펭귄 스터디 내용07

작성자 : 류성수

작성일자 : 24.07.01

url : https://www.youtube.com/watch?v=s7xnTuSb8U8&list=PLz—ENLG_8TMdMJIwyqDIpcEOysvNoonf&index=9

● 함수의 호출과 종료

함수의 호출과 종료 부분을 설명하시는 부분을 공부하였다.

일단 이를 알아보기 위해 간단한 함수를 작성하였다.

```
(gdb) l main
4       {
5               return a+b;
6       }
7
8       void main()
9       {
10              printf("start!\n");
11              int a= 1, b = 2;
12              int result = sum(a,b);
13      }
```

일단, 이론상 함수가 호출되려면,

1. 함수를 호출할때의 넘기는 파라미터들을 레지스터로 옮긴다.
2. caller 부분에서 callee의 함수 주소로 call 명령을 수행한다.
3. caller 함수의 코드 레지스터 주소값을 스택에 푸시하고, rsp의 주소값을 rbp로 옮겨서 새로운 스택영역으로 이동한다.
4. rbp의 값을 기준으로 주소값을 계산하면서, 파라미터들을 스택에 옮긴다.

함수가 종료 되려면,

1. 스택프레임을 정리한다. rbp에 저장된 caller의 주소를 pop한다.
2. 호출자에게 원하는 return 값을 eax 레지스터로 옮긴다.
3. rbp의 값으로 다시 이동한다.

<table>
<tr>
<td>

```
Dump of assembler code for function main:
   0x0000000000001161 <+0>:     endbr64
   0x0000000000001165 <+4>:     push   %rbp
   0x0000000000001166 <+5>:     mov    %rsp,%rbp
   0x0000000000001169 <+8>:     sub    $0x10,%rsp
   0x000000000000116d <+12>:    lea    0xe90(%rip),%rax        # 0x2004
   0x0000000000001174 <+19>:    mov    %rax,%rdi
   0x0000000000001177 <+22>:    call   0x1050 <puts@plt>
   0x000000000000117c <+27>:    movl   $0x1,-0xc(%rbp)
   0x0000000000001183 <+34>:    movl   $0x2,-0x8(%rbp)
   0x000000000000118a <+41>:    mov    -0x8(%rbp),%edx
   0x000000000000118d <+44>:    mov    -0xc(%rbp),%eax
   0x0000000000001190 <+47>:    mov    %edx,%esi
   0x0000000000001192 <+49>:    mov    %eax,%edi
   0x0000000000001194 <+51>:    call   0x1149 <sum>
   0x0000000000001199 <+56>:    mov    %eax,-0x4(%rbp)
   0x000000000000119c <+59>:    nop
   0x000000000000119d <+60>:    leave
   0x000000000000119e <+61>:    ret
```

</td>
<td>

```
Dump of assembler code for function sum:
   0x0000000000001149 <+0>:     endbr64
   0x000000000000114d <+4>:     push   %rbp
   0x000000000000114e <+5>:     mov    %rsp,%rbp
   0x0000000000001151 <+8>:     mov    %edi,-0x4(%rbp)
   0x0000000000001154 <+11>:    mov    %esi,-0x8(%rbp)
   0x0000000000001157 <+14>:    mov    -0x4(%rbp),%edx
   0x000000000000115a <+17>:    mov    -0x8(%rbp),%eax
   0x000000000000115d <+20>:    add    %edx,%eax
   0x000000000000115f <+22>:    pop    %rbp
   0x0000000000001160 <+23>:    ret
End of assembler dump.
```

</td>
</tr>
<tr>
<td>main 함수의 코드</td>
<td>sum 함수의 코드</td>
</tr>
</table>

어셈블리어를 확인하면, 해당 이론들이 정확히 반영되는 것을 확인할 수 있다.

● 놀라운 사실

이전에, VScode로 string 클래스의 문자들을 하나씩 받아 문자열에 저장하여 printf로 출력하는 코드를 작성하였는데,
실수로 NULL을 붙이지 않아 오류가 발생하였던 일을 기억하고 있다.
VScode의 디버거는 printf의 오류를 출력하지 않고, 생뚱맞게 puts의 오류를 출력하였는데, 이부분을 이해하지 않고 넘어갔었다.

chatGPT의 의견에 따르면, printf에 전달되는 문자열에서 정수나, 실수를 문자열로 바꿔야 하는 형식지정자(%d, %f)가 없다면, gcc 컴파일 단계에서 printf 대신 puts 함수로 컴파일 한다는 사실을 알게 되었다.