



C basic language

임베디드스쿨 2기

Lv1과정

2021. 04. 16

김효창

- 목 차 -

첨부1. Assembly Analysis

Assembly Analysis



✓ **switch**



✓ **continue**



✓ **break**



✓ **goto**



✓ **array**



✓ **doublepointer**

Assembly Analysis

jl : jump less // cmp a , b 에서 a 가 작을 때 점프

jle : jump less or equal // 결과가 작거나 같을 경우 점프

je : jump equal // 비교 결과가 같을 때 점프

jne : jump not equal // 비교 결과가 다를 때 점프

jg : jump greater // cmp a, b에서 a가 크면 점프

movl : 4바이트를 복사한 뒤 나머지 4바이트는 모두 0으로 할당

movslq : Move sign-extended double word to quad word 4바이트 또는 8바이트 단위로 복사한다

imul : signed multiply // 부호있는 곱셈

shl , shr : 부호가 없는 연산

0x20 크기만큼 rax를 오른쪽으로 각 비트를 시프트 한다 , 1번 이동할 때 마다 값이 2배씩 증가 / 감소

sal , sar : 부호가 있는 연산 ,

0x1f 크기만큼 rax를 오른쪽으로 각 비트를 시프트 한다 , 1번 이동할 때 마다 값이 2배씩 증가 /감소

test : Operand 1 과 Operand 2 를 AND 연산 하라는 것 , "eax 값이 0 인지 확인할 때 사용"

zero flag : 이전 명령어 결과에 대한 제어 흐름을 바꾸는 조건부 분기 명령어에 사용

cltq : %rax ← SignExtend(%eax) // Sign extend %eax to %rax

Assembly Analysis

break	goto
<+114> mov -0xc(rbp), eax	<+170> mov -0xc(rbp), eax
<+117> cmp -0x4(rbp), eax	<+173> cmp -0x4(rbp), eax
<+120> jne	<+176> jne
<+122> mov -0x4(rbp), eax	<+178> mov -0x4(rbp), eax
<+139> printf " data: 1, Error ! "	<+195> printf " data: 0 Error! "
<+144> movb 0x1, -0x11(rbp)	<+200> nop
<+148> jmp	<+201> endbr64
<+188> cmpb 0x0, -0x11(rbp)	<+212> puts " 에러 발생 "
<+192> jne	<+217> mov 0xffffffff ,eax
<+206> nop	<+222> jmp
<+207> cmpb 0x0, -0x11(rbp)	<+303> leaveq
<+211> jne	
<+225> nop	
<+231> leaveq	

goto 를 사용하면 다중 반복되는 jmp, jne, jle ... 등등 쉽게 탈출할 수 있다

별도 함수 작성하지 않고 예외 처리 가능 ...

이것 말고는 장점을 모르겠다 단점이 있을 거 같긴 한데....

goto 를 사용하지 않으면 어셈블리 분석 시 최적화 되지 않아 복잡해서 읽기가 어렵다

Assembly Analysis

```
void fun(int* y)      ip of assembler code for function main:      ip of assembler code for function fun:
{
    *y = 20;          0x000055555555162 <+0>:      endbr64      0x000055555555149 <+0>:      endbr64
}                    0x000055555555166 <+4>:      push    %rbp      0x00005555555514d <+4>:      push    %rbp
                    0x000055555555167 <+5>:      mov     %rsp,%rbp  0x00005555555514e <+5>:      mov     %rsp,%rbp
                    0x00005555555516a <+8>:      sub     $0x10,%rsp 0x000055555555151 <+8>:      mov     %rdi,-0x8(%rbp)
                    0x00005555555516e <+12>:     mov     %fs:0x28,%rax 0x000055555555155 <+12>:     mov     -0x8(%rbp),%rax
                    0x000055555555177 <+21>:     mov     %rax,-0x8(%rbp) 0x000055555555159 <+16>:     movl    $0x14,(%rax)
                    0x00005555555517b <+25>:     xor     %eax,%eax  0x00005555555515f <+22>:     nop
                    0x00005555555517d <+27>:     movl    $0xa,-0xc(%rbp) 0x000055555555160 <+23>:     pop     %rbp
int main(void)       0x000055555555184 <+34>:     lea     -0xc(%rbp),%rax 0x000055555555161 <+24>:     retq
{
    int x = 10;      0x000055555555188 <+38>:     mov     %rax,%rdi
    fun(&x);          0x00005555555518b <+41>:     callq   0x55555555149 <fu
    return 0;         0x000055555555190 <+46>:     mov     $0x0,%eax
                    0x000055555555195 <+51>:     mov     -0x8(%rbp),%rdx
                    0x000055555555199 <+55>:     xor     %fs:0x28,%rdx
                    0x0000555555551a2 <+64>:     je      0x555555551a9 <m
                    0x0000555555551a4 <+66>:     callq   0x55555555050 <
```

함수 진행 전

(gdb) i locals

x = 10

(gdb) x/x &x

0x7fffffffde24: 0x0000000a

(gdb) p &x

\$1 = (int *) 0x7fffffffde24

함수 진행 후

(gdb) i locals

x = 20

(gdb) x/x &x

0x7fffffffde24: 0x00000014

(gdb) p &x

\$2 = (int *) 0x7fffffffde24

reference (참조자 , C++)

자신이 참조하는 변수를 대신할 수 있는 또 하나의 이름

reference

- 같은 메모리 공간을 공유하므로 메모리 공간을 소모하지 않는다
- 선언과 동시에 초기화를 해야 한다 ???

Assembly Analysis

```
hyochangkim@hyochangkim-ThinkPad-X390-Yoga:
efer.c
#include <stdio.h>
int main(void)
{
    int x = 10;
    int *p;

    p = &x;

    *p = 20;

    return 0;
}
```

```
0x000055555555149 <+0>: endbr64
0x00005555555514d <+4>: push    %rbp
0x00005555555514e <+5>: mov     %rsp,%rbp
0x000055555555151 <+8>: sub     $0x20,%rsp
0x000055555555155 <+12>: mov     %fs:0x28,%rax
0x00005555555515e <+21>: mov     %rax,-0x8(%rbp)
0x000055555555162 <+25>: xor     %eax,%eax
0x000055555555164 <+27>: movl    $0xa,-0x14(%rbp)
0x00005555555516b <+34>: lea     -0x14(%rbp),%rax
0x00005555555516f <+38>: mov     %rax,-0x10(%rbp)
0x000055555555173 <+42>: mov     -0x10(%rbp),%rax
0x000055555555177 <+46>: movl    $0x14,(%rax)
0x00005555555517d <+52>: lea     -0x14(%rbp),%rax
0x000055555555181 <+56>: mov     %eax,%edx
0x000055555555183 <+58>: mov     -0x10(%rbp),%rax
0x000055555555187 <+62>: mov     %edx,(%rax)
0x000055555555189 <+64>: mov     $0x0,%eax
0x00005555555518e <+69>: mov     -0x8(%rbp),%rcx
0x000055555555192 <+73>: xor     %fs:0x28,%rcx
0x00005555555519b <+82>: je      0x555555551a2 <main+89>
0x00005555555519d <+84>: callq   0x55555555050 <_stack_chk_fail@plt>
```

```
(gdb) i locals
x = 20
p = 0x7fffffffdddec
(gdb) x/x &x
0x7fffffffdddec: 0x00000014
(gdb) x/x &p
0x7fffffffdddf0: 0xffffdddec
(gdb) p &x
$1 = (int *) 0x7fffffffdddec
(gdb) p &p
$2 = (int **) 0x7fffffffdddf0
```

(int *) 0xddec : 정수에 대한 포인터

(int **) 0xddf0 : 정수에 대한 포인터의 포인터

(int*) pointer : 주소 값이 나온다

((int)pointer) : 주소 값을 참조하여 값을 불러온다

