

Linux 内核模式下 C++ 语言的导入研究

彭 磊, 吴 磊, 叶娅兰

(电子科技大学计算机学院, 成都 610054)

摘 要: 使用 C++ 代替 C 作为 Linux 内核环境开发语言, 有利于将面向对象的设计与编程引入 Linux 内核。但是在 Linux 内核中保持 C++ 与内核的兼容性和自身的语言特性是一个具有挑战性的问题。该文对 Linux 内核模块装载机制和 C++ 语言在 Linux 内核模式下内存分配机制进行了分析, 实现了在 Linux 内核开发中应用 C++ 语言的切实可行的方法。

关键词: Linux; 内核模块; C++

Research on C++ Language Inside Linux Kernel

PENG Lei, WU Lei, YE Ya-lan

(School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610054)

【Abstract】 Rather than C language, C++ is chosen as the language working in Linux kernel with the purpose of realizing object-oriented design and programming in kernel environment. However, it is a challenge that how to keep the compatibility between Linux kernel and C++. This paper analyzes the mechanism of kernel modules loading. Also the measure of memory allocation adopted by C++ in the kernel is discussed. As a result, it finds a feasible way to apply C++ language to kernel modules development.

【Key words】 Linux; kernel module; C++

1 概述

Linux 作为当今世界上最大的开源操作系统, 其分布于各地的开发者从两个渠道对 Linux 内核进行升级和维护: 一是单纯的内核开发者基于 Linux 单体式(monolithic)内核^[1]本身进行维护, 不断发布内核补丁^[2]; 二是驱动设备提供商基于 Linux 内核的可装卸模块机制^[3]开发出支持各种外设的驱动程序, 提高 Linux 的可用性。而现今从事于嵌入式 Linux 产品研发的人员, 都是以开发内核模块为主^[4]。

由于历史的原因, Linux 内核一直是以 C 作为其开发语言, 而与内核密切相关的内核模块, 开发者们在基于如下的理由下, 也纷纷选择了 C 作为内核模块的开发语言: (1) 由于内核和模块都是用 C 写出的, 因此在动态链接的时候不会发生名字解析(naming convention)的错误。(2) 现今经典的指导性的书籍和资料^[5]都是以 C 作为内核模块开发语言。(3) 用 C 开发出的内核模块运行速度比较快。

然而在 Windows 上看到了这样的现象: Microsoft 为驱动程序的开发者们提供了 DDK, 该套件提供了 Windows 的内核函数接口, 供 Window 设备驱动程序(也是一种可动态装卸的模块)调用。但是, 直接使用 DDK 的开发人员并不多, 多数的开发人员转而使用一个基于 DDK 之上的第三方套件: Driver studio^[6]。该套件的最大特色在于使用面向对象的思想构建设备驱动程序, 将一个个具体的驱动程序实现为一个 C++ 语言中的类(class)。由于引入强大的 C++ 语言和面向对象的设计思想, 驱动程序的开发纳入了正规的软件开发管理流程, 包括使用 RUP^[7]来全程建模。

因此, 在 Linux 上也很有必要在内核模块的级别上引入 C++ 语言和面向对象的设计思想, 这种引入可以带来的预期收益包括: (1) 驱动程序的工程化特征大大增强, 可以获得强大 CASE 工具支援, 包括使用 UML 语言建模。而这种建模的

结果转化为 C++ 语言代码的成本很小。(2) 在驱动程序的设计过程中, 可以引入适合的设计模式, 大大增强代码可复用度^[8], 提高驱动程序的鲁棒性。(3) 可将 C++ 语言最新的一些特性和运用, 比如泛型编程^[9], 引入到内核级别, 实现一系列的容器和算法, 大大简化驱动程序的开发工作。

以 C++ 语言进行 Linux 内核开发成了上述前景的根本。但是, 由于内核模块和普通应用程序的差异以及内核缺乏 C++ 运行库的支持, 因此在内核中支持 C++ 变得很有挑战性。事实上, 很少有文献论述这一话题。笔者在对 Linux 内核和 C++ 对象模型作仔细的研究后, 使用基于 C++ 的内核模块, 实现了一系列满足 STL 标准接口的容器。

2 内核模式下运行 C++ 的基本要求

用 C++ 编写的内核模块必须满足以下要求才能提供上述优点: (1) 内核仍然能够对这些模块动态地加载和卸载; (2) C++ 语言重要的特性仍然得以保留, 诸如继承、虚函数等。

2.1 C++ 与内核的兼容

由 C 写成的内核, 必然会同用 C++ 写出的模块存在兼容性问题。一个典型的兼容问题会发生在编译期, 本文写出一个模块, 用如下编译参数^[5,10]进行编译时可以发现 g++ 会产生如下编译错误:

```
g++-DMODULE-D__KERNEL__-DMODVERSIONS
-MD-O-c-I/usr/src/linux/include
-include/usr/src/linux/include/linux/modversions.h -fno-rtti -fno-
exceptions -fcheck-new
```

这是由于在 C 写出的头文件中使用了 C++ 的一些关键字, 如 new, typename 等, 因此在引入任何原有的 Linux 内核

作者简介: 彭 磊(1977 -), 男, 博士研究生, 主研方向: 嵌入式软件构建; 吴 磊、叶娅兰, 博士研究生

收稿日期: 2006-12-10 **E-mail:** penglei77@126.com

头文件时，必须利用预处理宏将头文件中的 C++ 关键字重定义，引用结束后，再恢复 C++ 关键字原有的含义。

```
#ifndef __cplusplus
#define new          cxx_new
... //转义所有 C++特有的关键字
#endif
#include "linux_import.h"
#ifdef __cplusplus
#undef new
... //恢复所有 C++特有的关键字
#endif
```

为便于处理，可使用称为头文件包裹(head file wrapper)的技巧：(1)创建一个头文件，该头文件仅仅使用#include 包含其他的 Linux 内核头文件，即为包裹头文件；(2)创建另一个头文件，在此文件中统一处理关键字重定义。

编译完成后，如果使用 objdump 查看此模块信息，会发现和一般 C 写出的模块有两大不同：(1)在模块的节表中没有 .modinfo 节，模块延迟绑定的内核函数被 C++ 进行了名字解析；(2)模块的装载点和卸载点函数也被 C++ 进行了名字解析。

上述两种情况会造成实用工具 insmod 无法进行模块装载。insmod.c 表明其装载流程如图 1 所示。

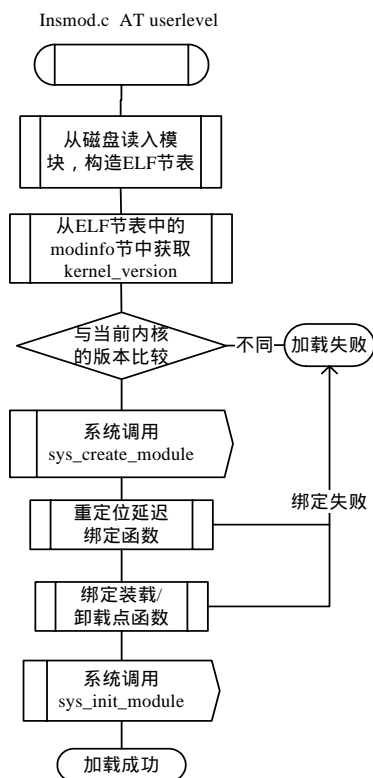


图 1 C++ 关键字兼容问题的解决方法

此过程中，如果不存在 .modinfo 节或者 .modinfo 节中没有 kernel_version 字段，则加载失败，insmod 会返回找不到模块内核版本的错误信息。如果 insmod 在重定位过程中发现不能在 kernel 导出函数表 (/proc/ksyms) 为模块的延迟绑定函数找到一个匹配项，insmod 则会返回不能定位函数的错误信息。最后，如果 insmod 不能在符号表中找到 init_module 和 cleanup_module 这两个符号，insmod 则会报告找不到加载点的错误。

由 C++ 编写的模块缺少 .modinfo 节，这是由 g++ 编译优化

造成的。在 module.h^[2] 文件中，原始的对 kernel_version 的定义如下：

```
static const char __module_kernel_version[] __attribute__((section(".modinfo"))) = "kernel_version=" UTS_RELEASE;
```

由于该变量从未被使用过，因此在编译时被 g++ 优化删除。解决这个问题，则必须显示指定该变量不能作优化删除，即为变量指定 volatile 属性。修改后如下：

```
volatile static char __module_kernel_version[] __attribute__((section(".modinfo"))) = "kernel_version=" UTS_RELEASE;
```

而对于名字解析的问题，需要显式抑制 C++ 的名字解析机制。方法是将必须保持名字不变的函数和变量放置于 extern "C" {} 语句块中，强制这些函数和变量以 C 的方式进行名字解析。

```
#ifndef __cplusplus
#define new          cxx_new
... //转义所有 C++特有的关键字
#endif
extern "C" {
#include "linux_import.h"
}
#ifdef __cplusplus
#undef new
... //恢复所有 C++特有的关键字
#endif
```

这样编译后的模块就可以顺利地加载和卸载了。

2.2 C++ 语言特性的保持

由 g++ 编译器产生的 C++ 对象的内存布局在操作系统内核和用户态并无两样，只不过一个放置于内核空间，而另一个放置于用户空间。基于此，C++ 语言的封装性和继承性在内核模式下得以保存。但是，多态性必须要依赖于 C++ 对象动态创建和销毁的能力，但是在内核模式下，没有用户态程序的堆区域，没有 C++ 运行库的支持，必须自己维护一个处于内核模式下的“堆区域”和分配策略。解决这一问题的唯一途径就是重载 operator new 及其 operator delete^[11]。

C++ 的关键字 new 在进行 C++ 对象动态创建时，分为两个阶段：

(1) 获取存储空间。在此阶段，new 关键字会查找调用要生成对象的 operator new，如果该操作符存在，则由后者完成分配内存的实质性工作。否则将使用全局的 operator new 获得内存。这部分内存空间在 C++ 运行库的支持下，默认属于用户程序的堆区域。

(2) 在已获得的内存空间里进行对象的初始化工作。即调用对象的构造函数。

因此，可以据此在内核中完成相似的功能。在 Linux 内核中，最为常见的获取内存的函数是 kmalloc，此函数使用了内核存储分配器 Slab 的普通高速缓存，分配对象的最大限制为 131 072 字节。可以使用它来作为全局的 operator new。同样，也可以使用 kfree 作为全局的 operator delete。

```
void* operator new(size_t size)
{ return kmalloc(size, GFP_KERNEL); }
void operator delete(void* deadobject)
{ kfree(deadobject); }
```

但是，对于一个具体的 C++ 对象而言，普通高速缓存并不能使内存利用率和分配速度达到最佳。要达到这个目的，必须定制对象专属的分配符和高速缓存。在 Linux 内核中，可使用 kmem_cache 函数族^[5]完成此功能。但是，作为一个可工

程化的方法,没有必要手工为所有C++对象添加这样的代码,而是依靠代码自动生成机制。C++中,能达到这种目的的手段有两种:宏(macro)和模板(template)。本文介绍以宏实现的形式。

operator new 和 operator delete 直接牵涉到对象专属的高速缓存,因此不能通过继承进行代码复用,而是应该用显式的专属函数实现。

```
#define DECLARE_CXXOBJECT(x) private: \
static kmem_cache_t* x##_objbuffer; \
public: \
void* operator new(size_t size); \
void operator delete(void* deadobject); \
static int x##_createobjectpool(); \
static int x##_destroyobjectpool();
```

上面的配置宏必须放置在一个类的声明中(.h 文件),除了声明这个类 x 的分配与销毁操作符外,同时声明了高速缓存的描述符指针以及创建高速缓存和销毁高速缓存的两个接口。由于高速缓存的声明与一个具体 x 类的名称相关,因此在代码自动生成时,不同的类具备不同的高速缓存操作接口,这保证了对象高速缓存的专属性。相应地,笔者制定了上述接口的实现宏,该宏应该在这个类的实现文件(.cc)中使用。其核心部分如下:

```
#define IMPLEMENT_CXXOBJECT(x) \
kmem_cache_t* x::x##_objbuffer=0; \
int x::x##_createobjectpool() \
{ \char s[256]={0}; \
sprintf(s,"%s",#x); \
x##_objbuffer=kmem_cache_create(s,sizeof(x),0,SLAB_HWCACHE_ALIGN,0,0); \
return 0; \} \
int x::x##_destroyobjectpool() \
{ \return (x##_objbuffer)? kmem_cache_destroy(x##_objbuffer):0; \} \
void* x::operator new(size_t size) \
{ \if(x##_objbuffer) \
return kmem_cache_alloc(x##_objbuffer,GFP_ATOMIC); \
else \
return NULL; \} \
void x::operator delete(void* deadobject) \
{ \if(x##_objbuffer) \
kmem_cache_free(x##_objbuffer,deadobject); \}
```

通常,应该在模块的装载点调用类的 Classname_createobjectpool 接口,创建专属缓存,便于动态创建对象之需。而在模块的卸载点销毁专属缓存,通过调用 Classname_destroyobjectpool 实现。

一个内核中 class 定义的例子如图 2 所示。

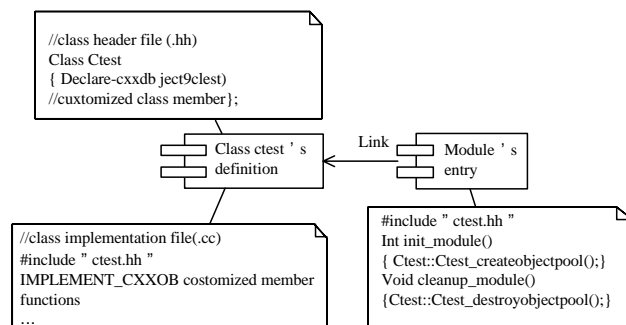


图 2 类定义的例子

3 内核模式下 C++的扩展问题

上一节所述的工作已经可以使得 C++在内核下具备了实用性,但是,本文的目的是最大化地保持 C++的外观,使开发人员在使用它时,并不会感觉到和在用户态下有巨大的差异。下面围绕这一目的讨论 C++全局对象和模板与泛型。

3.1 C++全局对象

在用户态程序下,开发人员往往会在 main 函数之外放置一些 C++全局对象。这些对象的内存空间位于.bss 段内,同时它们的构造函数的调用点位于 main 函数之前的一段自动链接的代码中。而它们的析构函数的调用点也是位于 main 函数结束之后一段隐藏的代码中。

在内核模块中,全局对象的内存空间仍然会布局在.bss 段中,但是从module.c^[2]中得知,模块执行的第 1 条语句必然位于init_module之中,这样,全局对象的初始化必须由开发者自己承担。可以利用 placement operator new完成这一功能。placement operator new 是C++中 operator new的一个特例,它使用调用者指定的地址进行内存分配。如果为new关键字指定这样的内存分配策略,就能为已经分配好空间的 C++全局对象调用该对象的构造函数。

```
//placement operator new
void* operator new(size_t size,void* p)
{
    return p;
}
...
CA a;
```

3.2 模板与泛型

模板属于编译期多态的技术,因此在内核模式下该特性不受任何影响。但是,这并不意味着开发人员可以顺利地在内核中使用 STL 来展现泛型的思想。不过,STL 设计得如此之好,以致它早就将算法、容器和内存分配分离了,若要将 STL 移植到内核模式下,本文所做的就是提供一个适合于内核模式下的空间配置器(allocator)。在本文的第 2 节已经讨论了一些在内核中分配内存的方法。可以根据它们做出一个空间配置器,其接口和实现可参见文献[12]。

4 结束语

使用 C++语言作为 Linux 内核模块开发,是一种新的尝试,它能给开发者带来诸多优点:可以提高模块设计水平,缩短模块开发时间,保证模块代码质量。本文通过分析模块工作机制,比较用户态程序和内核模块的区别,详细地探讨了如何将 C++应用于内核模式中,并尽可能保证 C++的语言特性。将 C++语言引入内核模块开发,强化了驱动程序面向对象的设计思想,为最终创建出统一的 Linux 驱动程序开发框架打下了坚实的技术基础。

参考文献

- 1 Bach M J. The Design of the UNIX Operating System[M]. [S. l.]: Prentice Hall, 1990.
- 2 Source Files of Linux Kernel[Z]. (2006-08). <http://www.kernel.orgs>.
- 3 Juan-Mariano de G. Loadable Kernel Modules[J]. IEEE Software, 1999, 16(1): 65-71.
- 4 Lennon A. Embedding Linux[J]. IEEE Review, 2001, 47(3): 336.
- 5 Rubini A. Linux Device Drivers[M]. 2nd ed. [S. l.]: O'Reilly & Associates, 2001.

(下转第 11 页)