

# strcpy( ) 구현



입력 : 한 줄의 문자열

출력 : bss 섹션 데이터에 문자열 저장하기

```
1 strcpy.asm
;file: strcpy.asm
2 ;입력받은 문자열을 복사하여 저장하는 프로그램
3 section .bss
4     str1 resb 64          ;문자열을 저장하기 위해 bss 세그먼트에 선언
5
6 section .text
7     global _start
8
9 _start:
10    xor rax, rax           ;rax, rbx, rcx, rdx 초기화
11    mov rbx, rax
12    mov rcx, rax
13    mov rdx, rax
14
15    sub rsp, 64            ;문자열 입력받을
16    mov rdi, 0
17    mov rsi, rsp
18    mov rdx, 63
19
20    syscall
21
22    xor r9, r9
23    while:
24        mov cl, [rsp + r9] ;NULL이 나올 때까지 입력받은 문자열을 돌며 str1 (주소) 가 가리키는 곳에 문자를 하나씩 저장함
25        mov [str1 + r9], cl
26        cmp cl, 0x00
27        je break
28        inc r9
29        jmp while
30    break:
31
32    mov rax, 1             ;str1 출력
33    mov rdi, 1
34    mov rsi, str1          ;rsi에 str1의 주소 전달해줌
35    mov rdx, 64
36
37    syscall
38
39    mov rax, 60            ;프로그램 종료
40
41    syscall
```

bss 섹션에 64bit의 크기를 갖는 데이터를 정의했다는 점 빼고는 strlen( ), strcat( )과 크게 다르지 않다.

```
;line 22 - 30
xor r9, r9
while:
    mov cl, [rsp + r9]
    mov [str1 + r9], cl
    cmp cl, 0x00
    je break
    inc r9
```

```
    jmp while
break:
```

문자열을 한 문자씩 `cl` 레지스터에 받아와 그걸 그대로 `str1`에 저장한다. `NULL`이 나올 때 까지 반복.

`str1`에 입력된 문자열이 잘 들어갔는지 확인하기 위해, `gdb`로 디버깅 해 보았다.

```
$ nasm -f elf64 strcpy.asm
$ gcc -g -o strcpy strcpy.o
$ gdb strcpy
```

```
(gdb) set disassembly-flavor intel
(gdb) disas main
Dump of assembler code for function main:
   0x00000000004004e0 <+0>:    xor     rax,rax
   0x00000000004004e3 <+3>:    mov     rbx,rax
   0x00000000004004e6 <+6>:    mov     rcx,rax
   0x00000000004004e9 <+9>:    mov     rdx,rax
   0x00000000004004ec <+12>:   sub     rsp,0x40
   0x00000000004004f0 <+16>:   mov     edi,0x0
   0x00000000004004f5 <+21>:   mov     rsi,rsp
   0x00000000004004f8 <+24>:   mov     edx,0x3f
   0x00000000004004fd <+29>:   syscall
   0x00000000004004ff <+31>:   xor     r9,r9
End of assembler dump.
(gdb) disas while
Dump of assembler code for function while:
   0x0000000000400502 <+0>:    mov     cl,BYTE PTR [rsp+r9*1]
   0x0000000000400506 <+4>:    mov     BYTE PTR [r9+0x601034],cl
   0x000000000040050d <+11>:   cmp     cl,0x0
   0x0000000000400510 <+14>:   je      0x400517 <break>
   0x0000000000400512 <+16>:   inc     r9
   0x0000000000400515 <+19>:   jmp     0x400502 <while>
End of assembler dump.
(gdb) disas break
Dump of assembler code for function break:
   0x0000000000400517 <+0>:    mov     eax,0x1
   0x000000000040051c <+5>:    mov     edi,0x1
   0x0000000000400521 <+10>:   movabs  rsi,0x601034
   0x000000000040052b <+20>:   mov     edx,0x40
   0x0000000000400530 <+25>:   syscall
   0x0000000000400532 <+27>:   mov     eax,0x3c
   0x0000000000400537 <+32>:   syscall
   0x0000000000400539 <+34>:   nop     DWORD PTR [rax+0x0]
End of assembler dump.
```

작성했던 코드들을 확인할 수 있다. 또한, while 안에서 내가 str1으로 정의했던 메모리 위치 0x601034를 확인 할 수 있었다.

```
(gdb) b break
Breakpoint 1 at 0x400517
(gdb) r
Starting program: /home/e_lumos/Desktop/Assembly/strcpy_f/strcpy
E_Lumos
Breakpoint 1, 0x000000000400517 in break ()
```

따라서 문자열 복사가 완료되는 시점인 break에 breakpoint를 잡고 실행시켜, 문자열이 모두 str1에 들어갈 수 있게 실행시켰다.

```
(gdb) x/7bx 0x601034
0x601034: 0x45 0x5f 0x4c 0x75 0x6d 0x6f 0x73
```

내가 넣어준 문자는 E\_Lumos로 7바이트. str1의 메모리 0x601034의 값을 1 byte씩 7 byte만큼, 16진수로 출력시켜 보았다.

이 값을 순서대로 아스키 코드로 바꾸면,

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int str[7] = {0x45, 0x5f, 0x4c, 0x75, 0x6d, 0x6f, 0x73};
6     for(int i = 0; i < 7; i++) cout << (char)str[i];
7     cout << endl;
8     return 0;
9 }
E_Lumos
logout
Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.

[프로세스 완료됨]
```

E\_Lumos가 됨을 확인할 수 있다.