深入理解 Linux 内核 第一章笔记

Contents

第一章 绪论

Linux 是 Unix-like **操作系统**大家族中的一员。从 20 世纪 90 年代末开始，**Linux** 突然变得非常流行，并且跻身于那些知名的商用 Unix 操作系统之列。这些 Unix 系统包括 AT&T 公司（现由 SCO 公司所有）开发的 System V Release 4 SRV4，加利福尼亚大学伯克利分校发布的 4.4 BSD，DEC 公司（现属于 HP）的 Digital Unix，IBM 公司的 AIX，HP 公司的 HP-UX，Sun 公司的 Solaris，以及 Apple 公司的 Mac OS X。除了 Linux 之外，还有一些其他的类 Unix 操作系统也是开放源代码的，如 FreeBSD、NetBSD 以及 OpenBSD。

1991 年，Linux Torvalds 开发出最初的 Linux，它作为一个适用于基于 Intel 80386 微处理器的 IBM PC 兼容机的操作系统。

Linux 最吸引人的一个有点就是它不是商业操作系统。它的源代码在 GNU 公共许可证（General Pwblic License）GPL 下是开放的，任何人都可以获得源代码并研究它。下载源代码的官方网站为 www.kernel.org。

Linux 与其他类 Unix 内核的比较

Linux 内核 2.6 版的目标是遵循 IEEE POSIX（Portable Operating Systems based on Unix，基于 Unix 的可移植操作系统）标准。这意味着，在 Linux 系统下，很容易编译和运行目前现有的大多数 Unix 程序，只需少许或根本无需为源代码打补丁。此外，Linux 包括了现代 Unix 操作系统的全部特点，诸如虚拟存储、虚拟文件系统、轻量级进程、Unix 信号量、SVR4 进程间通信、支持对称多处理器（Symmetric Multiprocessor SMP）系统等。

Linux 与一些著名的商用 Unix 内核到底如何竞争：

单块结构的内核

它是一个庞大、复杂的自我完善程序，由几个逻辑上独立的成分构成。大多数商用 Unix 变体也是单块结构（一个显著的例外是 Apple 的 Mac Os X 和 GNU 的 Hurd 操作系统，这两者遵循微内核的方法。）。

编译并静态连接的传统 Unix 内核

大部分现代操作系统内核可以动态地装载和卸载部分内核代码（典型的例子如设备驱动程序），通常把这部分代码称作模块 module。Linux 对模块的支持是很好的，在主要的商用 Unix 变体中，只有 SVR4.2 和 Solaris 内核有类似的特点。

内核线程

Linux 以一种十分有线的方式使用内核线程来周期性地执行几个内核函数，但是它们并不代表基本的执行上下文抽象。

多线程应用程序支持

一个多线程用户程序由很多轻量级进程 LWP 组成，这些进程可能对共同的地址空间、共同的物理内存页、共同的打开文件等等进行操作。Linux 把轻量级进程当做基本的执行上下文，通过非标准的 clone()系统调用来处理它们。

抢占式 preemptive 内核

Linux 可以随意交错执行处于特权模式的执行流。

多处理器支持

Linux 支持不同存储模式的对称多处理，包括 NUMA：系统不仅可以使用多处理器，而且每个处理器可以毫无区别地执行任何一个任务。

文件系统

Linux 标准文件系统呈现出多种风格。

与商业竞争对手相比，Linux 有如下优势：

- Linux 是免费的。
- Linux 的所有成分都可以充分地定制。
- Linux 可以运行在低档、便宜的硬件平台上。
- Linux 是强大的。Linux 的主要目标是效率。
- Linux 的开发者都是非常出色的程序员。内核非常稳定。
- Linux 内核非常小。
- Linux 与很多通用操作系统高度兼容。
- Linux 有很好的技术支持。

硬件的依赖性

Linux 试图在硬件无关的源代码与硬件相关的源代码直接保持清晰的界限。在 arch 和 include 目录下包含了很多子目录，对应 Linux 所支持的不同硬件平台。包括 arm、intel、mips 等。

Linux 版本

Linux 内核 2.6 之前，每个版本号用三个数字描述。用圆点分隔。前两个数字用来表示版本号，第三个数字表示发布号。第二位数字表示内核的类型：偶数表示稳定的内核，奇数表示开发中的内核。

Linux 内核 2.6 版之后，第二个数字已经不在用于表示一个内核是是稳定还是处在开发中的了。

操作系统基本概念

操作系统必须完成两个主要目标：

- 与硬件部分交互，为包含在硬件平台上的所有低层可编程部件提供服务。
- 为运行在计算机系统上的应用程序（即所谓用户程序）提供执行环境。

一些操作系统允许所有的用户程序都直接与硬件部分进行交互，如 MS-DOS。与此相反，类 Unix 操作系统把与计算机物理组织相关的所有低层细节都对用户运行的程序隐藏起来。当程序想使用硬件资源时，必须向操作系统发出一个请求，由内核代表应用程序与相关的硬件部分进行交互。

为了实施这种机制，现代操作系统依靠**特殊的硬件特性**来禁止用户程序直接与低层硬件部分进行交互，或者禁止直接访问任意的物理地址。硬件为 CPU 引入了至少两种不同的执行模式：用户程序的非特权模式和内核的特权模式。Unix 把它们分别称为用户态 User Mode 和内核态 Kernel Mode。

下面是一些基本概念:

多用户系统

一台能并发和独立地执行分别属于两个或多个用户的若干应用程序的计算机。

用户和组

操作系统必须保证用户空间的私有部门仅仅对其拥有者是可见的。每个用户用一个数字来表示，及用户标识符 User ID，UID。为了和其他用户有选择地共享资料，每个用户是一个或多个用户组的一名成员，组由唯一的用户组标识符 user group ID 标识。

类 Unix 操作系统都有一个特殊的用户，叫做 root，即超级用户 superuser。root 用户几乎无所不能。

进程

所有的操作系统都使用一种基本的抽象：进程 process。一个进程可以定义为：“程序执行时的一个实例”，或者一个运行程序的执行上下文。

内核体系结构

大部分 Unix 内核是单块结构：每一个内核层都被集成到整个内核程序中，并代表当前进程在内核态下运行。相反，微内核 microkernel 操作系统只需要内核有一个很小的函数集。运行在微内核之上的几个系统进程实现从前操作系统级实现的功能，如内存分配程序、设备驱动程序、系统调用处理程序等等。

微内核相对于宏内核来说具有一定的理论优势。宏内核的优势是效率高，因为微内核不同层次之间的消息传递等需要话费一定的代价。

Unix 文件系统概述

Unix 操作系统的设计集中反映在其文件系统上。

文件

Unix 文件是以字节序列组成的信息载体，内核不解释文件的内容。从用户的观点来看，文件被组织在一个数结构的命名空间中。树的根对应的目录被称为根目录。

Unix 的每个进程都由一个当前工作目录，它属于进程执行上下文，标识出进程所用的当前目录。

- 绝对路径： 路径名的第一个字符是‘/’
- 相对路径： 路径名的第一个字符不是‘/’

硬链接和软链接

硬连接指通过索引节点来进行连接。硬连接的作用是允许一个文件拥有多个有效路径名，这样用户就可以建立硬连接到重要文件，以防止“误删”的功能。硬链接有两方面的限制：

- 不允许用户给目录创建硬链接。避免出现环形目录结构体
- 只有在统一文件系统中的文件之间才能创建硬链接。

软链接也称符号连接 symbolic link。软链接文件有类似于 Windows 的快捷方式。它实际上是一个特殊的文件。在符号连接中，文件实际上是一个文本文件，其中包含的有另一文件的位置信息，可以是位于任意一个文件系统的任意文件或目录。

文件类型

可以是下列类型之一：

- 普通文件 regular file
- 目录
- 符号链接
- 面向块的设备文件 block-oriented device file
- 面向字符的设备文件 character-oriented device file
- 管道 pipe 和命名管道 named pipe，也教 FIFO
- 套接字 socket

文件描述符与索引节点

文件系统处理文件需要的所有信息包含在一个名为索引节点 inode 的**数据结构**体中。每个文件都有自己的索引节点，文件系统用索引节点来标识文件。索引节点至少提供如下信息：

- 文件类型
- 与文件相关的硬链接个数
- 以字节为单位的文件长度
- 设备标识符，即包含文件的设备的标识符
- 在文件系统中标识文件的索引节点号
- 文件拥有者的 UID
- 文件的用户组 ID
- 几个时间戳，表示索引节点状态改变的时间、最后访问时间及最后修改时间
- 访问权限和文件模式

访问权限和文件模式

文件的潜在用户分为三种类型：

- 作为文件所有者的用户
- 同组用户，不包括所有者
- 所有剩下的用户

文件的访问权限也有三种：

- 读
- 写
- 执行

因此，文件访问权限的组合就用 9 种不同的二进制来标记。

还有三种附加的标记，即 suid Set User ID、sgid Set Group ID 及 sticky 用来定义文件的模式。

- suid。 进程执行一个文件时通常保持进程拥有者的 UID。然而，如果设置了可执行文件 suid 的标志位，进程就获得了该文件拥有者的 UID。

- sgid。 进程执行一个文件时通常保持进程组的用户组 ID。然而，如果设置了可执行文件 sgid 的标志位，进程就获得了该文件用户组的 ID。

- sticky。设置了 sticky 标志位的可执行文件相当于向内核发出一个请求，当程序执行结束以后，依然将它保留在内存。该标志已经过时。

当文件由一个进程创建时，文件拥有者的 ID 就是该进程的 UID。而其用户组 ID 可以是进程创建者的 ID，也可以是父目录的 ID，这取决于父目录 sgid 标志位的值。

文件操作的系统调用

- 打开文件。进程只能访问打开的文件。
- 访问打开的文件。可以顺序/随机地访问。对设备文件和命名管道文件，通常只能顺序访问。
- 关闭文件。释放与文件描述符 fd 相对应的打开文件对象。当一个进程终止时，内核会关闭其所有仍然打开着的文件。
- 更名及删除文件。不需要打开就可以更名和删除文件。实际上，该操作并没有对这个文件的内容起作用，而是对一个或多个目录的内容起作用。

Unix 内核概述

进程/内核模式

内核本身并不是一个进程，而是进程的管理者。进程/内核模式假定：请求内核服务的进程使用所谓的系统调用 system call 的特殊编程机制。每个系统调用都设置了一组识别进程请求的参数，然后执行与硬件相关的 CPU 指令完成从用户态到内核态的转换。

Unix 系统还包括所谓内核线程 kernel thread 的特权进程，具有如下特点：

- 以内核态运行在内核地址空间。
- 不与用户直接交互，因此不需要终端设备。
- 通常在系统启动时创建，然后一直处于活跃状态直到系统关闭。

进程实现

为了让内核管理进程，每个进程由一个进程描述符 process descriptor 表示，这个描述符包含有关进程当前状态的信息。

当内核暂停一个进程的执行时，就把几个相关处理器寄存器的内容保存在进程描述符中。这些寄存器包括：

- 程序计数器 PC 和栈指针 SP 寄存器
- 通用寄存器
- 浮点寄存器
- 包含 CPU 状态信息的处理器控制寄存器，处理器状态字 processor status word
- 用来跟踪进程对 RAM 访问的内存管理寄存器

进程地址空间

每个进程运作在自身私有地址空间。用户态下运行的进程涉及到私有栈、数据区和代码区。在内核态运行时，进程访问内核的数据区和代码区，但使用另外的私有栈。

Linux 支持 mmap 系统调用，该系统调用允许存放在块设备上的文件或信息的一部分映射到进程的部分地址空间。

同步和临界区

一般来说，对于全局变量的安全访问通过源自操作 atomic operation 来保证。

临界区是这样的一段代码，进入这段代码的进程必须完成，之后另一个进程才能进入。

- 非抢占式内核。当进程在内核态执行时，不能被任意挂起，也不能被另一个进程代替。
- 禁止中断。在进入一个临界区之前禁止所有的硬件中断，离开时再重新启用中断。
- 信号量。semaphore 是一种广泛使用的同步机制。可以把信号量看成一个对象，其组成如下：一个整数变量；一个等待进程的链表；两个原子方法 down 和 up。每个要保护的数据结构都有自身的信号量，当内核希望访问这个数据结构时，在相应的信号量上执行 down 方法。如果信号量的当前值不是负数，则允许访问这个数据结构。否则，把执行内核控制路径的进程加入到这个信号量的链表并阻塞该进程。当另一个进程在那个信号量上执行 up 方法时，允许信号量链表上的一个进程继续执行。
- 自旋锁。信号量可能是很低效的，因为内核必须把进程插入到链表中，然后挂起它，操作比较耗时，这个时候其他内核控制路径可能已经释放了信号量。自旋锁 spin lock 和信号量很类似，但是它没有进程链表，当一个进程发现锁被另一进程锁着时，不停地旋转，直到锁打开。在单处理器下自旋锁是无效的。
- 避免死锁。Linux 通过按规定的顺序请求信号量来避免死锁 deadlock。

信号和进程间通信

Unix 信号 signal 提供了把系统事件报告给进程的一种机制。每种事件都由自己的信号编号，通常用一个符号常量来表示，例如 SIGTERM。有两种系统事件：

- 异步通告。
- 同步错误或异常。

用户态下进程间通信机制很多，通常有：信号量、消息队列及共享内存。共享内存为进程之间交换和共享数据提供了最快的方式。

进程管理

fork 系统调用用来创建一个新进程；exit 系统调用用来终止一个进程；exec 系统调用用来装入一个新程序。

- 僵死进程

  wait4 系统调用允许进程等待，直到其中的一个子进程结束，它返回已终止子进程的进程标识符。僵死进程表示进程已经终止，父进程还没有执行完 wait4。

- 进程组和登陆会话

  现代 Unix 操作系统引入了进程组 process group 的概念，以表示一种作业 job 的抽象。

  现代 Unix 内核也引入了登陆会话 login session。

内存管理

内存管理是 Unix 内核中最复杂的活动。

- 虚拟内存

  Virtual memory 作为一个逻辑层，处于应用程序的内核请求与硬件内存管理单元 MMU memory management unit 之间。现代 CPU 包含了能自动把虚拟地址转换成物理地址的硬件电路。

  它有很多用途和优点：

  - 若干个进程可以并发地执行
  - 应用程序所需内存大于可用物理内存时也可以运行
  - 程序只有部分代码装入内存时进程可以执行它
  - 运行每个进程访问可用物理内存的子集
  - 进程可以共享库函数或程序的一个单独内存映像

    - 程序是可重定位的，也就是说，可以把程序放在物理内存的任何地方
    - 程序员可编写与机器无关的代码
- 进程虚拟地址空间处理

  进程的虚拟地址空间包括了进程可以引用的所有虚拟内存地址。通常包括如下几个内存区：

    - 程序的可执行代码
    - 程序的初始化数据
    - 程序的未初始化数据
    - 初始程序栈
    - 所需共享库的可执行代码和数据
    - 堆
- 高速缓存

  物理内存的一大优势就是用作磁盘和其他块设备的高速缓存。

设备驱动程序

内核通过设备驱动程序 device driver 与 I/O 设备交互。