C++ Tricks 2.4 I386 平台C函数调用边界 的栈分配 □

从 farseerfc.wordpress.com 导入

2.4 I386平台C函数调用边界的栈分配

当调用一个函数时,主调函数将参数以声明中相反的顺序压栈,然后将当前的代码执行指针(eip)压栈,然后跳转到被调函数的入口点。在被调函数中,通过将ebp加上一个偏移量来访问函数参数,以声明中的顺序(即压栈的相反顺序)来确定参数偏移量。被调函数返回时,弹出主调函数压在栈中的代码执行指针,跳回主调函数。再由主调函数恢复到调用前的栈。

函数的返回值不同于函数参数,通过寄存器传递。如果返回值类型可以放入32位变量,比如int、short、char、指针等类型,通过eax寄存器传递。如果返回值类型是64位变量,如_int64,同过edx+eax传递,edx存储高32位,eax存储低32位。如果返回值是浮点类型,如float和double,通过专用的浮点数寄存器栈的栈顶返回。如果返回值类型是用户自定义结构,或C++类类型,通过修改函数签名,以引用型参数的形式传回。

```
同样以最简单的函数为例:void f(){
```

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

```
产生的汇编代码如下:
f:
push ebp;备份ebp
mov ebp,esp;建立栈底
sub esp,4;为i分配空间
mov eax,2;准备参数b的值2
push eax ;将b压栈
mov eax,1;准备参数a的值1
push eax ;将a压栈
call g;调用g
add esp,8;将a和b一起弹出,恢复调用前的栈
mov dword ptr[ebp-4],eax;将返回值保存进变量i
mov esp,ebp;恢复栈顶
pop ebp;恢复栈底
g:
push ebp;备份ebp
mov ebp,esp ;建立栈底
sub esp,4;为局部变量c在栈中分配内存
```

mov eax,dword ptr[ebp+8] ;通过ebp间接读取参数 a的值

mov ebx,dword ptr[ebp+12] ;通过ebp间接读取参数b的值

add eax,ebx ;将a和b的值相加,之和存在eax中 mov dword ptr[ebp-4],eax ;将和存入变量c

mov eax,dword ptr[ebp-4] ;将c作为返回值,代码 优化后会删除此句

add esp,4;销毁c的内存

mov esp,ebp;恢复栈顶

popebp;恢复栈底

ret;返回函数f

栈的内存布局如下:

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:旧ebp <-f的ebp

100104:....

注意在函数g的汇编代码中,访问函数的局部变量和访问函数参数的区别。局部变量总是通过将ebp减去偏移量来访问,函数参数总是通过将ebp加上偏移量来访问。对于32位变量而言,第一个局部变量位于ebp-4,第二个位于ebp-8,以此类推,32位局部变量在栈中形成一个逆序数组;第一个函数参数位于ebp+8,第二个位于ebp+12,以此类推,32位函数参数在栈中形成一个正序数组。

由于函数返回值通过寄存器返回,不需要空间分配等操作,所以返回值的代价很低。基于这个原因,旧的C语法约定,不写明返回值类型的函数,返回值类型为int。这一规则与现行的C++语法相违背,因为C++中,不写明返回值类型的函数返回值类型为void,表示不返回值。这种语法不兼容性是为了加强C++的类型安全,但同时也带来了一些问题。