2주차 - GDB를 활용하여 쉘 실행하기

학부: 컴퓨터공학부　학번: 2020136129　이름: 최수연

1. 디버깅할 소스 코드 생성

1) 실행 파일 생성

```
vim debug.c
```

```
#include <stdio.h>

void dont_call(void)
{
    printf("Good job~!\n");
}

void should_call(char *str)
{
    printf("%s\n", str);
}

int main(int argc, char **argv)
{
    void (*func)(char *);

    func = should_call;
```

```
    func("no way\n");

    return 0;
}
```

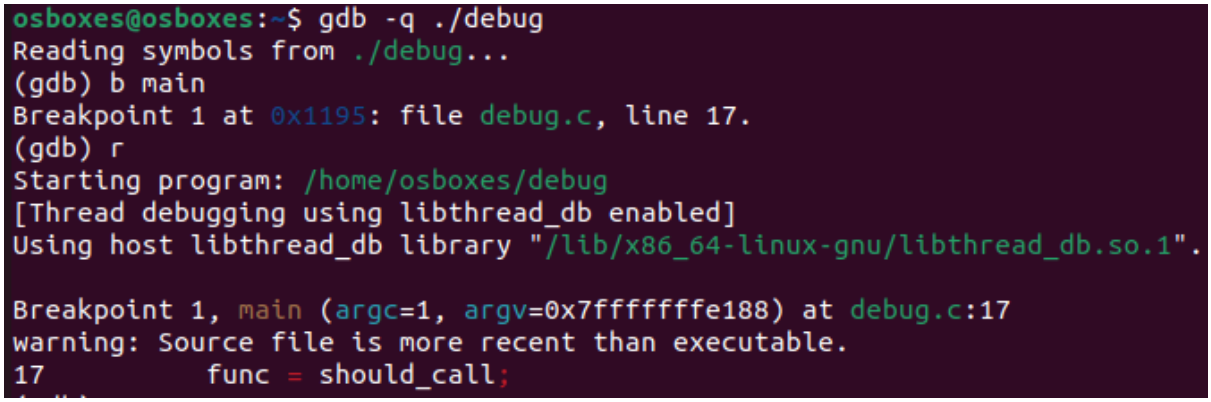
2) 컴파일

```
gcc debug.c -o debug -g
```

2. GDB 사용하여 쉘 실행

1) GDB를 사용하여 debug 파일 실행

```
gdb -q ./debug  # debug라는 이름을 가진 바이너리 파일을 gdb로 실행
b main          # main 함수에 breakpoint 걸기
r               # 프로그램 실행(run)
```



2) func을 system 함수로 변경

```
n # 다음 코드 func = should_call; 호출
print func # func 변수 값 출력
set var func = system # func을 system 함수로 변경
print func # 변경된 func 변수 값 확인
p system # 위의 변경된 func 변수 값의 주소가 system 함수의 주소와 동일한지 확인
```

> #### *. 명령어 n
>
> ‘set var func = system’ 호출 후에 ‘func = should_call;’ 호출되면,
>
> ‘func’이 ‘should_call’로 덮어씌워지므로,

> 명령어 n을 통해 다음 코드인 'func = should_call;' 부분을 먼저 실행해야 함

```
17          func = should_call;
(gdb) n
18          func("no way\n");
(gdb) print func
$1 = (void (*)(char *)) 0x555555555163 <should_call>
(gdb) set var func = system
(gdb) print func
$2 = (void (*)(char *)) 0x7ffff7c50d60 <__libc_system>
(gdb) p system
$3 = {int (const char *)} 0x7ffff7c50d60 <__libc_system>
```

3) main 함수 디버깅

```
disas main # main 함수를 디버깅하여 어셈블리어 출력
```

```
(gdb) disas main
Dump of assembler code for function main:
   0x0000555555555182 <+0>:     endbr64
   0x0000555555555186 <+4>:     push   %rbp
   0x0000555555555187 <+5>:     mov    %rsp,%rbp
   0x000055555555518a <+8>:     sub    $0x20,%rsp
   0x000055555555518e <+12>:    mov    %edi,-0x14(%rbp)
   0x0000555555555191 <+15>:    mov    %rsi,-0x20(%rbp)
   0x0000555555555195 <+19>:    lea    -0x39(%rip),%rax        # 0x555555555163 <should_call>
   0x000055555555519c <+26>:    mov    %rax,-0x8(%rbp)
=> 0x00005555555551a0 <+30>:    mov    -0x8(%rbp),%rax
   0x00005555555551a4 <+34>:    lea    0xe64(%rip),%rdx        # 0x55555555600f
   0x00005555555551ab <+41>:    mov    %rdx,%rdi
   0x00005555555551ae <+44>:    call   *%rax
   0x00005555555551b0 <+46>:    mov    $0x0,%eax
   0x00005555555551b5 <+51>:    leave
   0x00005555555551b6 <+52>:    ret
End of assembler dump.
```

`%rdi`: 함수의 첫 번째 인자 전달

`%rdx`: 함수의 두 번째 인자 전달

`%rax`: 반환하는 값 저장

`mov`: 값 복사 명령어

`call`: 함수 호출

`lea`: 주소 연산 수행 및 결과 주소를 레지스터에 저장

> 아래 명령어를 통해 `%rdx` 레지스터의 값을 `%rdi` 레지스터로 복사하기 때문에,

> `%rdx` 또는 `%rdi` 둘 중 어떤 레지스터 주소로 바꾸어도 상관 없음

```
mov %rdx,%rdi
```

(1) 명령어를 한 줄씩 실행하여 레지스터 문자열 변경

```
x/s $rdi  # $rdi 레지스터의 문자열 출력
x/s $rdx  # $rdx 레지스터의 문자열 출력
ni        # 다음 명령어 실행
```

```
(gdb) x/s $rdi
0x1:    <error: Cannot access memory at address 0x1>
(gdb) x/s $rdx
0x7fffffffe198: "\231\344\377\377\377\177"
(gdb) ni
0x000055555555551a4      18          func("no way\n");
(gdb) x/s $rdi
0x1:    <error: Cannot access memory at address 0x1>
(gdb) x/s $rdx
0x7fffffffe198: "\231\344\377\377\377\177"
(gdb) ni
0x00005555555551ab       18          func("no way\n");
(gdb) x/s $rdi
0x1:    <error: Cannot access memory at address 0x1>
(gdb) x/s $rdx
0x55555555600f: "no way\n"
(gdb) ni
0x00005555555551ae       18          func("no way\n");
(gdb) x/s $rdi
0x55555555600f: "no way\n"
(gdb) x/s $rdx
0x55555555600f: "no way\n"
(gdb)
```

(2) `call *%rax` 부분에 `breakpoint`를 걸어서 실행

```
(gdb) break *0x00005555555551ae
Breakpoint 2 at 0x5555555551ae: file debug.c, line 18.
(gdb) c
Continuing.

Breakpoint 2, 0x00005555555551ae in main (argc=1, argv=0x7fffffffe188) at debug.c:18
18          func("no way\n");
(gdb) x/s $rdi
0x55555555600f: "no way\n"
(gdb) x/s $rdx
0x55555555600f: "no way\n"
```

4) “bin/sh” 주소 찾기

```
find &system, +99999999, "/bin/sh"
# system 함수의 주소에서 메모리 검색하여 "/bin/sh" 문자열 주소 찾아 출력
```

```
(gdb) find &system, +99999999, "/bin/sh"
0x7ffff7dd8698
warning: Unable to access 16000 bytes of target memory at 0x7ffff7e268a0, halting search.
1 pattern found.
```

5) 문자열 주소 변경

> $rdi와 $rdx는 현재 문자열 “no way\n”로 지정되어 있는데,
> 해당 문자열의 주소를 위에서 찾은 “bin/sh” 주소로 변경하여 문자열이 변경되도록 함

```
set $rdi = 0x7ffff7dd8698 # $rdi의 주소를 “bin/sh” 주소로 변경, $rdx도 가능
c # 프로그램 계속 실행
# system("/bin/sh")가 실행됨
ps # 쉘 명령어 ps(프로세스 정보 표시)를 통해 /bin/sh가 잘 동작하는지 테스트
exit # 쉘 나가기
```

```
(gdb) set $rdi = 0x7ffff7dd8698
(gdb) c
Continuing.
[Detaching after vfork from child process 6560]
$ ps
    PID TTY          TIME CMD
   2361 pts/0    00:00:00 bash
   5216 pts/0    00:00:00 gdb
   5366 pts/0    00:00:00 debug
   6560 pts/0    00:00:00 sh
   6561 pts/0    00:00:00 sh
   6610 pts/0    00:00:00 ps
$ exit
[Inferior 1 (process 5366) exited normally]
(gdb)
```

```
(gdb) set $rdx = 0x7ffff7dd8698
(gdb) c
Continuing.
[Detaching after vfork from child process 4037]
$ ps
    PID TTY          TIME CMD
   2461 pts/0    00:00:00 bash
   2773 pts/0    00:00:00 gdb
   2852 pts/0    00:00:00 debug
   4037 pts/0    00:00:00 sh
   4038 pts/0    00:00:00 sh
   4063 pts/0    00:00:00 ps
$ exit
[Inferior 1 (process 2852) exited normally]
(gdb)
```

3. 궁금한 점

> 💡 system("/bin/sh") 코드 자체를 소스파일 안의 main에 넣어서 실행해도 되는건가?

⇒ 된다. 애초에 system("/bin/sh") 이 코드가 소스파일의 메인 함수 안에 있으면 실행했을 때 자동으로 실행된다.

> 💡 아예 set $rdx = "/bin/sh" 이렇게 문자열 자체를 바꿔서 실행해도 되나?

⇒ 된다. 앞에서 set var func = system으로 바꾸었기 때문에 현재 func에는 system 함수를 호출하는 것으로 바뀌어 있고, 원래 문자열대로라면 system(”no way\n”); 이런 방식인데, 이 부분의 문자열을 “/bin/sh”로 바꾸게 되면, system(”/bin/sh”)가 되기 때문에 쉘이 실행된다.

> 💡 교수님께서는 첫번째 인자를 전달하는 %rdi 레지스터의 주소를 바꾸라고 했는데,
> %rdx 레지스터 주소를 바꿔서 실행해도 될까?

⇒ 해본 결과 되는 것 같다. 단 call 전에 breakpoint를 걸면 두 레지스터 모두 문자열이 동일하게 변경되어 있지만, 한 줄씩 호출할 경우에는 각자 레지스터에 들어있는 문자열이 잘 바뀌어 있는지 확인해야 한다.