```
26          exit(1);
27      }
28
29      /* Now we can call addvec() just like any other function */
30      addvec(x, y, z, 2);
31      printf("z = [%d %d]\n", z[0], z[1]);
32
33      /* Unload the shared library */
34      if (dlclose(handle) < 0) {
35          fprintf(stderr, "%s\n", dlerror());
36          exit(1);
37      }
38      return 0;
39  }
```

code/link/dll.c

图 7-17 （续）

旁注 共享库和 Java 本地接口

Java 定义了一个标准调用规则，叫做 Java 本地接口(Java Native Interface，JNI)，它允许 Java 程序调用“本地的”C 和 C++ 函数。JNI 的基本思想是将本地 C 函数(如 foo)编译到一个共享库中(如 foo.so)。当一个正在运行的 Java 程序试图调用函数 foo 时，Java 解释器利用 dlopen 接口(或者与其类似的接口)动态链接和加载 foo.so，然后再调用 foo。

7.12 位置无关代码

共享库的一个主要目的就是允许多个正在运行的进程共享内存中相同的库代码，因而节约宝贵的内存资源。那么，多个进程是如何共享程序的一个副本的呢？一种方法是给每个共享库分配一个事先预备的专用的地址空间片，然后要求加载器总是在这个地址加载共享库。虽然这种方法很简单，但是它也造成了一些严重的问题。它对地址空间的使用效率不高，因为即使一个进程不使用这个库，那部分空间还是会被分配出来。它也难以管理。我们必须保证没有片会重叠。每次当一个库修改了之后，我们必须确认已分配给它的片还适合它的大小。如果不适合了，必须找一个新的片。并且，如果创建了一个新的库，我们还必须为它寻找空间。随着时间的进展，假设在一个系统中有了成百个库和库的各个版本库，就很难避免地址空间分裂成大量小的、未使用而又不再能使用的小洞。更糟的是，对每个系统而言，库在内存中的分配都是不同的，这就引起了更多令人头痛的管理问题。

要避免这些问题，现代系统以这样一种方式编译共享模块的代码段，使得可以把它们加载到内存的任何位置而无需链接器修改。使用这种方法，无限多个进程可以共享一个共享模块的代码段的单一副本。(当然，每个进程仍然会有它自己的读/写数据块。)

可以加载而无需重定位的代码称为*位置无关代码*(Position-Independent Code，PIC)。用户对 GCC 使用-fpic 选项指示 GNU 编译系统生成 PIC 代码。共享库的编译必须总是使用该选项。

在一个 x86-64 系统中，对同一个目标模块中符号的引用是不需要特殊处理使之成为 PIC。可以用 PC 相对寻址来编译这些引用，构造目标文件时由静态链接器重定位。然而，对共享模块定义的外部过程和对全局变量的引用需要一些特殊的技巧，接下来我们会谈到。