System Programming
(Assignment1)

과 목	시스템프로그래밍실습
담당교수	이기훈 교수님
학 과	컴퓨터공학과
학 번	2010720149
성 명	이동현
날 짜	2016. 03. 24 (목)

광운대학교
KwangWoon University

A. 제목및 목적

i. 제목

VMWare에 Ubuntu 12.04 LTS 설치

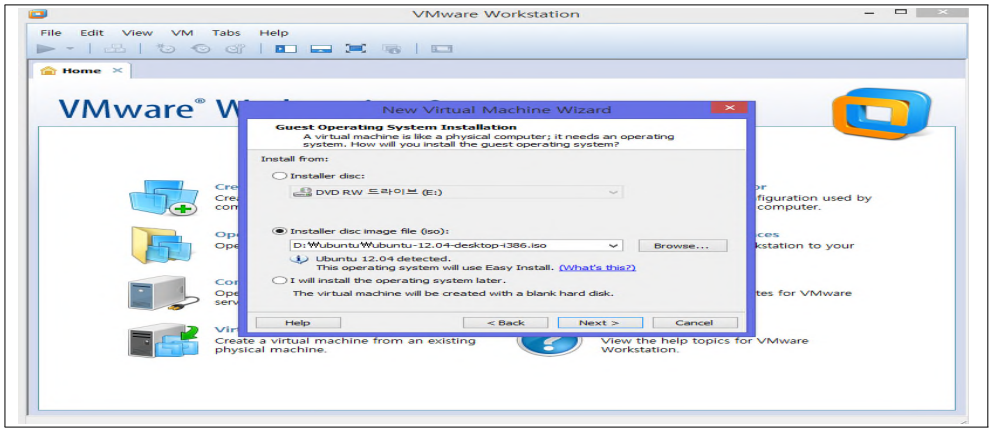
ii. 목적

시스템프로그래밍실습을 진행하기 전에 실습에 필요한 VMWare와 Ubuntu 12.04 LTS를 설치하여 Window 기반에서 Linux를 사용하기 위한 목적이 있다.

B. 결과

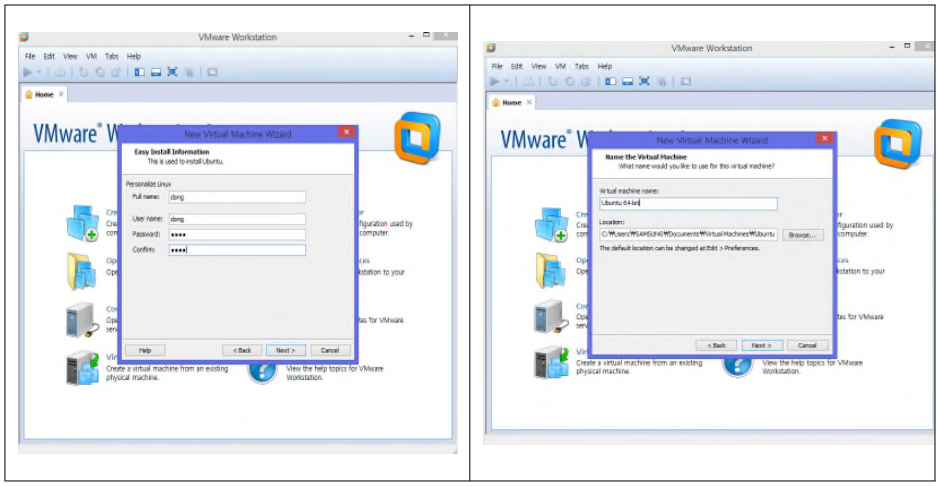
i. VMWare를 이용한 Ubuntu 설치

1)



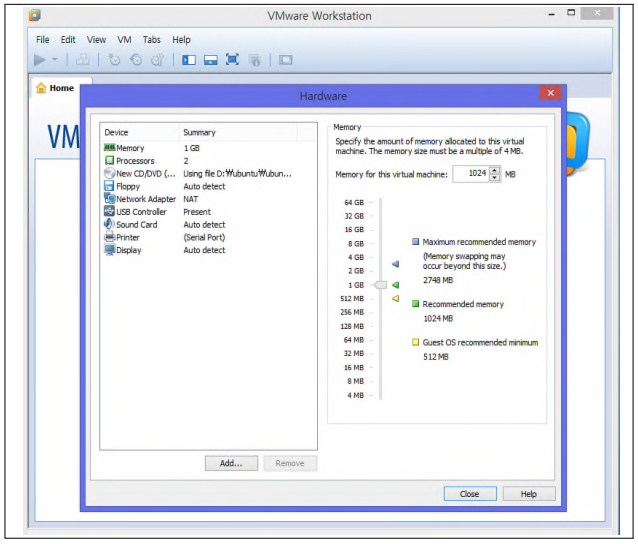
- VMWare에서 가상머신 생성후 Ubuntu ISO 파일 위치 지정.

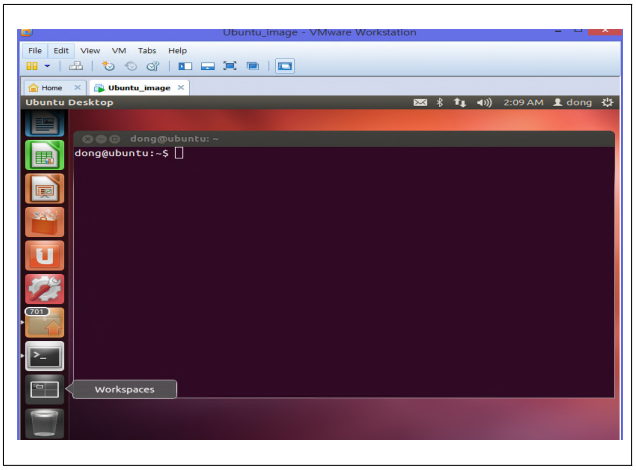
2)



- Ubuntu에서 로그인 할 계정과 비밀번호를 지정하고 가상머신의 이름과 설치가 될 위치를 지정.

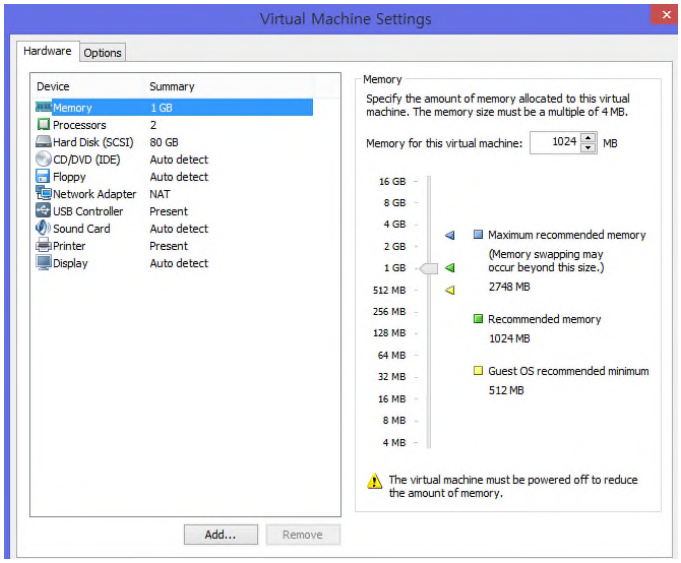
3)





- 가상머신에 할당한 저장장치 크기와 메인 메모리 크기, cpu 및 cpu core 수 설정.
- Ubuntu 설치후 Terminal Lock으로 고정.

ii. 가상 머신의 디스크 크기



B. 고찰 및 결과

- 이번 과제는 실습에서 사용할 Ubuntu를 설치하는 과정을 직접 실시해보았다. 졸업 작품에서 Ubuntu를 자주 사용하는데, 이용 중에 많은 에러와 오류가 발생하기 때문에 자주 재설치를 했었다. 예를 들어 메모리 사용량을 적게 잡고 설치하여 재 설치하는 경우 등 필요에 따라 재설치가 필요하기 때문에 설치 요령을 익힐 필요가 있다고 생각을 했다. 이번 시스템프로그래밍 실습을 진행을 시작 전에 설치를 통해 앞으로 과제를 수행할 기대감이 생겼다.

c. Reference

- 강의자료 ' 2016-1_SPLab_01_Introduction '

A. 제목및 목적

i. 제목

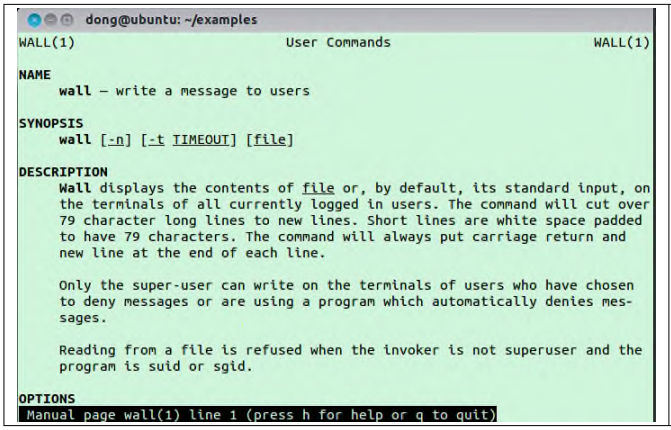
2주차 Linux 기본 명령어를 사용.

ii. 목적

Linux상의 shell 명령어와 그 것의 옵션을 함께 사용함으로써 명령어의 기능과 그 이후 결과에 대해 인지하고 앞으로 명령어 사용에 이를 숙달하기 위한 목표가 있다.

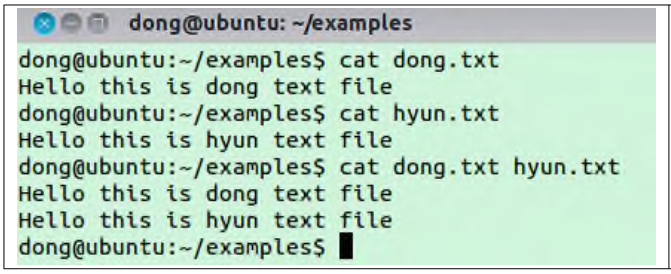
B. 결과

man)



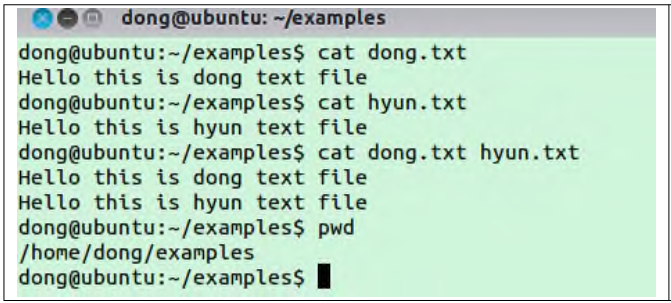
man wall 명령어 결과 wall 명령어에 대한 manual page를 확인 할 수 있다.

cat)



첫 번째 #cat 명령어는 하나의 파일에 대해서 명령어를 수행 했을 때 파일 내의 내용을 출력한다. 두 번째는 두 파일로 명령어를 수행을 했을 때 두 파일을 연결시켜 출력한다.

pwd)

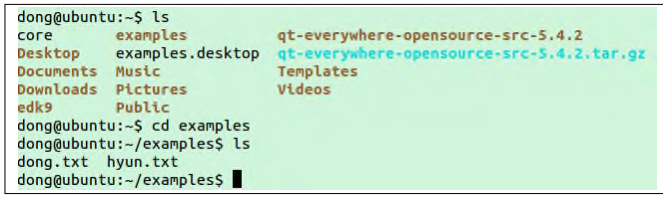


pwd 명령어는 내가 현재 작동 하고 있는 디렉토리를 보여준다. home/dong/examples 인 상태임을 알 수 있다.

cd)

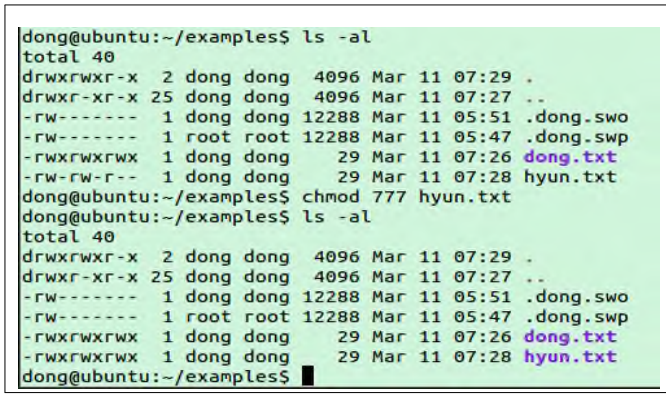
cd 명령어는 현재 디렉토리를 변경하는 명령어이다. 캡쳐 화면을 통해 examples 디렉토리로 이동함을 확인 할 수 있다.

ls)



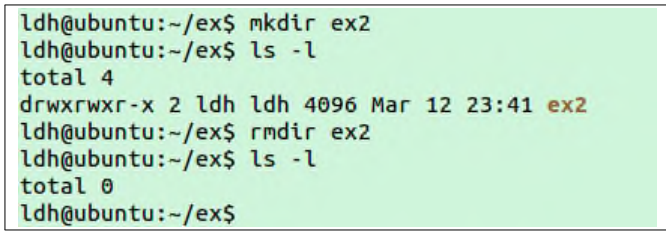
ls 명령어는 현재 디렉토리 내용물들을 리스트화하여 보여주는 명령어이다. 현재 리스트를 출력하여 확인 할 수 있다.

chmod)



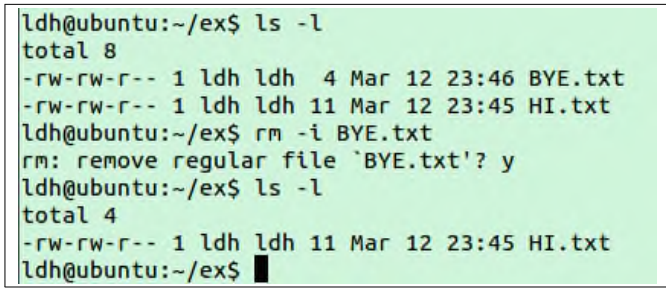
chmod 파일 접근 허가를 변경하는 명령어로 옆 캡쳐에서 숫자로 이용하여 hyun.txt의 권한이 rwxrwxrwx로 변경되었음 확인 할 수 있다.

mkdir, rmdir)



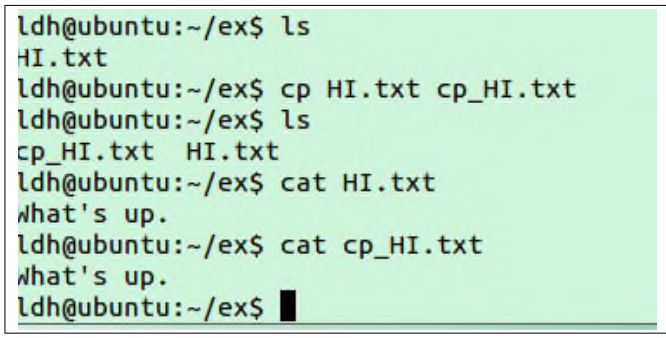
mkdir # rmdir 명령어는 디렉토리를 생성 및 삭제시키는 명령어로, 옆 캡쳐를 통해서 생성 후 지운 상태를 확인 할 수 있다.

rm)



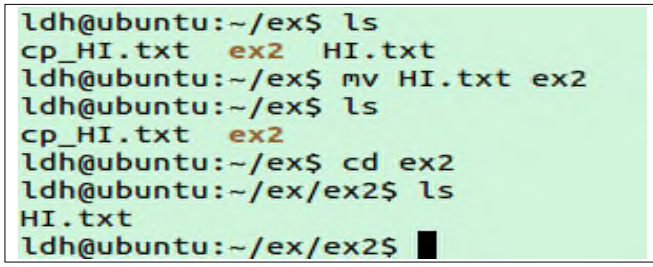
rm 명령어는 file을 삭제하는 명령어로, 옆 캡쳐에서는 -i 옵션을 통해 삭제 확인 여부를 되묻는 방법으로 삭제를 하였다.

cp)



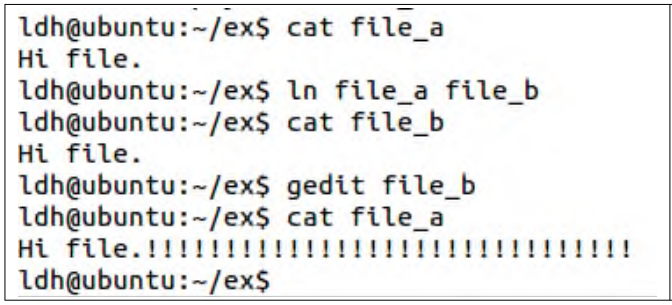
cp 명령어는 디렉토리 그리고 파일을 복사하는 명령어이다. 옆 캡쳐에서는 파일만 복사하는 것을 보여주었다.

mv)



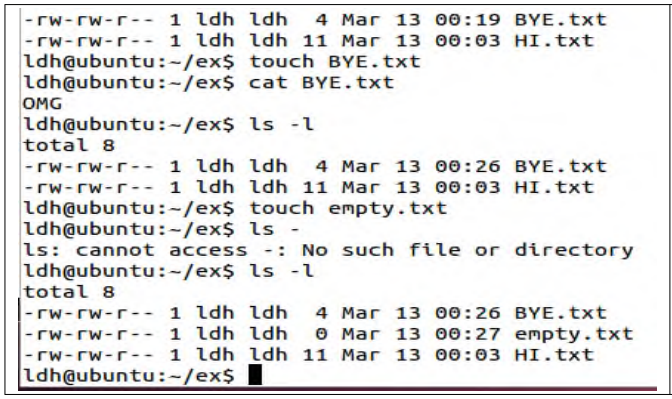
mv 명령어는 파일을 옮기거나 디렉토리의 이름을 재설정 할 수 있는 명령어이다.

ln)



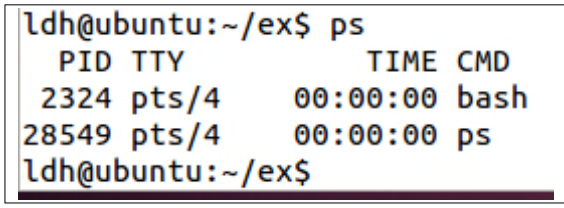
.ln 명령어는 파일 간의 Hard Link를 맺는 명령어로 Window에서 바로가기와 유사하다. 서로간의 링크로 하나의 파일로 두 파일이 연결되어 있음을 확인 할 수 있다.
그 이외에 Symbolic Link가 존재하는데, 이것은 파일간의 링크가 깨질 경우 해당 파일이 존재하지 않음을 알 수 있다.

touch)



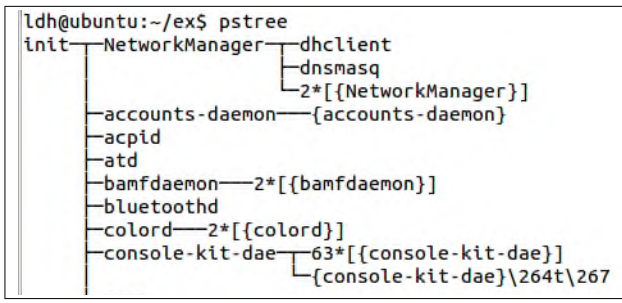
touch 명령어는 파일의 stamps를 변경하거나 빈 파일을 만드는 명령어이다. 옆 캡쳐와 같이 stamps변경으로 인한 시간 변경을 확인할 수 있고, 빈 파일이 생성되었음을 볼 수 있다.

ps)



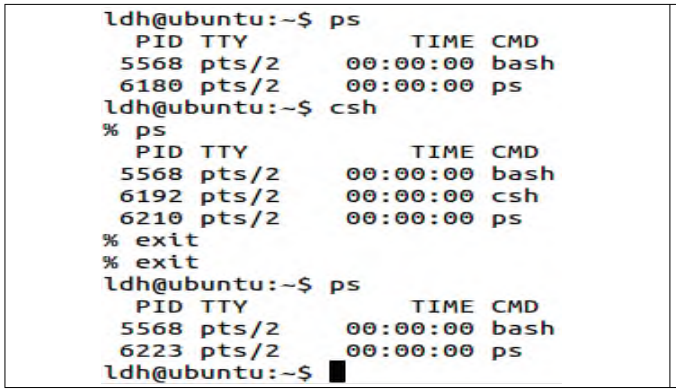
ps 명령어는 현재 프로세서 상태를 보여 준다. 여러 가지 옵션에 따라 확인 할 수 있지만, 캡쳐에 용이하기 위해 옵션을 없는 것만 첨부하였다.

pstree)



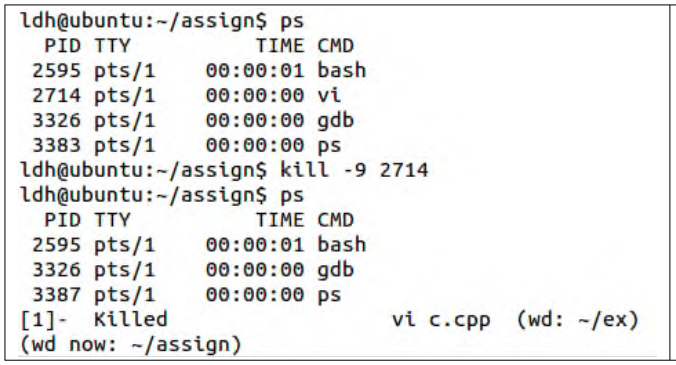
pstree 명령어는 프로세서의 tree를 보여주는 명령어로 이것 또한 일부분만 캡쳐를 했다.

exit)



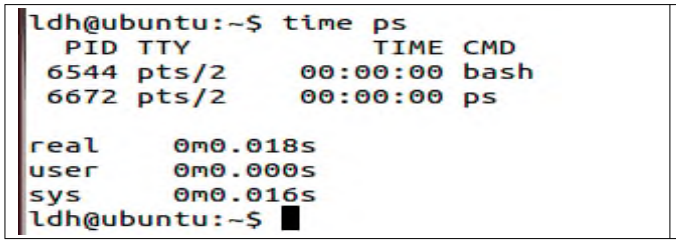
exit는 해당 shell을 종료하는 명령어로, csh shell일 경우 종료시키는 것음 캡쳐를 통해 확인 할 수 있다.

kill)



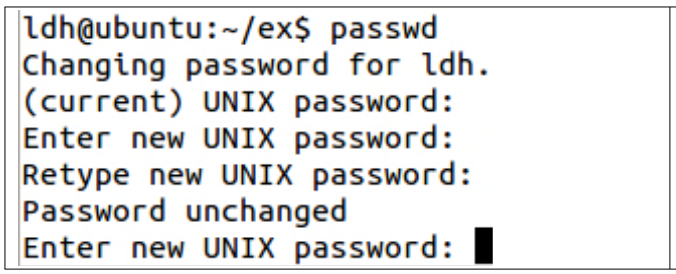
kill 명령어는 process에게 signal을 보내는 명령어이다. 옵션 -9는 해당 프로세스 ID를 중지시키는 명령어로, vi를 수행했던 프로세스를 중지시키는 것을 옆 캡쳐 화면을 통해 보여주었다.

time)



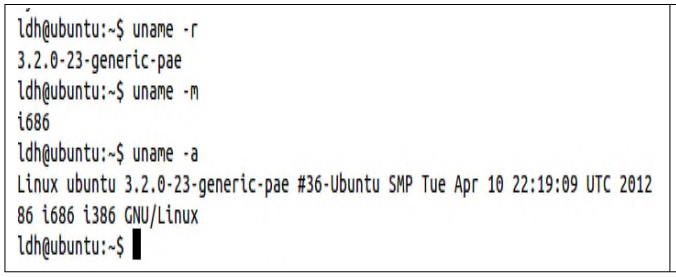
time 명령어는 해당 명령어에 대한 시간을 확인하는 명령어로 캡쳐에서는 ps에 관한 실제 소요시간, user영역에서 소비된 시간, 커널에서 소비된 시간을 확인 할 수 있다.

passwd)



passwd 명령어는 사용자의 비밀번호를 변경하는 명령어로, 옆 캡쳐를 통해 비밀번호 변경을 함을 확인 할 수 있다.

uname)



uname 명령어는 시스템 정보를 보여주는 명령어로 옵션 -r 과 -m 그리고 -a 따라 시스템 정보를 확인 할 수 있다.

wc)

ldh@ubuntu:~/ex$ ls BYE.txt empty.txt HI.txt ldh@ubuntu:~/ex$ wc HI.txt 1 2 11 HI.txt ldh@ubuntu:~/ex$	# wc 명령어는 해당 파일의 line, word byte의 수를 보여주는 명령어이다.

more)

Hi file.!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Hi file.!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Hi file.!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Hi file.!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Hi file.!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Hi file.!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Hi file.!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Hi file.!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Hi file.!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Hi file.!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Hi file.!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Hi file.!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Hi file.!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Hi file.!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Hi file.!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Hi file.!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Hi file.!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! --More--(48%)	# more 명령어는 파일 안의 내용을 보여주는 명령어이지만 특별한 기능을 갖추고 있다. 파일안의 내용을 한꺼번에 볼 수 있는 기능과 /TERM 입력을 통해 검색기능도 가능하다.

echo)

ldh@ubuntu:~/ex$ echo LeeDongHYun LeeDongHYun	# echo 명령어는 한줄 text를 그대로 보여주는 명령어이다.

alias)

ldh@ubuntu:~/ex$ alias myls='ls -al' ldh@ubuntu:~/ex$ myls total 16 drwxrwxr-x 2 ldh ldh 4096 Mar 13 00:27 . drwxr-xr-x 22 ldh ldh 4096 Mar 13 00:45 .. -rw-rw-r-- 1 ldh ldh 4 Mar 13 00:26 BYE.txt -rw-rw-r-- 1 ldh ldh 0 Mar 13 00:27 empty.txt -rw-rw-r-- 1 ldh ldh 11 Mar 13 00:03 HI.txt ldh@ubuntu:~/ex$	# alias 명령어는 별칭을 짓는 명령어로 예를 들면 ls −al 을 myls로 대체하여 사용할 수 있게 이름을 짓는 명령어이다.

find)

ldh@ubuntu:~/ex$ find -name '*.txt' ./BYE.txt ./empty.txt ./HI.txt ldh@ubuntu:~/ex$	# find 명령어는 계층의 디렉토리에서 파일을 찾는 명령어이다.

grep)

ldh@ubuntu:~/ex$ cat BYE.txt OMG!!!!!!!!!!! Hellow Linux! hahaha kikiki ldh@ubuntu:~/ex$ grep Linux BYE.txt Hellow Linux! ldh@ubuntu:~/ex$	# grep 명령어는 파일 내용의 일부의 단어를 포함하는 단어를 잡아주는 명령어로, 검색기능이라고 볼 수 있다.

B. 고찰 및 결과

- Linux shell의 많은 명령어를 직접 사용함으로써 기본적인 프로그램을 수행할 수 있는 능력을 배양하는데 큰 도움이 생각되는 과제라 생각이 들었다. 가상 프로그램으로 보는 Linux를 사용자의 인터페이스가 직접 클릭이 아닌 글로 인한 명령어가 새롭기도 하였지만, 그만큼 익숙지 않는 어려움도 있었다. 명령어를 직접 한번 씩 써봄으로써 각각의 명령어의 수행 기능이나 결과를 확인 할 수 있지만, 각 명령어에 대한 옵션이 많아 하나하나 외우고 입력해보기는 무리가 있었다. 명령어 + --help를 입력하면 기본적인 옵션을 알 수 있도록 도움을 주기 때문에 참고하기 좋은 방법이라고 생각이 들었다.
그 중에서 고민 했던 명령어가 ln 명령어와 kill 명령어였다. ln명령어는 파일들 사이에 Hard link를 만드는 명령어로 다른 파일을 통해 동일한 파일을 접근할 수 있는 링크라고 이해하면 쉽다. ln의 link는 한쪽으로만 종속된 링크가 아닌 둘 중 하나의 파일 내용을 수정할 경우 순서 상관없이 다 지워짐을 확인 할 수 있다. 이를 생각하면 cp의 명령어와는 당연히 차이점이 다름을 생각할 수 있을 것이다.
그와 반대로 심볼릭 링크는 cp와 완전 동일한 수행결과를 보여주지는 않지는 비슷한 결과를 나타낸다. 즉, 서로 심볼릭 링크가 되어있는 두 파일 중 하나를 지울 경우 심볼릭 링크가 되어있는 파일은 존재하지 않음을 확인 할 수 있다. 직접 명령어 예시를 수행함으로써 이해하기가 편했다.
kill 명령어는 단순히 프로세스를 죽이는 명령어로 알았지만, 치명적인 실수를 하였다. kill명령어는 해당 프로세스아이디에 신호를 보내는 명령어이기 때문에 -9라는 옵션을 통해 해당 프로세스를 멈추는 일부의 신호를 갖는 명령어임을 확인하였다.
나머지 명령어는 강의자료를 참고하여 쉽게 해결할 수 있었다. 앞으로 자주 사용 할 명령어임을 인지하고 공부를 더 해나가야겠다고 생각이 들었다.

c. Reference

- 강의 자료 2016-1_SPLab_03_Unix+Linux+Commands.pdf

A. 제목및 목적

i. 제목

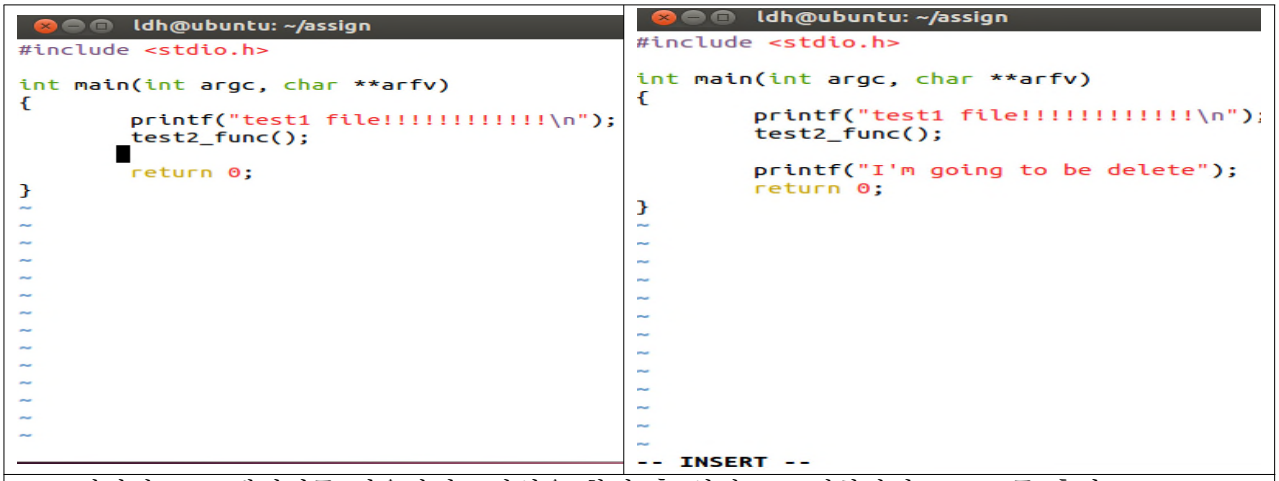
3주차 vi, make, gdb 명령어 사용

ii. 목적

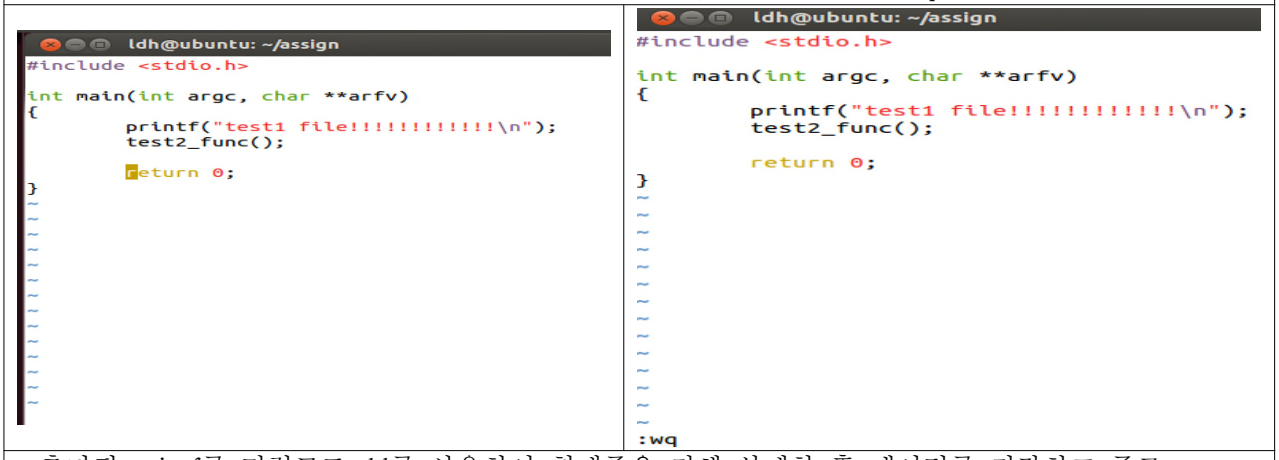
리눅스 기본 편집기인 vi 에디터 사용하여 컴파일 과정을 시행하는 make 명령어 사용하기 위해 Executable file을 생성 해본다. 디버깅 과정을 하는 gdb 명령어를 수행함으로써 수행할 프로그램을 직접 확인 해본다.

B. 결과

1) vi (vi editor)



- vi명령어로 vi 에디터를 이용하여 C파일을 확인 후 입력모드 전환하여 printf 를 추가.



-추가된 printf를 명령모드 dd를 사용하여 현재줄을 전체 삭제한 후 데이터를 저장하고 종료.

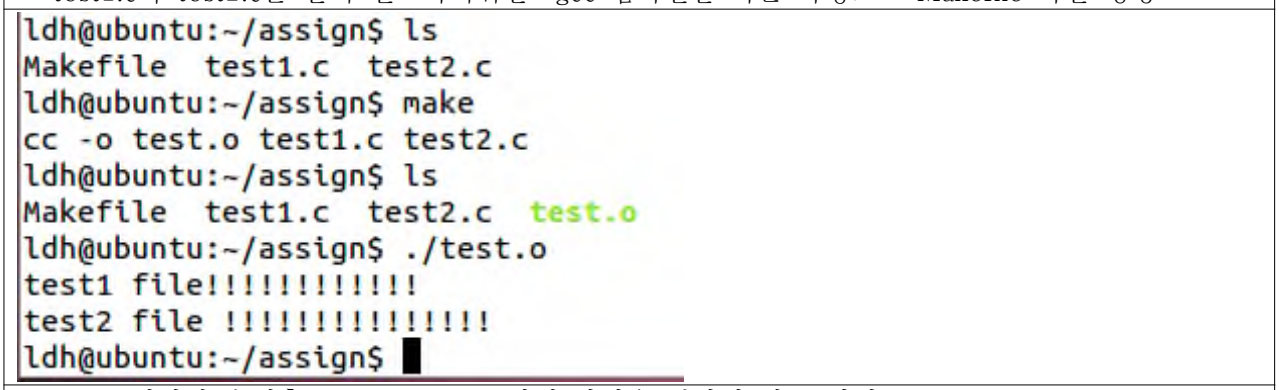
```
ldh@ubuntu:~/assign$ cat test1.c
#include <stdio.h>

int main(int argc, char **arfv)
{
        printf("test1 file!!!!!!!!!!!!!\n");
        test2_func();

        return 0;
}
ldh@ubuntu:~/assign$
```

-저장된 test1.c 파일을 cat 명령어를 통해 확인

-명령모드에서 패턴 int를 검색을 하였고, int를 char형으로 패턴에 의한 치환을 하였다.

2) make

- test1.c와 test2.c를 같이 빌드하기위한 gcc 컴파일을 하는 과정으로 Makefile 파일 생성



- make 명령어 수행후 Execution file 실행 파일을 컴파일 하는 과정.

3) gdb

```
ldh@ubuntu:~/assign1-1$ ls
a.out  example.c
ldh@ubuntu:~/assign1-1$ ./a.out Hi! My name is donghyun!
The number of inputted variable is 6
Segmentation fault (core dumped)
ldh@ubuntu:~/assign1-1$ gcc -g example.c
ldh@ubuntu:~/assign1-1$ gdb a.out
GNU gdb (Ubuntu/Linaro 7.4-2012.02-0ubuntu2) 7.4-2012.02
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/ldh/assign1-1/a.out...done.
(gdb) b 9
Breakpoint 1 at 0x8048455: file example.c, line 9.
```

- example.c 파일을 gcc 컴파일을 통해 실행파일 a.out을 만들고, 임의의 글자 인자를 가지고 프로그램을 실행한다. Segmentation fault를 통해 프로그램 에러가 떠있음을 확인 할 수 있다. 에러의 위치를 찾기위해서 gdb 디버깅을 수행해보자.

디버깅을 하기위해서 gdb 컴파일 옵션과 gdb 실행 시킨다. gdb 프롬프트에 Break point line 9번째로 설정하고 Breakpoint가 line 9번째 선택되었음을 확인할 수 있다.

```
(gdb) r Hi! My name is donghyun.
Starting program: /home/ldh/assign1-1/a.out Hi! My name is donghyun.

Breakpoint 1, main (argc=6, argv=0xbffff3e4) at example.c:9
9               printf("The number of inputted variable is %d\n", argc);
(gdb) n
The number of inputted variable is 6
10              printf("and they are");
(gdb) s
12              for(i=0 ; i<argc ; i++)
(gdb) step
14                      strcpy(ptemp, argv[i]);
(gdb) print 1
$1 = 1
(gdb) print argv[0]
$2 = 0xbffff560 "/home/ldh/assign1-1/a.out"
(gdb) next
```

-프롬프트에 매개변수를 사용하기 위해 r 명령과 임의의 인자를 같이 쓴다. 프로그램이 running이 되고, Break point로 인해 9번째 line까지 디버깅이 되었음을 확인할 수 있다. 프롬프트에 n(next), s(step)은 다음 line으로 넘어가는 것으로 한 줄 한 줄 확인한다. 해당 라인의 변수 값을 확인하기위해 print를 사용하여 각각 해당 변수의 값을 확인 할 수 있다.

```
Program received signal SIGSEGV, Segmentation fault.
0xb7eaf0d9 in ?? () from /lib/i386-linux-gnu/libc.so.6
(gdb) bt
#0  0xb7eaf0d9 in ?? () from /lib/i386-linux-gnu/libc.so.6
#1  0x0804849c in main (argc=6, argv=0xbffff3e4) at example.c:14
(gdb)
```

- line을 넘기다보면 해당 line에서 Segmentation fault의 에러가 발생함을 확인 할 수 있다. 대충 line을 세면서 보면 짐작을 할 수 있지만, 정확한 위치를 확인하기 위하여 bt를 통한 스택 전체를 확인 할 수 있다. ' at example.c:14 '에서 14가 14번째 line에서 오류가 생겼음을 확인 할 수 있고,

해당 코드에서 포인터 변수에 변수 값을 입력하려는 오류임을 파악을 할 수 있었다.

B. 고찰 및 결과

- 이번 과제는 vi, make, gdb를 수행을 함으로써 해당 c파일을 수행할 수 있는 능력을 배양할 수 있는 과제였다. vi는 파일 내의 내용을 수정하는 에디터이다. 다소 수정내용이 명령모드와 입력모드가 따로 있어 에디터에 익숙하기 어려운 점은 있었다. 그래서 기본적인으로 다루기 위해서는 약간의 명령모드의 암기가 필요했다. 데이터를 저장하고 데이터를 삭제하고 입력모드와 명령모드를 다룰 수 있는 명령어를 직접 사용함으로써 많이 익숙해졌다. 그리고 앞으로 많이 사용할 것 같은 명령어인 make는 컴파일 과정을 자동화하는 것으로 gcc 명령을 통해 직접 입력하는 시간보다 많은 시간을 절약할 수 있는 Makefile을 생성하는 법을 알아보았다. c file이 어느 과정을 통하여 execution file을 만드는 과정을 익숙하지 않으면, 단편적으로 이해하기엔 쉽지 않을 수 있다. 예전 교수님이 강조했었던 이 과정을 한번 더 되새김을 통해 완벽히 이해할 필요가 있다. C file은 컴파일러를 통해 어셈블리 파일이 생성이 되고 어셈블리는 이 파일을 읽어 Object file을 생성한다. 그리고 Linker가 Executable file을 생성하여 해당 파일에 대한 프로그램을 수행할 수 있다. 그리고 두 개 파일을 빌드 할 경우, 두 파일이 독립적으로 수행을 할 수 없을 경우 변수를 사용하여 대체하여 Makefile을 작성하지 않으면 컴파일 자체가 실행되지 않음을 주의를 해야 한다. 이어 에러를 직접 사용자가 확인하기위한 디버깅을 수행하는 gdb는 직접 사용자의 명령을 통해 에러의 위치를 파악해야하기 때문에 기본적인 gdb 프롬프트에 입력할 명령은 알아두어야 할 필요가 있다고 생각한다. 보이는 강의자료 대로 수행을 할 경우 큰 어려움은 없었던 디버깅 명령어였다. 앞으로 c파일을 자주 접할 것 같아서 이 명령어에 대해서 더 익숙해질 필요가 있다고 생각이 들었다.

c. Reference

- 강의자료 ' 2016-1_SPLab_03_Linux-based_programming.pdf