## C++ Tricks 2.4 I386 平臺C函數調用邊界 的棧分配 <sup>2</sup>

從 farseerfc.wordpress.com 導入

## 2.4 I386平臺C函數調用邊 界的棧分配

當調用一個函數時,主調函數將參數以聲明中相反的順序壓棧,然後將當前的代碼執行指針(eip)壓棧,然後跳轉到被調函數的入口點。在被調函數中,通過將ebp加上一個偏移量來訪問函數參數,以聲明中的順序(即壓棧的相反順序)來確定參數偏移量。被調函數返回時,彈出主調函數壓在棧中的代碼執行指針,跳回主調函數。再由主調函數恢復到調用前的棧。

函數的返回值不同於函數參數,通過寄存器傳遞。如果返回值類型可以放入32位變量,比如int、short、char、指針等類型,通過eax寄存器傳遞。如果返回值類型是64位變量,如\_int64,同過edx+eax傳遞,edx存儲高32位,eax存儲低32位。如果返回值是浮點類型,如float和double,通過專用的浮點數寄存器棧的棧頂返回。如果返回值類型是用戶自定義結構,或C++類類型,通過修改函數簽名,以引用型參數的形式傳回。

同樣以最簡單的函數爲例:

```
void f(){
int i=g(1,2);
}
int g(int a,int b){
int c=a+b;
return c;
}
```

```
產牛的彙編代碼如下:
f٠
push ebp ;備份ebp
mov ebp,esp;建立棧底
sub esp,4;爲i分配空間
mov eax,2:準備參數b的值2
push eax ;將b壓棧
mov eax,1:準備參數a的值1
push eax;將a壓棧
call g ;調用g
add esp,8;將a和b一起彈出,恢復調用前的棧
mov dword ptr[ebp-4],eax;將返回值保存進變量i
mov esp,ebp;恢復棧頂
pop ebp;恢復棧底
g:
push ebp ;備份ebp
mov ebp,esp;建立棧底
sub esp,4;爲局部變量c在棧中分配內存
```

mov eax,dword ptr[ebp+8] ;通過ebp間接讀取參數 a的值

mov ebx,dword ptr[ebp+12] ;通過ebp間接讀取參數b的值

add eax,ebx ;將a和b的值相加,之和存在eax中 mov dword ptr[ebp-4],eax ;將和存入變量c

mov eax,dword ptr[ebp-4] ;將c作爲返回值,代碼 優化後會刪除此句

add esp,4;銷燬c的內存

mov esp,ebp;恢復棧頂

popebp;恢復棧底

ret;返回函數f

棧的內存佈局如下:

100076:c <- g的esp

100080:f的ebp=100100 <- g的ebp

100084:f的eip

100088:a=1

100092:b=2

100096:i

100100:舊ebp <-f的ebp

100104:....

注意在函數g的彙編代碼中,訪問函數的局部變量和訪問函數參數的區別。局部變量總是通過將ebp減去偏移量來訪問,函數參數總是通過將ebp加上偏移量來訪問。對於32位變量而言,第一個局部變量位於ebp-4,第二個位於ebp-8,以此類推,32位局部變量在棧中形成一個逆序數組;第一個函數參數位於ebp+8,第二個位於ebp+12,以此類推,32位函數參數在棧中形成一個正序數組。

由於函數返回值通過寄存器返回,不需要空間分配等操作,所以返回值的代價很低。基於這個原因,舊的C語法約定,不寫明返回值類型的函數,返回值類型爲int。這一規則與現行的C++語法相違背,因爲C++中,不寫明返回值類型的函數返回值類型爲void,表示不返回值。這種語法不兼容性是爲了加強C++的類型安全,但同時也帶來了一些問題。