LINUX 개발환경(4)

- 소프트웨어 컴파일
- 소스 코드 디버깅

Computer & Software Engineering
Lee Jong Yeol
soyeum@kumoh.ac.kr

소프트웨어 컴파일

❑ GNU C 프로그래밍 (1)

- C 언어로 작성된 프로그램을 컴파일하기 위해 GNU C 컴파일러(gcc) 설치
- gcc는 C/C++ 모두 컴파일 가능함

```
<gcc 설치 확인>
linuxer@linuxer-PC:~$ gcc -v                    (컴파일러 버전 확인)

<설치되어 있지 않다면 설치>
linuxer@linuxer-PC:~$ sudo apt-get install gcc
linuxer@linuxer-PC:~$ man  gcc

<C++ 전용 컴파일러 설치하려면>
linuxer@linuxer-PC:~$ sudo apt-get install g++      (GNU C++ 컴파일러)
```

소프트웨어 컴파일

❑ GNU C 프로그래밍 (2)

▪ 간단한 C 코드 작성

```
linuxer@linuxer-PC:~$ vi  hello.c

#include <stdio.h>

int main( ) {
    printf("Hello, World.₩n");
    return 0;
}
```

▪ 컴파일 및 실행

```
linuxer@linuxer-PC:~$ gcc  hello.c
linuxer@linuxer-PC:~$ ./a.out
```

소프트웨어 컴파일

❑ GNU C 프로그래밍 (3)

▪ 단일 모듈 프로그램 작성 : long.c

```
#include <stdio.h>
#include <string.h>

#define MAXLINE 100

void copy(char from[], char to[]);
char line[MAXLINE];
char logest[MAXLINE];

int main()
{
    int len, index=1, max=0;

    printf("[%d] ", index++);
    fgets(line, MAXLINE, stdin);

    while(strlen(line) > 1) {
        len = strlen(line);
        if( len > max) {
            max = len;
            copy(line, longest);
        }
        printf("[%d] ", index++);
        fgets(line, MAXLINE, stdin);
    }
    if(max > 0)
        printf("logest: %s", longest);
    return 0;
}

void copy(char from[], char to[])
{
    int i = 0;
    while ((to[i] = from[i]) != '₩0')
            i++;
}
```

소프트웨어 컴파일

❑ GNU C 프로그래밍 (4)

▪ 컴파일과 링크 방법

```
<컴파일만 진행>
linuxer@linuxer-PC:~/long$ gcc  -c long.c
linuxer@linuxer-PC:~/long$ ls
long.c  long.o

<링크만 진행>
linuxer@linuxer-PC:~/long$ gcc  -o long  long.o
linuxer@linuxer-PC:~/long$ ls
long  long.c  long.o

<컴파일과 링크 동시에 진행(빌드)>
linuxer@linuxer-PC:~/long$ gcc  -o long  long.c
```

소프트웨어 컴파일

❑ GNU C 프로그래밍 (5)

▪ 다중 모듈 프로그램 작성 : main.c main.h copy.c copy.h

```
/* main.c */

#include "main.h"

int main()
{
    int len, index=1, max=0;
    printf("[%d] ", index++);
    fgets(line, MAXLINE, stdin);

    while(strlen(line) > 1) {
        len = strlen(line);
        if( len > max) {
            max = len;
            copy(line, longest);
        }
        printf("[%d] ", index++);
        fgets(line, MAXLINE, stdin);
    }
    if(max > 0)
        printf("logest: %s", longest);
    return 0;
}
```

```
/* main.h */
#include <stdio.h>
#include <string.h>
#include "copy.h"

#define MAXLINE 100

char line[MAXLINE];
char logest[MAXLINE];
```

```
/* copy.c */
#include "copy.h"

void copy(char from[], char to[])
{
    int i = 0;
    while ((to[i] = from[i]) != '₩0')
            i++;
}
```

```
/* copy.h */
void copy(char from[], char to[]);
```

소프트웨어 컴파일

❑ GNU C 프로그래밍 (6)

▪ 다중 모듈 컴파일과 링크

```
<컴파일만 진행>
linuxer@linuxer-PC:~/long2$ gcc  -c main.c  copy.c
linuxer@linuxer-PC:~/long2$ ls
copy.c  copy.h  copy.o  main.c  main.h  main.o

<링크만 진행>
linuxer@linuxer-PC:~/long2$ gcc  -o main  main.o  copy.o
linuxer@linuxer-PC:~/long2$ ls
copy.c  copy.h  copy.o  main  main.c  main.h  main.o

<컴파일과 링크 동시에 진행>
linuxer@linuxer-PC:~/long2$ gcc  -o main  *.c
```

소프트웨어 컴파일

❑ GNU C 프로그래밍 (7)

▪ 다중 모듈 프로그램 작성시 오류

```
/* main.h */

#include "copy.h"
...
```

```
/* copy.h */

#include "main.h"
...
```

```
/* main.c */

#include "main.h"

int main() {
    ....
    copy( ... );
    ...
}
```

```
/* copy.c */

#include "copy.h"

void copy( ... )
{
    ...
}
```

```
linuxer@linuxer-PC:~/long2$ gcc -o main *.c
In file included from main.h:4,
                 from copy.h:2,
                 from main.h:4,
                 from copy.h:2,
                 ...
copy.h:2:18: error: #include nested too deeply
    2 | #include "main.h"
In file included from copy.h:2,
                 from main.h:4,
                 from copy.h:2,
                 from main.h:4,
                 ...
main.h:2:19: error: #include nested too deeply
    2 | #include <stdio.h>
      |                   ^
main.h:3:20: error: #include nested too deeply
    3 | #include <string.h>
      |                    ^
main.h:4:18: error: #include nested too deeply
    4 | #include "copy.h"
      |                  ^
linuxer@linuxer-PC:~/long2$
```

소프트웨어 컴파일

❑ GNU C 프로그래밍 (8)

▪ 헤더 파일 중복 호출 문제 해결

- [방법-1] 전처리기에 중복 방지 요청
- [방법-2] 조건부 컴파일 방식 활용
- [방법-3] 두 가지 방법 모두 사용

```
/* main.h */

#ifndef __MAIN_H__
#define __MAIN_H__

#pragma once

#include "copy.h"
...

#endif  // __MAIN_H__
```

```
/* copy.h */

#ifndef __COPY_H__
#define __COPY_H__

#pragma once

#include "main.h"
...

#endif  // __COPY_H__
```

소프트웨어 컴파일

❑ GNU C 프로그래밍 (9)

- 대규모 프로그램의 컴파일 작업
  - 헤더 파일, 소스 파일, 목적 파일, 실행 파일 등의 모든 관계를 기억해야 함
  - 모든 파일들의 경로를 알고 있어야 컴파일 가능 함
    => **현실적으로 불가능**

<dir1/main.c>

```
extern void first( );
extern void second( );

void main( ) {
    first( );
    second( );
}
```

<dir2/first.c>

```
#include <stdio.h>

void first( ) {
  printf("first.c ---₩n");
}
```

<dir3/second.c>

```
#include <stdio.h>

void second( ) {
  printf("second.c ---₩n");
}
```

...

<빌드>

```
linuxer@linuxer-PC:~$ gcc -o ./main  ./dir1/main.c  ./dir2/first.c
                                      ./dir3/second.c  ......
```

소프트웨어 컴파일

❑ GNU C 프로그래밍 (10)

▪ make 시스템 활용

- 실행 파일을 만들기 위해 필요한 파일의 상호 의존 관계를 파악하여 실행 파일을 쉽게 만들 수 있도록 도와주는 시스템
- 컴파일러 지정 및 빌드 과정에서 생성된 불필요한 파일들 제거 가능
- $ make [-f 메이크 파일]

```
<옵션이 없으면 Makefile 또는 makefile 찾아서 빌드 진행>
linuxer@linuxer-PC:~$ make

<옵션으로 특정 메이크 파일 지정 가능>
linuxer@linuxer-PC:~$ make -f makefile
linuxer@linuxer-PC:~$ make -f Makefile
linuxer@linuxer-PC:~$ make -f mybuild.mak
```

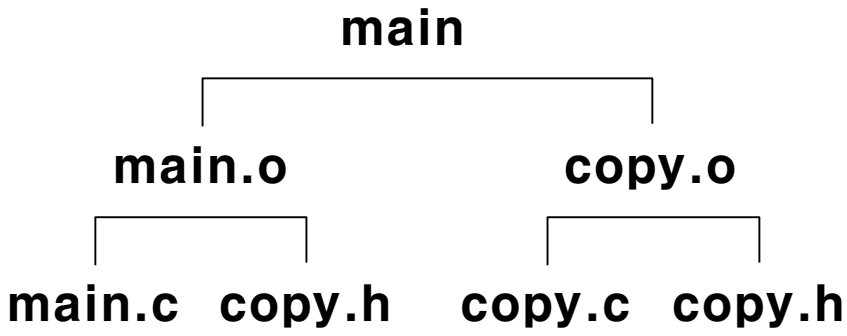
소프트웨어 컴파일

❑ GNU C 프로그래밍 (11)

▪ makefile 형식

```
대상리스트: 의존리스트
명령리스트
```

```
<makefile 예>
main: main.o copy.o
        gcc -o main main.o copy.o
main.o: main.c copy.h
        gcc -c main.c
copy.o: copy.c copy.h
        gcc -c copy.c
```



소프트웨어 컴파일

❑ GNU C 프로그래밍 (12)

▪ makefile을 활용한 컴파일

〈makefile〉

```
TARGET=main
OBJECTS=main.o first.o second.o

${TARGET} : ${OBJECTS}
    gcc -o ${TARGET} ${OBJECTS}

main.o : main.c
    gcc -c main.c
first.o : first.c
    gcc -c first.c
second.o : second.c
    gcc -c second.c
```

<컴파일 및 실행>

```
~$ make
~$ ./main
```

소프트웨어 컴파일

❑ GNU C 프로그래밍 (13)

▪ 일반적인 makefile 형태

```
CC=<컴파일러>
CFLAGS=<컴파일 옵션>
LDFLAGS=<링크 옵션>
LDLIBS=<링크 라이브러리 목록>
OBJS=<Object 파일 목록>
TARGET=<빌드 대상 이름>

all: $(TARGET)

clean:
    rm -f *.o
    rm -f $(TARGET)

$(TARGET): $(OBJS)
$(CC) -o $@ $(OBJS)
```

소스코드 디버깅

GNU debugger (gdb) (1)

- GNU 소프트웨어를 위한 기본 디버거 (1988년, 리처드 스톨먼)
- Ada, Assembly, C/C++, Objective-C 등의 12개 프로그래밍 언어 지원
- 다운로드: http://www.gnu.org/software/gdb/
  - 안정화 버전: 9.1 (2020년 2월 8일)
  - 최신 버전: 10.1 (2020년 10월 24일)
- 특징 및 주요 기능
  - 명령 입력 방식으로 한 줄씩 실행
  - 변수 접근 및 수정 가능
  - 함수 탐색 가능
  - 중단점(breakpoint) 설정 가능
  - 코드 추적(tracing) 가능

소스코드 디버깅

❑ GNU debugger (gdb) (2)

▪ gdb를 사용하기 위한 컴파일

```
<단일 모듈>
linuxer@linuxer-PC:~$ gcc -g -o long  long.c

<다중 모듈>
linuxer@linuxer-PC:~$ gcc -g -o main  main.c  copy.c
```

▪ gdb 실행: gdb [실행파일]

```
<독립 실행>
linuxer@linuxer-PC:~$ gdb

<디버깅 대상 실행파일 함께 로딩>
linuxer@linuxer-PC:~$ gdb long

(gdb) help
```

소스코드 디버깅

❑ gdb 명령 (1)

▪ 소스 보기: l (list)

명령	기능
l [줄번호]	지정된 줄 출력 (관련된 여러 줄 함께 출력됨)
l [파일명:][함수명]	지정된 파일에 있는 함수를 출력
set listsize n	출력되는 줄의 수를 n으로 변경 (기본 n=10)

```
inuxer@linuxer-PC:~/gcc_test/hello$ gdb hello
(gdb) set listsize 3
(gdb) l
1           #include <stdio.h>
2
3           int main()
(gdb) l main
3           int main()
4           {
5                       printf("Hello, World!₩n");
```

소스코드 디버깅

❑ gdb 명령 (2)

- 중단점 설정: b (break), clear, d (delete)

명령	기능
b [파일:]함수명	해당 파일의 함수 시작부분에 중단점 설정
b n	n 번째 줄에 중단점 설정
b +n	현재 줄에서 n 개 줄 이후에 중단점 설정
b -n	현재 줄에서 n 개 줄 이전에 중단점 설정
info b	현재 설정된 중단점 모두 출력
clear n	n 번째 줄에 설정된 중단점 삭제
d	모든 중단점 삭제

```
(gdb) b main
Breakpoint 1 at 0x1149: file hello.c, line 4.
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000001149 in main at hello.c:4
```

소스코드 디버깅

❑ gdb 명령 (3)

▪ 프로그램 실행 및 트레이스

명령	기능
r 인수	명령행 인수를 받아 프로그램 수행 (run)
n	멈춘 지점에서 다음 줄을 수행하고 멈춤 (next)
s	n과 같은 기능(함수 호출 시 함수 내부로 진입) (step)
c	중단점을 만날 때까지 계속 수행 (continue)
u	반복문에서 빠져 나옴
finish	현재 수행하는 함수의 끝으로 이동
return	현재 수행 중인 함수를 빠져 나옴
k	프로그램 수행을 강제로 종료 (kill)
quit	gdb 종료

소스코드 디버깅

❑ gdb 명령 (4)

- 변수 관련 명령: whatis, p (print), display

명령	기능
whatis 변수명	지정한 변수에 관련된 정보 출력
p [변수명]	해당 변수 값 출력
p 파일명[::변수명]	해당 파일의 전역 변수 출력
p [함수명]::[변수명]	해당 함수의 정적 변수 출력
display	변수 추적 목록을 출력
display 변수명	해당 변수를 추적 목록에 추가
info locals	현재 상태의 지역 변수 출력

```
(gdb) p copy
$2 = {void (char *, char *)}
0x555555555189 <copy>
(gdb) info locals
len = 0
index = 2
max = 0
```

```
(gdb) whatis len
type = int
(gdb) display max
1: max = 0
(gdb)
```

소스코드 디버깅

❑ [실습] gdb 활용 (1)

- 아래 코드를 작성하여 컴파일하고 실행해 본다.

```
#include <stdio.h>
#include <string.h>

void main()
{
    int i;
    double j;
    char *bug = NULL;

    for( i = 0; i < 5; i++) {
        j = i/2 + i;
        printf(" j is %lf ₩n", j );
    }
    strcpy(bug, "hi");
    printf("bug is %s ₩n", bug);
}
```

```
$ gcc -g -o test test.c
$ ./test
 j is 0.000000
 j is 1.000000
 j is 3.000000
 j is 4.000000
 j is 6.000000
세그멘테이션 오류 (core dumped)
$
```

소스코드 디버깅

❑ [실습] gdb 활용 (2)

- 오류가 어디에서 발생하는지 gdb로 확인한다.

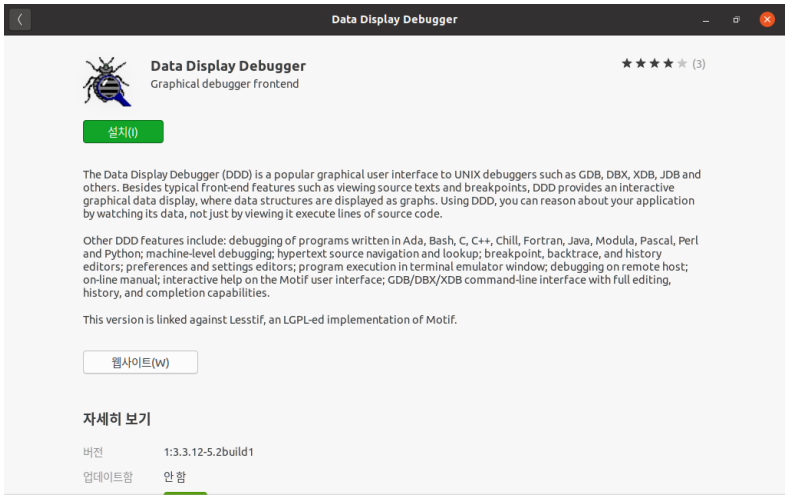
```
$ gdb test
(gdb) l main
4           void main()
5           {
6               int i;
7               double j;
8               char *bug = NULL;
9
10              for( i = 0; i < 5; i++) {
11                      j = i/2 + i;
(gdb) b 10
Breakpoint 1 at 0x115d: file test.c,
line 10.
(gdb) r
Starting program:
Breakpoint 1, main () at test.c:10
10              for( i = 0; i < 5; i++) {
```

```
(gdb) n
11          j = i/2 + i;
(gdb) info locals
i = 0
j = 0
bug = 0x0
(gdb) n
12          printf(" j is %lf ₩n", j );
(gdb) n
 j is 0.000000
...
(gdb) n
Program received signal SIGSEGV,
Segmentation fault.
0x00005555555551aa in main () at
test.c:14
14          strcpy(bug, "hi");
```

소스코드 디버깅

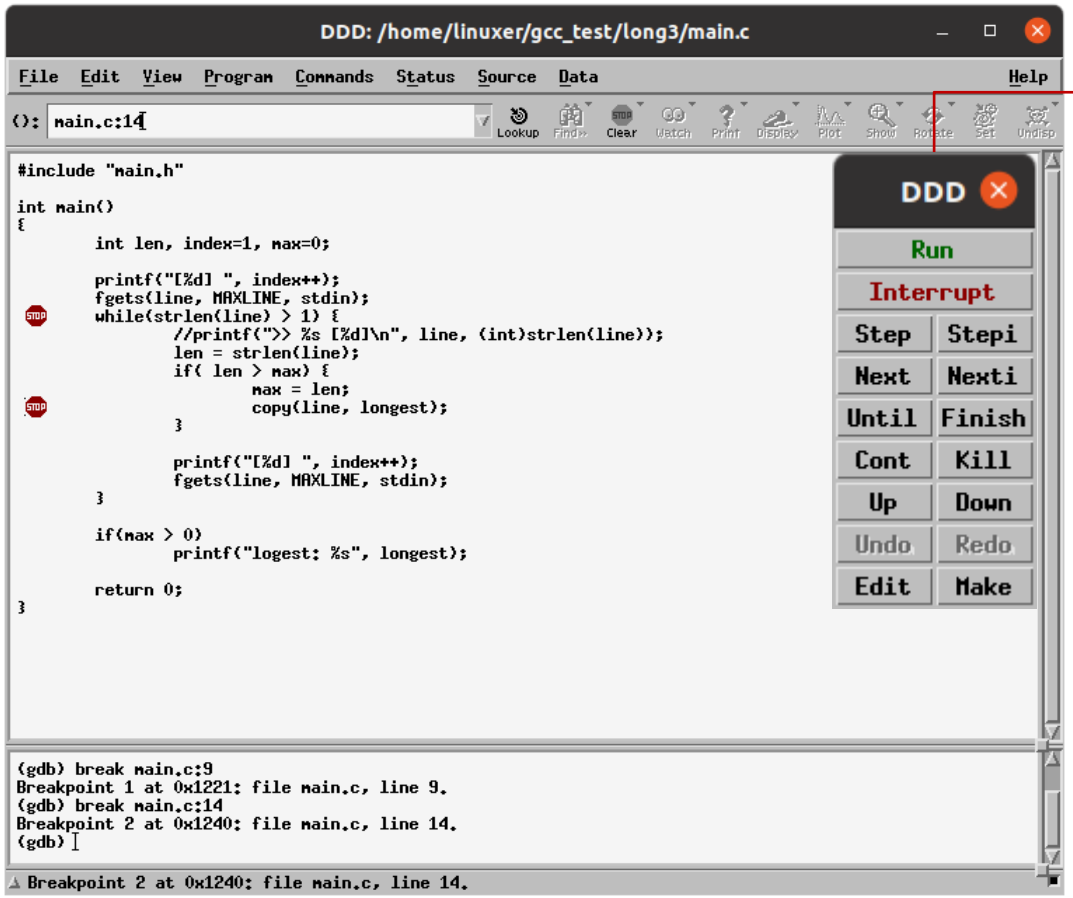
❑ Data Display Debugger(DDD) (1)

- gdb를 위한 그래픽 사용자 인터페이스
- 다운로드: http://www.gnu.org/software/ddd (상업용)
- Ada, Bash, C/C ++, Chill, Fortran, Java, Modula, Pascal, Perl, Python 등으로 작성된 프로그램 디버깅 가능
- 소스 및 중단점 보기와 같은 일반적인 프런트 엔드 기능 제공
- 데이터 구조가 그래프로 표시되는 대화형 그래픽 데이터 디스플레이 제공
- 하이퍼 텍스트 소스 탐색 및 조회
- 중단점, 역 추적 및 기록 편집기
- 기본 설정 및 설정 편집기
- 터미널 에뮬레이터 창에서 프로그램 실행
- 원격 호스트에서 디버깅



소스코드 디버깅

❑ Data Display Debugger(DDD) (2)



중단점
(breakpoint)

명령 도구상자
(Command Tool)

소스코드 표시창
(Source Window)

gdb 명령 입력 콘솔
(GDB console)

소스코드 디버깅

❑ Data Display Debugger(DDD) (3)

▪ 명령 도구 상자

명령	기능
Run	프로그램 디버깅 실행
Interrupt	프로그램 디버깅 일시 중지
Step	한 줄 진행 (함수 내부 진입)
Next	한 줄 진행
Until	중단점까지 진행
Cont.	계속 진행
Kill	프로그램 수행을 강제 종료
Up	호출 한 스택 프레임 선택 및 인쇄
Down	호출 된 스택 프레임 선택 및 인쇄