

2025년 1학기 시스템프로그래밍 실습 3주차

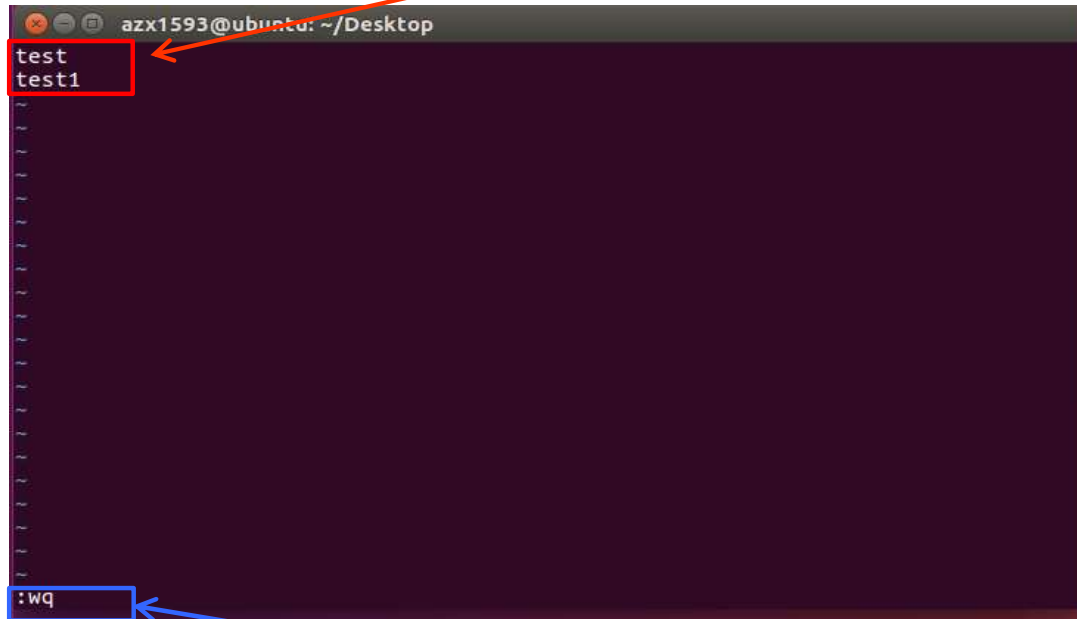
Linux – based Programming

System Software Laboratory
College of Software and Convergence
Kwangwoon Univ.

vi Editor (1/4)

- vi 에디터

- Linux 기본 편집기.
- 실행 화면



```
azx1593@ubuntu: ~/Desktop
test
test1
:wq
```

입력 모드 시, 소스 코드 등의 내용을 입력

명령 모드 시, 콜론(:) or 슬래시(/) 등으로 시작하는 vi 명령어 입력

- vim 패키지 설치

- 기본 설치된 vi는 기능이 제한적이므로 vim 패키지 설치 필요.
- # **sudo apt-get install vim**
 - Ubuntu 기준.

vi Editor (2/4)

- **vi 모드**
 - 명령모드
 - 한/두 문자로 구성된 vi 전용 명령어를 사용하는 모드
 - vi 진입 시 기본 모드
 - 입력모드에서 [esc] 키로 진입 가능
 - 입력모드
 - vi 편집 화면에서 문자를 입력 할 수 있는 상태를 의미
 - 명령 모드에서 [esc] 후 [i] 등의 키로 진입 가능
- **vi 시작과 종료**
 - 시작
 - vi file_name → vi를 시작하여 지정한 파일 편집
 - 종료
 - :wq → 데이터를 저장하고 종료
 - :q → 데이터를 저장하지 않고 종료
 - :q! → 데이터를 저장하지 않고 강제 종료

vi Editor Options (1/3)

■ 커서 이동

h	← 이동
j	↓ 이동
k	↑ 이동
l	→ 이동
Backspace	커서가 있는 행에서 커서를 왼쪽으로 옮김
Space	커서가 있는 행에서 커서를 오른쪽으로 옮김

■ 텍스트 변경

r	커서가 있는 문자를 변경
R	커서가 있는 부분부터 글자 덮어서 씀
s	한 글자를 삭제 한 후 문장 삽입
S	커서가 있는 문장을 삭제하고 문장 삽입
C	커서가 있는 행에서 커서를 왼쪽으로 옮김

■ 문자, 행, 삽입

i	커서 앞으로 문장 삽입
I	행의 시작 부분에서 문장 삽입
a	커서 뒤로 문장 삽입
A	행의 끝 부분에서 문장 삽입
o	커서가 위치한 행의 아래에 문장 삽입
O	커서가 위치한 행의 위에 문장 삽입

■ 텍스트 삭제

cc	현재 행 삭제 후 문장 입력
cw	커서가 있는 문자 삭제 후 문장 입력
x	커서가 있는 한 글자 삭제
X	커서 앞 한 글자 삭제
D	커서가 있는 부분의 뒷 부분의 행을 삭제
dd	커서가 있는 한 행 삭제

vi Editor Options (2/3)

■ 복사 및 붙여 놓기

nY	커서가 있는 행부터 n행 만큼 복사
yy	커서가 있는 행 복사
p	커서 아래 복사된 문자열 붙임 (소문자 p)
P	커서 위에 복사된 문자열 붙임 (대문자 P)

■ 파일 불러오기

:e 파일 이름	vi 를 종료하지 않고 해당 파일 편집
:e!	현재 편집하고 있는 파일 다시 부르기
:e#	한 글자를 삭제 한 후 문장 삽입

■ 파일 저장

:w	vi 파일을 저장
:w 파일 이름	파일이름으로 저장
:wq	저장 후 종료
:wq!	저장 후 강제 종료

■ 행 번호 설정

:set number	편집기의 라인 표시
:set nu	편집기의 라인 표시
:set non	편집기의 라인 표시 없애기

vi Editor 단축키

version 1.1

April 1st, 06

Esc

명령 모드

vi / vim

단축키 모음

~ 대소문자 전환

! 외부 명령

@ 매크로 실행

이전 검색

\$ 줄 끝으로 이동

% 일치하는 줄로 찾기

^ 줄의 첫 글자

& :s 반복

* 다음 검색

(문장 시작

) 문장 끝

아래 줄로 이동

+ 다음 줄

1

2

3

4

5

6

7

8

9

0 줄의 처음

- 이전 줄

= 자동 들여쓰기

Q 실행 모드

W 다음 WORD

E 끝 WORD

R 수정 모드

T 뒤로 검색

Y 줄단위 복사

U 줄 단위 실행 취소

I 줄 시작에서 삽입

O 행 위에 삽입

P 커서 이전에 붙여넣기

{ 문단 시작

} 문단 끝

q 매크로 기록

w 다음 단어

e 단어 끝

r 한 문자 교체

t 한 문자 검색

y 복사

u 실행 취소

i 편집 모드

o 행 아래에 삽입

p 커서 이후에 붙여넣기

[기타

] 기타

A 줄 끝에 덧붙이기

S 줄 삭제 후 편집 모드

D 줄 끝까지 삭제

F 뒤로 검색

G 파일 끝/줄로 이동

H 화면 상단

J 줄 랑잡기

K 다음 줄

L 화면 하단

: 명령줄

ex 레지스터 지정

| 열 이동

a 덧붙이기

s 단어 삭제 후 편집 모드

d 삭제

f 한 문자 찾기

g 확장 명령

h ←

j ↓

k ↑

l →

: 명령줄

> 명령 반복

! 매크로 이동

\ 사용 안함

Z 종료

X 백스페이스

C 줄 끝까지 바꾸기

V 줄단위 비주얼 모드

B WORD

N 이전 (찾기)

M 화면 가운데

< 내려쓰기

> 들여쓰기

? 찾기 (뒤로)

Z 확장 명령

X 글자 삭제

c 바꾸기

v 비주얼 모드

b 이전 단어

n 다음 (찾기)

m 매크로 설정

, 역T/F

. 명령 반복

/ 찾기

동작

커서를 이동하거나, 연산자가 동작할 범위를 지정합니다.

명령

바로 동작하는 명령.
빨간색은 편집 모드로 변경됩니다.

연산자

이동 관련 문자(숫자나 커서 이동)와 함께 사용하여 하며, 커서의 위치부터 목적지까지 연산합니다.

확장

특별한 키 함수로, 추가적인 키 입력이 필요합니다.

q

입력후 (숫자를 제외한 .으로 끝낼수 있는) 글자를 입력하여야 합니다.

words: 구분자로 공백, 특수기호 모두 사용

WORDS: 구분자로 공백 문자만 사용

words: quux(foo, bar, baz)

WORDS: quux(foo, bar, baz)

주요 명령행 명령 ('ex'):

:w (저장), :q (종료), :q! (저장하지 않고 종료)

:e f (파일 f 열기),

:%s/x/y/g (파일 전체에서 'x'를 'y'로 교체),

:h (vim 도움말), :new (새 파일)

그외 중요한 명령들:

CTRL-R: 세션행 (vim),

CTRL-F/-B: 페이지 위로/아래로,

CTRL-E/-Y: 줄 스크롤 위로/아래로,

CTRL-V: 블록-비주얼 모드 (vim 전용)

비주얼 모드:

커서를 움직여 지정한 범위에 연산자를 적용합니다. (vim 전용)

참고:

(1) 복사/붙여넣기/지우기 명령어를 사용하기 전에 "x"를 입력하여 레지스터(클립보드)를 지정하세요. (x는 a에서 z 또는 * 을 사용할 수 있음) (예: "ay\$를 입력하면 현재 커서에서 라인 끝까지의 내용을 레지스터 'a'에 저장합니다.)

(2) 어떤 명령을 입력하기 전에 횡수를 지정하면, 횡수만큼 반복하게 됩니다.(예: 2p, d2w, 5l, d4j)

(3) 연속으로 입력하는 명령은 현재의 라인에 반영됩니다. 예시: dd(현재 라인 지우기), >>(들여쓰기)

(4) ZZ는 저장후 종료, ZQ는 저장하지 않고 종료.

(5) zt : 커서가 위치한 곳을 제일위로 올리기, zb : 바닥으로, zz : 가운데로

(6) gg: 파일의 처음으로(Vim 전용), gf: 커서가 위치한 곳의 파일 열기(Vim 전용)

vi/vim 에 대한 더 많은 강좌나 팁을 얻으려면 www.viemu.com (ViEmu, MS 비주얼 스튜디오를 위한 vi/vim 에뮬레이션)을 방문하십시오.

- Reference
 - 영문 버전: http://www.viemu.com/a_vi_vim_graphical_cheat_sheet_tutorial.html
 - 한글 버전: <https://kldp.org/node/102947>

GCC Compiler

- 컴파일

- hello.c 파일 준비

```
#include <stdio.h>

void main()
{
    printf("Hello World! \n");
}
```

- 컴파일

- \$ gcc (파일명)
- e.g. \$ gcc hello.c

```
sslab@ubuntu:~/Desktop$ ls
hello.c
sslab@ubuntu:~/Desktop$ gcc hello.c
sslab@ubuntu:~/Desktop$ ls
a.out hello.c
```

- 실행파일 이름을 정해주지 않고 컴파일 한 경우에는 a.out으로 자동 생성됨

- 실행

- \$./a.out

```
sslab@ubuntu:~/Desktop$ ./a.out
Hello world!
```

- 실행 파일명 지정

- \$ gcc -o (실행파일명) (소스파일명)
- e.g. \$ gcc -o hell hello.c

```
sslab@ubuntu:~/Desktop$ gcc -o hell hello.c
sslab@ubuntu:~/Desktop$ ls
a.out hell hello.c
```

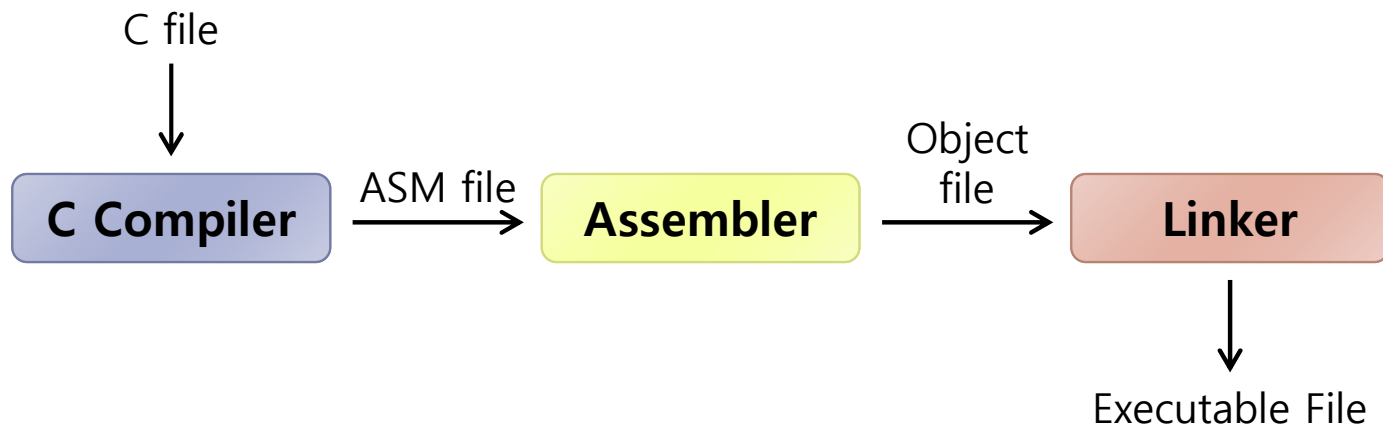
make (1/6)

- **필요성**

- 컴파일 과정을 자동화하기 위해 사용
 - gcc 컴파일러의 다양한 옵션들을 컴파일 할 때마다 입력한다면 vi 실행/편집/종료, 컴파일을 반복하는 데 많은 시간이 소요될 것임.

- **Makefile**

- 컴파일 할 소스 파일과 컴파일 옵션에 관해 정의해놓은 스크립트 파일
- 아래 그림과 같이 빌드는 여러 과정으로 구성되어 있는데, 이를 통해 한 번 만에 빌드를 수행할 수 있도록 함



make (2/6)

- Makefile (cont'd)

- 작업할 대상 → 타겟
 - 타겟명을 의미하기 위해 이름 뒤에 ':'
 - 타겟명 다음 줄에는 실행할 명령을 명시

타겟 → `hello:` `hello.c` ← 타겟 명
`gcc hello.c -o hello` ← 명령어

- 여러 개 빌드
 - 'all:' 이라는 타겟 뒤에 새로운 하위 타겟('hello hello2') 추가
 - make는 'all' 타겟에서 새로운 타겟 'hello hello2' 를 확인하고 실행
 - hello2만 빌드하고 싶다면, 'make hello2' 실행

```
all: hello1.c hello2.c

hello1: hello1.c
    gcc hello1.c -o hello1

hello2: hello2.c
    gcc hello2.c -o hello2
```

타겟(Target)	명령어가 수행되어 나온 결과를 저장한 파일
타겟 명(Target Name)	소스 파일 명
명령어(Command)	실행 명령어

make (3/6)

- **Makefile (cont'd)**
 - make 수행 시 타겟을 명시하지 않는 경우
 - i.e. \$ make
 - Makefile의 제일 첫 번째 타겟에 해당하는 명령 수행
 - make 수행 시 타겟을 명시하는 경우
 - i.e. \$ make hello1
 - 해당 타겟의 명령 수행

make (4/6)

- 변수사용과 대체

- 변수는 '이름 = 값'의 형태로 지정
- 타겟은 '\$@'로 대체, 타겟명은 '\$^'로 대체
- 변수는 '\$(이름)'형태로 사용
- E.g1.

```
test: hello.c
gcc -o $@ $^
```

타겟 ← test: hello.c → 타겟명

- E.g2.

```
OBJS = hello1.c hello2.c
CC   = gcc
EXEC = test

all: $(OBJS)
     $(CC) -o $(EXEC) $^

clean:
     rm -rf $(EXEC)
```

변수 명 ← OBJS → 값

make (5/6)

- 프로그램 실행 하기
 - 실행 결과

```
sslab@ubuntu:~/Desktop$ ls
hello1.c hello2.c hello2.h Makefile
sslab@ubuntu:~/Desktop$ make
gcc -o test hello1.c hello2.c
sslab@ubuntu:~/Desktop$ ls
hello1.c hello2.c hello2.h Makefile test
sslab@ubuntu:~/Desktop$ ./test
Hello world! 1
Hello world! 2
sslab@ubuntu:~/Desktop$
```

- 소스

```
#include <stdio.h>
#include "hello2.h"

int main(int argc, char **argv)
{
    printf("Hello World! 1 \n");
    hello2_func();
    return 0;
}
```

hello1.c

```
#include <stdio.h>

void hello2_func()
{
    printf("Hello World! 2 \n");
}
```

hello2.c

```
#ifndef __HELLO2_H__
#define __HELLO2_H__

int hello2_func(void);

#endif
```

hello2.h

```
OBJS = hello1.o hello2.o
CC = gcc
EXEC = test

all: $(OBJS)
    $(CC) -o $(EXEC) $^

clean:
    rm -rf $(EXEC)
```

Makefile

make (6/6)

- 매개변수

- Integer형 변수
 - 입력 인자의 개수를 저장
- char* 배열 형 변수
 - 입력 인자들을 배열 형으로 저장

example.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int i;
    printf("The Number Of Inputted Variable is %d \n", argc);
    printf("and they are ");

    for(i=0; i<argc; i++)
    {
        printf("%s ", argv[i]);
    }

    printf("\n");

    return 0;
}
```

수행 결과

```
sslab@ubuntu:~/Desktop$ ls
example.c
sslab@ubuntu:~/Desktop$ gcc example.c
sslab@ubuntu:~/Desktop$ ls
a.out example.c
sslab@ubuntu:~/Desktop$ ./a.out hello! system Programming lab. Class!
The Number of Inputted Variable is 6
and they are ./a.out hello! system Programming lab. Class!
sslab@ubuntu:~/Desktop$
```

Make 실습

- Makefile 작성

- Makefile을 생성하고 컴파일 후 "client, server"를 출력 하시오.

- 결과

```
sslab@ubuntu:~/Desktop$ ls
cli.c Makefile srv.c
sslab@ubuntu:~/Desktop$ make
gcc -o srv srv.c
gcc -o cli cli.c
sslab@ubuntu:~/Desktop$ ls
cli cli.c Makefile srv srv.c
sslab@ubuntu:~/Desktop$ ./cli
client
sslab@ubuntu:~/Desktop$ ./srv
server
sslab@ubuntu:~/Desktop$
```

- 코드

```
#include <stdio.h>

int main()
{
    printf("client\n");
    return 0;
}
```

cli.c

```
#include <stdio.h>

int main()
{
    printf("server\n");
    return 0;
}
```

srv.c

?

Makefile

2025년 1학기 시스템프로그래밍 & 시스템 프로그래밍 실습

Basic

System Software Laboratory
College of Software and Convergence
Kwangwoon Univ.

Contents

- **Basic-1. Ubuntu Installation**
- **Basic-2. Linux Commands**
- **Basic-3. Linux based Programming**
- **Report Requirements**

Basic-1. Ubuntu Installation

- 과제 내용

- Vmware 설치 과정 및 Ubuntu를 설치하는 과정을 캡처하고 설명

- 조건

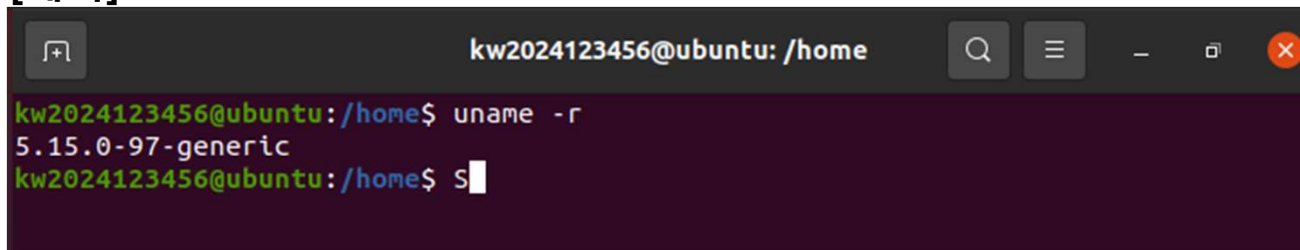
- 설치 하는 방법(multi-booting, virtual machine, ...)은 무관
 - 단, Virtual machine을 사용할 경우, tool(VMWare, ...) 설치 과정은 과제에서 제외

- Ubuntu 로그인 계정 설정 시 본인 학번 앞에 'kw' 입력 후 학번 입력

- ex) kw2024123456
- \$uname -r 명령어 실행

- 터미널 화면 캡처하여 보고서에 첨부

- [예시]



```
kw2024123456@ubuntu: /home
kw2024123456@ubuntu:/home$ uname -r
5.15.0-97-generic
kw2024123456@ubuntu:/home$ s
```

- 실습 강의자료 참고할 것.

- [실습강의자료]1. Introduction

Basic-2. Linux Commands

- 과제 내용

- 실습 시간에 배운 Linux 명령어를 사용하고, 이를 캡처하고 설명

- 조건

- 아래의 명령어를 모두 사용하고, 과정을 캡처 및 핵심 내용에 대한 설명 필수
- man, ls, pwd, cd , cat, chmod, mkdir, rmdir, rm, cp, mv, ln, touch, ps, exit, kill, ps, passwd, uname, wc, echo, alias, grep
- 다음 아래 명령어는 과제 수행 시 지정한 옵션 반드시 사용
 - ls, ln, uname
 - 강의 자료에 명시 된 옵션만 사용
 - rm
 - -r 옵션만 사용
- 실습 강의자료 참고할 것.
 - [실습강의자료]2. Unix/Linux commands

Basic-3. Linux based Programing

- 과제 내용
 - 실습 시간에 배운 vi, make 사용하고, 이를 캡처하고 설명
- 조건
 - Vi editor
 - **각 단계 수행 후 사용한 명령어 설명과 결과 화면 캡처 필수**
 1. 1라인: 본인 학번, 2라인: 본인 이름, 3라인: Kwangwoon University 입력
 2. Kwangwoon University 복사 후 본인 학번 다음 라인에 복사 붙어 놓기
 3. 편집기에 라인 표시
 4. 본인 학번을 파일 이름으로 저장
 - Make
 - 본인의 학번, 이름이 출력되는 c파일 Makefile로 컴파일
 - kw_hello.c : 본인 학번과 이름이 출력하는 프로그램(sprintf) 사용
 - 실행 파일명: hello
 - **실행 결과, Makefile, kw_hello.c 캡처 필수**
- 실습 강의자료 참고할 것.
 - [실습강의자료]3. Linux-based Programming

Report Requirements

- **Ubuntu 20.04.6 Desktop 64bits 환경에서 채점**
- **Copy 발견 시 0점 처리**
- **보고서 구성**
 - **보고서 표지**
 - 수업 명, 과제 이름, 담당 교수님, 학번, 이름 필히 명시
 - 과제 이름 → Basic
- **과제 내용**
 - Introduction
 - 과제 소개 - 4줄 이상(background 제외) 작성
 - 결과화면
 - 수행한 내용을 캡처 및 설명
 - 고찰
 - 과제를 수행하면서 느낀점 작성
 - Reference
 - 과제를 수행하면서 참고한 내용을 구체적으로 기록
 - 강의자료만 이용한 경우 생략 가능

Report Requirements

- Softcopy Upload

- 제출 파일
 - 보고서를 **pdf로 변환**하여 제출
 - 보고서 이름은 **Basic_수강분류코드_학번_이름**으로 작성

수강요일	이론1 월5수6	이론2 목4	실습1 목12
수강분류 코드	A	B	C

- 예시1 -이론1만 수강하는 학생인 경우
 - Basic_A_2024123456_홍길동.pdf
- 예시2 -이론1 & 실습1을 수강하는 학생인 경우

- 이론 :  실습수업때 제출했습니다. 2022-08-29 오후 3:58 텍스트 문서 0KB

- 실습 : Basic_C_2024123456_홍길동.pdf

- 예시3 -실습1만 수강하는 학생인 경우
 - Basic_C_2024123456_홍길동.pdf

- 양식에 따르지 않을 시 감점

Report Requirements

- 실습 수업을 수강하는 학생인 경우

- 실습 과목에 과제를 제출
- 이론 과목에 간단한 .txt 파일로 제출

실습수업때 제출했습니다.

2022-08-29 오후 3:58

텍스트 문서

0KB

- 이론 과목에 .txt 파일 미 제출 시 감점

- 과제 제출

- KLAS – 강의 과제 제출
- 2025년 3월 27일 목요일 23:59까지 제출
- 딜레이 받지 않음
 - 제출 마감 시간 내 미제출시 해당 과제 0점 처리
 - 교내 서버 문제 발생 시, 메일로 과제 제출 허용(제출 기한 내)

2025년 1학기 시스템프로그래밍 실습 3주차

Appendix

System Software Laboratory
College of Software and Convergence
Kwangwoon Univ.

| gdb (1/8)

- **GDB(GNU DeBugger) 란?**

- 어떤 프로그램이 수행되는 도중 그 프로그램 내에서 어떤 일이 일어나는지를 볼 수 있게 해 주는 툴.

- **GDB 기능**

- 프로그램을 수행시킨다.
- 어떤 특별한 조건에서 프로그램의 수행을 stop 시킨다.
- 프로그램이 stop 된 상태에서 그 프로그램의 내부를 볼 수 있다.
- 프로그램의 일부분을 수정한다.
- Stop 된 프로그램을 continue 시킨다.

| gdb (2/8)

- 디버깅하기 위한 컴파일 옵션
 - `$ gcc -g Executable_file`
- GDB 시작 – GDB 프롬프트"(gdb)" 가 나옴
 - `$ gdb 실행파일`
 - 현재 수행중인 프로그램(프로세스 번호)를 디버깅
 - 매개변수의 사용 여부에 따라 `run(r)` 혹은 `run arg1 arg2 ...`
- GDB 디버깅 진행
 - `step(s)` 혹은 `next(n)`
- GDB 종료
 - `quit(q)` 혹은 `^D (ctrl + D)`

| gdb (3/8)

- **Breakpoint 란?**
 - 프로그램의 수행을 정지시키는 지점
- **특정 함수에 breakpoint 설정**
 - `break function` 혹은 `b function`
- **프로그램 소스 줄에 breakpoint 설정**
 - `break linenum` 혹은 `b linenum`
- **현재 위치에서 상대적 위치에 breakpoint 설정**
 - `break +offset` 혹은 `break -offset`
- **조건 breakpoint 설정**
 - `break ... if cond`
 - e.g., **(gdb) `break 10 if i == 5`**
 - 변수 `i`의 값이 5일 경우 10번 행에 breakpoint 설정
- **Breakpoint 설정지점 보기**
 - `info breakpoints`

gdb (4/8)

- Watchpoint 란?

- 특정 식의 값이 변경되거나 읽혀질 때 프로그램의 수행이 stop 하는 특별한 breakpoint

- 프로그램에 의하여 특정 변수가 쓰여지면(write) breakpoint 형성

- watch 변수

- 프로그램에 의하여 특정 변수가 읽혀지면(read) breakpoint 형성

- rwatch 변수

- 특정 변수가 써지거나(write) 혹은 읽혀지면(read) breakpoint 형성

- awatch 변수

- 설정된 watchpoint 보기

- info watchpoints

실행 예시

```
(gdb) watch j
Hardware watchpoint 2: j
(gdb) c
Continuing.
j is 0.000000

Hardware watchpoint 2: j

Old value = 0
New value = 1
main () at example1.c:10
10      printf("j is %f \n" , j);
```

| gdb (5/8)

▪ 변수 값 보기

- `print or p 변수`
- `print /f expr` // f는 format(예: /d /o /x)
- `info locals` // 지역 변수들 정보 출력
- `info variables` // 전역 변수들 정보 출력

▪ 메모리 값 보기

- x 명령어는 메모리 특정 범위의 값들을 확인하는데 사용하는 명령어

▪ 사용 방법

- `x/[범위][출력format][단위]`
- `x/[범위][단위][출력format]`
- `x /nfu 주소값`
 - n : 개수
 - f : format
 - u :단위 (e.g. b(1), h(2), w(4))

B(1byte) h(2byte) w(4byte)	
/t	2진수
/o	8진수
/d	10진수
/u	Unsigned int
/x	16진수
/f	부동소수점 값
/s	문자열

▪ 변수 값 수정(대입)

- `print x=4`

▪ 다른 곳으로 jump

- `jump linespec`

gdb (6/8)

- 스택 전체 보기
 - backtrace 혹은 bt

```
#include <stdio.h>

int kk(void)
{
    int k = 5;
    printf("%d", k);
}

void main()
{
    int i;
    double j;

    kk();
    for( i=0; i<5 ; i++)
    {
        j= i / 2 + i;
        printf("j is %f \n" , j);
    }
}
```

example1.c

```
(gdb) list 10
5         int k = 5;
6         printf("%d", k);
7     }
8
9     void main()
10    {
11        int i;
12        double j;
13
14        kk();
(gdb) b 14
Breakpoint 1 at 0x400554: file example1.c, line 14.
(gdb) r
Starting program: /home/azx1593/Desktop/TT/gg/anything

Breakpoint 1, main () at example1.c:14
14        kk();
(gdb) n
15        for( i=0; i<5 ; i++)
(gdb) bt
#0  main () at example1.c:15
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/azx1593/Desktop/TT/gg/anything

Breakpoint 1, main () at example1.c:14
14        kk();
(gdb) s
kk () at example1.c:5
5         int k = 5;
(gdb) bt
#0  kk () at example1.c:5
#1  0x0000000000400559 in main () at example1.c:14
```

실행 결과

| gdb (7/8)

- 소스 라인 프린트
 - `list linenum`
 - 해당 줄을 기준으로 출력
 - `list function`
 - 해당 함수의 정의를 출력
 - `list`
 - 현재 줄의 아래쪽 소스를 출력
 - `list -`
 - 현재 줄의 위쪽 소스를 출력

gdb (8/8)

▪ gdb options

list	현재 위치에서 소스 파일의 내용을 10줄 보여준다 ex) list 2, 15 : 소스 파일의 2 ~ 15 까지를 보여준다.
run	프로그램을 시작한다.(break가 있다면 break까지 실행)
break	특정 라인이나 함수에 정지점을 설정한다. break function : 현재 파일 안의 함수 function에 정지점을 설정한다. break file:function : 파일 file안의 function에 정지점을 설정한다. watch : 감시점 설정(감시점은 어떤 사건이 일어날 때에만 작동한다) until : 실행 중 line까지 실행
clear	특정 라인이나 함수에 있던 정지점을 삭제한다.
delete	몇몇 정지점이나 자동으로 출력되는 표현을 삭제한다.
next	다음 행을 수행한다. 서브루틴을 호출하면서 계속 수행한다.
step	한 줄씩 실행 시킨다.
print	print expr : 수식의 값을 보여준다.
display	현재 display된 명령의 목록을 보여준다.
bt	프로그램 스택을 보여준다. (backtrace)
kill	디버깅 중인 프로그램의 실행을 취소한다.
file	file program : 디버깅할 프로그램으로서 파일을 사용한다.
count	continue : 현재 위치에서 프로그램을 계속 실행한다.
help	명령에 관한 정보를 보여주거나 일반적인 정보를 보여준다.
quit	gdb에서 빠져나간다.