浙江科技学院学报,第 18 卷第 2 期,2006 年 6 月
Journal of Zhejiang University of Science and Technology
Vol. 18 No. 2, June 2006

Linux 内核中面向对象思想的研究与应用

钱亚冠
(浙江科技学院 理学院,杭州 310023)

摘　要：运用 C 语言这样的面向过程语言进行程序设计时,传统上是使用结构化的分析设计方法,但是能否在 C 语言开发中运用面向对象的思想进行设计却是一个值得探索的问题。然而,通过分析 Linux 内核代码,C 语言在 Linux 内核开发中贯穿面向对象的设计思想的技术理念得到了阐述,而且还总结了将其应用于软件工程实践中的可行性。

关键词：Linux 内核;面向对象程序设计;封装;继承;多态
中图分类号：TP311.11　　**文献标识码**：A　　**文章编号**：1671-8798(2006)02-0111-03

Research and Application of Object-Oriented Thinking in Linux Kernel

QIAN Ya-guan
(School of Science, Zhejiang University of Science and Technology, Hangzhou 310023, China)

Abstract: When programming with procedure-oriented language like C, traditionally, the structure analysis and design method is usually used. But, it is worthy of exploring how to combine with Object-oriented (OO) thinking in programming in C. The technology and feasibility about a combination of programming in C and OOD is studied on the basis of analysis of Linux kernel codes.

Key words: Linux kernel; object-oriented programming; encapsulation; inheritance; polymorphism

C 语言程序具有高效、简洁、可移植性好等优点,因此广泛使用于操作系统等系统软件的实现。但 C 语言本身不具有面向对象语言的特征,而面向对象的设计思想已经在当今的软件工程领域中显示出了强大的优越性。如何在 C 语言开发的系统软件领域引入面向对象的设计思想呢? Linux 操作系统提供了一个很好的典范,本文正是结合分析 Linux 内核源代码,详细分析如何在 C 语言开发的软件中实现了 OO 思想,并将这种思想应用到笔者的软件开发实践中。

1　面向对象的程序设计

面向对象的程序设计具有三个基本特征:封装、继承、多态[1]。

1.1　封装(Encapsulation)

封装就是把过程(方法)和数据(属性)包装起来,实现信息隐蔽和抽象。定义了一个对象的属性,就是确定哪些属性对其他外部对象可见,哪些属性表示内部状态。对属性数据的访问只能通过已定义的界面。封装保证了模块具有较好的独立性,使得

收稿日期：2006-03-09
作者简介：钱亚冠(1976—　),男,浙江嵊州人,助教,硕士,主要从事计算机教学和嵌入式软件开发。

程序维护修改较为容易。

1.2 继承(Inheritance)

继承是一种联结类的层次模型,它提供了一种明确表述共性的方法,方便了类的重用。一个新类可以从现有的类中派生,这个过程称为类继承。新类继承了原始类的特性,新类称为原始类的派生类(子类),而原始类称为新类的基类(父类)。派生类可以从它的基类那里继承方法和实例变量,并且类可以修改或增加新的方法使之更适合特殊的需要[2]。

1.3 多态(Polymorphism)

派生类可以像基类对象一样使用,同样的消息即既可以发送给基类对象,也可以给派生类对象。即在类等级不同层次中可共享一个行为的名字,但不同层次中的类却按自己的需要来实现这个行为。当一个对象接收到发送给它的消息时,根据对象所属的类动态选用该类中定义的实现算法。

2 Linux 代码中的"类"与对象封装

面向对象语言中的类概念与传统过程语言的数据类型在数据封装和抽象上非常相似。C 语言提供 typedef 保留字用于定义新的数据类型,可以用它来模仿类的概念,以结构类型实现数据和方法的封装。以文件系统中的 file 对象[3]为例:

```
struct file {
    struct list_head    * f_list;
    struct dentry       * f_dentry;
    struct vfsmount         * f_vfsmnt;
    struct file_operations    * f_op;     / * 指向特定文件系统的操作集 * /
    ……
    loff_t              f_pos;
    unsigned long       f_reada, f_ramax, f_raend, f_ralen, f_rawin;
    ……
};
```

内核给 file 对象定义了很多的属性,其中方法通过指针 * f_op 指向它的操作集:

```
struct file_operations {        / * vfs 文件系统操作集,可看作是类方法定义 * /
    ……
    ssize_t ( * read) (struct file * , char * , size_t, loff_t * );
    ssize_t ( * write) (struct file * , const char * , size_t, loff_t * );
    int ( * open) (struct inode * , struct file * );
    ……
};
```

从上面的定义看,已经具有抽象类的概念:file 对象只定义了方法的原型(虚函数),而没有实现。具体方法的实现由具体的文件系统提供。内核中的重要数据结构都进行了封装,如文件系统中的 inode 对象,内存管理中的 page 对象、cache 对象和 slab 对象,网络实现部分中的 socket 缓冲区对象 struct sk_buff 等。

3 Linux 代码中的"继承"特性

由于 C 语言本身没有提供支持继承的语言要素,所以从程序语言的角度来看,是无法实现"继承"的。但从"继承"的思想和目的来看,就是让子类能够共享父类的数据和方法,同时又能在父类的基础上定义扩展新的数据成员和方法,从而消除类的重复定义,提高软件的可重用性。从这个角度分析容易发现其中的"继承"思想。典型的例子是内核 2.4 版本新提出的通用链表结构(linux/list.h)[3]:

```
struct list_head {        / * 通用双向链表对象 * /
    struct list_head  * next, * prev;
};
```

可以把这个对象看作一个基类,对它的基本操作是链表节点的插入,删除,链表的初始化和移动等。其他数据结构(可看作子类)如果要组织成双向链表,可以在链表节点中包含这个通用链表对象(可看作是继承)。一个简单的例子是时钟链表:

```
struct timer_list {
    struct list_head list;
    unsigned long expires;
    unsigned long data;
    void ( * function)(unsigned long);
};
```

这是个定时器队列,它"继承"了通用队列 list_head,见图 1。

链表的本质就是一个线性序列,其基本操作为插入和删除等,不同链表间的差别在于各个节点中存放的数据类型,因此把链表的特征抽象成这个通用链表,作为"父类"存在,具体不同的链表则"继承"这个"父类"

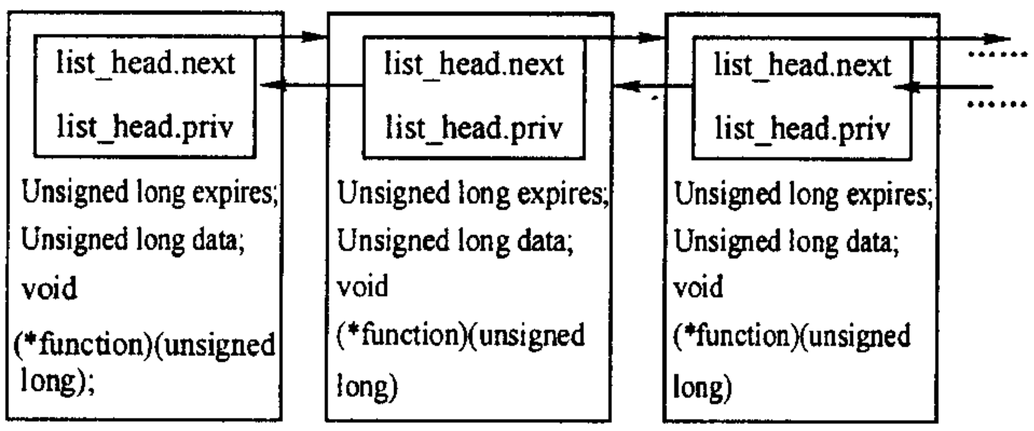

图 1　扩展(继承)了通用链表的定时器队列

的基本方法,同时扩充自己的私有属性。如本例中,定时器添加了自己的是私有数据:expires,data,还有方法 function()。使用通用链表的数据结构遍及内核的各个部分:进程调度,存储器管理,文件系统,网络协议栈等。

4　Linux 代码中的"多态"

这里借用 C++的多态概念来分析内核代码中的"多态"特性。多态的两种基本形式:编译时多态与运行时多态[1]。

4.1　运行时多态

运行时多态指程序运行时才可确定的多态性,主要通过继承和虚函数获得。虚函数就是允许其子类对其重新定义(overriding),具体运行时调用父类还是子类的这个同名函数,由运行时提供的具体对象类型决定,在编译阶段不能确定,因此称滞后联编,或动态绑定(dynamic binding)。

内核中的典型例子是虚拟文件系统(VFS)。Linux 为了支持不同的文件系统,引入了虚拟文件系统的概念。虚拟文件系统就是为用户程序提供一个统一、抽象的文件系统界面,这个界面由一组标准的文件操作构成,如 read(),write()等,并由各具体文件系统(如 ext2)实现[4]。这些对用户程序透明,他们见到的只是 VFS 提供的这组标准接口,而不关心具体由谁实现。

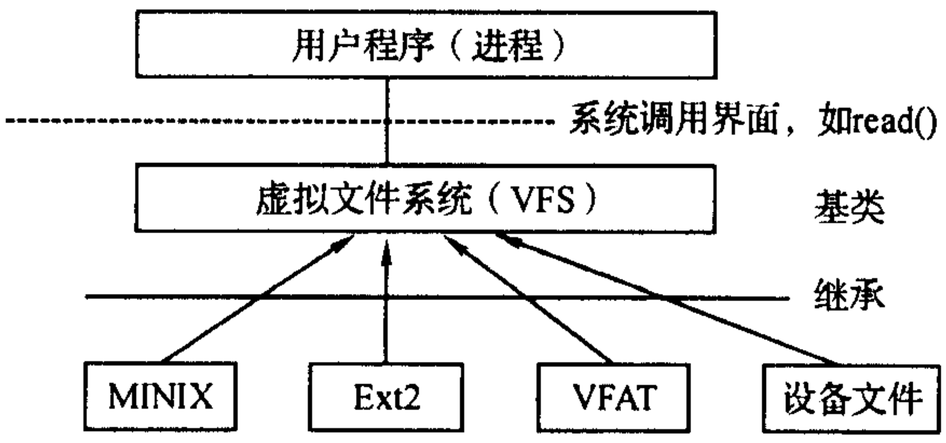

图 2　Linux 文件系统中的"类"层次

从图 2 中可以看出,VFS 可看作抽象基类,而具体文件系统如 Ext2、VFAT 可看作它的子类,VFS 提供的操作函数集(如 struct file_operations)是一个纯虚函数的概念,具体实现由实现文件系统提供。用户程序调用 VFS 提供的 API 进行编译时,并不能确定具体由哪个文件系统的实现函数来完成操作。而只有在系统运行时,根据用户进程具体访问的文件类型,OS 调用相应的文件系统提供的实现函数来完成。这个过程可视为"动态绑定"的过程。内核在很多时候不能确定具体的实现函数的调用,最简单就是有这么多的外设,用户会使用什么样的外设(驱动程序),事先根本不能确定,因此必须要有一种动态加载的机制。面向对象中的多态特性就是适应了这种灵活性。

4.2　编译时多态

编译时多态指在编译阶段即可确定的多态性,主要通过重载机制获得。函数重载使得在语义上相似的函数可用同样的标识符来命名[1]。C++编译器通过函数的参数类型或个数的不同进行区分,但 C 编译器不提供重载机制,为此内核在函数中增加一个参数来区分不同的实现。以 void * kmalloc (size_t size, int flags) 为例,这个函数的功能是请求系统分配若内存区域,标志变量 flags 用于区分不同的分配策略,由于语义相似,这族操作使用同一个函数名 kmalloc。这些标志变量预先定义为:GFP_KERNEL(分配一般的内核内存)、GFP_USER(代表用户分配内存)、GFP_DMA(分配用于 DMA 缓冲区的内存)等[5]。函数执行时将根据 flags 的不同而不同。如 kmalloc(size, GFP_DMA)就将分配用于 ISA 设备进行 DMA 传输用对缓冲区[5]。这种"重载"减少了名字空间的污染,在相同的语义上提供了一致的界面。尽管这种"重载"不能说是真正意义上的重载,但它却体现了面向对象程序设计关于重载的思想。

5　软件开发中的应用

利用 C 语言的高效性结合 OO 设计思想,是两者优点的有效结合。笔者在嵌入式软件开发实践中,结合这种思想,已成功开发了基于 DSP 的 MPEG-4 CODEC,多目标视频跟踪等嵌入式实时软件。

(下转第 117 页)