상일 때의 조건문을 추가한다. argument가 3개 이상 이란 것은 [mycat f1 f2]처럼 입력한 파일의 수가 두 개 이상인 것을 말한다. 이 조건을 만족할 때 argv[2]를 읽기전용으로 open하여 파일번호를 x2에 return한다. 그리고 x2가 -1이면(오류 발생시) 에러 메시지를 출력하고 프로그램을 종료한다. 에러가 없다면 무한루프를 시작한다. x2를 20bytes씩 read하여 buf에 저장하고 이어서 write로 buffer에서 앞서 read한 y bytes만큼 화면에 출력한다. 파일을 끝까지 읽을 때(y=0)까지 무한루프를 돌면서 위 과정을 반복한다.



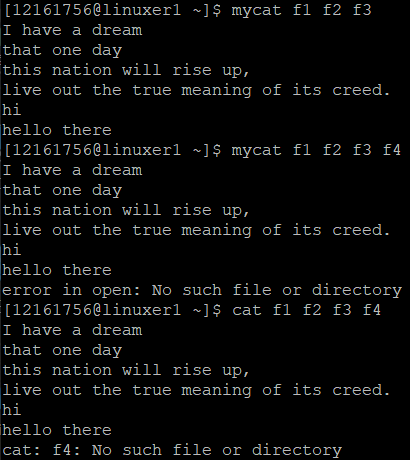
1. [mycat f1]으로 출력할 파일 하나만 넘기면 위 코드에서 if 조건문 이전까지만 실행되어 f1 파일 내용만 출력된다.

2. [mycat f1 f2]로 출력할 파일을 두 개 넘기자 if 조건문을 만족하여 f1에 이어서 f2의 파일 내용(‘hi’)도 출력된다. 이는 [cat f1 f2]의 결과와 같다.

**12) Modify mycat such that it can handle any number of files.**

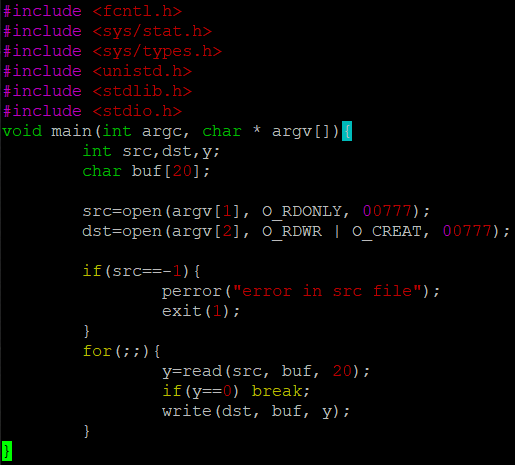


: 최초의 mycat.c 코드에서 변수 선언부 아래를 for문으로 한 번더 씌웠다. i는 1부터 시작하여 루프를 한 번 돌 때마다 1씩 증가한다. i가 argc 미만일 때까지 for루프를 돈다. argv[i]를 읽기 전용르로 open한 뒤 파일 번호를 x에 return한다. x가 -1이면 perror로 에러메시지를 출력하고 프로그램을 종료한다. -1이 아니라면 for루프를 또 돈다. x를 20bytes 읽어 buf에 저장하고 만약 파일을 다 읽었다면(y=0 이면) 루프를 탈출한다. 아니면 write(1,…,…)로 buf의 y bytes만큼을 화면에 출력하는 과정을 반복한다.



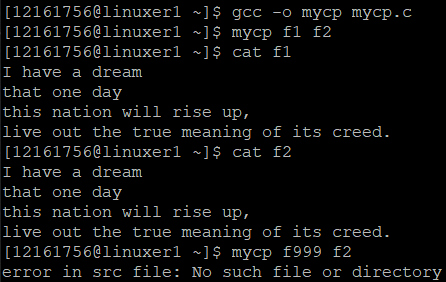
: mycat 뒤로 파일명 argument를 여러개 넘겨주어도 작동하는 것을 확인할 수 있다.

**13) Implement mycp that works similarly to “cp”.**



: 우리가 cp 명령어를 사용할 때 cp 바로 뒤에 위치하는 argv[1]에는 복사할 파일명이 기록된다. 따라서 src라는 변수에 해당 파일을 open한 후 생성되는 파일번호를 return한다. 그 다음 위치하는 argv[2]에는 생성할 복사본의 파일명을 가리키는 주소가 있다. 마찬가지로 dst변수에 파일번호를 담는다.

변수 선언 후 src=-1이면(argv[1] 파일 open시 문제 발생) 에러 메시지를 출력하고 프로그램을 종료한다. 그리고 for 루프를 이용해 src파일 내용을 20bytes씩 read해 buf에 저장하고 저장한 bytes 수를 y에 리턴한다. 그리고 buf에서 y bytes만큼 dst파일에 write하는 과정을 반복한다. y=0일 때(src를 모두 읽었을 때) 루프를 빠져나온다.

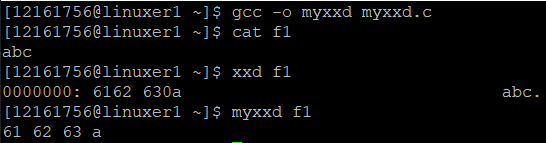


: 작성한 mycp를 이용해 f1파일 내용을 f2로 복사하였다. 존재하지 않는 파일을 복사하려할 때(mycp f999 f2) 오류메시지도 잘 출력되었다.

**14) Implement myxxd that works similarly to “xxd”. Run "myxxd mycat.c". Compare the result with "xxd mycat.c".**

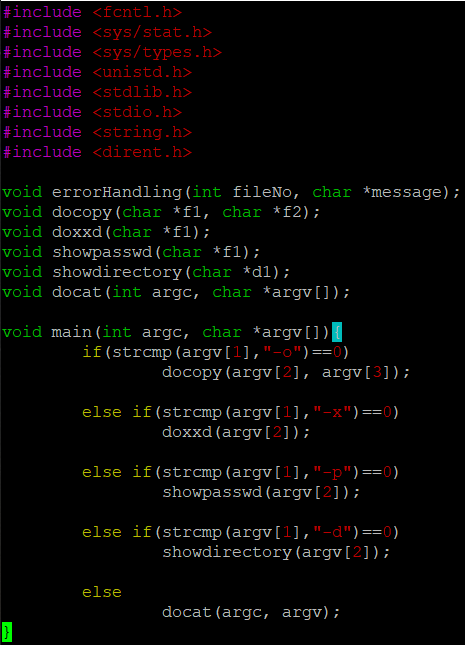


: argv[1]은 16진수로 출력할 파일의 이름을 가리키는 주소값이다. argv[1]을 읽기 전용으로 open한 후 해당 파일번호를 x에 return한다. open에 문제가 있으면 x에 -1이 return되는 것을 이용해 예외처리문을 작성했다. 이후 for문으로 1byte씩 파일 x를 읽어 buf에 저장 후 이어서 printf로 16진수로 출력하는 것을 반복한다.



: f1파일의 내용 ‘abc’가 xxd와 myxxd를 이용했을 때 모두 61 62 63 a로 동일하게 출력되었다.

**15) Modify mycat to handle various options. The second argument is either a file or an option. If it is a file, just display the contents. If it is an option (starting with ‘-‘), perform the following corresponding actions.**



1. showdirectory함수에서 opendir()과 readdir()를 사용하기 위해 <dirent.h>를 include한다.

2. main문에서 argc와 argv를 통해 command line argument의 정보를 넘겨받는다.

3. 명령창에서 mycat 뒤에 입력할 옵션값은 argv[1]로 넘겨지게 된다. 이 값을 strcmp로 비교하여 -o, -x, -p, -d일 때 각각 docopy, doxxd, showpasswd, showdirectory함수를 실행한다.

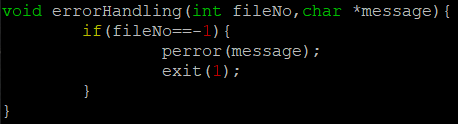
3. docopy함수에는 argv[2](source), argv[3](destination) argument를 넘긴다.

4. doxxd함수에는 open할 파일명(을 가리키는 포인터)인 argv[2]를 넘긴다.

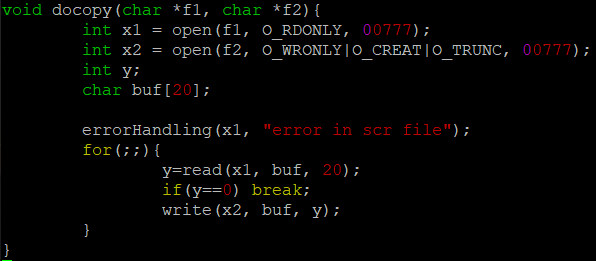
5. showpasswd함수에는 open할 /etc/passwd를 가리키는 argv[2]가 넘겨진다.

6. showdirectory함수에는 정보를 확인할 directory명을 가리키는 argv[2]가 넘겨진다.

7. 위의 옵션에 의한 함수에 해당되지 않는다면(else) 기존의 cat기능인 docat함수를 실행한다. 이 함수에는 argument수를 나타내는 argc와 각각의 argument를 가리키는 포인터 배열 argv 전체를 넘긴다.

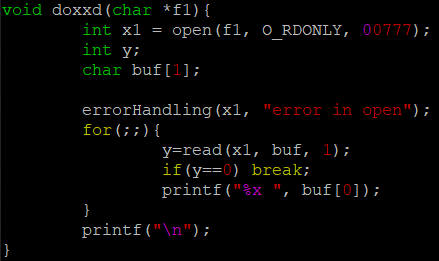


: 파일번호와 출력할 메시지를 넘겨받아 파일번호가 -1일 때(오류 발생) 해당 메시지를 perror로 출력한 후 프로그램을 종료하는 함수이다. docopy와 doxxd, cocat 함수 안에서 활용하였다.



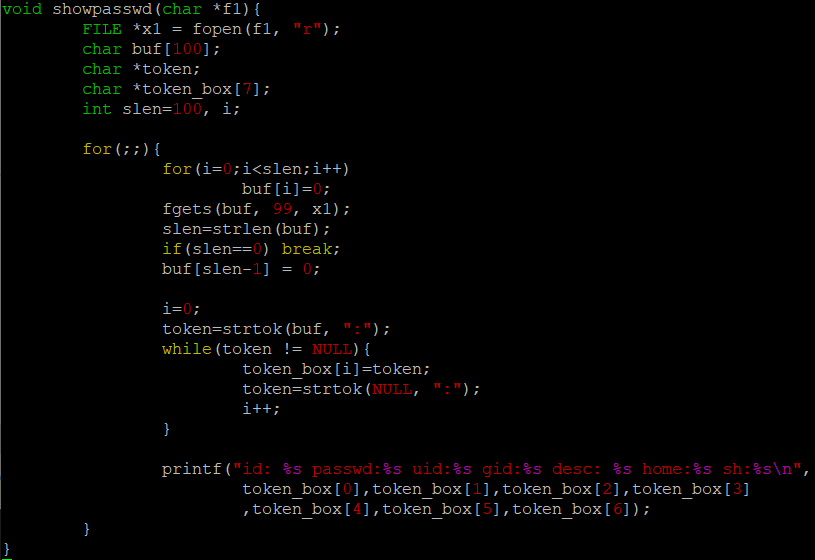
1. f1에서 f2파일로 copy하는 함수이다. f1은 읽기전용으로, f2는 쓰기전용으로 open한다. f2의 경우 기존 파일이 존재하지 않으면 생성하고(O\_CREAT), 있다면 비우고 새로 시작한다.(O\_TRUNC) 먼저 f1의 파일 번호인 x1을 errorHandling 함수로 넘겨주어 예외처리를 한다.

2. 이후 for루프를 돌며 f1의 내용을 20bytes씩 read하여 buf에 저장한다. 만약 f1파일을 모두 읽었다면(y=0) 루프를 탈출한다. 아니라면 buf에 저장된 y bytes만큼을 f2(x2)에 write하는 과정을 계속 반복한다.



1. f1파일을 16진수로 출력하는 함수이다. f1을 읽기전용으로 open하여 파일 번호를 x1에 return한다. 마찬가지로 x1을 이용해 errorHandling한다.

2. for루프를 돌면서 f1(x1)을 1bytes씩 읽어 buf에 저장한다. f1을 모두 읽었을 때(y=0) 루프를 탈출한다. 그렇지 않으면 buf[0]에 저장된 내용을 16진수(%x) 형태로 출력하는 과정을 반복한다.

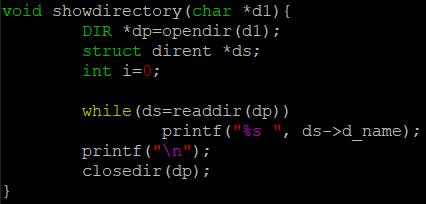


1. f1파일을 읽기전용으로 fopen한 뒤 그 결과를 파일 포인터 x1이 가리키도록 한다.

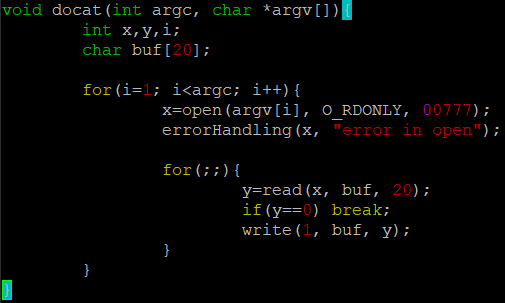
2. 변수 선언 후 무한루프로 시작하며 for문으로 buf 배열을 모두 0으로 초기화한다. 이후 fgets로 f1(x1)의 내용을 99bytes 가져와 buf에 저장한다. fgets는 내용 중 enter입력을 기준으로 한 줄만 불러온다. 변수 slen는 strlen를 이용해 buf의 길이를 저장한다. 만약 slen이 0이면(fgets가 f1의 모든 내용을 불러왔을 때) 루프를 탈출한다. buf[slen-1]=0으로 buf의 마지막 enter입력을 0으로 바꾼다.

3. strtok로 buf의 내용 중 ‘:’문자를 기준으로 token으로 생성한 후 token이 NULL값을 가리키지 않을 동안 while문을 실행한다. token\_box 포인터 배열에 token 포인터 내용을 담은 후 다시 strtok를 이용해 다음 ‘:’ 문자까지 token으로 만든다. 이후 i를 1씩 증가하여 token\_box 포인터 배열 칸을 이동한다.

4. while문이 종료되면 출력 예시에 맞게 token\_box의 내용을 출력한다. 이후 다음 줄을 출력하기 위해 최초 for문의 처음으로 돌아가 위의 과정을 계속 반복한다.

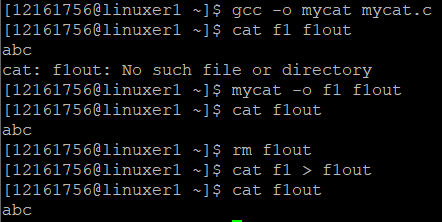


: directory pointer 변수 dp를 선언해 d1을 opendir로 연 결과를 return한다. dirent 구조체 포인터 ds를 선언한다. while문을 돌며 해당 directory를 읽고 dirent 구조체 항목 중 파일 이름이 저장되는 d\_name을 모두 출력한다. while문이 끝나면 closedir로 열었던 directory를 닫는다.



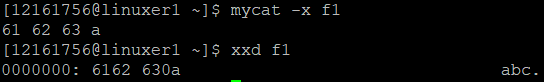
1. for문을 argc를 이용해 argv[0]을 제외한 argument수 만큼 돈다. argv[i]를 읽기 전용으로 열고 x로 파일번호를 return하고 이후 errorHandling함수로 예외처리를 한다.

2. 또 다시 for문으로 argv[i](x)를 20bytes씩 읽어 buf에 저장하고 모두 읽었으면(y=0이면) 루프를 탈출한다. 아니라면 write(1,…,…)를 이용해 화면에 buf를 y bytes만큼 출력하는 과정을 반복한다.

< 1. docopy 결과 >

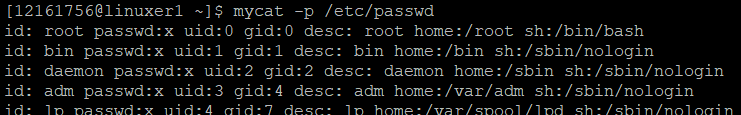
: [mycat -o f1 f1out]은 f1 내용을 f1out으로 복사한다. [cat f1 > f1out]과 결과가 같다.

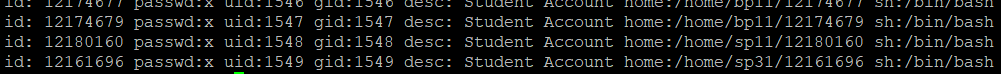
< 2. doxxd 결과 >



: [mycat -x f1]은 f1의 내용을 1byte씩 16진수로 나타낸다. [xxd f1]과 비슷하다.

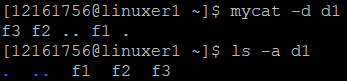
< 3. showpasswd 결과 >



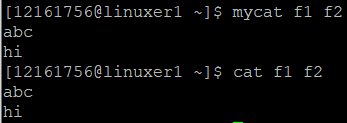


: [mycat -p /etc/passwd]는 /etc/passwd 내용을 사용자가 보기 쉽게 항목별로 제목을 붙여 출력한다.

< 4. showdirectory 결과 >

: [mycat -d d1]은 d1 directory안의 파일을 모두 출력한다. [ls -a d1]과 출력 순서는 다르나 같은 내용을 출력한다.

< 5. docat 결과 >

: [mycat f1 f2]는 f1과 f2의 내용을 차례로 출력한다. [cat f1 f2]과 결과가 똑같다.