用汇编语言解析C/C++函数调用中值传递、指针传递和引用传递的内在机制

火善栋[1]，杨旭东[2]

(1. 重庆三峡学院，重庆 404000；2. 重庆安全技术职业学院)

摘　要：为了让学生通过对汇编语言的学习加深对计算机内部原理的理解，借助汇编语言低级化的特点，详细分析了C/C++函数调用中值传递、指针传递和引用传递的内在实现机制。

关键词：汇编语言；函数调用；指针；引用；堆栈

中图分类号：N　　**文献标志码**：A　　**文章编号**：1006-8228(2012)09-49-02

Via assemble language to catch on mechanism of value transmit, pointer transmit and reference transmit in C/C++ function calling

Huo Shandong, Yang Xudong

(*1. Chongqing three gorges university, Chongqing 404000, China; 2. Chongqing safety technology profession college*)

Abstract: Since the assemble language is an important machine language of facing computer hardware directly, it can help to better understand the principle of computer. The characteristic of assemble language is introduced and it is explained in the paper how to realize value transmit, pointer transmit and reference transmit in the computer.

Key words: assemble language; function calling; pointer; reference; stack

0 引言

函数是C/C++程序设计中的一个很重要的知识点。在函数调用的过程中，函数参数的传递方式有三种，分别是值传递、引用传递和指针传递，这三者之间到底有什么区别，往往是很多初学者容易犯迷糊的地方。之所以会出现这种情况，其最终原因是由于他们没有从根本上理解这三种参数传递方式的内在实现原理。当然，很多教科书上已对其实现原理作了很详细的阐述，但总让初学者感到很抽象。正是基于这种原因，本文借助汇编语言低级化的特点，通过一个很简单的实例从底层对参数传递的实现原理作了很详细的分析和探讨，以帮助读者彻底理解这三者之间的区别。

1 实例分析

下面就借助VC6.0反汇编工具，对实现两个数据的交换的实例来详细的分析函数调用中三种参数传递方式的内在区别。

1.1 引用传递

相应数据交换的C++源代码和反汇编代码如表1所示。

从表1中可以看出，采用引用传递参数时，首先将变量m和n的值分别放入内存单元dword ptr [ebp-4]和dword ptr [ebp-8]中（由于m和n在main()函数中，因此也属于局部变量，所以实际上是将m、n的值压入到main()函数的堆栈中），如表一反汇编代码①所示。main()函数在调用swap函数以前，首先将实参m、n的地址按其所对应形参的顺序从右到左压入到被调用函数swap的堆栈中，如表1反汇编代码②所示；然后调用被调函数（调用以前先将返回地址eip的值压入堆栈），再将ebp的值压入堆栈并将esp的值赋给ebp；最后将esp的位置上移68个字节的空间以存放swap函数内局部变量的值（其反汇编代码省略）。其堆栈引用过程如图1所示。

表1　引用传递C++源代码和反汇编代码对照表

引用传递C++源代码	VC6.0对应的部分反汇编代码（……表示省略的代码）

#include <iostream.h>
void sawp(int &a,int &b);
void main(void)
{ int m=2,n=5;
sawp(m,n);
cout<<"m="<<m<<";"<<"n="<<n<<endl;
}
void sawp(int &a,int &b)
{ int temp=a;
a=b;
b=temp;
}
	0040107E call @ILT+10(sawp) (0040100f)
	00401083 add esp,8
	……
	void sawp(int &a,int &b)
	12: {
	00401110 push ebp
	00401111 mov ebp,esp
	……
	13: int temp=a;
	00401128 mov eax,dword ptr [ebp+8]
	0040112B mov ecx,dword ptr [eax] ③
	0040112D mov dword ptr [ebp-4],ecx
	14: a=b;
	00401130 mov edx,dword ptr [ebp+8]
	00401133 mov eax,dword ptr [ebp+0Ch] ④
	00401136 mov ecx,dword ptr [eax]
	00401138 mov dword ptr [edx],ecx
	15: b=temp;
	0040113A mov edx,dword ptr [ebp+0Ch]
	0040113D mov eax,dword ptr [ebp-4] ⑤
	00401140 mov dword ptr [edx],eax
	16: }
	……
	00401147 pop ebp
	00401148 ret

5: int m=2,n=5; 00401068 mov dword ptr [ebp-4],2 ① 0040106F mov dword ptr [ebp-8],5	
6: sawp(m,n); 00401076 lea eax,[ebp-8] 00401079 push eax 0040107A lea ecx,[ebp-4] ② 0040107D push ecx	

收稿日期：2012-6-8

作者简介：火善栋（1974-），男，湖北人，硕士，讲师，主要研究方向：智能信息系统。

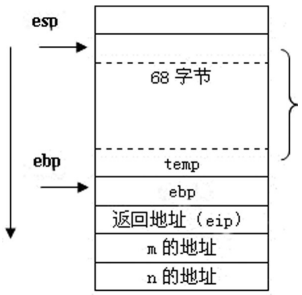

图1 传递引用时堆栈示意图(每一小行占有4个字节)

表1中,反汇编代码③、④、⑤通过ebp指针完成了变量a和b之间的数据交换。从反汇编代码中可以看出,这种交换实际上是完成了实参m、n所对应地址内容的交换,也就是说在swap函数中对形参a、b的操作实际上就是对实参m和n的操作。这是因为实参和形参共享同一个内存单元的缘故。

1.2 指针传递

相应的C++源代码和反汇编代码如表2所示。

表2 指针传递C++源代码和反汇编代码对照表

```
指针传递C++源代码
#include <iostream.h>
void sawp(int *a,int *b);
void main(void)
{ int m=2,n=5;
  sawp(&m,&n);
  cout<<"m="<<m<<":"<<"n="<<n<<endl;
}
void sawp(int *a,int *b)
{ int temp=*a;
  *a=*b;
  *b=temp;
}
VC6.0对应的部分反汇编代码
（……表示省略的代码）
……
5:          int m=2,n=5;
00401068    mov    dword ptr [ebp-4],2
0040106F    mov    dword ptr [ebp-8],5   ①
6:          sawp(&m,&n);
00401076    lea    eax,[ebp-8]
00401079    push   eax
0040107A    lea    ecx,[ebp-4]           ②
0040107D    push   ecx
0040107E    call   @ILT+30(sawp) (00401023)
00401083    add    esp,8
                ……
10:    void sawp(int *a,int *b)
11:    {
00401110    push   ebp
00401111    mov    ebp,esp
                ……
12:         int temp=*a;
00401128    mov    eax,dword ptr [ebp+8]
0040112B    mov    ecx,dword ptr [eax]   ③
0040112D    mov    dword ptr [ebp-4],ecx
13:         *a=*b;
00401130    mov    edx,dword ptr [ebp+8]
00401133    mov    eax,dword ptr [ebp+0Ch]
00401136    mov    ecx,dword ptr [eax]   ④
00401138    mov    dword ptr [edx],ecx
14:         *b=temp;
0040113A    mov    edx,dword ptr [ebp+0Ch]
0040113D    mov    eax,dword ptr [ebp-4] ⑤
00401140    mov    dword ptr [edx],eax
                ……
00401147    pop    ebp
00401148    ret
```

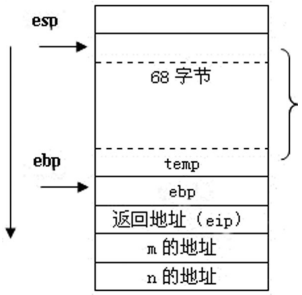

图2 传指针时堆栈示意图(每一小行占有4个字节)

从表2和图2中可以看出,通过指针传递实现两个数值的交换和通过引用传递实现两个数值的交换,其对应的反汇编代码和堆栈调用过程是完全相同的。也就是说,在函数的调用过程中引用传递和指针传递实际上都是将实际参数所在内存单元的地址压入到被调用函数的堆栈中,在被调用函数的内部,对形式参数操作是通过访问实际参数的地址来进行的,也就是说对形式参数的操作就是对实际参数的操作。引用传递和指针传递不同的是,引用传递时实际参数与形式参数完全等同;但指针传递时则要分两种情况,一种情况是对指针a、b(也就是地址)的操作,另一种就是对指针所指向的内存单元的内容*a、*b(也就是数值)的操作,后一种情况与引用传递是完全等同的,也就是本文中所示的情况,因此,也可以把引用传递看作是指针传递的一个特例。

虽然引用传递和指针传递本质上传的都是实际参数所对应内存单元的地址,但指针可以直接操作内存单元的地址,这无疑是指针的一大优势。在对数组和字符串操作上,指针则显得尤为方便和灵活,但这也会带来一些不安全的因素。因此,相比之下,引用传递比指针传递更安全;指针传递比引用传递功能更强大、更灵活。

1.3 值传递

相应C++源代码和反汇编代码如表3所示。

表3 值传递C++源代码和反汇编代码对照表

```
值传递C++源代码
#include <iostream.h>
void sawp(int a,int b);
void main(void)
{ int m=2,n=5;
  sawp(m,n);
  cout<<"m="<<m<<":"<<"n="<<n<<endl;
}
void sawp(int a,int b)
{ int temp=a;
  a=b;
  b=temp;
}
VC6.0对应的部分反汇编代码
（……表示省略的代码）
……
5:          int m=2,n=5;
00401068    mov    dword ptr [ebp-4],2
0040106F    mov    dword ptr [ebp-8],5   ①
6:          sawp(m,n);
00401076    mov    eax,dword ptr [ebp-8]
00401079    push   eax
0040107A    mov    ecx,dword ptr [ebp-4] ②
0040107D    push   ecx
0040107E    call   @ILT+35(sawp) (00401028)
00401083    add    esp,8
                ……
9:     void sawp(int a,int b)
10:    {
00401110    push   ebp
00401111    mov    ebp,esp
                ……
11:         int temp=a;
00401128    mov    eax,dword ptr [ebp+8]
0040112B    mov    dword ptr [ebp-4],eax ③
12:         a=b;
0040112E    mov    ecx,dword ptr [ebp+0Ch]
00401131    mov    dword ptr [ebp+8],ecx ④
13:         b=temp;
00401134    mov    edx,dword ptr [ebp-4]
00401137    mov    dword ptr [ebp+0Ch],edx ⑤
14:    }
                ……
0040113F    pop    ebp
00401140    ret
```

从表3中可以看出,采用值传递调用函数时,是将实际参数的值压入到被调用函数的堆栈中。其反汇编代码如表三①、②所示,其被调用函数堆栈示意图如图3所示。

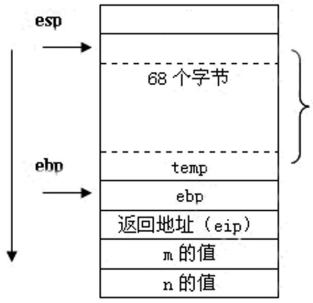

图3 传值调用时堆栈示意图(每一小行占有4个字节)

从堆栈数据分布情况来看,值传递与引用传递和指针传递惟一不同的是,在堆栈的底部前者存放的是实际参数的值,而后者存放的是实际参数在内存单元中的地址,而在函数内部,对形式参数a、b进行数值交换也是通过ebp指针来实现的(如表三③、④、⑤所示),这种交换也是完全通过调用函数堆栈内

(下转第54页)

点来梳理这两者之间的关系。就德州学院而言,应该设立一个大学生创新教研室,作为一个计算机类学生与社会之间的窗口,积极与社会联系,联合创办大学科技园。这不仅有利于产学研结合,把学生的创新成果转化为生产力,也利于培养优秀的创新创业人才,并能吸纳一大批毕业生到园中企业就业。

6.1 合理定位社会服务职能

德州学院属于地方院校,有着自己的地理优势,特别是计算机系理应成为德州地方IT行业的领头羊,切实把区域经济社会发展需求和自身情况进行合理的职能定位,切实将区域发展优势转变为自身优势,社会服务应该着眼于长久,不能为了短期的收益而放弃办学精神,也不能为服务而服务,应紧紧围绕"培养创新应用型人才"这一目标,以合作共赢,求真、务实精神积极吸取社会资源,进一步提升自身在地方上的竞争力。

6.2 针对社会需求积极参加社会性创业大赛

社会性大赛题目大都是通过对企业深入调查,急企业所需而选定的,参加大赛也是学校对外宣传和展示的一个社会服务窗口。最近三年,德州学院计算机系学生在各种赛事中取得了优异的成绩,在赛场上有的企业愿在现场与学生签定聘用合同,可以看出企业对具有创新能力人才的急需。但是在校学生还要继续在校学习,这就需要为企业和学校搭建平台,建立长期合作的实训实习基地或者共建研发实验室,让学生走出去,让企业进得来。要借助社会性创业大赛,做到以赛促教,以赛促发展,以赛促交流。

6.3 积极寻求切入点,建立共赢机制

地方高校要想得到发展,就要与地方的科研机构和企业进行合作,与他们共享科研成果,共谋发展。首先,高校的研究机构应积极地去企业进行调查研究,与他们共谋发展,共同解决技术难题;其次,在企业建立校企合作基地,有计划,有批次地派老师和学生参与企业的计划、生产;最后,将企业和高等学校的新课题、新技术互动共享,让学生积极参与课题的研究,使这些学生的成功体验起到表率作用,从而在全体学生中产生以点带面的效果。

7 结束语

地方高校要在目前大形势下把握机遇,开拓未来,只有不断地完善人才培养模式,使培养的人才不但能够胜任企业的现任岗位,也能很快适应不断变化的社会需求。要做到这一点,地方院校就应该积极拓展校企合作的途径,适时地调整和变换教学计划和培养模式,使高校真正成为培养优秀人才的摇篮,也成为地方企业发展的助力器。更进一步,地方高校还应积极参与企业创新共享资源平台的建设,成为地方科学技术创新的主导者和倡导者。

参考文献:

[1] 欧阳伦四.高校创业教育存在的问题及对策[J].教育探索,2011.3:103-105
[2] 冯理政,夏建国.校企合作应用型人才培养模式的创新探索[J].江苏技术师范学院学报,2009.3:68-69
[3] 王益萍.地方高校社会服务与应用型人才培养的互动模型研究[J].台州学院学报,2010.2:70-73
[4] 陆慧娟,高波涌.计算机专业创新型人才培养思考与实践[J].计算机教育,2008.20:156-158
[5] 刘迎春,熊志卿.应用型人才培养目标定位及其知识、能力、素质结构的研究[J].中国大学教学,2004.10:56-59
[6] 王旭东.论地方高校的社会服务职能的拓展[J].中国高教研究,2007.8:16-17
[7] 李海军.计算机类创新型应用人才培养模式与社会服务关系研究[J].计算机时代,2011.10:51-53
[8] 梁林梅,刘永贵,桑新民.高等教育信息化发展与研究论纲[J].现代教育技术,2012.1:5-9

(上接第50页)

的数据来进行的,只不过这种交换已经与实际参数没有任何关系了,这也是值传递与引用传递和指针传递所产生不同结果的根本原因所在。

2 结束语

在函数调用的过程中,引用传递和指针传递都是将实际参数所在内存单元的地址按其所对应的形式参数的顺序从右到左压入到被调用函数的堆栈中,而值传递只是将其值压入到被调用函数的堆栈中,在函数的内部,对形式参数的操作都是通过调用压入到被调用函数堆栈中的实际参数来实现的。

对于引用传递而言,实际参数和形式参数对应于相同的内存单元,因此,对形式参数的操作也就是对实际参数的操作;对于指针传递而言,在函数的内部,则分为两种情况,一种是对形式参数地址的操作,另一种是对形式参数数值的操作,后者完全等同于引用传递,前者则实际上是对实际参数地址的操作,这两种情况都对应于对实际参数的操作;对于值传递而言,在函数的内部,形式参数只是对应于被被压入的实际参数的值,而这个值只是实际参数的一个简单的拷贝,这个拷贝与实际参数没有任何联系,因此,对形式参数的操作对实际参数没有任何影响。

参考文献:

[1] 谭文等,天书夜读.从汇编语言到Windows内核编程[M].电子工业出版社,2008.
[2] 大善栋.通过汇编语言理解函数调用的内在机理[J].计算机时代,2010.7.
[3] 吕凤翥.C++语言程序设计[M].清华大学出版社,2003.
[4] 卜艳萍.汇编语言程序设计教程(第二版)[M].清华大学出版社,2008.
[5] 王晓东.C++程序设计简明教程[M].中国水利水电出版社,2008.
[6] 林锐.高质量程序设计指南—C++/C语言[M].电子工业出版社,2003.