第三篇　Linux系统内核分析

第8章　Linux内核简介

本章介绍有关Linux的内核内容，如系统初始化、运行以及内核提供的各种调用等。

8.1　系统初始化

当PC启动时，Intel系列的CPU首先进入的是实模式，并开始执行位于地址0xFFFF0处的代码，也就是ROM-BIOS起始位置的代码。BIOS先进行一系列的系统自检，然后初始化位于地址0的中断向量表。最后BIOS将启动盘的第一个扇区装入到0x7C00，并开始执行此处的代码。这就是对内核初始化过程的一个最简单的描述。

最初，Linux核心的最开始部分是用8086汇编语言编写的。当开始运行时，核心将自己装入到绝对地址0x90000，再将其后的2k字节装入到地址0x90200处，最后将核心的其余部分装入到0x10000。当系统装入时，会显示Loading...信息。装入完成后，控制转向另一个实模式下的汇编语言代码boot/Setup.S。

Setup部分首先设置一些系统的硬件设备，然后将核心从0x10000处移至0x1000处。这时系统转入保护模式，开始执行位于0x1000处的代码。

接下来是内核的解压缩。0x1000处的代码来自于文件zBoot/head.S，它用来初始化寄存器和调用decompress_kernel()程序。decompress_kernel()程序由zBoot/inflate.c, zBoot/unzip.c 和zBoot/misc.c组成。解压缩后的数据被装入到了0x100000处，这也是Linux不能在内存小于2M的环境下运行的主要原因。

解压后的代码在0x1010000处开始执行，紧接着所有的32位的设置都将完成：IDT、GDT和 LDT将被装入，处理器初始化完毕，设置好内存页面，最终调用start_kernel过程。这大概是整个内核中最为复杂的部分。

start_kernel()程序用于初始化系统内核的各个部分，包括：

- 设置内存边界，调用paging_init()初始化内存页面。
- 初始化陷阱，中断通道和调度。
- 对命令行进行语法分析。
- 初始化设备驱动程序和磁盘缓冲区。
- 校对延迟循环。

最后，系统核心转向move_to_user_mode()，以便创建初始化进程（init）。此后，进程0开始进入无限循环。

8.2　系统运行

初始化进程开始执行/etc/init、/bin/init 或 /sbin/init中的一个之后，系统内核就不再对程序进行直接控制了。之后系统内核的作用主要是给进程提供系统调用，以及提供异步中断事件的

处理。多任务机制已经建立起来，并开始处理多个用户的登录和 fork()创建的进程。

8.3 内核提供的各种系统调用

8.3.1 进程的基本概念和系统的基本数据结构

从系统内核的角度看来，一个进程仅仅是进程控制表（process table）中的一项。

进程控制表中的每一项都是一个 task_struct结构，而 task_struct结构本身是在include/linux/sched.h中定义的。在task_struct结构中存储各种低级和高级的信息，包括从一些硬件设备的寄存器拷贝到进程的工作目录的链接点。

进程控制表既是一个数组，又是一个双向链表，同时又是一个树。其物理实现是一个包括多个指针的静态数组。此数组的长度保存在 include/linux/tasks.h 定义的常量NR_TASKS中，其缺省值为128，数组中的结构则保存在系统预留的内存页中。链表是由 next_task 和 prev_task两个指针实现的，而树的实现则比较复杂。

系统启动后，内核通常作为某一个进程的代表。一个指向 task_struct的全局指针变量current用来记录正在运行的进程。变量 current只能由kernel/sched.c中的进程调度改变。当系统需要查看所有的进程时，则调用 for_each_task，这将比系统搜索数组的速度要快得多。

某一个进程只能运行在用户方式（user mode）或内核方式（kernel mode）下。用户程序运行在用户方式下，而系统调用运行在内核方式下。在这两种方式下所用的堆栈不一样：用户方式下用的是一般的堆栈，而内核方式下用的是固定大小的堆栈（一般为一个内存页的大小）。

8.3.2 创建和撤消进程

Linux系统使用系统调用 fork()来创建一个进程，使用 exit()来结束进程。 fork()和exit()的源程序保存在kernel/fork.c and kernel/exit.c中。fork()的主要任务是初始化要创建进程的数据结构，其主要的步骤有：

1）申请一个空闲的页面来保存task_struct。
2）查找一个空的进程槽（find_empty_process()）。
3）为kernel_stack_page申请另一个空闲的内存页作为堆栈。
4）将父进程的LDT表拷贝给子进程。
5）复制父进程的内存映射信息。
6）管理文件描述符和链接点。

撤消一个进程可能稍微复杂些，因为撤消子进程必须通知父进程。另外，使用 kill()也可以结束一个进程。 sys_kill()、sys_wait()和sys_exit()都保存在文件exit.c中。

8.3.3 执行程序

使用fork()创建一个进程后，程序的两个拷贝都在运行。通常一个拷贝使用 exec()调用另一个拷贝。系统调用exec()负责定位可执行文件的二进制代码，并负责装入和运行。

Linux系统中的exec()通过使用linux_binfmt结构支持多种二进制格式。每种二进制格式都代表可执行代码和链接库。 linux_binfmt结构种包含两个指针，一个指向装入可执行代码的函数，另一个指向装入链接库的函数。

Unix系统提供给程序员 6种调用 exec()的方法。其中的 5种是作为库函数实现，而

sys_execve()是由系统内核实现的。它执行一个十分简单的任务：装入可执行文件的文件头，并试图执行它。如果文件的头两个字节是#!,那么它就调用在文件第一行中所指定的解释器，否则，它将逐个尝试注册的二进制格式。

8.4 存取文件系统

众所周知，文件系统是Unix系统最基本的资源。最初的Unix系统一般都只支持一种单一类型的文件系统，在这种情况下，文件系统的结构深入到整个系统内核中。而现在的系统大多都在系统内核和文件系统之间提供一个标准的接口，这样不同文件结构之间的数据可以十分方便地交换。Linux也在系统内核和文件系统之间提供了一种叫做VFS（virtual file system）的标准接口。

这样，文件系统的代码就分成了两部分：上层用于处理系统内核的各种表格和数据结构；而下层用来实现文件系统本身的函数，并通过VFS来调用。这些函数主要包括：

- 管理缓冲区(buffer.c)。
- 响应系统调用fcntl() 和 ioctl()(fcntl.c and ioctl.c)。
- 将管道和文件输入/输出映射到索引节点和缓冲区(fifo.c, pipe.c)。
- 锁定和不锁定文件和记录(locks.c)。
- 映射名字到索引节点(namei.c, open.c)。
- 实现select()函数(select.c)。
- 提供各种信息(stat.c)。
- 挂接和卸载文件系统(super.c)。
- 调用可执行代码和转存核心(exec.c)。
- 装入各种二进制格式(bin_fmt*.c)。

VFS接口则由一系列相对高级的操作组成，这些操作由和文件系统无关的代码调用，并且由不同的文件系统执行。其中最主要的结构有inode_operations 和 file_operations。

file_system_type是系统内核中指向真正文件系统的结构。每挂接一次文件系统，都将使用file_system_type组成的数组。file_system_type组成的数组嵌入到了fs/filesystems.c中。相关文件系统的read_super函数负责填充super_block结构。