AspectC 在 Linux 内核开发中的应用研究

Research on AspectC in the Development of Linux Kernel

夏守姬 张琼声 黄亭宇 (中国石油大学(华东)计算机科学与技术系 山东东营 257061)

摘 要: AspectC 是一种新的编程技术,是基于 C 语言的面向方面扩展。本文简述了面向方面编程的基本思想,介绍了 AspectC 的基本概念以及实现机制,重构了 Linux 内核中的典型贯穿特性——同步锁关注点,并对比分析了原始实现与方面实现对系统代码质量的影响,论述了面向方面技术对于设计和开发操作系统内核所具有的应用价值。

关键词: 关注点分离 模块性 AspectC 操作系统 方面

计算机软件设计的一个重要原则,就是要清晰地分离各种关注点,然后分而治之,各个击破,最后形成统一的解决方案。所谓关注点,是指一个特定的目标、概念或者兴趣域,是软件系统开发的基本处理模块。从面向过程到面向对象开发方法的先后提出,事实证明关注点分离技术是开发软件系统的一种有效的办法。

然而,对于大型复杂的操作系统来说,其本身存在着大量的贯穿特性,既有高层次的概念,如安全、服务质量等,也有较低层次的概念,如缓冲、同步、日志等。贯穿特性,简单地说就是一个贯穿多个基本系统模块的系统元素,实现上表现为其代码遍布若干不同模块,但与其所横跨的模块代码在功能上没有相关性。它们无法用传统的面向过程或面向对象技术进行有效的抽象和模块化,导致了代码交织和代码散布现象,严重影响了系统模块的内聚性和模块之间的独立性。

为了解决贯穿特性的模块化实现问题,20 世纪 90 年代人们提出了一种新的面向 aspect 的软件开发技术,该方法引入 aspect 来描述贯穿特性,并能够自动地将贯穿特性织入软件系统中,实现了贯穿特性的模块化,使开发的系统代码变得更清晰,系统变得更容易维护、移植和扩展,弥补了传统开发方法的不足。

1 面向方面编程和 AspectC

1.1 面向方面编程(AOP)简介

面向方面编程(aspect – oriented programming, AOP)是由 Xerox PARC 研究中心开发的一种编程范式,它被视为是"后"面向对象时代的一种全新的编程技术。AOP 是一种关注点分离技术,它从编程方法学的角度对贯穿特性问题进行有效解决,使开发者可以更好的将那些本不应该纠缠在一起的任务分离开,从而为系统提供了更好的封装性和互操作性,真正达到"关注点分离,分而治之"的目的。

AOP 构建在面向对象技术的基础之上,提供了新的 aspect 机制对系统贯穿特性进行描述、设计和实现。它的基本思想是分别描述系统的不同关注点及其关系,通常使用传统的结构化方法或面向对象方法对核心关注点进行处理;使用 aspect 机制对贯穿特性进行处理,以一种松耦合的方式实现单个关注点,然后依靠 AOP 环境的支撑机制,将这些关注点组织或编排成最终的可运行程序。

一个 AOP 的语言实现可以借助其它编程范型作为它的基础,从而保留其基础范型的优点。目前,比较成熟的 AOP 实现有基于 Java 语言的 AspectJ、基于 C 语言的 AspectC 和基于 C++语言的 AspectC++等。

1.2 AspectC

AspectC(Aspect Oriented C)是 C 语言的一种面向方面编程实现,是 C 的一个简单扩展。它通过将 AOP 概念引入 C 语言中使得面向方面思想能够运用于 C 系统的开发。

1.2.1 AspectC 的基本概念

AspectC 语言是一个可免费获得、由多伦多大学中间件系统研究组织 Gong、Jacobsen 等人开发的,其设

计思想主要遵循 AspectJ 语言以及 Coady 等人的建议。为了支持 aspect 机制，AspectC 中引入了一些新的语言构造。下面以 AspectC0.5 作为示例，简单介绍一下 AspectC 语言的基本组件。

(1) 连接点(Join Point)

连接点是 AspectC 的胶合剂，它提供一个框架使得普通 C 代码和 aspect 代码可以协调一致的工作。具体实现时连接点是指程序执行上下文中明确定义的点，在这些点中可以执行 aspect 代码。目前 AspectC 支持的连接点类型主要有函数调用连接点 call 和函数执行连接点 execution。

(2) 切入点(Pointcut)

切入点是处理 aspect 贯穿关系的关键元素，是实现连接点的语言结构。它能够灵活地描述程序静态结构和动态控制流中的连接点。技术实现上，切入点通过切入点表达式描述特定连接点。其语法声明如下：

```
pointcut pointcut – name( parameter – listopt ):
  pointcut – description;
```

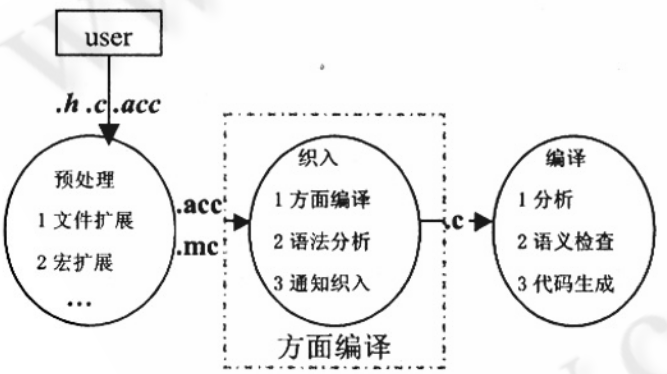

图 1　AspectC 编译过程

(3) 通知(Advice)

通知是 AspectC 真正的关键，它用于声明在切入点表达式中定义的连接点被捕获时应执行的动作，是一个被特定事件触发的逻辑。它能被嵌入到调用者和被调用者之间以及方法的调用者和方法本身之间，这就允许开发者定义贯穿的动作模块，从而可以很方便的将贯穿特性行为透明的嵌入到已存在的其他相关功能性模块中。

目前 AspectC 提供了三种将通知关联到连接点的方式：before、around 和 after。before 表示通知在连接点之前运行，after 表示通知在连接点之后运行，around 表示通知围绕着连接点运行，它有权决定是否运行其连接点，可以修改连接点的上下文环境，并且返回与原连接点处的方法类型一致的值。通知声明的基本语法是：

```
type – specifieropt before | after | around ( parameter –
type – listopt ): pointcuts { function – body }
```

(4) 方面(Aspect)

方面是体现贯穿关系的模块单元，它支持将贯穿系统的关注点封装到单独的模块中。它在 AspectC 中的地位类似于类在面向对象中的地位。它可以简单的看作是连接点集和通知的综合体。

1.2.2 AspectC 机制

运用 AspectC 开发系统软件的过程包括三个步骤：(1)分解并标识出系统需求中的 aspect 即贯穿特性；(2)运用 aspect 语言构造实现标识出的 aspect；(3)通过 AspectC 编译器将所有单元编译重组在一起，形成最终可运行系统。为了支持上述机制，AspectC 提供了一个 C 的面向方面扩展构造以及一个用于实现该构造的编译器。

(1) AspectC 基本结构

一个标准的 AspectC 程序包括两部分：核心/基本程序和方面程序。其中核心程序用于实现系统的基本逻辑功能，采用标准 C 书写；而方面程序用来模块化实现横跨多个基本模块的贯穿特性，采用 AspectC 和 C 书写。为了帮助区分核心和方面程序，AspectC 建议核心程序采用“.mc”后缀，方面程序采用“.acc”后缀。

(2) AspectC 编译机制

AspectC 的编译过程包括 3 个阶段：预处理、方面织入以及编译。预处理的任务主要包括文件扩展和宏扩展等操作，用户输入的 AspectC 文件和 C 源文件经由 C 预处理完后，生成的“.mc”和“.acc”后缀文件作为输入传递给 AspectC 编译器进行方面织入。AspectC 编译器是一个源对源的解释器，主要操作包括方面编译、语法分析以及通知织入，核心任务就是要将方面中定义的通知织入到切入点指定的连接点处，输出为 ANSI – C 的标准 C 文件。方面织入后的 C 文件可以由任何 C 编译器(如 gcc、cc 或 xlc)编译从而生成可执行文件。具体编译过程如图 1 所示，整个编译过程展现了采用面向方面技术进行软件开发的核心思想。

2 AspectC 在重构 Linux 内核中的应用

随着软件编程方法学的发展，面向过程、面向对象技术的先后出现使得开发者能够更高效地建立更加复杂的操作系统。然而，传统开发方法处理操作系统中的贯穿特性还不够理想，而面向方面技术运用方面机制分离并模块化实现贯穿特性，为操作系统的研究与设计开发工作提供了新的应用途径。

下面主要介绍运用 AspectC 重构 Linux 内核中的一个典型贯穿特性——file_list_lock 同步锁的案例研究。

2.1 同步锁关注点 file_list_lock( ) 和 file_list_unlock( )

贯穿特性就是指分布在系统多个模块中的关注点，它的基本特征就是代码遍布若干功能模块，并且影响被横切模块的实现。同步是 Linux 内核中贯穿特性的一个典型。如今的 Linux 大都支持多处理器环境，为了实现多个处理器对共享数据的并发访问，Linux 引入了同步锁的概念，也就是说，某个时刻，当一个对象要访问共享数据时，该对象必须上锁，访问完后立即解锁。由此可见，同步锁操作被分散到不同的本不相关的模块中，这些模块在执行核心操作之外还得执行锁操作。因此，同步锁关注点满足了贯穿特性的特征，下面我们将以 Linux 中广泛分布的同步锁 file_list_lock( ) 和 file_list_unlock( ) 为案例，首先分析该关注点的原始实现，然后采用 AspectC 语言对其进行重构，通过对比分析重构前后系统的变化，从而确定 AspectC 对于提高 Linux 内核源代码的模块性方面的应用价值以及应用途径。

2.2 Linux 同步锁的实现

在 Linux 内核的原始实现中，当一个内核对象操作共享数据 file 队列时，必须先通过 file_list_lock( ) 获得文件锁，待操作完成以后则通过 file_list_unlock( ) 释放该锁以供其他对象使用，这样就保证了对共享数据的并发访问。然而，file_list_lock( ) 和 file_list_unlock( ) 操作必须散布在内核中任何需要的地方，代码的分散现象严重，模块性很差。

根据统计，此关注点的分布涉及 Linux 内核中的 5 个文件 10 个函数的 11 处代码，具体函数分布如图 2 所示。

2.3 Linux 同步锁的 AspectC 实现

运用 AspectC，重构 file_list_lock( ) 关注点为一个方面涉及的操作有：首先将贯穿特性 file_list_lock( ) 和 file_list_unlock( ) 两个函数调用从源代码（即 file_list_lock 横切的 5 个文件 10 个函数的 11 处发生地）中移走，然后使用 AspectC 构造将此关注点封装在一个方面文件中，以模块化的方式实现该贯穿特性。

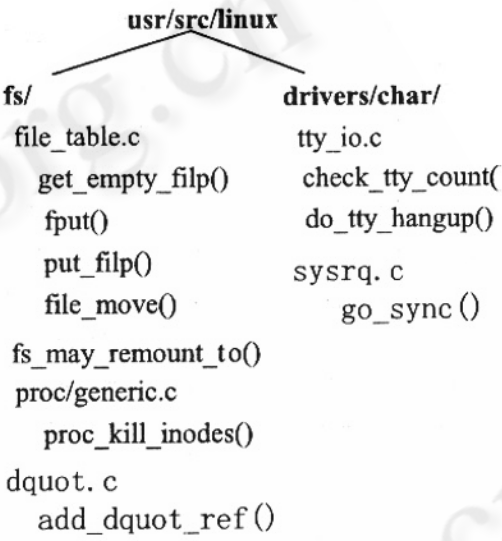

图 2　同步锁 file_list_lock( ) 的内核分布

图 3 显示了 file_list_lock( ) 方面文件的部分核心实现。方面代码中 3 个命名的切入点明确指定了内核中需要加锁的特定连接点，4 个通知声明则分别根据相应的切入点定义了在对象访问共享数据时的加锁以及解锁操作。

2.3.1 方面文件中的切入点

方面文件中，第一个命名切入点 callProc_kill_inodes_and_Fs_may_remount_to 指定了函数 proc_kill_inodes( ) 或 fs_may_remount_ro( ) 执行时的连接点。第二个和第三个命名切入点 callFput_and_Put_filp、callFile_move 分别指定了函数 fput( ) 或 put_filp( ) 执行时的连接点以及 file_move 函数执行连接点，同时它们还通过 args 关键字将选择的连接点的调用参数暴露给通知上下文，使得通知体内可以合法使用该参数值。

2.3.2 方面文件中的通知

前两个通知 before( ) 和 after( ) 声明了当捕获到第一个切入点指定的连接点时要执行的操作，即在连接点执行之前（before）加锁，连接点完成之后（after）解锁。后两个通知 around( ) 则定义了各自切入点指

定的连接点匹配时的执行动作。在 AspectC 中使用 around 通知可以修改连接点处的执行代码，即可以取代源程序中的代码或者继续原来的执行。若开发者希望在 around 通知中仍然执行被捕获连接点处的逻辑，必须使用 proceed 关键字。

把 file_list_lock( ) 关注点重构为方面文件，实现了将分散的应用组织为一个单独的模块。接下来的主要工作就是要检验重构后的方面文件是否正确，验证的标准就是检查重构前后的同步锁功能是否保持一致。可采用的一种方式是编写与同步锁横切的函数以及对应的方面文件具有相似结构的简单函数，在 cygwin 环境中采用 AspectC 编译器进行编译运行，通过验证输出字符串的结果来检查重构后是否保持原有的功能。另一种方式就是重新编译更改后的 Linux 内核，检验编译后的内核是否正常运行。经过以上两种方式检验，证明了重构后的系统完全保持了原有的特性。

```
pointcut callProc_kill_inodes_and_Fs_may_remount_to( ) :
  execution( static void
      proc_kill_inodes( struct proc_dir_entry * ) ) ||
  execution( int fs_may_remount_ro( struct super_block * ) ) ;
pointcut callFput_and_Put_filp( struct file * file) :
  ( execution( void fput( struct file * ) ) ||
    execution( void put_filp( struct file * ) ) ) &&args( file) ) ;
pointcut callFile_move( struct list_head * list) :
  execution( void file_move( struct list_head * ) ) &&args( list) ) ;

 before( ) : callProc_kill_inodes_and_Fs_may_remount_to ( ) {
    file_list_lock( ) ; }
 after( ) : callProc_kill_inodes_and_Fs_may_remount_to ( ) {
    file_list_unlock( ) ; }
 void around( struct file * filp) : callFput_and_Put_filp ( filp) {
      if( atomic_dec_and_test( &filp - > f_count) ) {
          file_list_lock( ) ;
          proceed( ) ;
          file_list_unlock( ) ; } }
 void around( struct list_head * list) : callFile_Move( list) {
      if( ! list) return( ) ;
      file_list_lock( ) ;
      proceed( ) ;
      file_list_unlock( ) ;
} ...
```

图 3　file_list_lock( ) 的部分核心方面实现

2.4 重构前后的对比分析

重构前，Linux 内核中同步锁关注点 file_list_lock 的实现散布在系统的多个功能函数中，代码的模块性极差，这种分散和交织的代码直接导致了代码冗余高、可重用率低，系统难以维护和扩展。

采用 AspectC 将贯穿特性重构为一个方面文件，带来的好处是：在访问对象的业务逻辑中只需要关心核心功能，无需显式增加同步锁操作，将同步锁操作彻底从核心逻辑中分离出来，系统结构变得更加清晰，真正实现了关注点分离的目标。同时，同步锁被封装在一个独立的方面文件中，代码变得更加紧凑，消除了冗余代码，改善了系统的模块结构，保证了模块间的松散耦合，也增强了内核源代码的可读性、可维护性。

事实证明，AspectC 的方面机制能够提高 Linux 内核源代码的质量。因此，AspectC 技术在 Linux 操作系统的设计与开发中具有很好的应用前景。

3 AspectC 存在的问题以及发展趋势

从 AspectC 的提出到真正发布不过短短几年时间，尽管其发展迅速，但是仍然存在很多问题：(1) AspectC 是一种新的编程技术，理论和实践方面都很不完善，没有完整的文档，没有得到良好的测试和大量实际项目的应用，作为一种开放的新技术，还需要在实际工作中得到更多的检验。(2) AspectC 的效率问题。如果采用 AspectC 重构系统中的某些贯穿特性后影响了系统的性能，那么改进操作系统内核代码的模块性是没有任何帮助的，相反对系统还是有害的。因此，重构系统代码后还需要严格的效率检验。(3) AspectC 的开发工具还很缺乏，编译器的功能过于简单，缺乏稳定完善的功能，而且，确定系统中贯穿特性的表现特征以及判定依据的标准也有待于进一步的解决。(4) 目前 AspectC 的研究工作主要是围绕已有的操作系统展开，即针对已有的贯穿特性进行 AspectC 重构，方面与内核代码的设计有先后次序。如果内核代码的设计不合适，那么切入点定义有时会包含很多函数名，结果造成了极大的维护问题。因此，方面与内核代码应该同时设计。

随着越来越多的组织对 AspectC 的关注并积极加入到 AspectC 的研究开发中，极大地推动了 AspectC 技术的发展。未来 AspectC 技术的发展趋势表现在以下

几个方面：

(1) 研究适合于 C 的关注点分离支持以及面向方面的语言特征，进一步完善 AspectC 的理论概念，将 AspectC 运用到更多的实际项目中，通过实践来检验 AspectC 技术的应用价值。

(2) 丰富 AspectC 编译器的功能，制定相对完善的说明文档以及贯穿特性的判定标准。

(3) 制定完整、规范的 AspectC 软件度量及程序测试方法。

(4) 研究开发 AspectC 配套的开发工具，方便 AspectC 的开发和使用。

(5) 尝试从操作系统的设计阶段就开始运用 AspectC 思想，考虑系统中的各种贯穿特性，同时设计实现构件代码和 AspectC 代码，扩大 AspectC 的应用领域。

4 总结

AspectC 是 C 的面向方面扩展，通过重构 Linux 内核中的同步锁关注点证明了 AspectC 有助于提高操作系统内核源代码的质量。面向方面技术的发展，为操作系统的研究和设计开发工作提供了新的思路和研究空间，具有极高的应用价值和可行的应用途径。

参考文献

1 Kiczales G, Lamping J, Mendhekar A. Aspect – oriented Programming. In: Proc. of the European Conf. on Object – Oriented Programming (ECOOP). Berlin: Springer – Verlag. 1997. 220 – 242.

2 Kiczales G, Hilsdale E, Hugunin J, Kersten M, Palm J, Griswold W. An Overview of AspectJ. In: Knudsen JL, ed. Proc. of the European Conf. on Object Oriented Programming. Berlin: Springer – Verlag. 2001. 327 – 353.

3 Spinczyk O, Gal A, Schroder – Preikschat W. AspectC + +: An AOP extension for C + +. In: Proc. of the 40th International conference on Tools Pacific. 2002. 53 – 60.

4 AspectC. http://www.aspectc.net

5 毛德操、胡希明，Linux 内核源代码情景分析，杭州：浙江大学出版社，2001.

6 Coady Y, Kiczales G, Feeley M, Smolyn G. Using AspectC to improve the modularity of path – specific customization in operating system code. In: Proc. of Joint ESEC and FSE – 9. New York: ACM Press, 2001. 88 – 98.

7 Coady Y, Kiczales G, Feeley M, Hutchinson N, Ong JS. Structuring Operating System Aspects. Communications of the ACM, 2001. 79 – 82.

8 Machrenholz D, Spinczyk O, Gal A, Schroder – Preikschat W. An Aspect – Oriented implementation of interrupt synchronization in the PURE operating system family. In: Proc. of the 5th ECOOP Workshop on Object Orientation and Operating Systems. 2002. 49 – 54.