이전포스트

[Tip] gdb의 간단한 사용법.

2010. 2. 23. 16:48 *by* freemmer

gdb는 Linux에서 사용되는 디버깅 툴 입니다.
linux에서 이 툴을 이용해서 디버깅을 좀 더 수월하게 진행할 수 있습니다.
다음은 간단한 사용법입니다.

01. 우선 컴파일 시에 디버깅 옵션을 주어 컴파일 합니다.

```
# g++ -g -o <App Name> <Source File>
```

-g : 디버깅 옵션 (최적화 옵션인 -O는 주지 않는다)
-o : 출력할 프로그램의 이름.
<App Name>의 이름대로 프로그램이 만들어진다.
<Source File>에 *.c 혹은 *.cpp와 같이 소스코드 파일명이 들어간다.

02. gdb를 사용합니다.

gdb를 사용하는 방법은 크게 3가지로 있습니다.

```
1) # gdb ./<App Name>
2) # gdb ./<App Name> ./<Core Name>
```

```
3) # gdb ./<App Name> ./<pid Name>
```

가장 일반적으로 사용되는 것은 1번과 2번 입니다. 1,2번을 간단히 설명하자면.

1번 사용법은 gdb를 사용하여 프로그램을 실행시켜 디버깅하는 방법으로 VC등의 윈도우 디버깅을 사용하는 방법과 유사합니다.

2번 사용법은 프로그램이 비정상 종료할 때 코어 파일을 만들게 설정한 경우, 해당 코어 파일을 확인하여 종료된 원인을 파악할때 사용하는 방법으로 Dr. Watson에서 만들어주는 덤프파일과 같다고 할 수 있습니다.
참고로, 코어 파일을 만들도록 했는지 확인하는 방법과 설정 방법은 다음과 같습니다.

```
# ulimit -c            //현재 설정된 코어 파일 크기를 확인 합니다.
# ulimit -c <Size>  //<Size>에 1024와 같이 직접 크기를 지정하거나 unlimited로 무제한으로 설정할 수 있습니다.
```

3번 사용법은 <pid Name>에 해당하는 프로세스에 연결하여 디버깅을 합니다. 이 방법은 실행중인 프로세스를 디버깅할때 사용됩니다.

03. gdb 명령어.

여기에서는 비교적 자주 사용하는 명령어만 기술하도록 하겠습니다.

run (r)

프로그램을 시작합니다. 이때, 인자값 (arg)를 포함 할 수도 있습니다. 또한, '<' , '>', '<<', '>>'와 같은 입출력 방향 재지정 기호는 물론 쉘의 사용까지도 확장 사용할 수 있습니다.

quit

프로그램을 종료합니다.

kill (k)

프로그램을 종료합니다.

break (b)

중단점을 설정합니다. b <function / file:function / file:line numbe r / *memory adress>등의 인수를 전달하여 설정 할 수 있습니다. 간략햔 예는 다음과 같습니다.

```
(gdb) b main.c:func // main.c의 func함수에 브레이크 설정.
(gdb) b CMain::func // CMain클래스의 func맴버함수에 브레이크 설정.
(gdb) b main.c:10    // main.c의 10번 라인에 브레이크 설정.
(gdb) b +2           // 현재행에서 2라인 이후의 지점에 브레이크 설정.
(gdb) b *0x00000001 // 0x00000001주소에 브레이크 설정.
```

info

각종 정보를 디스플레이 합니다. 더욱 자세한 정보는 help를 통해 확인 할 수 있습니다.

```
(gdb) info b           // 현재 설정된 브레이크 포인트의 정보를 보여준다.
(gdb) info locals      // 현재 상태의 지역변수 정보을 보여준다.
(gdb) info variables   // 현재 상태의 전역변수 정보를 보여준다.
(gdb) info registers   // 레지스터 정보를 보여준다.
(gdb) info frame       // 현재 스택 프레임 정보 보여준다.
(gdb) info args        // 현재 스택 프레임의 함수가 호출될때 인자를 보여준다.
(gdb) info catch       // 현재 스택 프레임의 함수내에 예외 핸들러를 보여준다.
(gdb) info signals     // 보낼 수 있는 시그널의 종류를 보여준다.
(gdb) info set         // 변경 가능한 환경설정 정보를 보여준다.
```

```
(gdb) info functions // 함수 리스트를 보여준다.
```

condition

브레이크 포인트에 조건을 설정합니다.

```
(gdb) condition 1 bCheck == true // 1번 브레이크 포인트 (info b로 확인)에 bCheck가 true일 때 동작하도록 설정.
```

clear (cl)

브레이크 포인트를 삭제합니다.

```
(gdb) cl 1                  // 1번 브레이크 포인트 삭제.
(gdb) cl CMain::func // CMain클래스의 func맴버함수의 브레이크 포인트 삭제.
(gdb) cl                    // 모든 브레이크 포인트 삭제.
```

enable / disable

브레이크 포인트를 활성화 하거나 비활성화 합니다.

```
(gdb) enable 1  // 1번 브레이크 포인트를 활성화 한다.
(gdb) disable 1 // 1번 브레이크 포인트를 비활성화 한다.
```

step (s) / next (n)

브레이크 포인트에 걸린 후 프로그램을 한 라인 실행시킨다.

```
(gdb) s       // 현재 행을 수행한다. (함수의 경우 내부로 진입한다)
(gdb) s 10    // step 명령을 10번 수행한다.
(gdb) n       // 현재 행을 수행한다. (함수의 경우 진입하지 않고 함수를 실행한 후 넘어간다)
(gdb) n 10    // next 명령을 10번 수행한다.
```

continue (c) / unitl (u)

```
(gdb) c   // 다음 브레이크를 만날때 까지 계속 진행한다.
(gdb) u   // for문에서 빠져나와 다음 브레이크까지 진행한다.
```

finish / return

```
(gdb) finish          // 현재 함수를 실행하고 빠져나간다.
(gdb) return          // 현재 함수를 실행하지 않고 빠져나간다.
(gdb) return false // 함수를 빠져나갈때 리턴값을 false로 준다.
```

watch

```
(gdb) watch bCheck   // bCheck의 값이 변경될 때마다 브레이크가 걸리도록 한다.
```

print (p)

```
(gdb) p bCheck        // bCheck 값을 확인한다.
(gdb) p func          // func함수의 주소를 확인한다.
(gdb) p tVal          // tVal 구조체의 주소를 확인한다. (*의 포인터로 값을 확인할 수 있다)
// 아래와 같이 포인터, 캐스팅등이 가능하다.
// p[/출력형식] <변수명/함수명> [@배열크기]
(gdb) p &eax          // eax레지스터 값을 확인한다.
(gdb) p *pByte@8      // 8크기 배열로 가져와 확인한다.
(gdb) p/t  val        // 변수를 2진수로 출력.
(gdb) p/o val         // 변수를 8진수로 출력.
(gdb) p/d val         // 변수를 signed 10진수로 출력.
(gdb) p/u val         // 변수를 unsigned 10진수로 출력.
(gdb) p/x val         // 변수를 16진수로 출력.
(gdb) p/a addr        // addr주소와 가장 가까운 심볼의 오프셋 출력. (예: main+10)
```

display / undisplay

```
(gdb) display pVal        // pVal을 매번 화면에 표시한다.
(gdb) undisplay 1         // 1번 디스플레이 설정을 지운다.
(gdb) enable display 1    // 1번 디스플레이를 활성화 한다.
(gdb) disable display 1   // 1번 디스플레이를 비활성화 한다.
```

frame / up / down

```
(gdb) frame [N]   // n번 스택 프레임으로 이동.
(gdb) up [N]      // 상위 프레임으로 이동.
(gdb) down[N]     // 하위 프레임으로 이동.
```

메모리 상태 검사

x[/범위][출력형식][범위의 단위] 형식으로 메모리의 특정 범위를 검사할 수 있습니다.

이때, [/범위]의 기본단위는 word로 4바이트 입니다. b(1바이트), h(2바이트), w(4바이트), g(8바이트)의 단위를 지정할 수 있습니다.

더욱 자세한 정보는 help를 통해 확인 할 수 있습니다.

```
(gdb) x/10d func    // func함수의 주소부터 40바이트를 signed 10진수로 출력한다.
```

signal

```
(gdb) signal SIGKILL    // KILL 시그널을 보낸다.
```

disass

```
(gdb) disass func
```

call

```
(gdb) call func(123)    // func함수를 호출하고 반환값 출력.
```

set

메모리의 특정 영역에 값을 설정합니다.

set {타입}[주소] = [값] 형식으로 작성되며, 예는 다음과 같습니다.

```
(gdb) set {int} 0x0000001 = 100    // 해당 주소에 100을 세팅한다.
```

backtrace (bt)

프로그램의 스텍을 보여줍니다.

위의 사용법은 간략하게 작성되어 있으므로 help로 정보를 얻기를 권장합니다.

TAG C++ , linux , Programming , Tip

이전포스트 카테고리의 다른글

[Tip] Doxygen을 사용하여 소스코드에서 레퍼런스 문서를 자동으로 만들자.

[Tip] SyntaxHighlighter - 블로그에 소스 코드를 보기 좋게 삽입해보자

[Tip] gdb의 간단한 사용법.

내 생애 처음이었던 해외여행~~

[Tip] Borland C++에서 Window를 포함하는 DLL을 만들어VC++에서 사용하기. (그 반대의 ...

내가 직업을 프로그래머로 선택한 이유...

댓글 3 , 엮인글 1

댓글을 남겨주세요

...!?
2016.03.29 22:03

비밀댓글입니다

↘답글달기

...!?
2016.03.29 22:03

비밀댓글입니다

↘답글달기

tanta
2016.07.07 12:05

감사합니다 잘보고 가요

↘답글달기

이전 116 / 119 다음

홈으로 방명록 로그아웃 맨위로