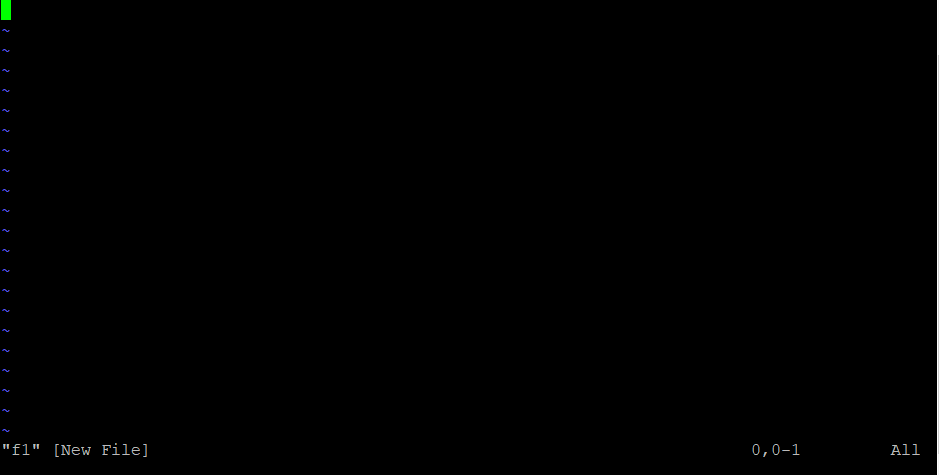
**System Programming Lect3,4 HW**

**001분반 / 12161756 / 윤성호**

<Lect3>

2. Homework

1) Make a file with vi.



**$ vi f1 : f1파일을 생성 후 vi 편집기를 통해 연다.**

**= vi 편집기로 f1 파일을 연 모습이다.**

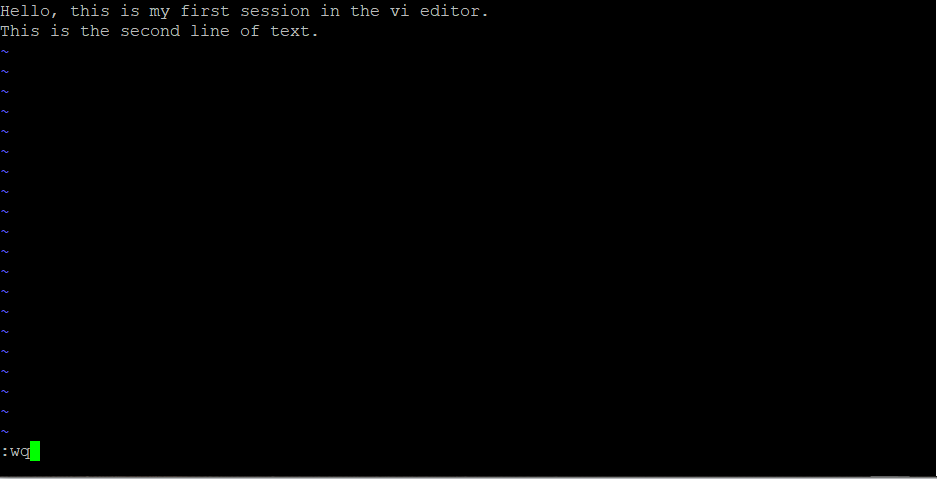
2) Start insertion with ‘i’ key and type following.



**[i] : vi 편집기에서 command mode -> insert mode로 전환한다.**

**= 좌측 하단에 표시된 ‘-- INSERT --’를 통해 insert mode로 전환됐음을 확인할 수 있다. insert mode로 전환하면 텍스트 입력이 가능하다.**

3) Return to command mode with ‘esc’. Save and exit with ‘:wq’



**[esc] : vi편집기에서 insert mode -> command mode로 전환한다.**

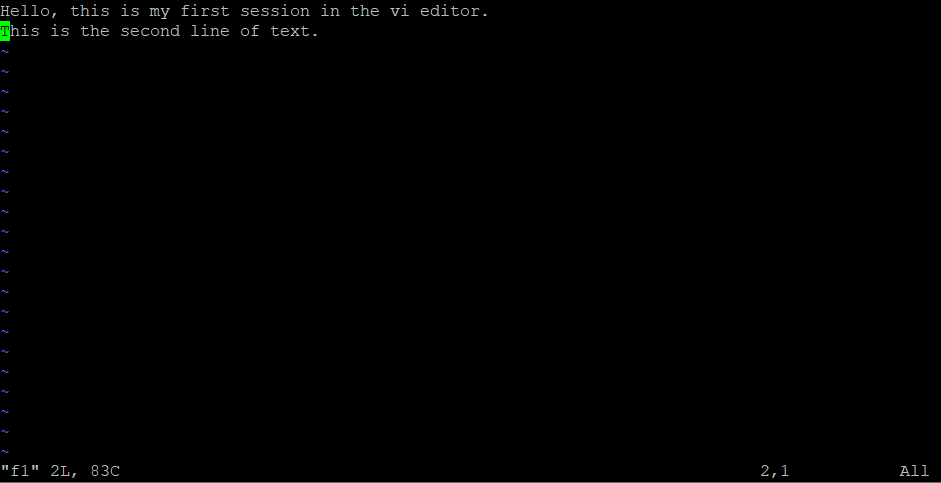
**[:wq] : 현재 상태를 저장 후 vi편집기를 종료한다.**

**= [esc]와 [:wq] 입력 후 enter를 누르기 전 화면이다.**



**= 이 상태에서 enter를 누르면 정상적으로 vi 편집기에서 빠져나온다.**

4) Reopen the file



**= [vi f1] 명령어를 이용해 다시 vi편집기로 f1 파일을 열었다.**

5) Use ‘j’, ‘k’, ‘h’, ‘l’ to move the cursor around. Move the cursor to the word “first”. Use ‘x’ key to delete the word “first”. The result should be as follows.



**<in command mode>**

**[j] : 커서를 위로 이동 / [k] : 아래로 이동 / [h] : 좌로 이동 / [l] : 우로 이동**

**[x] : 현재 커서에 들어온 문자를 제거**

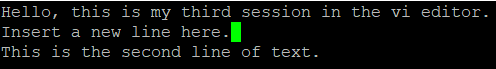
**= j, k, h, l키를 이용해 커서를 ‘first’에 위치한 후 x키를 이용해 제거하였다.**

6) Insert “third” as follows.



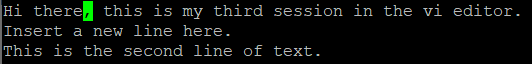
**= [i]를 눌러 insert 모드로 이동한 후 ‘third’를 입력하였다. 이후 [esc], [:wq]를 이용해 빠져나온다.**

7) Add a new line as follows. Use ‘o’ key.



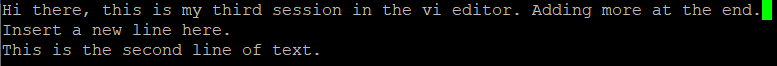
**[o] : 현재 커서가 위치한 line 바로 아래 새로운 줄을 삽입 후insert mode로 전환한다.**

8) Change the beginning as follows. Use ‘x’ and ‘i’ key.



**= [x]를 이용해 ‘ello’를 제거 후 [i]를 이용해 insert mode로 전환하여 ‘i there’ 입력한다.**

9) Add more at the end. Use ‘a’ key.



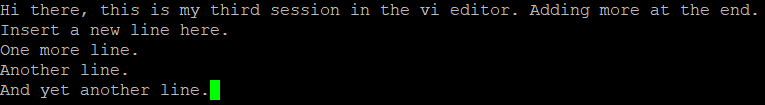
**[a] : 커서를 우측으로 한 칸 이동 후 insert mode로 전환한다.**

10) Delete the last line. Use ‘dd’.



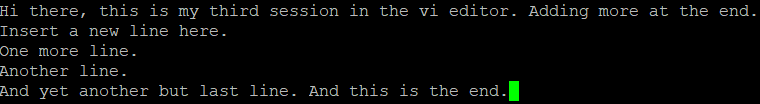
**[dd] : 현재 커서가 있는 한 줄 전체 제거한다.**

11) Add few more lines. Use ‘o’.



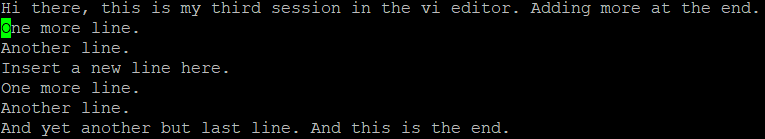
**= [o]를 이용해 insert mode로 전환 후 텍스트를 작성하였다.**

12) Change the last line.



**= 현재의 insert mode에서 마지막 줄 내용을 변경하였다.**

13) Copy and paste as follows. Use ‘2yy’ to copy two lines at the current cursor; move the cursor to another location and use ‘p’ to paste them at that location.

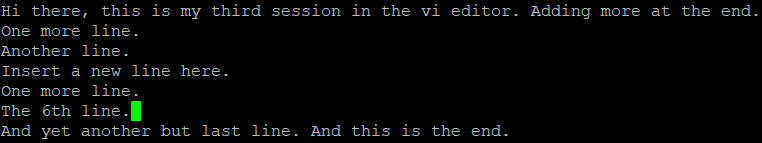


**[2yy] : 현재 커서가 있는 줄과 그 아래줄, 총 두 줄을 복사한다.**

**[p] : 현재 커서가 있는 줄의 다음 줄에 붙여넣는다.**

**= 기존의 3번째 줄(One more line.)에 커서를 둔 후 [2yy]를 누른다. 그 후1번째 줄에 커서를 둔 후 [p]를 누르자 복사한 내용이 1~2번 줄 사이에 삽입되었다.**

14) Go to line 6 with ‘:6’ and make change as follows.



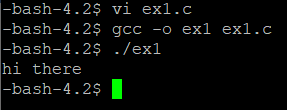
**[:6] : 6번째 줄로 커서가 이동한다.**

**= [:6]로 커서를 이동 후 해당 줄을 수정하였다.**

15) Write a simple c program and compile and run.



**= vi 편집기로 ex1.c라는 파일을 생성 후 ‘hi there’을 출력하는 C프로그램을 작성하였다.**



**$ gcc -o [생성할 실행파일명] [컴파일할 파일명] : C프로그램을 컴파일하여 실행파일을 생성한다.**

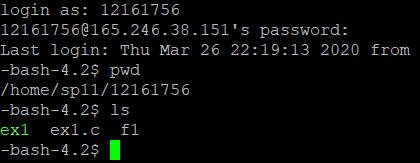
**$ ./[실행할 파일명] : 실행파일을 실행한다.**

**= ex1.c를 gcc 컴파일러로 컴파일하여 ex1이라는 실행파일을 생성한다. [./ex1]을 입력하여 ex1파일이 정상실행되는 것을 확인하였다.**

<lect4>

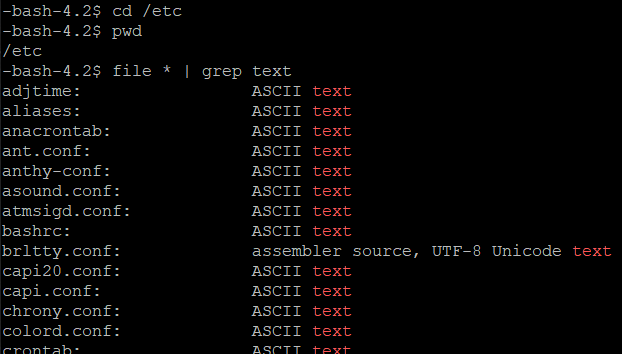
2. Homework

1) Login to the system. Show the current directory. Show what files are in your directory.



**= [pwd]를 통해 현재 디렉토리가 ‘/home/sp11/12161756’임을 확인했다. [ls]를 이용해 현재 디렉토리에 ex1, ex1.c, f1 파일이 있음을 확인했다.**

2) Go to “/etc” directory. "file \*" will show the information for all files in the current directory. Combine "file \*" and "grep" using the pipe symbol(|) to display file information only for text files.

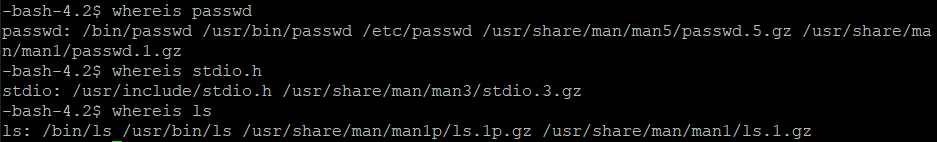


**= [cd /etc]로 절대경로를 입력하여 해당 디렉토리로 이동하였다.**

**$ file \* | grep [찾을 문자] : 현재 디텍토리에 있는 파일명과 타입의 출력물 중 찾을 문자를 포함한 결과만 표시한다.**

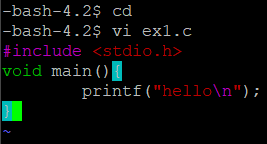
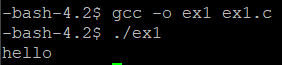
**= ‘text’가 빨간색으로 강조되며 해당 문자를 포함한 결과만 추려진 모습이다.**

3) Find the location of the password file (“passwd”), the location of C header files such as "stdio.h", and the location of utility programs (or Linux commands) such as “ls”. Use "whereis" commad.



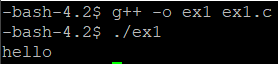
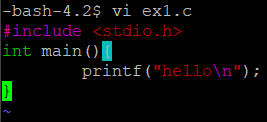
**$ whereis [파일명 or 명령어] : 파일명이나 명령어의 절대경로를 표시한다.**

4) Go to your login directory ("cd" without arguments will move you to your login directory). Make ex1.c using vi. Compile with "gcc" and run.

**= [cd]로 로그인 디렉토리로 이동하였다. [vi ex1.c]로 ex1.c 파일을 vi 편집기로 연 후 [i]를 눌러 insert모드로 전환 후 C프로그램을 작성하였다. 이후 [esc]와 [:wq]로 저장 후 빠져나왔다.**

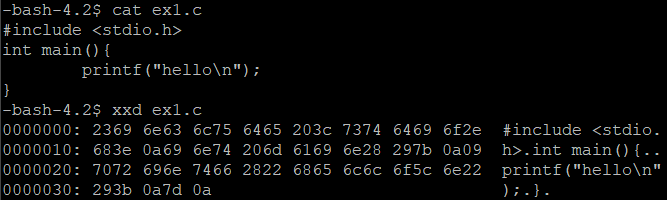
**= [gcc -o ex1 ex1.c]로 ex1.c 파일을 컴파일하여 ex1 실행파일을 생성하였다. [./ex1]으로 정상 실행되는 것을 확인하였다.**



**$ g++ -o [생성할 실행파일명] [컴파일할 파일명] : c++ 프로그램을 컴파일한다.**

**= ex1.c 코드를 수정한 후 [g++]을 이용해 c++ 컴파일을 진행하였고 실행도 성공하였다.**

5) Display the contents of ex1.c using cat and xxd. With xxd, you can see the ascii code for each character in ex1.c. Find the ascii codes for the first line of the program: "#include <stdio.h>".



**$ cat [파일] : 파일 내용을 문자로 취급(아스키 코드 이용)하여 출력한다.**

**$ xxd [파일] : 파일 내용을 바이트 단위로 끊어 16진수로 출력한다.**

**= [cat ex1.c]를 이용하자 이전에 vi 편집기로 작성한 c프로그램 내용이 문자 형태로 출력되었다. [xxd ex1.c]는 왼쪽에서부터 각 라인별 주소값과 각 byte의 헥사값, 아스키 코드로 해석한 값이 출력되었다. 아스키코드로 해석한 값들 역시 각 라인별로 나눠어져있다.**

**= 이를 통해 ‘#include <stdio.h>’에 해당하는 아스키 코드값은 23 69 6e 63 6c 75 64 65 20 3c 73 74 64 69 6f 2e 68 3e임을 유추할 수 있다.**

6) Display the contents of ex1 (the executable file). You cannot use "cat" to see ex1. Why?



**= [cat]를 이용하면 파일 내용이 깨져 나온다. ex1.c 파일을 컴파일하여 생성된 ex1파일에는 기계어가 포함되어 있기에 아스키 코드로 해석하는 [cat]를 이용하면 내용을 제대로 볼 수 없다.**

6-1) The compiler has translated the C statements in ex1.c into machine instructions and stored in ex1:

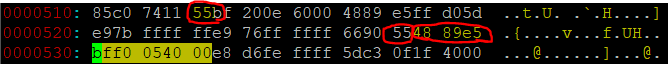
55 -- push rbp

48 89 e5 -- mov rbp, rsp

bf f0 05 40 00 -– mov edi, 0x400df0

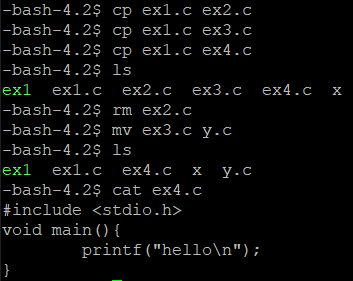
.........

Find these machine instructions in ex1 with xxd. Use "/pattern" command in vi to search for a string.



**= 82~84번째 줄에서 세 개의 기계어를 확인할 수 있었다. g++로 컴파일한 파일에서는 ‘bf f0 05 40 00’이 발견되지 않았다. 다시 c로 작성하여 gcc로 컴파일한 파일을 확인에서는 확인할 수 있었다.**

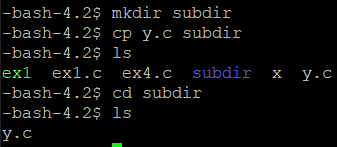
7) Copy ex1.c to ex2.c, ex3.c, and ex4.c. Remove ex2.c. Rename ex3.c to y.c.



**= [cp]로 ex1의 복사본 ex2.c, ex3.c, ex4.c를 생성했다. [rm]으로 ex2.c를 제거하였으며 [mv]로 ex3.c의 이름을 y.c로 변경하였다.**

**$ mv [변경할 파일명] [변경후 파일명] : 파일명을 변경한다.**

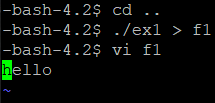
8) Make a subdirectory. Copy y.c in this subdirectory.



**= [mkdir]로 ‘subdir’이라는 디렉터리를 생성하고 [cp]로 subdir안에 y.c의 복사본을 생성한다.**

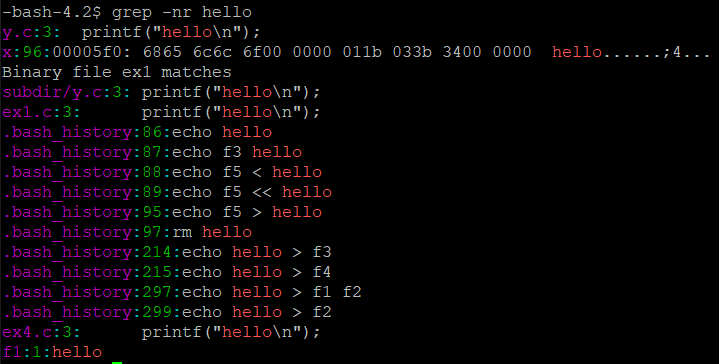
9) Redirect the output of ex1 to another file using ">" symbol.

$ ./ex1 > f1



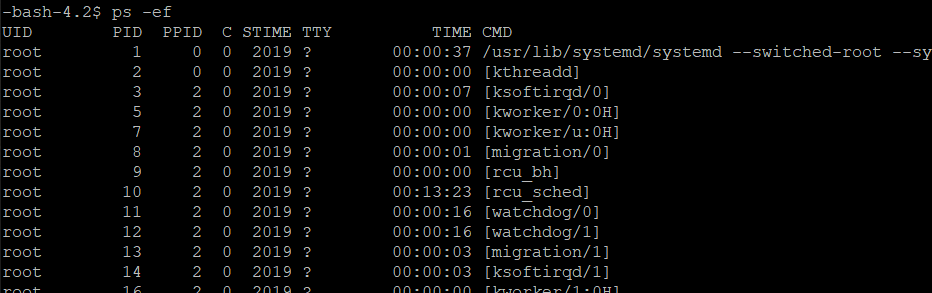
**= [cd ..]으로 ex1 파일이 있는 이전 디렉토리로 돌아간 후 redirection(>)로 ex1의 실행결과를 f1 파일에 저장한다. vi로 확인해보니 ex1의 실행결과인 ‘hello’가 출력되었다.**

10) Use grep to search "hello" in all files (use -nr option).



**= [grep -nr]을 이용해 ‘hello’가 포함된 파일을 모두 찾았다. 이전 명령어 입력 기록을 담고 있는 .bash\_history 파일에서도 찾을 수 있었다.**

11) Find out what processes exist in your system. Use "ps -ef".



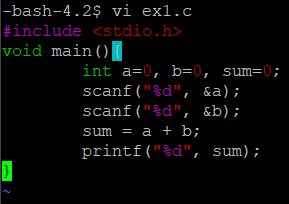
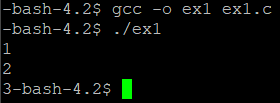
**= [ps -ef]를 이용하여 모든 사용자들의 프로세스 목록을 모두 출력하였다.**

12) "ps -ef" shows all the processes in the system. How do you know which ones are yours? Use "tty" for this purpose. Note that when a user logs in, the system allocates a terminal, and you can find the terminal number with "tty" command.

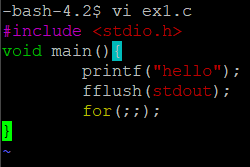
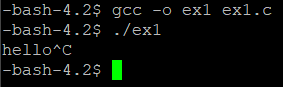
**= [tty]로 현재 사용 중인 터미널 이름을 출력하였다. [ps -ef]의 출력에서 pts/12를 찾아보니 UID에 내 계정명이 표시된 것을 확인할 수 있다.**

13) Modify ex1.c so that it receives two numbers from the user and prints the sum. Use scanf() for this.

**= ex1.c 파일을 scanf와 printf로 두 개의 숫자를 입력받고 합을 출력하는 코드로 수정하였다. 1+2=3이 출력되는 모습이다.**

14) Modify ex1.c so that it contains an infinite loop after printing "hello".

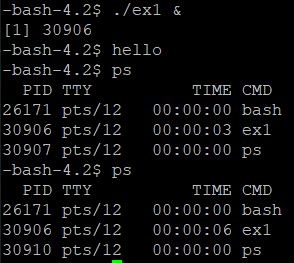
 

**= ‘hello’ 출력 후 for 무한 루프를 도는 C프로그램을 작성하였다. 실행해보니 ‘hello’ 출력 후 무한 루프를 도느라 프로그램이 끝나지 않았고 [ctrl + c]로 빠져나올 수 있었다.**

15) Run the program with & at the end, and use ps to check its status.

$ ./ex1 &

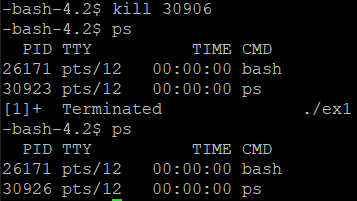
$ ps



**$ ./실행파일명 & : 실행파일을 백그라운드에서 실행한다.**

**= [./ex1 &]를 입력하자 해당 PID를 출력해준 후 [./ex1]과는 달리 다른 명령어를 입력할 수 있는 상태가 된다. 이후 [ps]로 실행 중인 프로세스 목록을 확인해보니 ex1이 실행 중인 것을 확인할 수 있었다.**

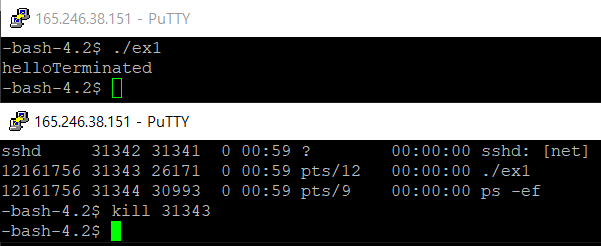
16) Kill it with "kill" command.



**$ kill [PID] : 해당 PID의 프로세스를 종료한다.**

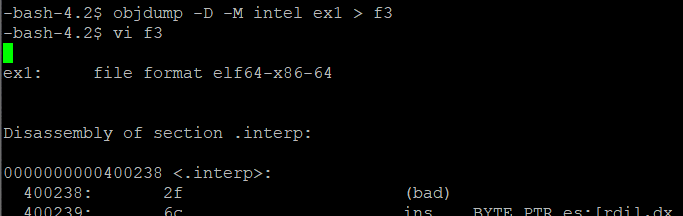
**= 30906 PID를 가진 ex1 프로세스가 종료됐다.**

17) Run the program again without & at the end. Open another login window, find out the process ID of the process running in the first window, and kill it.



**= [./ex1]을 실행해 무한루프를 도는 것을 다른 로그인 창에서 [ps -ef]를 입력해 해당 프로세스의 PID가 31343인 것을 확인했다. 이후 [kill 31343]을 입력하자 첫 번째 로그인 창에서 ‘Terminated’가 표시되면서 무한 루프가 멈췄다.**

18) Use "objdump -D -M intel ex1" to dump the assembly code of ex1.c. Find <main>.

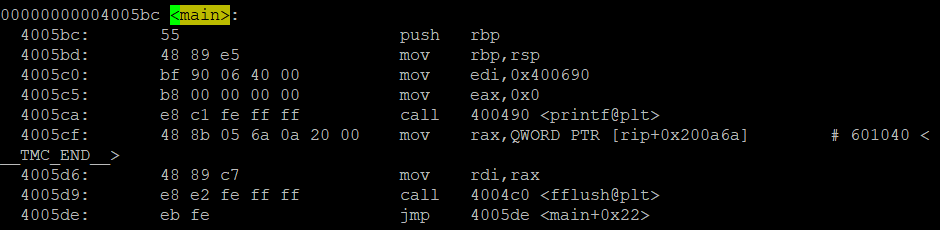


**$ objdump : 바이너리에 있는 기계어를 어셈블리 코드로 변환한다.**

**[-D] : 바이너리의 모든 정보를 어셈블리어로 변환한다. (--disassemble-all)**

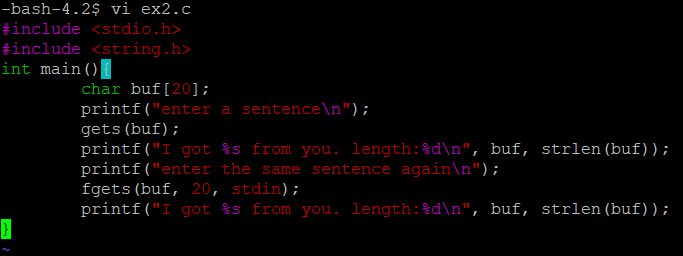
**[-M intel] : intel 문법으로 disassemble한다.**

**= [objdump -D -M intel ex1]의 결과를 f3 파일에 저장 후 vi 편집기로 실행한다.**

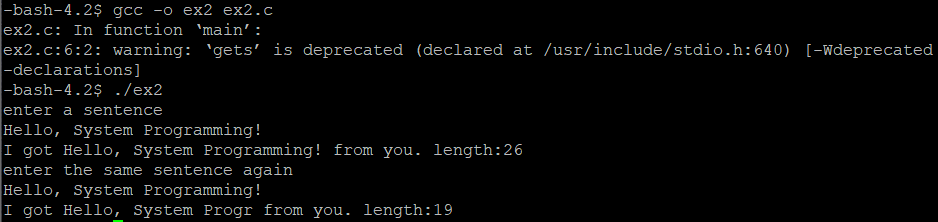


**= [/<main>]을 이용해 <main>의 변환 결과를 확인할 수 있다.**

19) Run following and tell the difference between gets and fgets



**= vi 편집기를 이용해 해당 C프로그램을 작성하였다.**

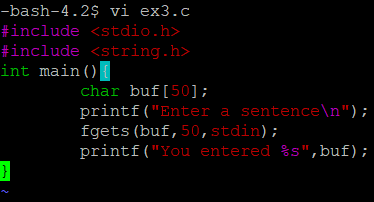


**= [gcc]로 컴파일하였더니 main함수 안의 ‘gets’에 대해 사라질 명령어를 사용하고 있다는 경고메시지가 나타났다.**

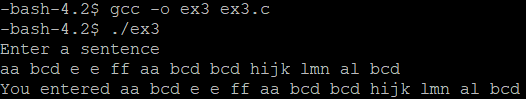
**= 경고메시지를 무시하고 [./ex2]로 만들어진 실행파일을 실행하였다.**

**= 실행 결과 [gets]는 선언한 char형의 20개가 넘은 문자도 입력받아 모두 출력할 수 있었다. 반면에 [fgets]는 19개까지 밖에 출력하지 못하였다. fgets는 인자에 20이라는 문자열 길이도 같이 넘겨주기에 이런 결과가 나타나는 것으로 보인다.**

20) Write a program to read a sentence and echo it as follows. Use gets() or fgets(). Do "man gets" or "man fgets" to find out the usage of them.



**= [fgets]를 이용해 작성하였다. buf라는 문자열은 50개를 할당하였다.**



**= [gcc]로 컴파일 후 실행하였다. 50개로 넉넉히 할당하였더니 끊기지 않고 모두 출력되었다.**