gdb 사용법

Debugging

Debug라는 말은 bug를 없앤다는 말이다. Bug란, 컴퓨터 프로그램 상의 논리적 오류를 말하며, 이것을 찾아 해결하는 과정이 바로, debugging이다. 초기 컴퓨터들은 실제 벌레가 컴퓨터에 들어가서 오작동을 일으키는 경우가 있었다고 하며, 여기서 'debug'이라는 말이 나왔다 한다.

Debugging을 하는 가장 원초적 방법은 프로그램 소스를 눈으로 따라가며, 머리로 실행시켜 논리적 오류를 찾아내는 것이다. 이것은 프로그래머로 하여금 정말 많은 에너지를 소비하게 하며, 자신이 작성한 프로그램인 경우 선입견으로 인해 오류를 찾아내기 힘든 경우가 많다.

Debugging에 컴퓨터의 도움을 받는 가장 단순한 방법은 프로그램의 중간 중간에 실행 상태를 출력하도록 하는 것이다. 이렇게 함으로써 변수에 잘못된 값이 들어가는지 여부를 사람이 직접 계산 하는 것 보다 좀 더 쉽게 알 수 있다.

Debugging 과정에 컴퓨터의 도움을 받는 가장 강력한 방법은 바로 debugging tool을 이용하는 것이다. Debugging tool들은 프로그램이 실행 중일 때, 변수들의 값이나 수행되는 statement을 보여주며, 필요에 따라 변수 값을 변경하여 실험을 해 볼 수 있도록 해준다.

gdb

GDB 같은 디버거의 목적은 다른 프로그램 수행중에 그 프로그램 내부에서 무슨 일이 일어나고 있는지 보여주거나 프로그램이 고장났을 때 무슨 일이 일어나고 있는지 보여주는 것이다. gdb 로 C, C++, Modula-2 로 짠 프로그램을 디버그할 수 있다. 버그를 잡는 걸 돕기위해 gdb 는 다음과 같은 작업들을 가능하게 한다.

1. 프로그램의 행동에 영향을 줄 수 있는 각종 조건을 설정한 후, 프로그램을 시작한다.
2. 특정 조건을 만나면 프로그램을 정지시킨다.
3. 프로그램이 정지됐을 때 무슨 일이 일어났는지 검사한다.
4. 프로그램 내부 설정을 바꾸어서 버그를 수정함으로써 다른 버그를 계속 찾아나간다.

Starting gdb

gdb를 시작하는 방법은 shell prompt에서 단순히 'gdb'라고 치는 것이다. 하지만, 일반적으로는 뒤에 인자로 실행파일을 주게 되며 필요에 다라 core 파일이나 process id를 주기도 한다.

$ **gdb** [*prog*] [*core*|*pid*]

Debugging하고자 하는 실행파일은 gcc에서 –g 옵션을 주어 compile함으로써 gdb가 필요로 하는 부가 정보들이 추가된 것을 사용해야 제대로 debugging을 수행할 수 있다.

core파일은 프로그램이 비정상적으로 종료될 때, 그때의 시스템의 내부 상태를 그대로 저장해 놓은 것으로, 이 파일을 인자로 주면 실행파일이 비정상적으로 종료된 곳이 소스코드의 어느 부분인지를 표시해 준다.

이미 실행중인 프로그램을 debugging하려면 해당 process의 id를 주면 되는데, 이 때 주의할 것은 pid와 같은 이름의 파일이 있을 경우 gdb가 core파일로 여기게 되는 점이다.

일단 gdb가 실행되면 gdb prompt가 표시되며, 여기에 gdb command를 주어 debugging을 진행 할 수 있게 된다.

gdb commands

· **list** [[*file***:**]*n*|*func*]

source file의 내용을 10줄씩 보여준다. 행 번호를 지정하면*n*번 행의 주변 10라인이 출력된다. 함수이름을 지정하면 그 함수의 내용이 출력된다. 두 개의 행 번호를 ‘,’로 분리해서 쓰면 첫 행 번호에서 시작해서 두 번째 행 번호까지의 소스가 출력된다.

· **help** [*name*]

*name*으로 지정된 gdb의 command나 관련 정보를 보여준다.

· **break** [[*file***:**]*n*|*function*] <-> **d (del)**

지정된 file의 *n*번 행 또는 *function*에 breakpoint를 설정한다.

· **clear** [[*file***:**]*n*|*function*]

지정된 *file*의 *n*번 행 또는 *function*에 설정된 breakpoint를 제거한다.

· **run** [*arglist*]

program을 시작한다.

· **print** *expr*

*expr*의 값을 한번 보여준다.

· **disp** *expr* <-> **undisp**

*expr*의 값을 실행되는 동안 계속 보여준다.

· **c** [*n*] (또는 **continue**)

breakpoint등에 의해 멈춰진 프로그램의 실행을 계속한다. n이 지정될 경우, 이후 n-1번은 breakpoint는 무시하고, n번째 breakpoint에 걸릴 때 멈춘다.

· **next** [*n*]

멈춰진 프로그램에서 프로그램의 다음 *n*(default=1)개의 문장을 실행하고 다음 번에 실행할 문장을 출력한다. 함수일 경우 함수 전체를 실행한다.

· **step** [*n*]

멈춰진 프로그램에서 프로그램의 다음 *n*(default=1)개의 문장을 실행하고 다음 번에 실행할 문장을 출력한다. 함수일 경우, 함수 내부로 들어가 한 문장씩 실행한다.

· **bt** (또는 **backtrace**)

program에서 function들이 불린 순서의 program stack을 보여준다.

· **up**

현재의 함수를 호출한 함수를 보여준다.

· **down**

현재의 함수가 호출한 함수를 보여준다.

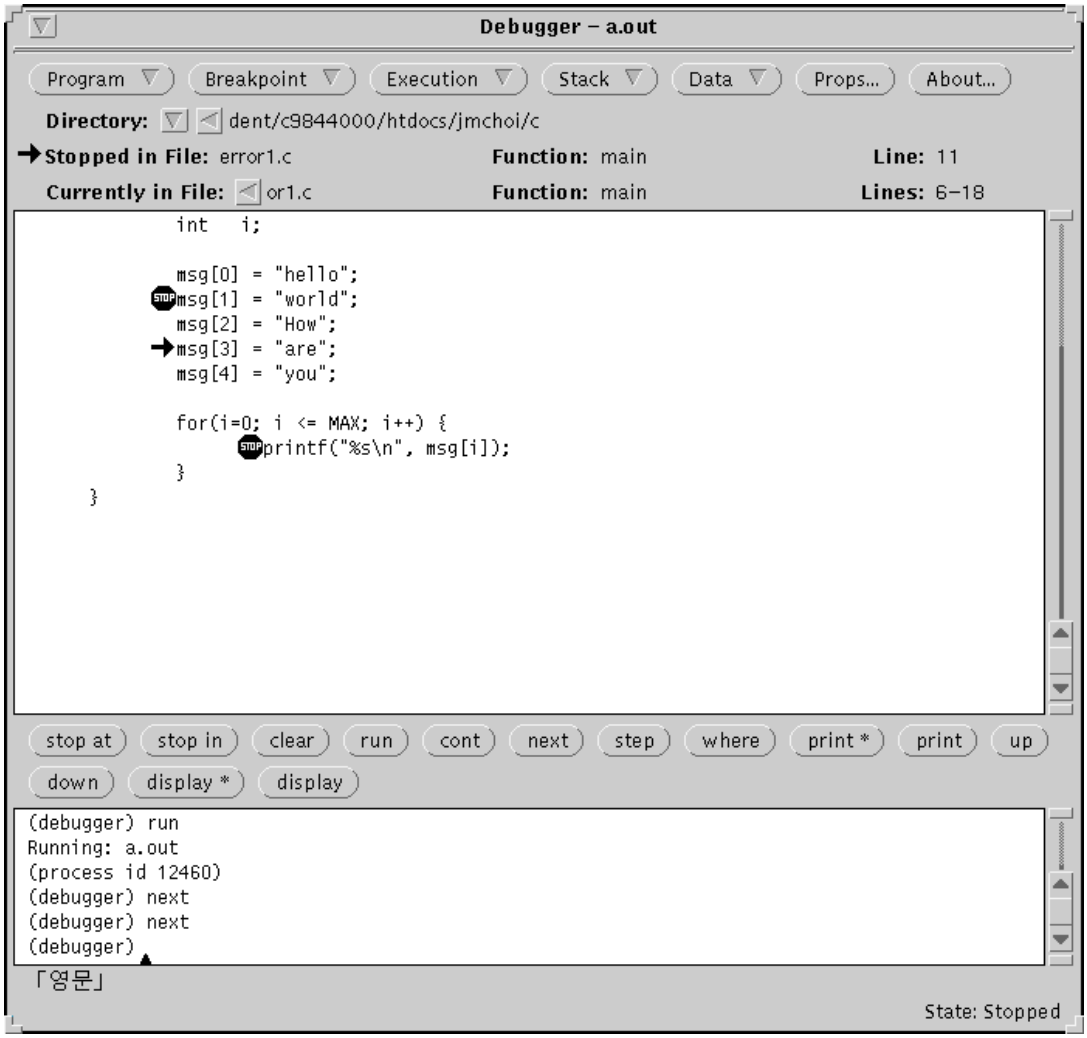
· **ret** (또는 **return**)

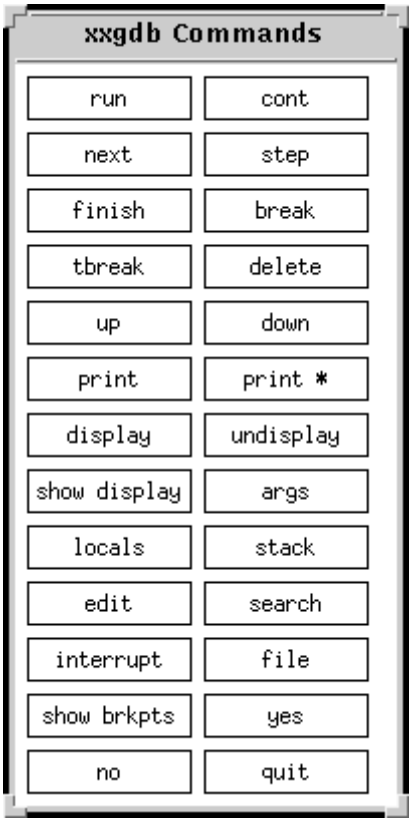
현재 함수를 반환하며 빠져나간다.

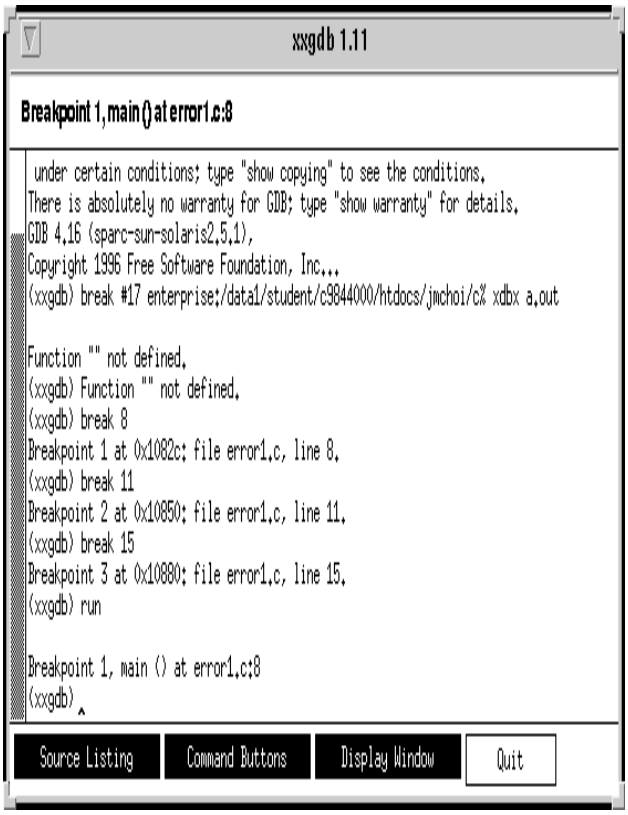
· **quit**

gdb를 종료한다.

xdbx


Debugger – a.out
Program
Breakpoint
Execution
Stack
Data
Props...
About...
Directory: dent/c9844000/htdocs/jmchoi/c
Stopped in File: error1.c
Function: main
Line: 11
Currently in File: or1.c
Function: main
Lines: 6–18
int i;
msg[0] = "hello";
msg[1] = "world";
msg[2] = "How";
msg[3] = "are";
msg[4] = "you";
for(i=0; i <= MAX; i++) {
printf("%s\n", msg[i]);
}
}
stop at
stop in
clear
run
cont
next
step
where
print *
print
up
down
display *
display
(debugger) run
Running: a.out
(process id 12460)
(debugger) next
(debugger) next
(debugger)
「영문」
State: Stopped





실행 예제

자신의 submit directory에 복사되어 있는 bugshome.c 파일을 gdb를 사용하여 debugging해 보자. 이 프로그램은 factorial과 power를 계산해 주는 함수를 구현하고 이를 사용하는 프로그램이다.

먼저 –g option으로 gcc를 사용해 컴파일 한 후, 일단 실행 시켜 보면 대충 무엇이 문제인지를 보게 된다. 물론 눈으로 따라가 보아도 쉽게 버그를 알아낼 수 있을 정도의 단순한 code이지만, 우리의 목적은 gdb를 사용하는 것이라는 것을 기억하자. 우선, power함수에서 결과가 나오지 않으므로 power함수를 breakpoint로 지정하여 시작해보자.

```
shell/tmt21 ~/tmp/bugshome {217} gdb a.out
GNU gdb 5.0
Copyright 2000 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.    Type "show warranty" for details.
```

```
This GDB was configured as "hppa2.0-hp-hpux10.20"...
(gdb) break power
Breakpoint 1 at 0x32a8: file bugshome.c, line 16.
(gdb) run
Starting program: /afs/postech.ac.kr/home/std/tmt21/tmp/bugshome/a.out
[NOTE] Your password will expire 119 days later.
warning: Unable to find __d_pid symbol in object file.
warning: Suggest linking with /opt/langtools/lib/end.o.
warning: GDB will be unable to track shl_load/shl_unload calls
11! = 39916800
This is f3().
Breakpoint 1, power (b=11, e=7) at bugshome.c:16
16              return b * power(b, e-1);
(gdb) c
Continuing.
Breakpoint 1, power (b=11, e=6) at bugshome.c:16
16              return b * power(b, e-1);
(gdb) c 3
Will ignore next 2 crossings of breakpoint 1.    Continuing.
Breakpoint 1, power (b=11, e=3) at bugshome.c:16
16              return b * power(b, e-1);
(gdb) c
Continuing.
Breakpoint 1, power (b=11, e=2) at bugshome.c:16
16              return b * power(b, e-1);
(gdb) c
Continuing.
Breakpoint 1, power (b=11, e=1) at bugshome.c:16
16              return b * power(b, e-1);
(gdb) c
Continuing.
Breakpoint 1, power (b=11, e=0) at bugshome.c:16
16              return b * power(b, e-1);
(gdb) ret 1
Make power return now? (y or n) y
#0    0x32c0 in power (b=11, e=1)
```

```
at bugshome.c:16
16              return b * power(b, e-1);
(gdb) l
11          }
12
13          /* returns b^e , e>=0 */
14          int power(int b, int e)
15          {
16              return b * power(b, e-1);
17          }
18
19          void f3()
20          {
(gdb) l
21              printf("This is f3().\n");
22          }
23
24          void f2()
25          {
26              f3();
27              printf("11^7 = %d\n", power(11, 7));
28          }
29
30          void f1()
(gdb) break 28
Breakpoint 2 at 0x3388: file bugshome.c, line 28.
(gdb) c
Continuing.
11^7 = 19487171
Breakpoint 2, f2 () at bugshome.c:28
28          }
(gdb) n
f1 () at bugshome.c:34
34          }
(gdb) n
main () at bugshome.c:40
```

```
40                  printf("0! = %d\n", factorial(0));
(gdb) s
factorial (n=0) at bugshome.c:7
7                   if(n == 1)
(gdb) l
2               #include <stdlib.h>
3
4               /* returns n * (n-1) * ... * 1    */
5               int factorial(int n)
6               {
7                   if(n == 1)
8                       return 1;
9
10                  return n * factorial(n - 1);
11              }
(gdb) ret 1
Make factorial return now? (y or n) y
#0    0x342c in main () at bugshome.c:40
40                  printf("0! = %d\n", factorial(0));
(gdb) c
Continuing.
0! = 1
Program exited normally.
(gdb) quit
```

References

http://www.delorie.com/gnu/docs/gdb/gdb_toc.html

http://sources.redhat.com/gdb/

http://www.cs.umd.edu/class/sum2002/cmsc420/gdb.html