EDDI
Electronic Design
Development Institute

에디로봇아카데미

임베디드 마스터 Lv1 과정

제 4기

2022. 09. 2

유진선

Gdb 디버깅 방법

사용법 : gcc [컴파일할 파일명] → ex)gcc func.c
a.out 이라는 실행파일이 생성되고, ./a.out 파일 실행 됩니다.

옵션

1. gcc -o : 실행 파일 이름 보정 / 사용법: gcc-o[실행파일이름][소스파일이름]
이 옵션을 사용하지 않으면 기본으로 a.out 실행파일이 생성된다.
ex) gcc -o func func.c

2. gcc -g : 디버깅 활성화 / 사용법 : gcc -g[소스파일이름] → ex) gcc -g func.c
이 옵션을 지정하지 않으면 디버깅을 할 수 없다.

3. gdb : 디버깅 시작 / 사용법 : gdb [실행파일] ex) gdb a.out

Gdb 명령어

EDDI
Electronic Design Development Institute

r : (run) **디버깅 실행** / ex) r
b : (break) break point **설정** / ex) b main
n : (next) **다음행 실행** (step over) / ex) n
s : (step) **다음행 실행** (step into) **단위** : C**언어** / ex) s
si : (step instruction) **단위** : **어셈블리 명령어 기준으로 한줄씩 실행한다**. / ex)si
disas : (disassemble) **디스어셈플리 명령** / ex) disas
p/x : 16**진수로 특정 결과를 출력한다**.
x : **메모리의 내용을 살펴본다**.
q : (quit) gdb **프로세스 종료** / ex) q

디버깅 (func.c)

기계어 분석시 반드시 알아야 하는 레지스터들
ax : 함수의 리턴값 저장
sp : 현재 스택의 최상위
bp : 스택의 base(기준점)
ip : pc(program counter) 다음에 실행할 명령어의 주소 포인팅

info registers : 실제 CPU 레지스터 정보

rsp : 현재스택의 최상위
rbp : 현재스택의 기준점
rip : 다음에 실행할 instruction의 주소값을 가르킴
rax : 무조건적으로 함수의 리턴값이 저장되며 연산용으로도 활용 가능
rcx : 보편적으로 for 루프의 카운터에 활용이 되며 연산용으로도 활용 가능

기계어 분석 - 디버깅 (func.c)

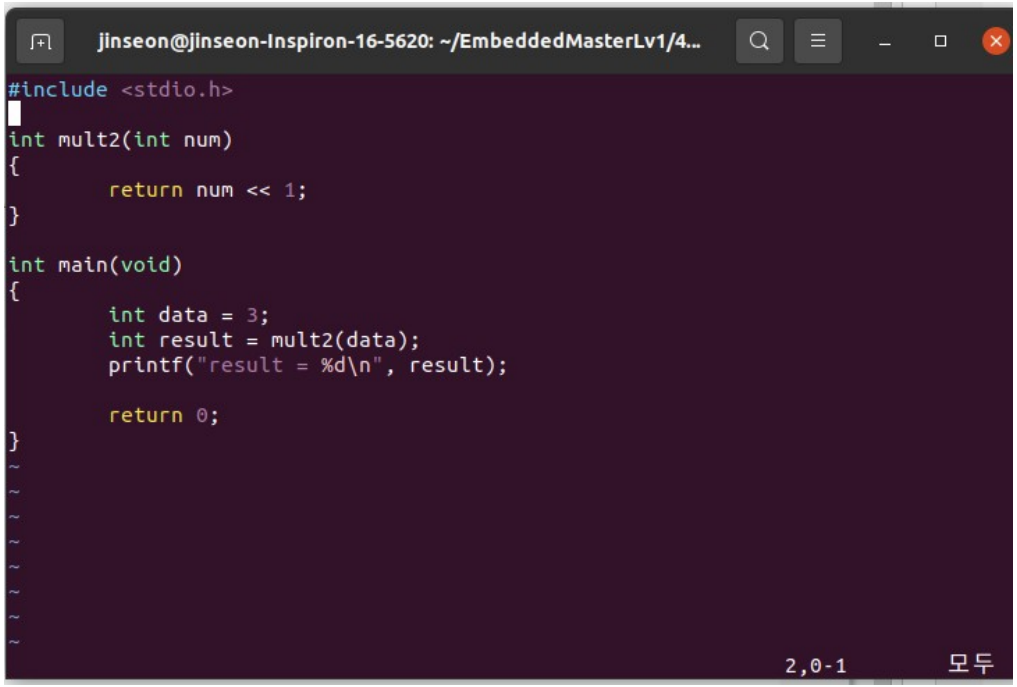
EDDI
Electronic Design Development Institute

규칙
1. 스택은 거꾸로 자란다.
2. 모든것은 메모리다. 메모리 단위 연산은 포인터 크기 단위(ALU가 결정)로 이루어진다.
기계어 동작 과정 파악을 위해 필요

push : 메모리에 정보를 배치
　　현재 스택의 최상위(sp)
포인터의 크기 → CPU, ALU
8byte 왜? →리눅스 메모리 설계 구조 때문

기계어 분석 - 디버깅 (func.c)

EDDI
Electronic Design Development Institute



위의 C 코드를 분석

기계어 분석 - 디버깅 (func.c)

디버깅 진입 방법

1. 진입 하려는 C코드의 디버깅 활성화 : gcc -g func.c
2. 디버깅 시작 : gdb a.out

```
jinseon@jinseon-Inspiron-16-5620:~/EmbeddedMasterLv1/4기/YuJinSeon/c/2$ gdb a.out
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...
(gdb)
```

기계어 분석 - 디버깅 (func.c)

디버깅 진입 방법

3. break point 설정 : b main → main에 break를 설정 하겠다.
4. r (run 프로그램 동작 실행) 입력
5. disas (어블리어로 표현) 입력
6. si step instruction) 어셈블리 명령어 한줄씩 실행

```
(gdb) b main
Breakpoint 1 at 0x115b: file func.c, line 9.
(gdb) r
Starting program: /home/jinseon/EmbeddedMasterLv1/4기/YuJinSeon/c/2/a.out

Breakpoint 1, main () at func.c:9
9       {
(gdb) diasa
Undefined command: "diasa".  Try "help".
(gdb) disas
Dump of assembler code for function main:
=> 0x000055555555515b <+0>:     endbr64
   0x000055555555515f <+4>:     push   %rbp
   0x0000555555555160 <+5>:     mov    %rsp,%rbp
   0x0000555555555163 <+8>:     sub    $0x10,%rsp
   0x0000555555555167 <+12>:    movl   $0x3,-0x8(%rbp)
   0x000055555555516e <+19>:    mov    -0x8(%rbp),%eax
   0x0000555555555171 <+22>:    mov    %eax,%edi
   0x0000555555555173 <+24>:    callq  0x555555555149 <mult2>
   0x0000555555555178 <+29>:    mov    %eax,-0x4(%rbp)
   0x000055555555517b <+32>:    mov    -0x4(%rbp),%eax
   0x000055555555517e <+35>:    mov    %eax,%esi
   0x0000555555555180 <+37>:    lea    0xe7d(%rip),%rdi        # 0x555555556004
   0x0000555555555187 <+44>:    mov    $0x0,%eax
   0x000055555555518c <+49>:    callq  0x555555555050 <printf@plt>
   0x0000555555555191 <+54>:    mov    $0x0,%eax
   0x0000555555555196 <+59>:    leaveq
   0x0000555555555197 <+60>:    retq
End of assembler dump.
(gdb)
```

```
(gdb) info registers
rax            0x55555555515b      93824992235867
rbx            0x5555555551a0      93824992235936
rcx            0x5555555551a0      93824992235936
rdx            0x7fffffffdf28      140737488346920
rsi            0x7fffffffdf18      140737488346904
rdi            0x1                 1
rbp            0x0                 0x0
rsp            0x7fffffffde28      0x7fffffffde28
r8             0x0                 0
r9             0x7ffff7fe0d60      140737354009952
r10            0xf                 15
r11            0x2                 2
r12            0x555555555060      93824992235616
r13            0x7fffffffdf10      140737488346896
r14            0x0                 0
r15            0x0                 0
rip            0x55555555515b      0x55555555515b <main>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb) disas
```

기계어 분석 - 디버깅 (func.c)

```
Dump of assembler code for function main:
=> 0x000055555555515b <+0>:     endbr64
   0x000055555555515f <+4>:     push   %rbp
```

push : 현재 스택의 최상위 (rsp가 가리키는 메모리 공간)에 뒤쪽의 메모리 값을 넣는다.

기계어 분석 - 디버깅 (func.c)

EDDI
Electronic Design Development Institute

```
Dump of assembler code for function main:
=> 0x000055555555515b <+0>:     endbr64
   0x000055555555515f <+4>:     push   %rbp
   0x0000555555555160 <+5>:     mov    %rsp,%rbp
```

mov : 좌측의 값을 우측에 복사한다.

rsp 값을 rbp에 넣는다.

기계어 분석 - 디버깅 (func.c)

EDDI
Electronic Design Development Institute

```
Dump of assembler code for function main:
=> 0x000055555555515b <+0>:     endbr64
   0x000055555555515f <+4>:     push   %rbp
   0x0000555555555160 <+5>:     mov    %rsp,%rbp
   0x0000555555555163 <+8>:     sub    $0x10,%rsp
```

sub : 뺄샘 rsp-0x10을 rsp에 대입

기계어 분석 - 디버깅 (func.c)

EDDI
Electronic Design Development Institute

```
Dump of assembler code for function main:
=> 0x000055555555515b <+0>:     endbr64
   0x000055555555515f <+4>:     push   %rbp
   0x0000555555555160 <+5>:     mov    %rsp,%rbp
   0x0000555555555163 <+8>:     sub    $0x10,%rsp
   0x0000555555555167 <+12>:    movl   $0x3,-0x8(%rbp)
```

movl : mov 와 같음 l 이 붙어서 4byte 크기로 복사

rsp-ox10을 rsp에 대입

기계어 분석 - 디버깅 (func.c)

EDDI
Electronic Design Development Institute

```
Dump of assembler code for function main:
=> 0x000055555555515b <+0>:     endbr64
   0x000055555555515f <+4>:     push   %rbp
   0x0000555555555160 <+5>:     mov    %rsp,%rbp
   0x0000555555555163 <+8>:     sub    $0x10,%rsp
   0x0000555555555167 <+12>:    movl   $0x3,-0x8(%rbp)
   0x000055555555516e <+19>:    mov    -0x8(%rbp),%eax
```

mov : 좌측의 값을 우측에 복사한다.

rbp-0x8의 메모리 값을 eax에 복사

기계어 분석 - 디버깅 (func.c)

EDDI
Electronic Design Development Institute

```
Dump of assembler code for function main:
=> 0x000055555555515b <+0>:     endbr64
   0x000055555555515f <+4>:     push   %rbp
   0x0000555555555160 <+5>:     mov    %rsp,%rbp
   0x0000555555555163 <+8>:     sub    $0x10,%rsp
   0x0000555555555167 <+12>:    movl   $0x3,-0x8(%rbp)
   0x000055555555516e <+19>:    mov    -0x8(%rbp),%eax
   0x0000555555555171 <+22>:    mov    %eax,%edi
```

mov : 좌측의 값을 우측에 복사한다.

eax의 값을 edi에 복사

기계어 분석 - 디버깅 (func.c)

EDDI
Electronic Design Development Institute

```
Dump of assembler code for function main:
=> 0x000055555555515b <+0>:     endbr64
   0x000055555555515f <+4>:     push   %rbp
   0x0000555555555160 <+5>:     mov    %rsp,%rbp
   0x0000555555555163 <+8>:     sub    $0x10,%rsp
   0x0000555555555167 <+12>:    movl   $0x3,-0x8(%rbp)
   0x000055555555516e <+19>:    mov    -0x8(%rbp),%eax
   0x0000555555555171 <+22>:    mov    %eax,%edi
   0x0000555555555173 <+24>:    callq  0x555555555149 <mult2>
```

callq : 함수 호출시 동작. 복귀주소 push 후 함수로 jump. q가 붙어 8byte 크기로 복사

복귀주소 push후 mult2 함수로 jump

기계어 분석 - 디버깅 (func.c)

EDDI
Electronic Design Development Institute

```
Dump of assembler code for function mult2:
=> 0x0000555555555149 <+0>:     endbr64
   0x000055555555514d <+4>:     push   %rbp
   0x000055555555514e <+5>:     mov    %rsp,%rbp
   0x0000555555555151 <+8>:     mov    %edi,-0x4(%rbp)
   0x0000555555555154 <+11>:    mov    -0x4(%rbp),%eax
   0x0000555555555157 <+14>:    add    %eax,%eax
   0x0000555555555159 <+16>:    pop    %rbp
   0x000055555555515a <+17>:    retq
End of assembler dump.
(gdb)
```

main 함수 내의 mult2 함수의 어셈블리어

기계어 분석 - 디버깅 (func.c)

EDDI
Electronic Design Development Institute

```
Dump of assembler code for function mult2:
=> 0x0000555555555149 <+0>:     endbr64
   0x000055555555514d <+4>:     push   %rbp
```

push : 현재 스택의 최상위에 뒤쪽의 메모리 값을 넣는다.
스택의 최상위에 rbp의 메모리 값을 넣고 rsp 이동

기계어 분석 - 디버깅 (func.c)

EDDI
Electronic Design Development Institute

```
Dump of assembler code for function mult2:
=> 0x0000555555555149 <+0>:     endbr64
   0x000055555555514d <+4>:     push   %rbp
   0x000055555555514e <+5>:     mov    %rsp,%rbp
```

mov : 좌측의 값을 우측에 복사
rsp의 값을 rbp에 복사

기계어 분석 - 디버깅 (func.c)

EDDI
Electronic Design Development Institute

```
Dump of assembler code for function mult2:
=> 0x0000555555555149 <+0>:     endbr64
   0x000055555555514d <+4>:     push   %rbp
   0x000055555555514e <+5>:     mov    %rsp,%rbp
   0x0000555555555151 <+8>:     mov    %edi,-0x4(%rbp)
```

mov : 좌측의 값을 우측에 복사
edi의 값을 rbp-ox4의 메모리에 복사

기계어 분석 - 디버깅 (func.c)

EDDI
Electronic Design Development Institute

```
Dump of assembler code for function mult2:
=> 0x0000555555555149 <+0>:     endbr64
   0x000055555555514d <+4>:     push   %rbp
   0x000055555555514e <+5>:     mov    %rsp,%rbp
   0x0000555555555151 <+8>:     mov    %edi,-0x4(%rbp)
   0x0000555555555154 <+11>:    mov    -0x4(%rbp),%eax
```

mov : 좌측의 값을 우측에 복사
rbp-ox4의 값을 reax에 복사

기계어 분석 - 디버깅 (func.c)

EDDI
Electronic Design Development Institute

```
Dump of assembler code for function mult2:
=> 0x0000555555555149 <+0>:     endbr64
   0x000055555555514d <+4>:     push   %rbp
   0x000055555555514e <+5>:     mov    %rsp,%rbp
   0x0000555555555151 <+8>:     mov    %edi,-0x4(%rbp)
   0x0000555555555154 <+11>:    mov    -0x4(%rbp),%eax
   0x0000555555555157 <+14>:    add    %eax,%eax
```

add 덧샘
eax값과 eax 값을 더하고 eax에 복사

기계어 분석 - 디버깅 (func.c)

EDDI
Electronic Design Development Institute

```
Dump of assembler code for function mult2:
=> 0x0000555555555149 <+0>:     endbr64
   0x000055555555514d <+4>:     push   %rbp
   0x000055555555514e <+5>:     mov    %rsp,%rbp
   0x0000555555555151 <+8>:     mov    %edi,-0x4(%rbp)
   0x0000555555555154 <+11>:    mov    -0x4(%rbp),%eax
   0x0000555555555157 <+14>:    add    %eax,%eax
   0x0000555555555159 <+16>:    pop    %rbp
```

pop 현재 스택의 최상위(rsp가 가리키는 메모리 공간)에서 값을 꺼내 뒤에 넣는다.

스택의 최상위에서 값을 꺼내 rbp에 복사하고 rsp 이동

기계어 분석 - 디버깅 (func.c)

EDDI
Electronic Design Development Institute

```
Dump of assembler code for function mult2:
=> 0x0000555555555149 <+0>:     endbr64
   0x000055555555514d <+4>:     push   %rbp
   0x000055555555514e <+5>:     mov    %rsp,%rbp
   0x0000555555555151 <+8>:     mov    %edi,-0x4(%rbp)
   0x0000555555555154 <+11>:    mov    -0x4(%rbp),%eax
   0x0000555555555157 <+14>:    add    %eax,%eax
   0x0000555555555159 <+16>:    pop    %rbp
   0x000055555555515a <+17>:    retq
End of assembler dump.
(gdb)
```

retq : pop $rip 와 같은말
스택의 최상위에서 값을 꺼내 rip에 복사하고 rsp 이동
mult2 의 Stack Frame 해제하고 다시 main 함수로 이동

기계어 분석 - 디버깅 (func.c)

EDDI
Electronic Design Development Institute

```
Dump of assembler code for function main:
=> 0x000055555555515b <+0>:     endbr64
   0x000055555555515f <+4>:     push   %rbp
   0x0000555555555160 <+5>:     mov    %rsp,%rbp
   0x0000555555555163 <+8>:     sub    $0x10,%rsp
   0x0000555555555167 <+12>:    movl   $0x3,-0x8(%rbp)
   0x000055555555516e <+19>:    mov    -0x8(%rbp),%eax
   0x0000555555555171 <+22>:    mov    %eax,%edi
   0x0000555555555173 <+24>:    callq  0x555555555149 <mult2>
   0x0000555555555178 <+29>:    mov    %eax,-0x4(%rbp)
```

mov : 좌측의 값을 우측에 복사한다.