EDDI
Electronic Design
Development Institute

에디로봇아카데미

임베디드 마스터 Lv1 과정

Ch1.

제 3기

2021. 12. 16

여건

CONTENTS

EDDI
Electronics Design Development Institute

* 변수와 메모리

* 디버거 (debugger)

* gdb 실습

변수와 메모리

* 변수란

특정한 데이터 타입을 가지고 있으며 정보를 저장할 수 있는 메모리 공간
컴퓨터에서 사용하는 모든 정보를 메모리에 적재 되어진다. PC에서 메모리라 부르는 것이 DRAM에 해당되며 DRAM에 데이터가 올라가는 것 자체가 변수에 해당함

* 데이터 타입

```
yeo@yeo-15Z980-GA50K:~/EmbeddedMasterLv1/3기/GYY/c/ch1$ vi var_type.c
yeo@yeo-15Z980-GA50K:~/EmbeddedMasterLv1/3기/GYY/c/ch1$ gcc var_type.c
yeo@yeo-15Z980-GA50K:~/EmbeddedMasterLv1/3기/GYY/c/ch1$ ls
a.out  var_type.c
yeo@yeo-15Z980-GA50K:~/EmbeddedMasterLv1/3기/GYY/c/ch1$ rm -f a.out
yeo@yeo-15Z980-GA50K:~/EmbeddedMasterLv1/3기/GYY/c/ch1$ gcc -o var_type var_type.c
yeo@yeo-15Z980-GA50K:~/EmbeddedMasterLv1/3기/GYY/c/ch1$ ./var_type
변수 타입 char의 할당되는 메모리 크기는 1
변수 타입 short의 할당되는 메모리 크기는 2
변수 타입 int의 할당되는 메모리 크기는 4
변수 타입 float의 할당되는 메모리 크기는 4
변수 타입 double의 할당되는 메모리 크기는 8
변수 타입 long double의 할당되는 메모리 크기는 16
```

디버거 (debugger)

* 컴퓨터 프로그램의 정확성이나 논리적인 오류(버그)를 찾아내는 과정

* 디버거의 기본 기능
  - 프로그램 코드의 단계적 실행 (step execution)
  - 설정된 중단점까지 실행 (break point)
  - 메모리나 레지스터의 값을 확인 (memory examination)

Gdb 실습

1. $gcc -o mem_addr mem_addr.c -g
// 우측 소스코드의 실행 파일 생성 및 디버깅 옵션 부여

2. $gdb mem_addr

3. (gdb)b *main
// 중단점을 지정하면 run 명령을 통해 프로그램을 실행
시킬 때마다 해당 함수가 시작되는 지점에서 멈추게 됨
// b ***메모리주소,*** b *offset+n ***으로도 중단점 지정 가능***

```c
#include <stdio.h>

int main(void)
{
        int var = 7;
        printf("var = %d\n", var);
        printf("var mem addr = 0x%x\n", &var);
        var = 10;
        return 0;
}
```

```
(gdb) b *main
Breakpoint 1 at 0x1169: file mem_addr.c, line 4.
(gdb) b *0x0000000000001169
Note: breakpoint 1 also set at pc 0x1169.
Breakpoint 2 at 0x1169: file mem_addr.c, line 4.
(gdb) b *main+0
Note: breakpoints 1 and 2 also set at pc 0x1169.
Breakpoint 3 at 0x1169: file mem_addr.c, line 4.
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000001169 in main at mem_addr.c:4
2       breakpoint     keep y   0x0000000000001169 in main at mem_addr.c:4
3       breakpoint     keep y   0x0000000000001169 in main at mem_addr.c:4
(gdb) d 2
(gdb) d 3
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000001169 in main at mem_addr.c:4
(gdb) r
Starting program: /home/yeo/EnbeddedMasterLv1/3기/GYY/review/ch1/mem_addr

Breakpoint 1, main () at mem_addr.c:4
warning: Source file is more recent than executable.
4       {
```

Gdb 실습

4. (gdb) disas main
// 함수의 어셈블리 코드를 볼 수 있음
// 1번째 행, 메모리 주소
// <+0> 과 같은 숫자는 해당 함수를 기준으로
몇 번째에 있는 명령어인지 나타냄
// 2번째 행, 어셈블리 언어에서 사용되는 명령어
// 3번째 행, 레지스터

// 처음에 breakpoint로 지정해준
main함수의 첫 번째 줄에 화살표(=>)가
가리키고 있다.
현재 위치가 그 곳이라는 뜻

```
(gdb) disas main
Dump of assembler code for function main:
=> 0x0000555555555169 <+0>:     endbr64
   0x000055555555516d <+4>:     push   %rbp
   0x000055555555516e <+5>:     mov    %rsp,%rbp
   0x0000555555555171 <+8>:     sub    $0x10,%rsp
   0x0000555555555175 <+12>:    mov    %fs:0x28,%rax
   0x000055555555517e <+21>:    mov    %rax,-0x8(%rbp)
   0x0000555555555182 <+25>:    xor    %eax,%eax
   0x0000555555555184 <+27>:    movl   $0x7,-0xc(%rbp)
   0x000055555555518b <+34>:    mov    -0xc(%rbp),%eax
   0x000055555555518e <+37>:    mov    %eax,%esi
   0x0000555555555190 <+39>:    lea    0xe6d(%rip),%rdi        # 0x555555556004
   0x0000555555555197 <+46>:    mov    $0x0,%eax
   0x000055555555519c <+51>:    callq  0x555555555070 <printf@plt>
   0x00005555555551a1 <+56>:    lea    -0xc(%rbp),%rax
   0x00005555555551a5 <+60>:    mov    %rax,%rsi
   0x00005555555551a8 <+63>:    lea    0xe5f(%rip),%rdi        # 0x55555555600e
   0x00005555555551af <+70>:    mov    $0x0,%eax
   0x00005555555551b4 <+75>:    callq  0x555555555070 <printf@plt>
   0x00005555555551b9 <+80>:    movl   $0xa,-0xc(%rbp)
   0x00005555555551c0 <+87>:    mov    $0x0,%eax
   0x00005555555551c5 <+92>:    mov    -0x8(%rbp),%rdx
   0x00005555555551c9 <+96>:    xor    %fs:0x28,%rdx
   0x00005555555551d2 <+105>:   je     0x5555555551d9 <main+112>
   0x00005555555551d4 <+107>:   callq  0x555555555060 <__stack_chk_fail@plt>
   0x00005555555551d9 <+112>:   leaveq
   0x00005555555551da <+113>:   retq
End of assembler dump.
```

Gdb 실습

5. (gdb) i r $rbp $rsp
// 특정 레지스터의 정보를 출력

6. (gdb) ni
// ni와 si 명령어는 기계어 instruction 단위로 실행한다.
// s와 n은 소스 줄 단위로 실행한다.
// s(si)는 함수 내부를 타고 들어갈 수 있고, n(ni)는 함수가 나오면 건너뛴다.

```
(gdb) i r $rbp $rsp
rbp            0x0                 0x0
rsp            0x7fffffffdf98      0x7fffffffdf98
(gdb) ni
0x000055555555510d      4       {
(gdb) ni
0x000055555555510e      4       {
(gdb) ni
0x0000555555555171      4       {
(gdb) disas main
Dump of assembler code for function main:
   0x0000555555555169 <+0>:     endbr64
   0x000055555555516d <+4>:     push   %rbp
   0x000055555555516e <+5>:     mov    %rsp,%rbp
=> 0x0000555555555171 <+8>:     sub    $0x10,%rsp
   0x0000555555555175 <+12>:    mov    %fs:0x28,%rax
   0x000055555555517e <+21>:    mov    %rax,-0x8(%rbp)
   0x0000555555555182 <+25>:    xor    %eax,%eax
```

7. (gdb) x/4wx $rbp, (gdb)x/4wx $rsp
// 특정 레지스터의 메모리 값 확인

// (gdb) x/b x/h x/w
// 각각 1, 2, 4 바이트 출력
// (gdb) x/x x/u
// 각각 16, 10진수로 출력
// (gdb)wx (gdb)x/4b
// info reg : 모든 레지스터들의 정보를 출력

```
(gdb) i r $rbp $rsp
rbp            0x7fffffffdf90      0x7fffffffdf90
rsp            0x7fffffffdf90      0x7fffffffdf90
(gdb) x/4wx $rbp
0x7fffffffdf90: 0x00000000      0x00000000      0xf7deb0b3      0x00007fff
(gdb) x/4wx $rsp
0x7fffffffdf90: 0x00000000      0x00000000      0xf7deb0b3      0x00007fff
```

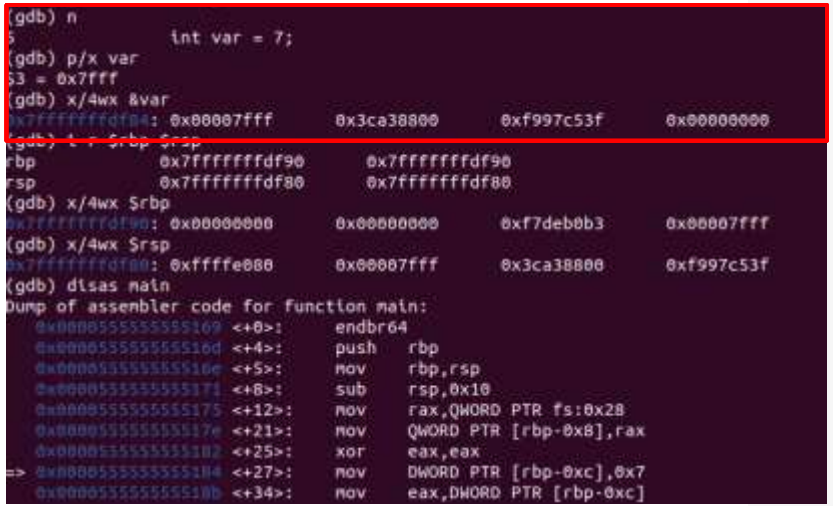
//

Gdb 실습

8. (gdb) n
// 소스코드 한 줄 실행

9. (gdb)p/x var, (gdb)x/4wx &var
// 변수 var의 값을 확인

10. (gdb) n
// 다시 소스코드 한 줄 실행 후,
// 변수 var 값의 변화를 확인




```
(gdb) n
6               printf("var = %d\n", var);
(gdb) p/x var
$4 = 0x7
(gdb) x/4wx &var
0x7fffffffdf84: 0x00000007      0x3ca38800      0xf997c53f      0x00000000
```