计算机科学1997 Vol.24 №.3

操作系统设计和实现中的面向对象技术的应用问题

潘 清

（国防科工委指挥技术学院 北京 101407）

摘 要 本文对面向对象技术和client/server技术在当前操作系统中的应用进行了分析，指出了面向对象技术和client/server技术在紧偶合体系结构的系统中应用的局限性。提出了引入集合进行系统管理的思想，说明了集合和对象结合的优越性，并采用这种思想对微内核操作系统对象进行了重新定义，简要说明了这种方法在操作系统中的应用。

关键词 操作系统，面向对象，集合

1 引言

目前在许多操作系统的设计和实现中都采用了面向对象的方法。最常见的是：在操作系统内核中定义基本对象——内核基本抽象。如Mach系统定义了五种基本抽象：任务，线程，端口，消息，存储对象；Plan9系统定义了两种基本抽象：进程，文件；系统中的其它功能都用这些基本对象来实现。在Windows-NT中还专门设立了一个对象管理模块，来管理系统中所有对象的创建和释放操作。采用面向对象的技术对传统的操作系统内核结构和功能进行重新组织和实现，使整个操作系统的模块化程度，可移植性，安全性大大提高了一步。但是，面向对象技术和client/server技术是基于消息通讯的技术，用户一般是通过向服务器或对象发送一个消息来获得服务，这种服务方式是松散偶合型系统体系结构的一个特点。传统的操作系统核心中的各种功能模块的关系非常密切，核心中的一个功能模块可以直接调用另一个模块中的功能。当引入面向对象及client/server技术对它进行改造时，会引起许多问题，其中最重要的问题是性能问题。这些技术如果使用不当，会使内核的开销增大，从而影响应用程序的运行效率。本文着重讨论面向对象技术在操作系统核心中的应用问题。首先介绍操作系统内核中的对象管理问题，然后引入集合的概念来管理操作系统中的对象，最后说明使用这种技术来重构Mach3.0内核对象的方法。

2 面向对象及client/server技术应用问题

在Mach3.0中，消息机制是Mach的基础，任务通过端口进行通讯，系统将内核也看成一个任务——内核任务。每个内核对象都有一个内核端口来标识它，对某个内核对象的操作都是通过向内核对象的端口发送消息来完成的。因此，从用户的角度来看，Mach3.0是一个由对象构成的系统。但是，从系统的实现来看，对内核对象的管理仍采用传统的方法。WindowsNT的设计和实现比Mach3.0有了新的突破，它在系统中设立了专门的对象管理模块，管理所有对象的创建和删除操作，不管是用户还是系统，系统内部通过该模块提供统一的接口调用。由于有了统一的对象管理模块，系统中对象的创建和删除操作以统一的方式进行，系统在创建对象的时候，根据创建者的身份创建不同安全级的对象，使系统的安全性大大提高了。但是，除了对象的创建和删除之外，对象在它的生存周期中的其它管理工作，都由各个管理模块自己负责。

当前client/server技术在操作系统设计和实现中使用得非常广泛。通过使用client/server技术可以将传统的操作系统结构重新结构化，将内核中的一部份功能移到内核之外实现，这样就使操作系统内核功能大为减少，使系统的可靠性和可维护性程度大大增强。采用client/server技术导致了微内核体系结构的操作系统的产生，微内核结构已经成为新一代操作系统体系结构的代名词。client/server技术也是建立在消息机制的基础上的，用户通过消息来请求服务器服务，服务器将服务结果通过消息传给用户。

由于面向对象和client/server技术都是以消息机制为基础的，比较适合于松散偶合体系结构的系

统.对于属于紧偶合结构,各部分关系密切的操作系统内核来说,如果采用纯对象或 client/server 技术会影响系统性能.因此,目前看到的系统都只采用了面向对象的一部分思想,在实现时并没有完全采用面向对象的技术,对象的管理还基本上是用传统的方法,内核中各种模块还是通过直接调用的方式相互调用.各个系统中采用 client/server 技术的层次也不一样.例如 WindowsNT 的文件系统是放在执行层实现的,文件系统服务器通过内部接口直接和底层驱动器打交道,NT 的用户环境服务器作为用户态的服务器实现,应用程序通过动态库来请求环境服务器的服务.Mach3.0的文件系统是以用户态服务器的方式实现的,服务器必须通过设备接口以消息的方式来请求设备的服务;应用程序通过仿真库发消息请求操作系统服务.从系统性能来看,前者性能要好一些,也就是说单纯地追求用 client/server 技术,尽可能地将原来在内核中实现的功能移到内核之外,以服务器的方式实现的做法并不十分理想.作者认为产生这种结果的一个原因是松偶合体系结构和功能模块间紧偶合关系的矛盾,采用这些技术需要在系统设计目标和系统性能之间进行权衡.

3 集合的引入

虽然在当前的操作系统设计和实现中引入了面向对象的概念,但是传统的方法是采用某种特殊的结构来进行系统管理.如在进程管理中,设立了进程的就绪队列和等待队列;为了管理内存页面,设立了自由页面队列和修改页面队列.我们通过引入集合的概念,可以达到系统管理的规范化目的.下面先以 Unix 文件系统为例来说明对象的管理.

从用户的使用角度来看,用户通过目录来管理各种文件,文件存放在文件目录中,为了管理的方便同一个目录中的文件被放在一起.例如,某个目录中存放的是实用程序,另一个目录中存放的是临时文件.目录中的文件之间可以没有任何关系,目录只是用户进行文件管理的有力工具.

从系统实现的角度来看,Unix 文件系统提供了一种抽象——文件.文件可以是一般的数据文件、目录文件和特殊文件.文件按照目录的方式组织在一起,通过目录来存放,查找和管理文件.这样,整个文件系统采用这一抽象就构成了一个树型结构.文件作为一个对象有如下操作:创建文件 create_file();打开文件 open_file();读文件 read_file();写文件 write_file();取文件信息 get_file_info();删除文件 delete_file();关闭文件 close_file().对一般的数据文件来说,这些操作就足够了.可是对于目录文件还需要其它一些操作,如目录项的查询、创建和删除.因此,目录应该作为文件对象的一个子类,即它既具有文件的一般操作又具有自己的特殊操作.目录文件的这些操作是用来管理系统中的文件的.

从上面的分析,作者的意图是象引入对象的思想一样,引入集合的概念,以便管理系统中的所有对象,重新组织、描述和构造系统,达到更高层次的模块化和可移植性.从 Mach3.0中,我们可以看到这种思想的某种体现,例如处理机集和端口集.

集合也可以看成一种对象,具有一般对象的属性,如创建一个集合,撤消一个集合.同时,它又有自己的特殊属性,如向集合中加入一些元素,移出一些元素等等.引入集合后,可以将操作系统核心中不便用对象来表示和管理的内容,使用集合来统一管理.例如,在操作系统中任务是一个对象,调度程序使用就绪队列和等待队列来调度和管理任务,可以把各种队列形式化地表示成集合的形式.选择就绪任务就等价于从就绪集合中取出一个元素,任务的等待操作就等价于将任务放入任务的等待集合.对于操作系统中的其它一些杂项管理工作如内存页面的管理等内容,都可以使用集合来统一管理.

3.1 集合和对象的结合

集合用来管理对象,一个对象的集合可以看成是处于同一种状态下的对象的集合,或是与某种事件相关的对象的集合.这样,集合就可以用来表示状态,对象在生存周期中的状态的变化,就等价于对象在不同的集合中不断迁移的过程.例如一个简单的进程状态变迁图(图1),它表示进程的状态变化,圆圈表示状态,圆圈之间的有向弧表示进程状态的变迁方向.引入集合后,将集合和状态相对应,一个集合管理处于同一种状态的进程,系统中的对象由集合来管理,集合和它管理的对象可以组成一个更大的对象,这个新的对象再由一个新的集合来管理,这样就构成了一个非常清晰的层次结构.如果将文件目录看成一个集合,Unix 文件系统就可以用上述方法清晰地表示出来.当集合用来管理对象时,它表现为集合的属性;当它被别人管理时,它表现为对象的属性.

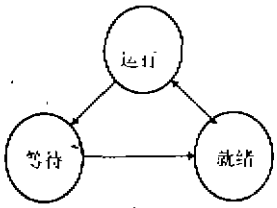

图1

3.2 集合的操作

在具体的实现中，集合可以用各种数据结构来表示，如队列，树，表等等。集合中对象之间有一定的关系，即集合中的元素是有一定次序的。这就与数学中的集合操作有一定的差距，在数学中集合的运算主要是集合和集合之间的操作，如集合的交，并，差，补和迪卡尔运算；而在软件中具体的操作都是集合和集合元素的操作，如将一个元素放入集合，或将一个元素从集合中取出。为了和具体的实现相联系，需要定义新的操作。为了不和集合的运算相矛盾，这里将集合看成是一个序列(sequence)。因此，这里定义的操作是序列的操作，简称是s_op。

get(s, name, element) 取序列s中名字为name的元素，若名字为空，则按规定的存取策略取一个元素。

find(s,element) 在序列s中查找是否有元素element。

put(s,element) 将元素element放入序列(按照规定的存取策略)。

test_empty(s) 判断s是否为空。

test_full(s) 判断s是否满了。

有了这些操作之后，就能很容易地描述系统中的各种管理功能。同一类型的操作可以统一实现，避免重复劳动，使设计者可以将注意力集中到关键的问题上。集合操作的统一命名和实现也为将来的维护提供方便，维护人员能比较容易地弄清整个系统的体系结构，而不会被一些杂乱无章的代码所困惑。这一点在操作系统中犹为重要。

一个大粒度的对象中可能包含许多集合，例如，在一个多线程的操作系统中，一个任务可以有多个线程，任务还有一个离散的虚拟地址空间，具有多个通讯端口等等。这样，任务就是一个具有集合属性的对象，

task=={线程},{虚拟地址},{端口},…)。

一个对象可以在多个集合中，如一个线程必须在它的任务的线程集合中，还可能在线程的就绪或等待集合中。通过对对象的属性的查询，可以确定对象所在的集合。因此，集合总是和对象的某个属性相联系，对象属性的变化，必须引起对象管理的操作；反之亦然。

4 集合概念在操作系统中的应用

通过继承Mach3.0的思想，作者以对偶的方式对Mach3.0中的对象进行了重新定义。在Mach3.0中，有五个基本抽象：线程，任务，端口，存储对象，消息。重新定义的方法是，只要系统中有一个对象，就定义一个集合来管理它。具体方法如下：

(1)定义基本对象：线程--线程集；任务--任务集；端口--端口集；存贮对象--存储对象集；消息--消息集。对于内核中一些不是用对象来表示的，但是又是独立的实体，如自由页面等等，也用上述方式来定义。

(2)定义各种独立的实体：自由页面--自由页面集；修改页面--修改页面集；局部端口名--局部端口名集；…。

(3)定义扩充对象：就绪线程--就绪线程集；等待线程--等待线程集；挂起线程--挂起线程集；运行线程--运行线程集。

(4)确定各类对象的管理机制。

(5)相同机制归类。

(6)选择合适的数据结构。

(7)实现对象的管理。

采取这种对偶方式的描述和实现，使得系统中的管理工作更加规范了。下面通过我们对MACH3.0调度系统的优化处理来说明集合的应用。

在MACH3.0中，线程是基本调度单位，处理器在进行线程调度时，是根据线程的优先级从处理机或处理机集中的就绪线程集合中选取的。当选取的线程和当前线程不在同一个任务中时，必须进行任务切换，这样，线程切换操作的开销就和任务一样了。为了能尽量从本任务中的线程中选取，我们增加了一个任务就绪线程集合，来进行调度的优化处理，以减少由于线程的切换而引起的任务的切换操作。这个例子说明了集合是对象管理的强有力的工具，通过集合可以灵活地管理各种各样的对象。