* 7번

main

0x0000000000400534 <+0>: push %rbp // 운영체제가 임의로 초기화한 rbp 값을 스택에 밀어넣는다
0x0000000000400535 <+1>: mov %rsp, %rbp // 현재 rsp 값을 rbp에 저장함
0x0000000000400538 <+4>: sub $0x10, %rsp // 현재 rsp 위치에서 16을 빼서 16byte의 공간을 확보함.
0x000000000040053c <+8>: movl $0x0, -0x8(%rbp) // rbp에서 0x8 만큼 아래에 0x0을 저장함.
0x0000000000400543 <+15>: movl $0x0, -0xc(%rbp) // rbp에서 0xc 만큼 아래에 0x0을 저장함.
0x000000000040054a <+22>: jmp 0x400556 <main+34> // 0x0000000000400556 의 pc로 이동함.
0x000000000040054c <+24>: mov -0xc(%rbp), %eax // rbp에서 0xc 만큼 아래의 값을 eax에 저장함.
0x000000000040054f <+27>: add %eax, -0x8(%rbp) // rbp에서 0x8 만큼 아래의 값을 eax값과 더하고 저장함.
0x0000000000400552 <+30>: addl $0x1, -0xc(%rbp) // rbp에서 0xc만큼 아래의 값을 0x1 과 더하고 저장함.
0x0000000000400556 <+34>: cmpl $0x4, -0xc(%rbp) // rbp에서 0xc만큼 아래의 값과 0x4 와 같은지 확인함.
0x000000000040055a <+38>: jle 0x40054c <main+24> // 0x000000000040054c 의 pc로 이동함.
0x000000000040055c <+40>: mov -0x8(%rbp), %eax // rbp에서 0x8만큼 아래의 값을 eax에 저장함.
0x000000000040055f <+43>: mov %eax, %edi // eax 값을 edi 값에 저장함.
0x0000000000400561 <+45>: callq 0x400526 <mult2> // 0x0000000000400523 의 pc의 mult2 함수로 이동함.
0x0000000000400566 <+50>: mov %eax, -0x4(%rbp) // eax값을 rbp에서 0x4만큼 아래의 공간에 저장함.
0x0000000000400569 <+53>: mov -0x4(%rbp), %eax // rbp에서 0x4만큼 아래의 값을 eax에 저장함.
0x000000000040056c <+56>: mov %eax, %esi // eax값을 esi 값에 저장함.
0x000000000040056e <+58>: mov $0x400614,%edi // 0x400614값을 edi에 저장함.
0x0000000000400573 <+63>: mov $0x0, %eax // 0x0을 eax에 저장함.
0x0000000000400578 <+68>: callq 0x400400 <printf@plt> // 0x400400 의 pc의 printf 함수로 이동함
0x000000000040057d <+73>: mov $0x0, %eax // 0x0의 값을 eax주소에 저장함.
0x0000000000400582 <+78>: leaveq

multi2

0x0000000000400526 <+0>: push %rbp // rbp주소를 스택에 밀어넣음.
0x0000000000400527 <+1>: mov %rsp, %rbp // 현재 rsp 값을 rbp 값에 저장함.
0x000000000040052a <+4>: mov %edi, -0x4(%rbp) // edi 값을 rbp에서 0x4만큼 아래의 공간에 저장함.
0x000000000040052d <+7>: mov -0x4(%rbp), %eax // rbp에서 0x4만큼 아래의 값을 eax에 저장함.
0x0000000000400530 <+10>: add %eax, %eax // eax값 자기자신을 더해서 저장함.
0x0000000000400532 <+12>: pop %rbp // 스택에서 rbp값을 빼내옴.
0x0000000000400533 <+13>: retq // 리턴

printf

0x0000000000400400 <+0>: jmpq *0x200c12(%rip) // 0x200c12 주소로 이동함.
0x0000000000400406 <+6>: pushq $0x0 // 0x0를 스택에 밀어넣음.
0x000000000040040b <+11>: jmpq 0x4003f0 // 0x4003f0 으로 이동함.

* 12번

*** 리눅스 디버깅 절차**

1. gcc -g -o debug [소스파일명]을 통해 debug 프로그램을 생성한다. gcc -g -o0 -o debug [소스파일명]을 하면 최적화를 강제로 방지할 수 있다.
2. gdb debug 명령어를 통해 디버거를 켠다.
3. 중단점을 사용해서 확인해보려는 명령어에서 프로그램이 멈추도록 한다.
4. 프로그램이 현재 멈춰있는 명령어에서 현재 레지스터의 주소, 값을 확인할 수 있다.
5. 계속 실행하면서 디버깅을 한다.
6. 디버깅이 끝났으면 종료한다.

*** 리눅스 디버깅 명령어**

1. 우선 컴파일 시에 디버깅 정보를 담아야 한다.
 gcc -g -o [프로그램명] [소스파일명]
 디버깅 옵션인 -g 으로 컴파일하며, 최적화 옵션인 -O 은 주지 않도록 한다.

2. 실행방법
 gdb [프로그램명]
 gdb [프로그램명] [core파일명]
 gdb [프로그램명] [실행중인프로세스pid]

3. 종료방법
 q
 Ctrl + d

4. 소스 찾아가기 (list)
 l : main 함수를 기점으로 소스의 내용이 출력된다
 l 10 : 10 행 주변의 소스가 출력되는데 10 - 5 행부터 10 + 5행까지 총 10행이 출려된다.
 l func : func 함수의 소스를 출력
 l -5 : 기본값으로 10줄이 출력된다고 가정하고, 다음에 출력될 라인이 11라인이라면, 10(마지막라인) - 5 라인을 중심으로 출력된다. 즉, 그대로 1~10라인이 출력된다.
 l a.c:func : a.c 파일의 func 함수부분을 출력
 l a.c:10 : a.c 파일의 10행을 기준으로 출력

5. 옵션
 set listsize 20 : 한번에 출력하는 행의 갯수를 20개로 늘린다.
 Enter : 마지막으로 수행한 명령어를 다시 수행한다

6. 프로그램 실행, 종료 (run, kill)
 r : 프로그램 수행 (재시작)
 r arg1 arg2 : arg1과 arg2를 인자로 프로그램 수행
 k : 프로그램 수행종료

7. 역추적하기 (backtrace)
 bt : 오류가 발생한 함수를 역으로 찾아간다.

8. 중단점 사용하기 (breakpoint, temporary breakpoint)
 b func : func 함수에 브레이크 포인트 설정

b 10 : 10행에 브레이크 포인트 설정
b a.c:func : a.c파일의 func함수에 브레이크 포인트 설정
b a.c:10 : a.c파일의 10행에 브레이크 포인트 설정
b +2 : 현재 행에서 2개 행 이후 지점에 브레이크 포인트 설정
b -2 : 현재 행에서 2개 행 이전 지점에 브레이크 포인트 설정
b *0x8049000 : 0x8049000 주소에 브레이크 포인트 설정 (어셈블리로 디버깅 시 사용)
b 10 if var == 0 : 10행에 브레이크 포인트를 설정해되, var 변수 값이 0일 때 작동
tb : 임시 중단점을 사용하는 것으로 한번만 설정되며, 그 이후에는 삭제된다.

9. 중단점 설정하기 (condition)
condition 2 var == 0 : 고유번호가 2번인 브레이크포인트에 var변수가 0일 때 동작하라고 설정

10. 중단점 삭제하기 (clear, delete)
cl func : func 함수의 시작 부분에 브레이크 포인트 지움
cl 10 : 10행의 브레이크 포인트 지움
delete 1 : 고유번호 1번의 브레이크 포인트를 지운
cl a.c:func : a.c 파일의 func함수의 브레이크 포인트 지움
cl a.c:10 : a.c 파일의 10행의 브레이크 포인트 지움
cl : 모든 브레이크 포인트 지움
11. 중단점 정보보기 (information)
info b : 현재 설정된 브레이크 포인트의 정보를 보여준다
방향키Up/Down : 방향키 Up/Down을 누르면 히스토리 기능을 제공한다
info br + TAB : info br 로 시작하는 키워드가 히스토리에 있다면 뿌려준다
info TAB + TAB : info 뒤에 올 수 있는 인자 리스트를 보여준다
TAB + TAB : 현재 사용가능한 모든 명령어 리스트를 보여준다

12. 중단점 비활성화, 활성화 하기 (enable, disable)
disable 2 : 고유번호 2번인 브레이크 포인트 비활성화
enable 2 : 고유번호 2번인 브레이크 포인트 활성화

13. 디버깅 하기 (step, next, continue, until, finish, return, step instruction, next instruction)
s : 현재 출력된 행을 수행하고 멈추지만, 함수의 경우 함수의 내부로 들어가서 수행된다
s 5 : s를 5번 입력한 것과 동일
n : 현재 행을 수행하고 멈추지만, 함수의 경우 함수를 수행하고 넘어간다
n 5 : n을 5번 입력한 것과 동일
c : 다음 브레이크 포인트를 만날때 까지 계속 수행한다
u : for 문에서 빠져나와서 다음 브레이크 포인트까지 수행한다.
finish : 현재 함수를 수행하고 빠져나감
return : 현재 함수를 수행하지 않고 빠져나감
return 123 : 현재 함수를 수행하지 않고 빠져나감, 단, 리턴값은 123
si : 현재의 인스트럭션을 수행, 함수 호출 시 내부로 들어간다.
ni : 현재의 인스트럭션을 수행, 함수 호출 시 내부로 들어가지 않는다.

14. 감시점 설정 (watch)
watch i : i변수에 와치포인트를 설정하고 i변수가 바뀔 때마다 브레이크가 걸리면서 이전값과 현재값을 출력한다.

15. 변수 정보보기 (info, print)

info locals : 현재 상태에서 어떤 지역변수들이 있으며, 값은 어떠한지를 알 수 있다.
info variables : 현재 상태에서의 전역변수 리스트를 확인할 수 있다.
p lval : lval 값을 확인한다.
p func : func 함수의 주소값을 확인한다.
p pt : pt가 구조체라면 구조체의 주소를 확인한다
p *pt : pt가 구조체라면 구조체의 값을 확인한다.
p **pt : *pt가 구조체라면 구조체의 값을 확인한다.
info registers : 레지스트 값 전체를 한번에 확인한다.

16. 레지스트 값 및 포인터가 가리키는 구조체의 배열을 출력 (info, print)
info all-registers : MMX 레지스트를포함하여 거의 대부분의 레지스트 값을 확인한다.
p $eax : eax 레지스트의 값을 확인한다. (ex_ eax, ebx, ecx, edx, eip)
p *pt@4 : 4크기의 배열로 gdb가 알 수 있으므로 4개의 크기만큼 가져와서 확인할 수 있다.

17. 중복된 변수명이 있는 경우 특정 변수를 지정해서 출력 (print)
p 'main.c'::var : main.c 파일에 있는 전역변수인 var 변수의 값을 출력
p hello::var : hello 함수에 포함된 static 변수인 var 변수의 값을 출력

18. 출력 형식의 지정
p/t var : var 변수를 2진수로 출력
p/o var : var 변수를 8진수로 출력
p/d var : var 변수를 부호가 있는 10진수로 출력 (int)
p/u var : var 변수를 부호가 없는 10진수로 출력 (unsigned int)
p/x var : var 변수를 16진수로 출력
p/c var : var 변수를 최초 1바이트 값을 문자형으로 출력
p/f var : var 변수를 부동 소수점 값 형식으로 출력
p/a addr : addr주소와 가장 가까운 심볼의 오프셋을 출력 (ex_ main + 15)

19. 타입이 틀릴 경우 타입을 변환하여 출력
p (char*)vstr : 실제 컴파일 시에 (void *)형으로 되어있었다고 하더라도 (char *)로 캐스팅 하여 보여줌

20. 특정한 위치 지정
p lstr + 4 : 예를 들어 lstr = "I like you." 라는 문자열은 "ke you."가 출력된다.

21. 변수 값 설정
p lval = 1000 : 변수값 확인 이외에는 설정도 가능하다.

22. 출력명령 요약 (print)
p [변수명] : 변수 값을 출력
p [함수명] : 함수의 주소를 출력
p/[출력형식] [변수명] : 변수 값을 출력 형식으로 출력
p '[파일명]'::[변수명] : 파일명에 있는 전역변수 값을 출력
p [함수명]::[변수명] : 함수에 있는 변수 값을 출력
p [변수명]@[배열크기] : 변수의 내용을 변수 배열의 크기 형태로 출력

23. 디스플레이 명령 (display, undisplay)
display [변수명] : 변수 값을 매번 화면에 디스플레이

display/[출력형식] [변수명] : 변수 값을 출력 형식으로 디스플레이
undisplay [디스플레이번호] : 디스플레이 설정을 없앤다
disable display [디스플레이번호] : 디스플레이를 일시 중단한다.
enable display [디스플레이번호] : 디스플레이를 다시 활성화한다.