1차 과제 - 리눅스 기초

학　　과 : 컴퓨터 공학과
담당교수 : 황호영
분　　반 : 목34
학　　번 : 2016722092
성　　명 : 정동호

목차

Ubuntu Installation

Introduction

리눅스 기초를 다지기 위해 Ubuntu 설치부터 시작한다. 그리고 설치 과정을 캡쳐하고 설명한다.

Result

기존에 이미 Ubuntu 16.04.03 가상 호스트가 있었지만 과제 수행을 위해 다시 설치하였다.
하드웨어 설정은 다음과 같다:

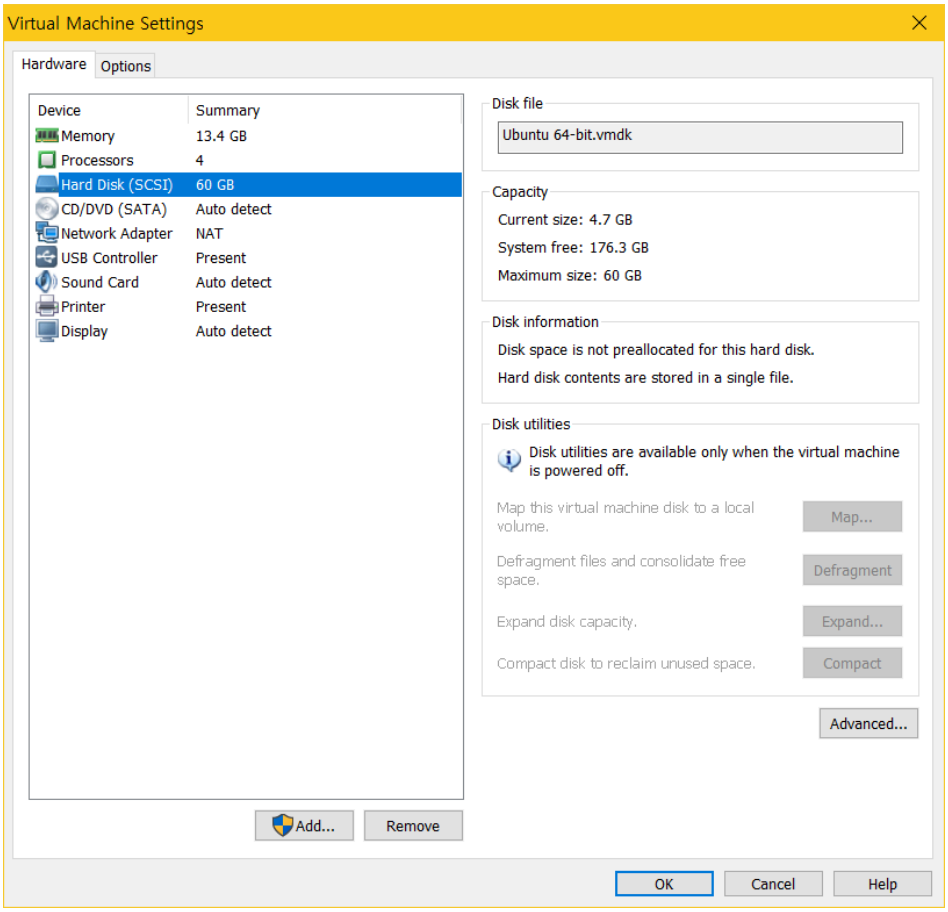

그림 1-1. 하드웨어 설정

다음은 설치 과정 중 언어 팩을 다운로드 하는 장면이다:

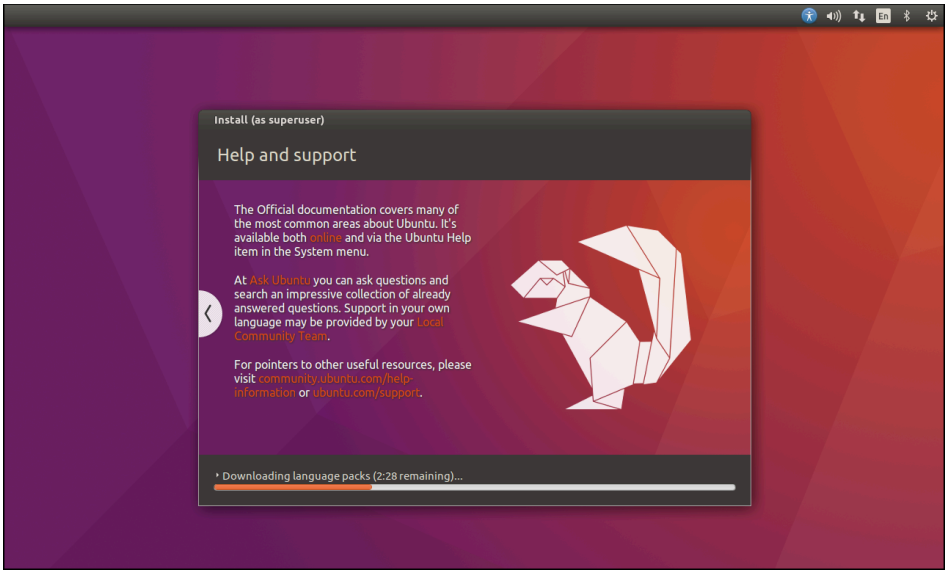

그림 1-2. 언어 팩 다운로드

다음과 같이 깔끔한 초기 화면을 볼 수 있었다:

그림 1-3. 초기 화면

Reference

https://www.ubuntu.com/download/desktop — Ubuntu 16.04.4 LTS (Xenial Xerus) 이미지 제공

Usage of Linux Commands

Introduction

우분투를 설치하고 난 후 리눅스의 기본적인 명령어들은 어떤 것들이 있는 지 알아보고 직접 터미널에서 따라 쳐 보면서 다양한 명령어와 옵션들을 실습한다. 실습에서 사용할 명령어들은 다음과 같다:

- 2주차 : `man`, `cat`, `pwd`, `cd`, `ls`, `chmod`, `mkdir`, `rmdir`, `rm`, `cp`, `mv`, `ln`, `touch`, `exit`, `kill`, `ps`, `pstree`, `time`, `passwd`, `uname`, `wc`, `more`, `echo`, `alias`, `find`, `grep`
- 3주차 : `vi`, `make`, `gdb`

Result

2주차

![man wall](thu_2016722092_정동호.assets/man wall.png)

1. `man`

그림 2-1-1. man wall

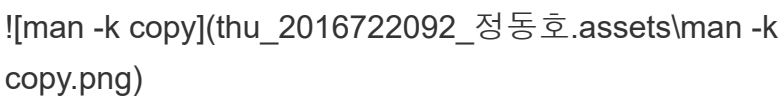

그림 2-1-2. man -k copy

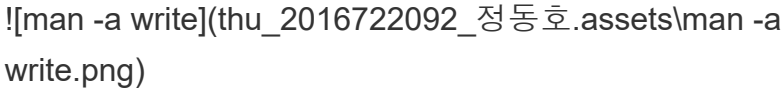

그림 2-1-3. man -a write

manual, 먼저 *man [OPTION...][SECTION] PAGE...* 에서 옵션 없이 `wall`을 인자로 주었을 때는 `wall`에 대한 설명이 나타나는 것을 볼 수 있었다.
그리고 -k 옵션을 주고서 `copy`를 인자로 주었을때는 `copy`가 포함된 모든 페이지들의 리스트가 반환되었다.
마지막으로 `man`에는 여러 섹션이 있고, 동일한 이름의 페이지가 존재할 수 있는데 _[SECTION]_을 명시하지 않고 a 옵션을 준다면 그림 2-1-3과 같이 한 페이지의 조회가 끝나면 다른 페이지도 조회할 것인지 선택할 수 있다는 것을 볼 수 있었다.
`man`의 섹션은 일반적으로 다음과 같이 8가지로 나뉜다:

섹션	설명
1	일반적인 명령어
2	시스템 콜
3	라이브러리 함수—특히 C 표준 라이브러리의
4	특수 파일(보통 /dev에서 볼 수 있는 장치)과 드라이버
5	파일포맷 및 규약
6	게임 및 화면보호기
7	잡다한 것들

섹션	설명
8	시스템 관리 명령어 및 대몬

모든 `man` page들은 일관성과 간결함을 위해 공통된 양식을 따르고 있다. 이 양식을 통해 페이지들은 가능한한 하이라이팅이나 폰트제어 없이 ASCII만으로 출력하는데 최적화 되고 있다. 양식은 다음과 같다:

- **NAME** : 명령어 또는 함수의 이름, 뒤이어 무엇을 하는지 한줄 짜리 설명이 나온다.
- **SYNOPSIS** : 명령어의 경우 어떻게 실행시키는지에 대한 설명을 출력하고, 프로그램 함수의 경우 함수가 취하는 인자들과 해당 함수를 정의한 헤더 파일들에 대한 목록을 나열한다.
- **DESCRIPTION** : 명령어 또는 함수의 기능에 대한 설명
- **EXAMPLES** : 일반적인 용례
- **SEE ALSO** : 관련된 명령어나 함수

물론 위의 절뿐만 아니라 OPTIONS, EXIT STATUS, ENVIRONMENT, ... 등도 포함할 수 있다.

2. `cat`

    ![cat](thu_2016722092_정동호.assets\cat.png)

    그림 2-2. cat

    con**cat**enate, 사용법은 *cat [OPTION] [FILE]...* 이다. 파일들을 잇고(concatenate) 표준 출력으로 내보낸다. 즉 파일의 내용을 출력하는 명령어이며 리다이렉션 `>`을 사용해 결과를 파일이나 다른 곳으로 내보낼 수 있다.

3. `pwd`

    ![pwd](thu_2016722092_정동호.assets\pwd.png)

    그림 2-3. pwd

    Print **w**orking **d**irectory, 현재 작업 디렉터리의 이름(경로)을 출력한다.

4. `cd`

    ![cd](thu_2016722092_정동호.assets\cd.png)

    그림 2-4. cd

    Change the current **d**irectory, 즉 현재 작업 디렉터리를 변경한다. 이때 인자로 ~를 주면 아무 인자 없는 `cd`와 같은 동작을 하며 현재 유저의 home 디렉터리로 이동한다. -를 주면 환경변수 $OLDPWD에 저장된 이전 경로로 이동한다.

5. `ls`

    ![ls](thu_2016722092_정동호.assets\ls.png)

그림 2-5. ls

디렉터리의 내용들을 나열한다. a 옵션을 주면 숨김 파일도 출력해주고 F 옵션을 주면 파일 종류를 표시해준다. 마지막으로 l옵션을 주면 권한, 소유자, 그룹, 이름 등등의 더 자세한 정보를 표시해준다.

6. `chmod`

    ![chmod](thu_2016722092_정동호.assets/chmod.png)

    그림 2-6. chmod

    Change file **mod** bits, 사용 문법은 *chmod [options] mode[,mode] file1 [file2 ...]* 이다. 권한을 지정해줄때 그림 2-6의 처음과 같이 *Symbolic* 모드를 사용하거나 마지막처럼 *Octal* 모드를 사용할 수 있다.

    파일과 디렉터리에 적용되는 권한의 효과는 다음과 같다:

    - **read**(4)
        - **파일** : 파일의 내용을 읽을 수 있다.
        - **디렉터리** : 디렉터리의 내용(안에 존재하는 파일 또는 하위 디렉터리)을 나열 할 수 있다.
    - **write**(2)
        - **파일** : 파일의 내용을 바꿀 수 있다.
        - **디렉터리** : 디렉터리내 임의의 파일을 만들거나 지울 수 있다.
    - **exec**(1)
        - **파일** : 명령어로써 파일을 실행시킬 수 있다.
        - **디렉터리** : 디렉터리의 내용에 접근할 수 있다. 즉 `cd`로 접근이 가능하다.

7. `mkdir`

    ![mkdir](thu_2016722092_정동호.assets/mkdir.png)

    그림 2-7. mkdir

    make **directories, 디렉터리를 생성한다. directories** 라서 인자를 여러개 주면 한번에 여러 폴더를 만들 수 있다. 또는 p옵션을 주면 한번에 subdirectory까지 생성이 가능하다.

8. `rmdir`

    ![rmdir](thu_2016722092_정동호.assets/rmdir.png)

    그림 2-8. rmdir

    remove empty **dir**ectories, 디렉터리를 삭제한다. 단, **디렉터리가 비어있어야 한다**. 그렇지 않은 경우에는 `rm`에 [rR] 옵션을 주면 된다.

9. `rm`

그림 2-9. rm

remove, 파일 또는 디렉터리들을 삭제한다. [rR] 옵션을 주면 비어있지 않은 디렉터리도 하위 폴더를 DFS로 탐색하며 삭제한다. i 옵션을 줄 경우 지울때마다 확인을 거쳐 좀 더 안전한 작업을 할 수 있다.

10. `mv`

그림 2-10. mv

move, 사용 문법은 *mv [OPTION]... SRC DST* 로 원본이 먼저 나오고 목적지가 마지막에 나온다. 기본적으로 파일을 이동시키지만 이름을 바꾸는 데에도 사용할 수 있다.

11. `ln`

그림 2-11-1. ln

그림 2-11-2. ln vs cp

그림 2-11-3. ln symbolic link

**link, `ln`은 *ln [OPTION]... TARGET [LINK_NAME]* 로 사용한다. 파일들 간에 link 를 만들어주는 역할을 하며 그림 2-11-1,2 를 통해 `cp`와의 차이점을 알 수 있다. link에는 symbolic—또는 soft— 과 hard 두 종류가 있다. 둘의 차이는 다음 그림과 같다:

그림 2-11-4. soft vs hard

이외에 hardlink는 같은 파일 시스템에서만 사용하다는 차이도 있다.

12. `touch`

그림 2-12. touch

빈 파일을 만들거나 파일의 [ma]time을 현재시간으로 변경한다.

13. `ps`

그림 2-13. ps

현재 프로세스들의 상태를 보여준다. 강의자료에서 나온 두가지 예를 실행시켜 보았는데 아무 옵션을 주지 않을 경우 t 옵션을 준것과 동일하며 현재 터미널에서 실행

자의 EUID를 가진 프로세스들을 나열한다. [eA] 옵션은 모든 프로세스를 출력하고 f 옵션은 더 자세한 정보를 출력해준다.
필드별 의미는 다음과 같다:

필드명	의미
UID	프로세스 소유자의 유저네임
PID	프로세스 ID
PPID	부모 프로세스 ID
C	CPU 사용량 및 스케줄링 정보
STIME	프로세스가 시작한 시간
TTY	프로세스가 관련된 터미널
TIME	전체 CPU 사용량
CMD	인자를 포함한 프로세스의 이름

14. `pstree`

    ![pstree](thu_2016722092_정동호.assets/pstree.png)

    그림 2-14. pstree

    프로세스들을 트리형태로 출력한다. 전체 프로세스 구조를 파악하는데는 시각적으로 `ps`보다 좋아보인다.
15. `exit`

    ![exit](thu_2016722092_정동호.assets/exit.png)

    그림 2-15. exit

    현재 쉘에서 나온다. 그림 2-15에서 처럼 csh 쉘에서 나와 bash로 돌아온 것을 확인할 수 있다.
16. `kill`

    ![SIGTERM](thu_2016722092_정동호.assets/SIGTERM.png)

    그림 2-16-1. SIGTERM

    `kill`은 *kill [-s sigspec | -n signum | -sigspec] pid | jobspec ...* 또는 *kill -l [sigspec]*로 사용한다. sigspec을 명시하지 않을 경우 기본으로 15) SIGTERM이 사용된다. 이 경우 `ps`에선 Terminated로 표시되며 다음과 같이 SIGTERM으로 종료되지 않을

경우 9) SIGKILL을 사용한다:

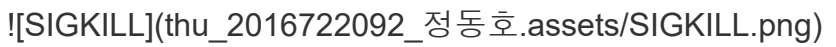

그림 2-16-2. SIGKILL

17. `time`

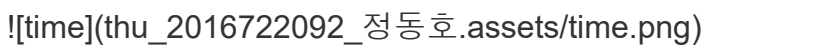

그림 2-17. time

인자로 주어진 프로그램을 실행하고 시스템 자원 사용량을 요약해서 보여주는 명령어이다. 요약 정보에는 real, user, sys가 있는데 각 의미는 다음과 같다.

- **real** : 실행 시작부터 종료까지의 시간. I/O 지연 및 스케줄링에 인한 대기등도 포함.
- **user** : 유저모드에서 사용한 CPU 시간.
- **sys** : 커널에서 사용한 CPU 시간.

user + sys 값은 멀티스레드의 경우 각각의 시간을 모두 합하므로 real보다 커질 수 있다.

18. `passwd`

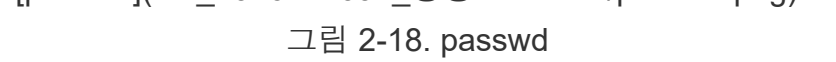

그림 2-18. passwd

passwd, 유저의 비밀번호를 변경한다. 일반유저는 자신의 비밀번호만 수정할 수 있고 superuser의 경우 모든 계정의 비밀번호를 변경할 수 있다.

19. `uname`

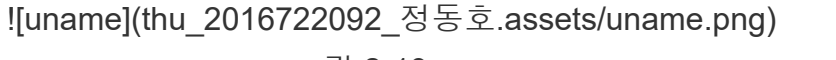

그림 2-19. uname

시스템 정보를 출력한다. 아무 옵션도 주지 않을 경우 s 옵션을 준것과 동일하며 기본적인 옵션들의 의미는 다음과 같다.

- **s** : 커널 이름 출력
- **r** : 커널 릴리즈 출력
- **m** : 장비 하드웨어 이름 출력
- **a** : 모든 정보 출력—커널 이름, 네트워크 노드 호스트이름, 커널 릴리즈, 커널 버전, 장비 하드웨어 이름, 프로세서 타입, 하드웨어 플랫폼, os 순으로.

20. `wc`

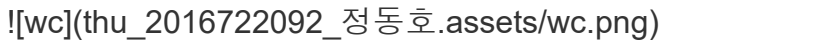

그림 2-20. wc

각 파일의 개행, 단어, 바이트 수를 알려준다. 단어 수만 알고 싶다면 w옵션을 주면 된다. 옵션과 파일은 동시에 여러개를 줄 수 있다.

21. `more`

    ![more](thu_2016722092_정동호.assets/more.png)

    그림 2-21. more

    출력 결과가 터미널 크기를 넘어가는 경우 사용자 편의를 위해 자동으로 멈춰준다. 눈에 띄는 옵션으로는 **f**(너무 길어서 개행된 줄은 무시), **s**(연속된 빈줄은 하나로 압축), **-number**(사용할 스크린 크기), **+number**(출력을 시작할 줄번호), **+/string**(출력하기 전에 탐색할 문자열)가 있다.

22. `echo`

    ![echo](thu_2016722092_정동호.assets/echo.png)

    그림 2-22. echo

    한 줄 짜리 텍스트를 출력한다. 단, e옵션을 줄 경우 backslash escape를 통해 여러 줄도 가능하다.

23. `alias`

    ![alias](thu_2016722092_정동호.assets/alias.png)

    그림 2-23. alias

    `alias`의 인자로 이름=값 쌍을 제공함으로써 비영구적인 별칭(alias)을 만들 수 있다. 나중에도 계속 쓰고 싶으면 ~/.bash_aliases에 등록한다.

24. `find`

    ![find](thu_2016722092_정동호.assets/find.png)

    그림 2-24. find

    `find`는 *find [-H] [-L] [-P] [-D debugopts] [-Olevel] [starting-point...] [expression]* 로 사용하며 디렉터리 계층에서 파일들을 탐색한다. 그림 2-24에서 나온 name옵션은 파일을 탐색할 때 따라오는 경로는 제외할때 쓴다.

    `man find`를 해보니 생각보다 내용이 많아서 다는 못해도 어느정도 정리할 필요를 느꼈다. --delete 옵션을 주면 찾은 파일들을 지운다. -exec em -i {} ; 옵션을 주면 똑같이 지우긴 하나 매번 물어보므로 더 안전하다. -type d 옵션을 줄 경우 디렉터리를 탐색한다. [+-]mtime [+-]num 옵션을 통해 마지막 수정시간으로 탐색이 가능하다. -perm 옵션을 통해 특정 권한을 가진 파일들을 탐색할 수 있다.

25. `grep`

    ![grep](thu_2016722092_정동호.assets/grep.png)

    그림 2-25. grep

    패턴과 매치되는 줄들을 출력한다. 사용법은 두가지가 있는데

    grep [OPTIONS] PATTERN [FILE...]

    grep [OPTIONS] [-e PATTERN]... [-f FILE]... [FILE...]

    가 있다. 각각 E, F, r 옵션에 대응하는 egrep, fgrep, rgrep 변종이 있지만

deprecated되었고 하위호환성을 위해서만 존재한다고 한다. 기본 grep의 사용법은 그림 2-25와 같고 두번째 처럼 e 옵션을 여러개 줄수도 있다. ?, +, | 등등의 확장 정규식 표현을 사용하려면 E 옵션을 사용한다.

3주차

1. `vi`

   ![vi-insert](thu_2016722092_정동호.assets/vi-insert.png)

   그림 3-1-1. vi - insert

   Nomal 모드에서 `i`를 입력하여 Insert mode로 진입한 뒤 insert를 입력했다.

   ![vi-delete](thu_2016722092_정동호.assets/vi-delete.png)

   그림 3-1-2. vi - delete

   e에 커서를 두고 Normal 모드에서 `D`를 입력하여 줄 끝까지 삭제했다.

   ![vi-save](thu_2016722092_정동호.assets/vi-save.png)

   그림 3-1-3. vi - save

   Normal 모드에서 `:w` 명령을 통해 현재 내용을 저장했다.

   ![vi-search](thu_2016722092_정동호.assets/vi-search.png)

   그림 3-1-4. vi - search

   `/test` 를 통해 test라는 문자열을 찾았다.

   ![vi-replace](thu_2016722092_정동호.assets/vi-replace.png)

   그림 3-1-5. vi - replace by pattern

   과제에서는 패턴에 의한 치환이 있었으나 강의자료에는 없어서 그냥 적당한 명령을 실행하여 모든 줄의 모든 =를 -로 치환하는 작업을 수행하였다.

2. `make`

   ![gcc](thu_2016722092_정동호.assets/gcc.png)

   그림 3-2-1. gcc

   강의 자료에 나온대로 `gcc`를 사용해 보았다. 다음으로 Makefile을 사용해본다. 명명 일관성을 위해 makefile이라 하려다가 `man make`에서 디렉터리 항목을 나열할 때 Makefile을 쓰면 README처럼 앞쪽에 강조된다는 이유로 Makefile을 권장하길래 Makefile로 만들었다.

   언뜻 보면 Makefile은 C나 쉘 스크립트보다 어려워 보인다. 그래서 이것저것 찾아봤고 그 중 강의자료에 나온것들을 정리해보았다. 일단 `make`의 존재 의의는 서로 의존관계에 있는 파일들을 자동으로 관리해 준다는 것이다. `make`는 자동으로 파일들

의 마지막 수정시간(mtime)을 비교하여 target보다 dependency가 최신이거나 target이 존재하지 않으면 해당 rule을 실행시키고 만약 해당 dependency가 없으면 그 dependency가 target인 rule을 찾아 먼저 실행시킨다. 따라서 `make` & Makefile을 사용하면 여러 파일들이 서로 의존하고 있는 프로젝트에서 매번 모두 재컴파일을 해야할 수고를 덜 수 있다.

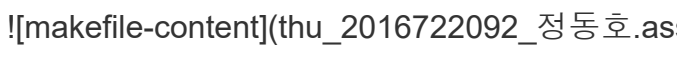

그림 3-2-2. makefile content

OBJS ... CC는 변수로써 $()를 통해 참조될 수 있다. 들여쓰기 없이 :(콜론)이 포함된 줄은 rule로써 일반적인 경우엔 `make`를 할 시 첫번째 rule만 실행된다. :를 기준으로 왼쪽은 target, 오른쪽은 dependency이다. 그 밑에는 Tab으로 들여쓰기된 command가 나오고 위에서 정의된 값으로 풀어쓰면
`gcc -o test.o test1.c test2.c`가 된다. 따라서 기본적으로 `make`를 실행할 경우 현재경로에서 test1.c 와 test2.c를 찾아 test.o를 빌드하고 `make clean`으로 실행할 경우 백업파일들을 포함한 관련된 모든 파일들을 지운다.
`make`의 실행 결과는 다음과 같다:

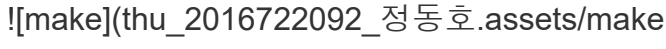

그림 3-2-3. make

3. `gdb`

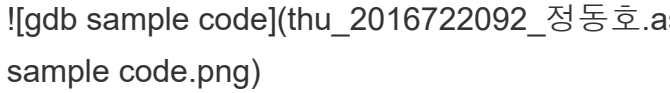

그림 3-3-1. gdb sample code

`gdb`를 사용하기 앞서 강의자료에 나온대로 코드를 작성했다. 실행하면 다음과 같은 오류가 발생한다:

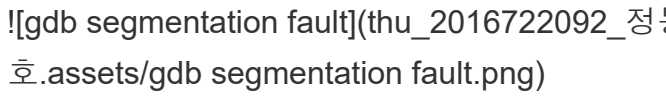

그림 3-3-2. gdb segmentation fault

그래서 강의자료에 나온대로 `gdb`를 사용했다. `gdb`를 통한 디버깅 과정은 다음과 같다:

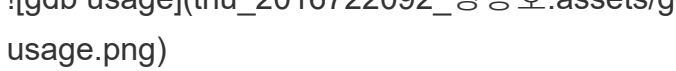

그림 3-3-3. gdb usage

`b 9` 로 9번째 라인에 중단점을 걸고 `r 11 22 33`으로 인자 11, 22, 33을 주어 프로그램을 실행했다. for문 전후로 i의 값을 관찰해보고 segmentation fault가 일어나는 지점에서 `bt` 를 통해 스택을 확인할 수 있었다.

Reference

https://en.wikipedia.org/wiki/Man_page — man page 부가 설명

https://www.interserver.net/tips/kb/learn-linux-file-system-permissions/, 강의자료실 -2018-1_SSLab_week02_Unix Linux Commands.pdf — 전반적인 내용, File permission 부가 설명

`man mkdir` — 어떻게 mkdir로 subdirectory까지 생성 가능한지 정보

https://askubuntu.com/a/801191 — soft link vs hard link 부가 설명

`man ps` — 아무 옵션도 안준 ps는 무엇을 출력하는지 설명

https://kb.iu.edu/d/afnv — ps의 각 필드의 의미

https://stackoverflow.com/a/556411/7899226 — time의 각 요약 정보의 의미

`man uname` — uname 옵션들의 의미

`man more` — more 옵션들의 의미

https://shapeshed.com/unix-find/ — find 부가 설명

http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/
http://mrbook.org/blog/tutorials/make/
http://makefiletutorial.com/ — make & Makefile 부가 설명