Assignment 02: Stack Frames

Date: 2024. 09. 16
Student ID:
Name: 김태훈

1. 전형적 스택 프레임 구조[1]

(그림을 사용하여 설명하기 바랍니다.)

local variable (stack bottom)
Saved register
매개변수
return address(stack top)

(그림1. 일반적인 스택 프레임 구조)

일반적인 스택 프레임은 작은 주소 방향으로 성장하며 x86에서 %RSP 레지스터는 스택의 최상위 원소를 가리킨다. 한 프로시저가 사용하는 스택프레임에 가장 높은 주소에는 리턴주소(return address)가 저장된다. 이는 프로시저가 종료될 때 이동해야할 메모리 주소를 가리킨다.

그 다음으로 높은 주소에는 매개변수가 저장된다. x86에서 일반적으로 6개 이하의 매개변수는 레지스터(64비트 기준 %rdi, %rsi, %rdx, %rcx, %r8 %r9)에 저장되므로, 7번째 매개변수부터 스택프레임에 저장된다.

그 다음 주소에는 피호출함수(callee function)가 저장해야하는 레지스터를 저장한다. 일반적으로, x86에서 레지스터 %rbx, %rbp, %r12, %r13, %r14, %r15는 피호출자가 그 값을 보존해야한다. 그래서 피호출함수가 해당 레지스터에 값을 덮어쓰기 위해서는 미리 레지스터를 스택에 저장하여야한다.

마지막으로, 지역변수(local variable)들을 저장한다. 지역변수는 해당 함수에서만 쓰이는 변수들을 의미한다.

2. 함수의 어셈블리 코드 분석

(함수 twice의 어셈블리 코드를 적고 분석합니다.)

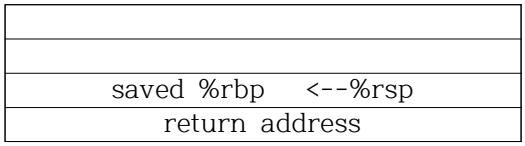
```
twice:
pushq    %rbp
.seh_pushreg     %rbp
movq     %rsp, %rbp
.seh_setframe    %rbp, 0
.seh_endprologue
movl     %ecx, 16(%rbp)
movl     16(%rbp), %eax
addl     %eax, %eax
popq     %rbp
ret
```

(코드1. twice 함수의 어셈블리 코드)

twice 함수의 어셈블리를 분석하면 아래와 같다.

(1) pushq   %rbp

%rbp 레지스터는 callee-saved 레지스터이므로, %rbp에 값을 저장하기 위해서는 원래 값을 스택에 푸시해야한다. 스택에 푸시하면 스택 최상단에 %rbp가 저장되고, %rsp 값이 8(64비트 기준)만큼 증가한다.



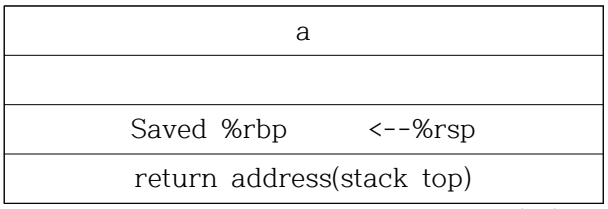
(그림2. pushq   %rbp 실행 후 스택프레임. %rsp는 saved %rbp를 가리킨다)

(2) movq   %rsp, %rbp

%rsp 레지스터의 값을 %rbp로 복사한다. %rsp는 현재 스택프레임의 최상단을 가리키고 있다. 따라서 %rbp도 스택프레임의 최상단을 가리킨다.

(3) movl   %ecx, 16(%rbp)

지역변수 a(%ecx)의 값을 스택프레임 최상단에서 16바이트 위에 저장한다. 스택 프레임 한 칸이 8바이트일 때, 스택프레임은 다음과 같이 된다.



(그림3. movl %ecx, 16(%rbp) 실행 후 twice 스택프레임. 지역변수 a(=%ecx)가 스택프레임에 저장되어있다)

(4) movl 16(%rbp), %eax

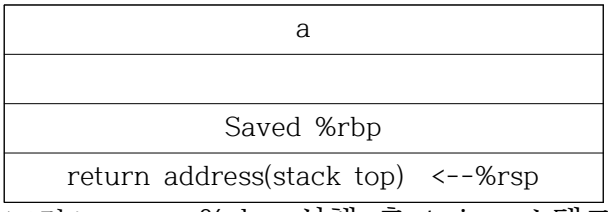
메모리에 저장된 지역변수 a값을 %eax에 저장한다. x86에서 %eax(64비트에서 %rax)는 리턴 값(return value)을 저장하는 레지스터이다[1].

(5) addl %eax, %eax

%eax 레지스터에 %eax + %eax 값을 저장한다. 따라서 %eax에는 a+a(=2*a)가 저장된다.

(6) popq %rbp

%rsp가 가리키고 있는 값을 %rbp로 복사하고, %rbp의 값을 8(64비트 기준)만큼 뺀다. %rsp가 가리키고 있는 값은 스택에 저장된 %rbp(saved %rbp)이므로, twice 함수가 호출되기 전 %rbp 값이 복원된다.



(그림4. popq %rbp 실행 후 twice 스택프레임. %rsp는 return address를 가리킨다.)

(7) ret

함수를 종료하고 값을 리턴한다. 값은 %eax에 저장되어있는 값을 리턴한다.

(8) SEH(Structured Exception Handling)**[2]**

이 어셈블리 코드에는 윈도우가 C/C++ 예외처리를 하기위해 .pdata와 .xdata 내용을 설정하기 위한 코드도 포함되어 있다.

1) .seh_pushreg %rbp

스택프레임에 %rbp 레지스터 값을 push하였음을 SEH에 알려준다.

2) .seh_setframe %rbp, 0

%rbp 레지스터 값이 %rsp값(=%rsp에서 0만큼 offset된 위치의 값)으로 설정되었음을 SEH에 알려준다.

3) .seh_endprologue

.seh 프롤로그 종료를 나타낸다.

3. 앵무새 만들기

(함수 triple에 대한 어셈블리 코드를 적고, twice의 어셈블리 코드와 비교하여 설명합니다.)

```
triple:
pushq    %rbp
.seh_pushreg     %rbp
movq     %rsp, %rbp
.seh_setframe    %rbp, 0
.seh_endprologue
movl     %ecx, 16(%rbp)
movl     16(%rbp), %edx
movl     %edx, %eax
addl     %eax, %eax
addl     %edx, %eax
popq     %rbp
ret
```

함수 twice와 차이나는 부분은 아래와 같다.

(1) movl 16(%rbp), %edx

매개변수 a값을 %edx 레지스터로 복사한다.

(2) movl %edx, %eax

%edx에 저장되어있는 값(= 매개변수 a)을 %eax 레지스터로 복사한다.

(3) addl %eax, %eax

%eax에 (%eax+%eax) 값을 저장한다. 따라서 %eax에는 a+a(=2*a)가 저장된다.

(4) addl %edx, %eax

%eax에 (%eax+%edx) 값을 저장한다. %eax에는 a+a, %edx에는 a가 저장되어있으므로, %eax에 a+a+a(=3*a)값이 저장된다.

즉 twice 함수는 매개변수 a에 대하여 (a+a)를 수행하고, triple 함수는 ((a+a)+a)를 수행한다.

4. 결론

(실험을 통해 배운 것을 정리합니다. 아래 Reference에 참고문헌을 추가합니다.)

스택프레임의 구조와, x86 어셈블리어를 다시 복습할 수 있었고, SEH(Structured Exception Handling)에 대해 새롭게 알게 되었다.

References

[1] Bryant, Randal E., and O'Hallaron, David R. “컴퓨터시스템”. 3판. 김형신 역. 퍼스트북. 2016

[2] Microsoft Learn. “x64 exception handling”. https://learn.microsoft.com/en-us/cpp/build/exception-handling-x64?view=msvc-170 (visited on 2024-09-14)