유닉스 프로그래밍 및 실습

gdb 사용법

단순 디버깅

- fprintf 이용
  - 확인하고자 하는 코드 부분에 fprintf(stderr, "…")를 이용하여 그 지점까지 도달했는지 여부와 관심있는 변수의 값을 확인
  - 여러 유형의 단순한 문제를 확인할 수 있음
  - 그러나 자세히 살펴보기 위해서는 디버깅 툴 필요

```
int main(void)
{
            int count;
            long large_no;
            double real_no;

            init_vars();
fprintf(stderr, "$$$ %d %ld %lf", count, large_no, read_no);

…

fprintf(stderr, "### %d %ld %lf", count, large_no, read_no);

            calc_f1(&count, &large_no);
…
fprintf(stderr, "@@@ %d %ld %lf", count, large_no, read_no);

}
```

gdb(GNU debugger)

- 컴파일 과정에서 반드시 –g 옵션 사용

```
$ gcc -g test.c -o test
```

- gdb 실행

```
$ gdb test
```

- gdb 명령으로 디버깅
  - 주요 지점에 break point 설정
  - 실행시키면 정지점에서 중지
  - 중단된 상태에서 주요 변수의 값 확인
  - 프로그램 실행 계속
  - 한 단계씩 진행
  - 함수 수행 후 결과값 표시
  - 스택 추적

gdb 주요 명령

명령	약어	설명
attach	at	실행 중인 프로세스에 디버거 붙이기
backtrace	bt	스택 추적(stack trace) 출력
break	b	정지점(break point) 설정
clear		정지점 해제
continue	c	프로그램 실행 계속
delete		번호로 정지점 설정 해제 (번호 생략시 전체 해제)
detach		현재 실행 중인 프로세스에서 디버거 떼기
display		실행을 중단할 때마다 표현값 출력
finish		함수 끝까지 실행하고 함수 반환값 표시
help		도움말 출력
jump		특정 주소로 분기하여 실행 계속
list		원시코드 다음 10줄 출력
next	n	프로그램 한 단계 수행. 함수인 경우 함수 내부로 들어가지 않고 다음 행으로 넘어감
print	p	표현값 출력
run	r	현재 프로그램을 처음부터 실행
set		변수값 변경
step	s	프로그램을 다음 소스 코드가 나올 때까지 실행. 함수일 경우 함수 내부로 진입
where	w	스택 추적 출력

gdb 기본 이용 (1)

- 실행

```
[kgu@lily sample2]$ gdb a.out
GNU gdb (GDB) Fedora (7.3.50.20110722-16.fc16)
Copyright (C) 2011 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 "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/kgu/2012U2/ch05/sample2/a.out...done.
(gdb)
```

- list 명령 실행

```
(gdb) list
23                                  line_sum += data;
24                          }
25
26                          return line_sum;
27          }
28
29          /*************************************************************************/
30          int main(void)
31          {
32                          pid_t pid = 1;
(gdb)
```

gdb 기본 이용 (2)

- list 명령 실행

```
(gdb) list
33                      int i;
34                      int read_lines;
35                      int fd1;
36                      int fd2;
37                      int fd3;
38                      char buffer[BUFFER_SIZE];
39                      char result[RESULT_SIZE];
40                      char file_name[FILENAME_SIZE];
41
42                      int line_sum;
(gdb)
```

- break 명령으로 calc_line_sum 루틴에 정지점 설정

```
(gdb) break calc_line_sum
Breakpoint 1 at 0x4009e0: file mp_calc.c, line 18.
```

- break 명령으로 60번째 라인에 정지점 설정

```
(gdb) break 60
Breakpoint 2 at 0x400ab7: file mp_calc.c, line 60.
```

gdb 기본 이용 (3)

- run 명령 실행

```
(gdb) r
Starting program: /home/kgu/2012U2/ch05/sample2/a.out
Breakpoint 2, main () at mp_calc.c:60
60                                  pid = fork();
(gdb)
```

- print 명령으로 현재 스택 내부의 변수 값 등 확인

```
(gdb) print pid
$1 = 1
(gdb) print read_lines
$2 = 32767
(gdb) print fd1
$3 = 7
(gdb) print line_sum
$4 = 51
(gdb)
```

- break 명령으로 23번째 라인에 정지점 설정

```
(gdb) break 23
Breakpoint 3 at 0x400a19: file mp_calc.c, line 23.
```

gdb 기본 이용 (4)

- continue 명령으로 계속 진행

```
(gdb) c
Continuing.
Detaching after fork from child process 8161.

Breakpoint 2, main () at mp_calc.c:60
60                                  pid = fork();
(gdb)
```

- delete 명령으로 정지점 2 삭제

```
(gdb) delete 2
```

- continue 명령으로 계속 진행

```
(gdb) c
Continuing.
Detaching after fork from child process 8384.
Detaching after fork from child process 8385.
Detaching after fork from child process 8386.
Detaching after fork from child process 8387.

Breakpoint 1, calc_line_sum (
    line=0x7fffffffe320 "753 443 469 159 449 819 784 137 803 351\np\r@") at mp_calc.c:18
18                      int line_sum = 0;
(gdb)
```

gdb 기본 이용 (5)

- where 또는 backtrace 명령으로 스택 추적 출력

```
(gdb) where
#0  calc_line_sum (line=0x7fffffffe320 "753 443 469 159 449 819 784 137 803 351\np\r@")
    at mp_calc.c:18
#1  0x0000000000400b43 in main () at mp_calc.c:74
(gdb) backtrace
#0  calc_line_sum (line=0x7fffffffe320 "753 443 469 159 449 819 784 137 803 351\np\r@")
    at mp_calc.c:18
#1  0x0000000000400b43 in main () at mp_calc.c:74
(gdb)
```

- step 또는 next 명령으로 다음 단계 진행

```
(gdb) s
21                      for (i = 0; i < 10; i++) {
(gdb) s
22                              sscanf(&line[i*4], "%d", &data);
(gdb) s

Breakpoint 3, calc_line_sum (
    line=0x7fffffffe320 "753 443 469 159 449 819 784 137 803 351\np\r@") at mp_calc.c:23
23                              line_sum += data;
(gdb) next
21                      for (i = 0; i < 10; i++) {
(gdb)
```

gdb 기본 이용 (6)

- display명령으로 멈출 때마다 출력할 변수 지정

```
(gdb) display i
2: i = 0
(gdb) c
Continuing.
Detaching after fork from child process 9108.

Breakpoint 2, main () at mp_calc.c:60
60                                  pid = fork();
2: i = 1
1: pid = 9108
(gdb)
```

- quit 명령으로 종료

```
(gdb) quit
A debugging session is active.

            Inferior 1 [process 9501] will be killed.

Quit anyway? (y or n) y
[kgu@lily sample2]$
```

기본 사용법

- 관심 있는 함수 마지막에서 확인하는 방법
  - list 함수이름을 수행하여 함수 마지막 줄 번호 찾기
  - break “마지막 줄 번호” 수행하여 정지점 설정
  - run 수행하면 함수 마지막 줄에서 중단
  - print로 함수 내부의 변수와 전역변수 값 확인
- 프로그램이 죽은 경우 정확한 위치 찾기
  - run 실행하면 오류 발생한 지점에서 중단
  - where명령으로 스택 추적
  - list 명령으로 프로그램 소스 확인
- 프로그램 전체를 한 단계씩 실행하면서 확인
  - break main으로 정지점 설정
  - run 실행하면 main() 첫 번째 줄에서 중지
  - step 명령으로 한 단계씩 진행하면서 변수 확인 (print 또는 display 이용)