混合编程时应注意的几个问题*

李海奎，郎璞玫
(中国林业科学研究院 资源信息所，北京 100091)

摘　要: 混合编程可以充分利用各种程序设计语言的优势。结合数值分析与统计软件 FORSTAT 设计和开发中的实践,针对混合编程时必须解决而又容易被忽略的几个问题:软件产品的形式、避免内存泄漏和协调异常处理机制,分析它们产生的原因,并给出了相应的解决方法,对于混合编程有一定的借鉴作用和指导意义。

关键词: 混合编程; 内存泄露; 异常处理
中图法分类号: TP311.52　　**文献标识码**: A　　**文章编号**: 1001-3695(2005)01-0167-02

Several Problems in the Mixed-Language Programming

LI Hai-kui, LANG Pu-mei
(Research Institute of Forest Resource & Information Techniques, CAF, Beijing 100091, *China)*

Abstract: Mixed-language programming could take use of advantage of various kind of programming language. Based on experiences of design and development in the process of FORSTAT: the numeric analysis and statistics software, several problems, which must be to solved and neglected easily in the mixed-language programming, including form of software, avoiding memory leak and reconciling exception handling, have been put forward. The paper analyses the context of their occurrence and gives solution alternatively, which has instructional significance to mixed-language programming.

Key words: Mixed-Language Programming; Memory Leak; Exception Handling

混合编程是指使用两种或两种以上的程序设计语言来开发应用程序的过程。目前,计算机上流行的程序设计语言有多种,它们有各自的优势和不足,如: VB 简单易学,界面开发能力强但执行速度较慢; C++可以操纵系统底层,运行速度快但编程麻烦; FORTRAN 擅长数值计算能力但编程可视化程度不强; MATLAB 矩阵计算能力强,但解释执行的特性使其不能脱离编程环境独立执行。程序设计语言的特性决定了在大型系统的开发中,一种语言往往不能满足项目的要求,混合编程不仅可以发挥多种语言的优势,实现代码共享,缩短开发周期,而且有利于多人协作,共同完成同一项目。混合编程时应遵循的规则在相关的文献中[1~3]多有讨论,包括函数的调用约定、参数的传递机制、过程和函数的命名约定等,在这里不作介绍。结合我们在数值分析和统计软件 FORSTAT 设计和开发中的实践,着重讨论混合编程时必须解决而又往往容易被忽略的几个问题。

1　软件产品的形式

混合编程最终形成的应用程序是否以软件产品的形式存在、软件产品是否发布、是独立发布还是必须外挂相应的编程语言,是混合编程时应该首先考虑的问题。如果混合编程的目的只是解决工作中的某一具体问题,最终不形成软件产品,则可以充分利用各种编程语言的集成开发环境,而不必考虑相关组件安装,因为这些组件已存在于各自的开发环境中。如果是以软件产品的形式存在,则必须考虑产品的发布形式: 独立发布时,产品必须能够脱离混合编程时各种程序语言的开发环境而运行; 外挂相应的编程语言时,除非要求用户必须安装相应程序语言,否则必须解决相应的版权问题,这是一个非常重要的问题。

在什么水平上的混合编程也是必须考虑的问题。如果在汇编层次上进行混合编程,即通过不同的编译器生成各自的目标文件(.obj 文件),然后链接生成可执行文件,则必须解决不同编译器内部的函数名约定问题或函数的修饰名问题; 如果使用组件和动态链接库等编程,则必须注意动态链接库的安装路径和组件的注册问题; 如果不同的程序设计语言生成的可执行文件之间互相调用时,要解决好进程间的同步或异步问题[4,5]以及数据的动态交换问题[6]。

2　避免内存泄漏

2.1　产生内存泄漏的原因

动态链接库是实现混合编程的主要方法之一。在这里,我们称生成动态链接库的设计语言为服务语言,调用动态链接库的程序设计语言为宿主语言。混合编程时,除了要求这两种语言的函数调用约定、参数数据类型和传递方式完全兼容外,内存管理机制的协调一致是一个必须解决的问题,否则极易造成内存泄漏。

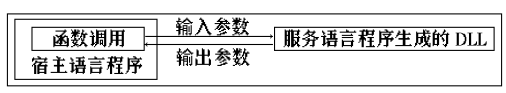

图 1　DLL 混合编程应用程序的执行方式

在图 1 中,当输入参数为数组时,实际传递到 DLL 的是指针,输入数组内存的分配和释放在宿主语言中完成; 当输出参

收稿日期: 2004-02-29; 修返日期: 2004-04-24
基金项目: 国家林业局“948”基金资助项目(2001-13)

数为数组时,如果数组的内存分配是在服务语言中动态完成的,而传回到宿主语言时的是指向数组第一个元素的指针,则存在内存泄漏的可能,因为若释放数组元素所占的内存,则宿主语言使用指针得不到正确的结果,而不释放内存,则下次调用此函数时,上次在服务语言中分配的内存不能被应用,这就造成了内存泄漏[7]。如 VB, VC 混合编程时,用 VC 编写矩阵相乘的函数,生成动态链接库,在 VB 中调用。

VC 代码的矩阵相乘函数为:

```
long_declspec( dllexport) _stdcall VCdllFunctionA( double * lpMatrix1, long Row1, long Col1, double * lpMatrix2, long Row2, long Col2, long * lpOutMatrix)
{   long i, j, k;
    if ( Col1! = Row2)
        return Error_Code;
    double * tt, ss;
    tt = new double[ Row1* Col2];    //动态分配所需的内存
    for ( i = 1; i < = Row1; i + + )
      for ( j = 1; j < = Col2; j + + )
      {  s = 0;
        for ( k = 1; k < = Row2; k + + )
          ss + = lpMatrix1[ ( k - 1) * Row1 + i - 1] * lpMatrix2[ ( j - 1) * Row2 + k - 1];
          tt[ ( j - 1) * Row1 + i - 1] = ss;
      }
      * lpOutMatrix = ( long) &tt[ 0];
      //获得数组首个元素的地址
      return Success_Code;
}
```

VB 调用时首先要进行声明:

```
Option Base 1
Private Declare Function VCdllFunctionA Lib "vcdll.dll" (IntputArray1 As Double, ByVal row1 As Long, ByVal col1 As Long, InputArray2 As Double, ByVal row2 As Long, ByVal col2 As Long, OutputArray As Long) As Long
Private Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMe～mory" (Destination As Any, Source As Any, ByVal Length As Long)
```

相应的调用过程为:

```
Success = VCdllFunctionA( Matrix1( 1, 1), RowofMatrix1, ColofMatrix1, Matrix2( 1, 1), RowofMatrix2, ColofMatrix2, lpOutMatrix)
If Success = Success_code Then
CopyMemory ByVal VarPtr( OutMatrix( 1, 1)), ByVal lpOutMatrix, RowofMatrix1 * ColofMatrix2 * 8
End If
```

调用时,假设相应的输入变量已经赋值,输出变量已经定义。

2.2 解决方法

一般来讲,可以用三种方法来解决内存泄漏问题:一是在宿主语言中操纵服务语言,释放在服务语言中分配的内存;二是在宿主语言中分配服务语言所需要的内存,这种方法要求宿主语言能够动态地分配内存;三是宿主语言和服务语言遵循共同的规范,如 COM。在上面的例子中,只需把 VC 函数参数的最后一个改为指向 Double 型的指针,而实际的计算语句改为 lpOutMatrix[(j - 1) * Row1 + i - 1] + = lpMatrix1[(k - 1) * Row1 + i - 1] * lpMatrix2[(j - 1) * Row2 + k - 1];动态分配内存的语句 tt = New Double[Row1 * Col2] 和指针赋值 * lpOutMatrix = (long) &tt[0] 均不需要;调用只需最后的一个参数改为需要返回数组的第一个元素即可(API 函数 CopyMemory 已不需要)。

在 VB 和 MATLAB 混合编程时,也可使用同样的方法解决内存泄漏问题。

3 协调异常处理机制

运行时出现异常是任何程序都不可避免的。在使用动态链接库的方法进行混合编程时,由于宿主语言和服务语言异常处理机制不同,必须协调一致,才能及时地根据异常类型进行相应的处理,保持程序的健壮性。一般来讲,有两种不同的异常处理机制:一是通过函数的返回值来确定异常情况,如 Win32 API 函数;二是通过填充全局的异常结构来确定异常的类型、来源和描述,如 VB 中的 Err 结构。

3.1 数值计算时运行异常

数值计算时的运行异常主要包括函数超出定义域、被零除、计算结果溢出(上溢和下溢) 。

表 1 不同程序设计语言的运行时异常返回信息

异常类型	VB(错误号, 描述)	VC++	MATLAB
函数超出定义域	5 Invalid Procedure Call or Argument	- 1. #IND	由于在复数范围进行计算不会出现
a/b(a = b =0)	6 Overflow	- 1. #IND	NaN
a/0(a≠ 0)	11 Division by zero	1. #INF(a > 0) - 1. #INF(a < 0)	Inf(a > 0) - Inf(a < 0)
溢出	6 Overflow	1. #INF, - 1. #INF	Inf, - Inf

在表 1 中,由于 MATLAB 计算的数域是复数,所以在实数域中可能出现的函数超出定义域的运行异常,在 MATLAB 中不会出现,但由于其计算结果是由实数部和虚数部两部分组成,我们仍可以通过判断虚数部是否存在来确定计算在实数域中是否超出定义域。对于其他运行异常,MATLAB 的计算结果遵循 IEEE 的数值计算标准。函数超出定义域和 a/b(a = b = 0) 其实都属于函数参数取值的错误,在 VC++ 中,它们是一致的,但在 VB 中分属不同的异常类型;上溢和被 0 除的结果都是数值超出计算机中所能表示的最大数(指绝对值) , VB 和 VC++ 的处理也不相同。实际上,数值计算发生运行异常时, VC++ 和 MATLAB 均通过返回值来体现异常类型,计算还会继续;而 VB 则立即停止,并没有返回值。

3.2 协调异常处理

所谓协调异常处理,就是混合编程时,在宿主语言中兼容服务语言的异常处理机制,把在服务语言中出现的运行异常,正确地传递给宿主语言,按宿主语言的异常处理机制确定相应的处理方法。设宿主语言是 VB,服务语言是 VC++ ,下面是一个实际的例子:

```
On Error GoTo hanErr '激活处理处理机制
…
ReturnVal = vcDllFunction( par1, par2) '调用 VC 函数
If InStr( CStr( ReturnVal), ". #IND") > 0 then
    ErrDescription = "函数超出定义域"
    ErrNumber = 5
  GoTo hanErr
End If
If InStr( CStr( ReturnVal), ". #INF") > 0 then
    ErrDescription = "计算结果溢出"
    ErrNumber = 6
```

(下转第 181 页)