Linux 内核组成和架构

Linux内核是Linux最核心的部分，Linux操作系统就是在Linux内核上发展壮大起来的，而内核的组成和架构则是任何试图涉及Linux Kernel的开源爱好者共同关心的。

□ Intel开源技术中心 刘宏

Linux内核从90年代初由Linus发布第一版开始，经过开源社区十几年的共同努力，现在已经发展到了最新的2.6内核。它已经从Linus作为学习Intel 386架构的一个“玩具”项目发展成一个成熟的可以商用的操作系统。由于Linux是一款开源的操作系统，这就给了我们一个非常难得的机会去学习一个成熟的、商用的操作系统是如何实现的。

进程管理

Linux内核支持所谓的多任务运行(multiprogramming)，即可以有多个用户程序同时运行，用户可以一边听音乐一边写文档。为了在一个单CPU的系统中支持多任务，Linux内核使用了进程的概念（内核中用task_struct的结构表示），一个进程就是一个运行中的用户程序。

整个系统运行时，每个进程都会分得一定的时间片（从几十毫秒到几百毫秒），Linux内核的调度器负责选择不同的进程运行。当这个进程的时间片用完后，调度器会选择一个新的进程运行。由于切换的时间和频率非常快，给用户的感觉是有多个程序在同时运行。而实际上，在一个CPU上每一个时间点上其实只有一个进程在运行。

1.Linux的进程

一个用户程序开始运行时，内核会负责把存储在磁盘上的二进制可运行程序调入内存中，同时会分配一个task_struct代表这个要运行的用户程序。task_struct是一个非常重要的数据结构，所有关于这个正在运行的用户程序信息都会保存在这个结构中，包括用户程序打开的文件，所使用的内存空间等。Linux内核所有对于进程的管理都是通过这个结构进行的。

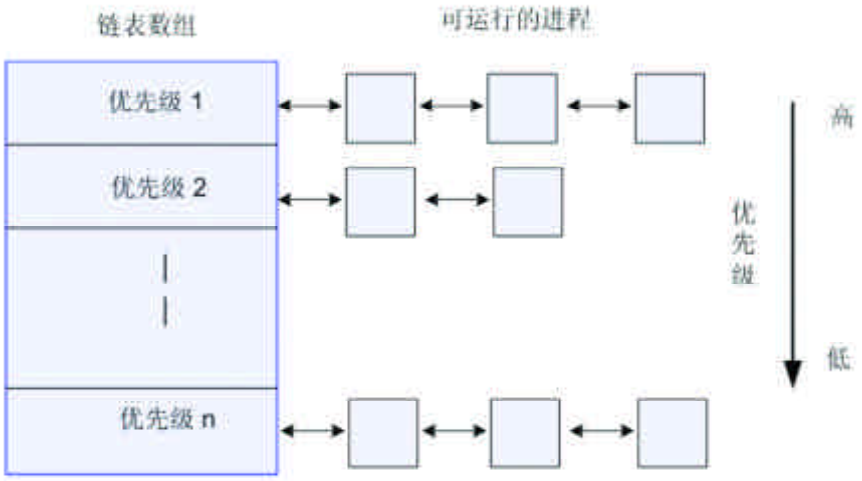

图1 2.6内核O(1)调度器

2.进程调度器

进程调度器是整个系统非常关键的一个子系统，在系统中有多个进程可以运行时，它负责选择哪个进程运行。每个进程都是CPU计算和I/O请求的交替，不同的是，有的进程大部分的时间是浪费在CPU计算上，而有的则把几乎所有的时间用在了I/O等待上。一个好的调度器，既能够让用户交互程序有很快的反应时间（如编辑程序必须能够快速地将用户的敲键快速地反映到屏幕上），也能够让CPU密集型（CPU-bound）的进程有很好的吞吐量(throughput)。

Linux内核调度器是基于优先级的调度器，通过一套非常复杂的算法决定一个进程当前的优先级，同时进程的优先级又是动态调整的：一个很久没有得到运行的进程的优先级会被暂时增加，以增加得到运行的机会；一个花了较多时间睡眠的进程则会被认为是用户交互程序，优先级也会被增加，当这类进程需要运行的时候就会有较大的机会被调度器选用，这样整个系统对用户的输入就会有较快的响应。

早先2.4版本的内核把所有可运行的进程连接在一个链表上，每次调度时都要循环遍历链表上的每一个进程，计算其优先级以决定选择哪一个进程运行。这种方法的扩展性不是很好，调度器选择一个可运行的进程的时间将随着系统内进程数的增加而线性增加。在2.5开发版本中，

© 1994-2012 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

Linux内核采用了由Ingo Molnar开发的O(1)调度器。O(1)调度器采用一个链表数组，根据进程的优先级把进程放进不同的链表中，每次调度时，总是从优先级最高的链表中取出第一个进程来运行。同时使用一个位图表（bitmap）来记录每一个优先级链表中是否有可运行的进程，搜寻这张位图表就可以找到优先级最高的链表。O(1)调度器的调度时间并不会因为系统中进程数的增加而改变，拥有极好的可扩展性。

内存管理

如何有效的管理整个系统的物理内存，快速响应内核各个子系统对内存分配的请求是操作系统一个很重要的功能。

1.页帧

在操作系统启动的初期，内核会通过一定的方法来得到系统中物理内存的分布信息（x86 PC从BIOS得到物理内存的信息，包括总共有多少内存，各个分块的地址信息等，可以通过cat /proc/meminfo看到整个系统的物理内存分布）。Linux内核是以一个物理页帧为单位对系统的物理内存进行管理的，对每一个物理页帧通过一个Page的结构进行状态的跟踪。页的大小由不同的CPU体系结构决定（在Intel x86上通常大小为4K字节）。

2.内存区域

Linux内核把所有的动态的RAM分为几个区域，ZONE_DMA、ZONE_NORMAL和ZONE_HIGHMEM。根据实际的内存情况，内核将内存分配给不同的区域。每个Zone会分配一个Page的数组来跟踪其所管理的物理内存页的信息。为了有效地管理、同时又能快速地响应内存请求，Zone通过一种叫buddy system的算法来组织这些页。

3.Slab分配器

由于Zone的内存分配是以页为单位进行的，对于小的内存分配请求（几十到几百字节），如果也使用页为单位进行分配就会造成内存的大量浪费。而且内核中，分配和释放某种数据结构是非常常见的操作，必须快速地完成这些操作。

在Zone的页分配器的基础上，Linux内核实现了Slab分配器很好地解决了这些问题。Slab分配器以对象为单位进行管理，把不同的对象组织在不同的Cache中。这种方式很好地解决了内存浪费的问题，同时又能快速的分配各种不同大小的对象。

虚拟文件系统VFS

Linux内核支持许多不同的文件系统，在Linux系统里，用户可以访问Windows系统的文件。实现这一功能的关键是Linux内核中的虚拟文件子系统(Virtual Filesystem)。VFS的关键是用一个通用的文件模型来表示各种不同的文件系统。这个通用模型主要由以下几个对象来表示superblock、inode、dentry和file，用户可以用面向对象的概念来进行理解，VFS定义的这些对象就像是基类（base class），定义了一些基本的信息，同时为每个类定义了一系列的操作函数。

各个不同的文件系统的主要工作就是给每个类加上自己的信息（继承这些基本类），同时提供自己的操作函数的实现。在硬盘上如何组织存放整个的文件系统，如文件系统控制信息、数据文件及根目录存放位置等都因各个文件系统的实现而异。Linux内核只能依赖于各个文件系统自己实现的操作函数来读取这些信息以达到支持各种不同文件系统的目的。

1.文件系统的安装

在Linux系统中，为了要操作某个磁盘上分区的文件，首先必须要将这个磁盘分区上的文件系统安装到根文件系统（通常为ext2/ext3）的某个目录上。

每种文件系统都必须定义一个file_system_type结构，这个结构最重要的就是get_sb函数的实现。在安装一个新的文件系统时，Linux内核会通过这个文件系统的名称来找到相应的file_system_type，调用其get_sb来得到这个文件系统的super_block（如vfat文件系统的vfat_fs_type→vfat_get_sb）。

Linux内核同时也会读取这个文件系统的根目录（利用刚得到的super_block的read_inode来读取）。这些信息都会被存储起来，以后在这个文件系统中进行文件操作时，需

链 接

GPLv3对用户的专利保护

GPLv3最重要的特性，就是明确了专利许可。任何分发GPLv3软件的人必须自动为软件用户提供许可证，让他们可以使用所有相关的专利。从理论上来讲，这样做可以保护所有GPL专利享有者的诉讼权利，防止Linux经销商与专利持有者进行独家经营。

根据FSF的要求，新版本的GPL意味着微软公司的专利许可会涉及到Novell、Xandros、LG电子和Linspire。也就是说，不仅是这四家厂商的客户，而是所有的用户，都不会再面临内含GPLv3软件的微软公司的起诉。而微软公司对此并不表示赞同，如果微软公司并未从实际行动上对侵害专利的Linux经销商或用户提出诉讼，那么这场争论绝不会被解决，但这种情况似乎根本不可能发生。

© 1994-2012 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

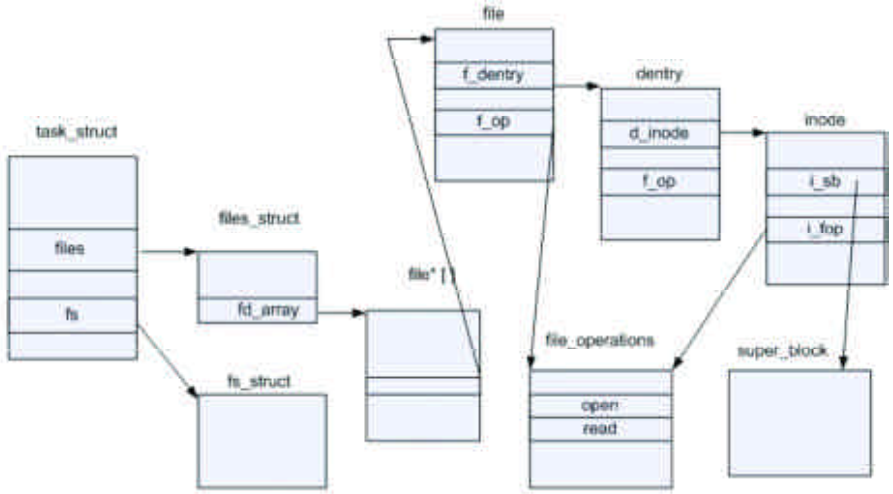

图2 Linux内核VFS子系统数据结构示意图

要找到这个super_block进行操作。super_block中存储的是文件系统级别的信息，比如这个分区上一共有多少数据块，哪些是空闲的，那些已经被文件占据了；以及根据这些信息，在创建新文件决定如何分配数据块。这些都因各个文件系统的实现而异。

2.文件的打开

假设我们想访问刚才安装的文件系统中的一个文件/mnt/tmp/test，内核需要进行一系列的操作。Linux内核得到了用户想要打开的文件名称后，从根目录“/”开始一级一级的往下查询。

首先，在dentry缓存中查找“mnt”目录项，如果没找到，则说明以前还没有别的用户查找过这个目录项。这时，内核就需要从硬盘上读取mnt这个目录项相应的inode，这就得借助各个文件系统的不同实现来完成这个操作。

通过mnt的父目录即根目录（dentry→d_inode→i_sb）可以找到这个文件系统的super_block，通过super_block的read_inode负责读取相应的inode。同时建立mnt目录相应的dentry并把它加入到dentry缓存中，以备以后的查询使用。

如果找到，则在dentry缓存中继续从mnt目录里查找tmp的目录项。找到tmp这个目录项后，内核会发现在这个目录项上安装了一个文件系统，这时会把当前找到的tmp目录项替换成所安装文件系统的根目录，从这个根目录开始继续查找。这时如果再读取某个文件的inode时，会找到这个新安装的文件系统的super_block，所有查询、读取等inode的操作都会使用这个文件系统的实现。正是通过这种方式，Linux内核实现了多种文件系统的支持。

最后会在tmp目录里找到test目录项，并会将表示这个文件的inode从磁盘上读出。最终将所有的这些数据结构通过图2的组织方式连接到表示用户进程的task_struct上。同时，file→f_op会被更新成文件系统特定的实现。以后用户对这个文件的所有操作都会最终调用file→f_op中的函数来完成。

进程间通信

Linux提供了几种进程间的通信方式，用于不同的进程之间在用户空间进行同步、数据的共享和交换。因为不同的用户进程都拥有自己独立的进程空间，进程间的通信就必须通过内核来进行中转，进行数据的共享和传递。

1.管道和命名管道

用户可以通过系统调用pipe()来创建一个管道，这个系统掉调用会返回两个文件描述符，一个用于读一个用于写。通常管道会用于父子进程之间，因为它需要两个进程能访问同一个files_struct，读的一方会关闭写文件描述符，写的一方关闭读文件描述符；这样两个进程之间就形成了一个单向的数据通道。

命名管道是为了在任意两个进程之间进行通信而设计地，通过在当前的文件系统中创建一个节点（通过mknod系统调用，指定S_FIFO文件类型），任何进程只要去打开这个文件就可以进行同管道的另一方进行通信了。在Linux内核中，管道的实现是通过pipefs文件系统实现的。

2.System V IPC

AT&T的System V UNIX中提供的进程间通信接口，Linux内核实现了信号量、信息队列和共享内存这三种IPC机制。

当用户进程需要一个IPC资源的时候，需要指定一个惟一的IPC KEY，Linux内核会创建一个相应的数据结构并记住这个KEY，同时返回一个32位标识符，以后用户进程就通过这个标识符引用这个IPC资源，进行同步操作。

以信息对列为例，为了发送一个信息报文，用户进程将32位标识符、报文内容和大小通过msgsnd系统调用传给内核，内核通过标识符找到相应的数据结构，同时将数据报文从用户空间拷贝到内核空间，并将这个报文放入这个数据结构的链表中。

接收时，进程通过msgrcv系统调用，同样指定这个32位的标识符，内核找到相应的数据结构后，把保存在链表中的信息报文传递给这个接收的进程。

用户进程之间就必须通过一定的机制来共享这个IPC KEY，通过这个KEY并借助内核的帮助来达到进程间通信的目的。软

责任编辑:陈杰 E-mail:chenjie@swm.com.cn 美术编辑:赵庆琨 E-mail:zqk@swm.com.cn

© 1994-2012 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net