第9章 系 统 进 程

本章介绍进程的基本概念、进程结构、进程调度以及进程使用的文件等方面的内容。

9.1 什么是进程

进程是运行于自己的虚拟地址空间的一个程序。可以说，任何在 Linix 系统下运行的都是进程。

Linux系统中包括下面几种类型的进程：

- 交互进程：该进程是由shell控制和运行的。它即可以在前台运行，也可以在后台运行。
- 批处理进程：该进程不属于某个终端，它被提交到一个队列中以便顺序执行。
- 守护进程：该进程只有在需要时才被唤起在后台运行。它一般在 Linux启动时开始执行。

进程是动态的，在处理器执行机器代码时进程一直在变化。进程不但包括程序的指令和数据，而且包括程序计数器和CPU的所有寄存器以及存储临时数据的进程堆栈。所以，正在执行的进程包括处理器当前的一切活动。 Linux是一个多进程的操作系统。每个进程都有自己的权限和任务。某一进程的失败一般不会导致其他进程的失败。进程之间可以通过由内核控制的机制相互通讯。

在进程的整个运行期间，它将会用到各种系统资源，会用到 CPU运行它的指令，需要物理内存保存它的数据。它可能打开和使用各种文件，直接或间接地使用系统中的各种物理设备。Linux系统内核必须了解进程本身的情况和进程所用到的各种资源，以便在多个进程之间合理地分配系统资源。

系统中最为宝贵的资源是 CPU，因为一般情况下一个系统只有一个 CPU。Linux是一个多进程的操作系统，所以，其他的进程必须等到正在运行的进程空闲 CPU后才能运行。当正在运行的进程等待其他的系统资源时， Linux内核将取得CPU的控制权，并将CPU分配给其他正在等待的进程。内核中的调度算法决定将CPU分配给那一个进程。

9.2 进程的结构

Linux系统中的每一个进程都包括一个叫做 task_struct的数据结构，而所有指向这些数据结构的指针组成系统中的一个进程向量数组。

缺省情况下，系统的进程向量数组大小是512，这表示系统中同时最多容纳的进程为512个。每当一个新的进程创建时，一个新的 task_struct结构将分配给该进程，并同时增加到进程向量数组中。系统还有一个当前进程指针，用来指向正在运行的进程。

进程的task_struct结构分为以下几个字段：

1) 状态

进程在运行时总是在不停地改变它的状态。在Linux系统中，有以下几个状态：

- 运行态：此时进程或者正在运行，或者准备运行。
- 等待态：此时进程在等待一个事件的发生或某种系统资源。Linux系统分为两种等待进程：可中断的和不可中断的。可中断的等待进程可以被某一信号中断，而不可中断的等待进

程将一直等待硬件状态的改变。

- 停止态：此时进程已经被中止。
- 死亡态：这是一个停止的进程，但还在进程向量数组中占有一个task_struct结构。

2) 调度信息

调度算法需要此信息来决定系统中的那一个进程需要执行。

3) 标识符

系统中的每一个进程都有一个进程标识符。进程标识符并不是指向进程向量的索引。每个进程同时还包括用户标识符和工作组标识符。

4）内部进程通讯

Linux系统支持信号、管道、信号量等内部进程通讯机制。

5）链接

在Linux系统中，每个进程都和其他的进程有所联系。除了初始化进程，其他的进程都有父进程。一个新的进程一般都是由其他的进程复制而来的。 task_struct结构中包括指向父进程，兄弟进程和子进程的指针。

6）时间和计时器

内核需要记录进程的创建时间和进程运行所占用的 CPU的时间。Linux系统支持进程特殊间隔计时器。进程可以使用系统调用设置计时器，并当计时器失效时给进程一个信号。计时器可以是一次性的或周期性的。

7）文件系统

进程在运行时可以打开和关闭文件。task_struct结构中包括指向每个打开文件的文件描述符的指针，并且包括两个指向VFS索引节点的指针。VFS的索引节点用来在文件系统内唯一地描述一个文件或目录，并且提供文件系统操作的统一的接口。第一个索引节点是进程的根目录，第二个节点是当前的工作目录。两个VFS索引节点都有一个计数字段用来表明指向节点的进程数。

8）虚拟内存

大多数的进程都需要虚拟内存。 Linux系统必须了解如何将虚拟内存映射到系统的物理内存。

9）处理器的内容

一个进程可以说是系统当前状态的总和。每当一个进程正在运行时，它都要使用处理器的寄存器及堆栈等资源。当一个进程挂起时，所有有关处理器的内容都要保存到进程的task_struct中。当进程恢复运行时，所有保存的内容再装入到处理器中。

在一个进程的task_struct中，有4对进程和组标识符：

- uid, gid：正在运行的进程用户标识符和组标识符。
- 有效uid和 gid：有些程序可能将正在运行的进程的uid和gid改为自己所有。这些程序一般被称为setuid程序，它们十分有用，因为这是一种限制服务存取权限的方法。有效 uid和gid是那些setuid程序的uid 和gid，并且它们保持不变。每当内核检查权限时，都要检查有效uid和gid。
- 文件系统uid 和gid：文件系统uid和gid一般和有效uid 和gid相同，它们用于检查文件系统的存取权限。
- 保留 uid 和gid：它们用来和POSIX标准兼容。在程序通过系统调用改变进程的 uid和gid时，它们可以保存真正的uid 和gid 。

9.3 进程调度

所有进程都是既运行于用户方式下，又运行于系统方式下。这就需要有一个安全机制便于在两种方式下切换。用户方式的权限要比系统方式的权限小得多。进程每次调用一个系统调用时，都要从用户方式切换到系统方式，并继续执行。在 Linux系统中，进程没有绝对的优先权，也就是说一个进程不能停止另一个正在运行的进程以便运行它自己。每个进程根据自己是否需要等待某些系统资源以决定是否放弃所占用的 CPU。例如，一个进程需要等待从一个文件中读取字符，该等待在系统方式下的系统调用中发生，此时，该进程将被挂起，系统选择另一个进程继续运行。

虽然进程需要经常调用系统调用，但如果一个进程要直到系统调用发生时才放弃 CPU,那么它也会占用过多的CPU时间。因此， Linux系统使用一种时间片的调度算法。每个进程只能运行200ms，然后放弃CPU给其他可以运行的进程，该进程将在以后恢复运行。

可以运行的进程是指只需要 CPU后就可以运行的进程。 Linux系统使用一个基于调度算法的简单权限来决定运行哪个进程。每个进程的task_struct结构中包括下面这些调度信息：

1）策略。

这是进程将会使用的调度策略。 Linux系统共有两种类型的进程：一般进程和实时进程。实时进程的权限要比其他进程的权限高。如果有一个实时进程等待运行，那么一般情况下都将首先运行。实时进程也有两种策略：轮流策略和先进先出策略。在轮流策略中，每个进程轮流运行，而在先进先出策略中，进程按照运行队列中的顺序执行，并且运行队列的顺序永远不变。

2）优先权。

这是调度算法给予进程的优先权，也即当进程被允许运行时能够运行的时间的长短。你可以通过系统调用改变此优先权。

3）实时优先权。

这是给予实时进程之间的一个相对的优先权。你也可以通过系统调用改变此实时进程的优先权。

4）计数器。

这是进程允许运行的时间。当进程第一次运行时，它将被设置为进程的优先权，并且在每个时钟周期后减一。

调度算法可以在多种情况下发生。它可以在将当前进程放入等待队列时发生，也可以在一个系统调用后发生。当调度算法发生时，它将执行以下几个步骤：

1）处理内核中的工作。

2）处理当前进程。

在其他进程运行之前，当前进程必须处理好。

- 如果当前进程使用的是轮流策略，则当前进程被放到运行队列的最后。
- 如果当前进程是可以被中断的，并且在最后一次调度之后接收到过中断信号，则进程的状态被设置为可运行。
- 如果当前进程的运行时间用完，则进程的状态设置为可运行。
- 如果当前进程为可运行，则进程状态保持为可运行。
- 那些既不是可运行也不是可中断的进程将被从运行队列中移走。

3）选择进程。

调度算法查找运行队列以找出最需要运行的进程。如果队列中有实时进程，那么实时进程将优先运行。一个普通进程的优先权是其计数器的值，而实时进程的优先权是其计数器的值加上1000。当前进程由于已经消耗了一些时间片，所以和其他的具有相同优先权的进程相比将处于不利的位置。如果有几个进程具有相同的优先权，则最靠近队列前端的进程将被执行。

4）进程交换。

如果最需要执行的进程不是当前进程，那么当前进程就会被挂起，同时一个新的进程将被执行。在结束当前进程时，进程所涉及的一切机器状态，包括程序计数器以及 CPU寄存器将保存到进程的task_struct中，而即将运行的进程的task_struct中的状态将装入到机器中。如果当前进程和即将运行的进程使用了虚拟内存的话，系统的内存页面表页会被更新。

9.4 进程使用的文件

系统中的每一个进程都包括两个描述文件系统特定信息的数据结构，如图 9-1所示。一个是fs_struct，它包括指向进程的VFS索引节点和指向进程的umask的指针。umask是创建新文件时的缺省模式，可以通过系统调用来改变。另一个数据结构是 files_struct，它包括进程正在使用的所有文件的信息。程序从标准输入设备中读入信息，并将输出信息写入标准输出设备中。这些设备在程序看来都是文件，每一个文件都包括自己的文件描述符。 files_struct结构中包括多达256个的文件数据结构，每个文件数据结构都描述一个进程正在使用的文件。文件数据结构中的f_mode用来描述创建文件的方式，例如只读，可读写或只写。 f_pos保存文件中下一个读写操作将要发生的位置。 f_inode指向VFS的索引节点。 f_op是一个指向包括一系列文件操作程序地址的向量的指针。

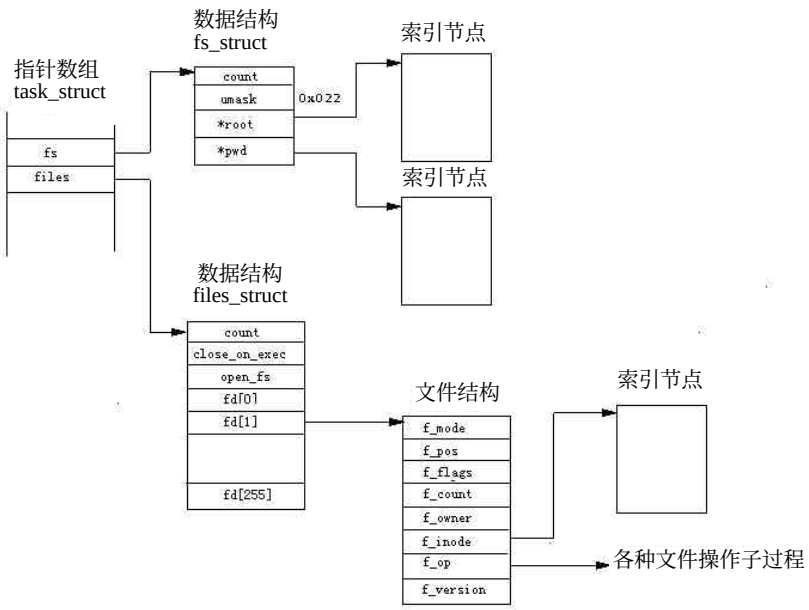

图9-1 文件数据结构示意图

每当进程打开一个文件时， files_struct的一个空的文件指针就用来指向新文件结构。在Linux系统中，一个进程在打开时就已经存在三个文件描述符，这三个文件描述符分别是标准输入、标准输出和标准错误文件，它们在进程的文件数据结构向量中分别是 0、1、2。

9.5 进程使用的虚拟内存

进程的虚拟内存中包括可执行代码和数据。首先，程序的镜像被装入，它包括可执行代码和数据。然后，进程分配处理过程中需要使用的虚拟内存。这些新分配的虚拟内存应该被链接到进程已经存在的虚拟内存。最后， Linux系统使用共享库，这些共享库中的代码和数据也应该被链接到进程的虚拟内存地址空间中。

因为在一个固定的时间中，进程只用到一部分代码和数据，所以把进程的全部代码和数据都装入到物理内存将是十分浪费的。 Linux使用一种需求装入的技术，也就是说，只有当进程试图存取一个虚拟内存页时，此虚拟内存页才被装入到物理内存中。所以， Linux内核修改进程的内存页面表，标记出不在物理内存中的虚拟内存页。当进程试图存取虚拟内存页时，系统硬件将产生一个页面错误，并将系统的控制权交给 Linux内核以处理此错误。因此，对于进程的地址空间中的每一个虚拟内存页， Linux内核都将知道它的来源以及如何将它调入到物理内存，以便修正页面错误。

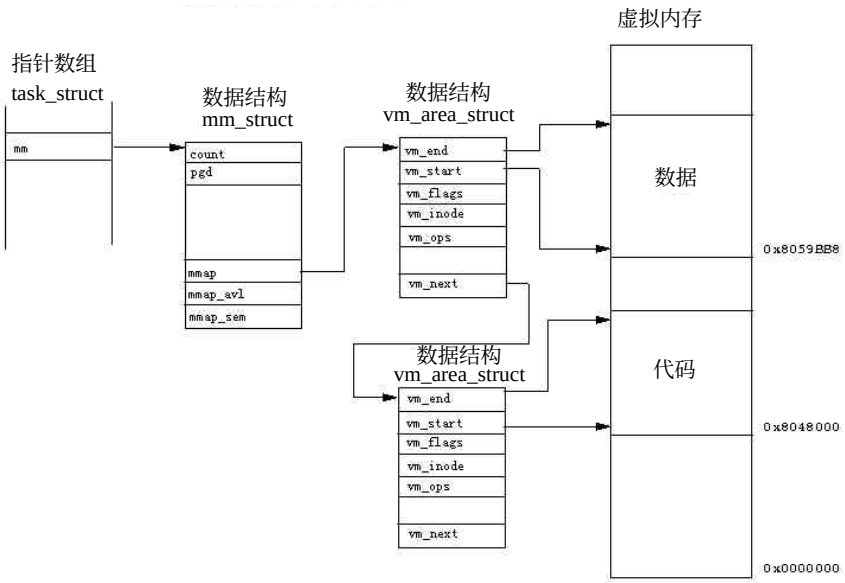

图9-2 进程使用的虚拟内存示意图

每一个进程的虚拟内存的内容都由进程的 task_struct中的mm指针指向的mm_struct结构来描述，如图9-2所示。进程的mm_struct结构包括已装入的可执行镜像的信息和一个指向进程页面表的指针。进程页面表由一系列的 vm_area_struct结构组成，每一个vm_area_struct结构代表进程内一片虚拟内存区域。 vm_area_struct结构中还包括vm_ops指针，用来指向一系列的虚拟内存处理程序。

当Linux内核为进程创建新的虚拟内存区或当 Linux内核处理虚拟内存页面错误时，内核都将经常存取进程的vm_area_struct结构。所以快速查找正确的 vm_area_struct就成为系统性能的关键。为了加快系统的查找时间， Linux同时将vm_area_struct组成了AVL (Adelson-Velskii and Landis)树。这种树的结构使得每一个 vm_area_struct节点都有各一个左右指针指向节点的左右节点。这样，当内核查找正确的 vm_area_struct时，可以从根节点开始从左右两个方向同时查找，大大提高了查找的效率。当然，插入一个vm_area_struct时，将会需要额外的处理时间。

9.6 创建进程

当系统刚刚启动时，系统运行于内核方式，也就是只有一个初始化进程在运行和其他进程一样，初始化进程也是由一系列机器状态组成的。当其他进程创建并运行时，初始化进程的状态被保存到初始化进程的task_struct结构中。

初始化进程的进程标识符是 1，它是系统的第一个真正的进程。它首先做一些系统的初始化设置（例如打开系统控制台和挂接根目录文件系统），然后执行系统初始化程序。初始化程序是/etc/init、/bin/init 或 /sbin/init中的一个。初始化程序使用/etc/inittab作为脚本文件来创建新的进程。这些新的进程同样可以创建其他新的进程。系统中所有的进程都是初始化进程的子进程。

新进程是通过复制老进程或当前进程而创建的。新进程的创建使用系统调用 fork()或者clone(),并且是在内核内部的内核方式下完成的。系统调用结束时，如果调度算法选择的话，新进程就可以准备运行。系统从物理内存中分配给新进程一个 task_struct数据结构和进程堆栈。新的task_struct结构加入到进程向量中。进程还得到一个和系统中的其他进程不同的唯一的标识符。

在复制进程时，Linux允许两个进程共享系统资源，包括进程要用到的文件、信号处理程序以及虚拟内存等。当资源共享时，共享资源的计数字器将会增加。这样，除非所有使用该资源的进程都停止运行，否则该资源不会被删除。

复制一个进程的虚拟内存是十分麻烦的。首先需要建立一系列的 vm_area_struct数据结构和它们的所有者 mm_struct，还需复制进程的页面表。此时并没有真正复制进程的虚拟内存，因为此时复制虚拟内存是十分麻烦的：一部分虚拟内存在物理内存中，一部分虚拟内存在进程可执行镜像中，还可能一部分内存在交换文件中。因此， Linux使用一种叫做“写入时复制”的技术，也就是只有当父进程或子进程试图写入虚拟内存时，子进程的虚拟内存才将会被复制。任何虚拟内存只要不执行写入，即使可以写入，也是可以被进程共享的。例如，可执行代码通常都是共享的。使用“写入时复制”技术时，可以写入的虚拟内存区的页面表入口标记为 read only，而虚拟内存的vm_area_struct数据结构标记为copy on write。当一个进程试图写入这样的虚拟内存页时，将会发生一个页面错误。这时， Linux将会复制虚拟内存，并修改两个进程的页面表和虚拟内存数据结构。

9.7 进程的时间和计时器

内核中保存有进程的创建时间和进程运行时消耗的 CPU时间。除了计时器， Linux还支持一些间隔时钟，用来在一定的时间间隔后产生信号中断。系统中的间隔时钟共有三种：

9.7.1 实时时钟

该时钟按实时方式运行，失效时产生一个SIGALRM信号。

9.7.2 虚拟时钟

该时钟只在进程运行时才运行，失效时产生一个SIGVTALRM信号。

9.7.3 形象时钟

该时钟在进程运行和在系统代表进程运行时都运行，失效时产生一个 SIGPROF 信号。

9.8 程序的执行

在Linux中，程序和命令是由命令解释器 shell解释执行的。当键入一个命令时， shell搜索保存在PATH环境变量中的进程及路径里的目录，找出和命令名相同的可执行镜像，如果发现则装入并执行。系统的 shell首先使用 fork()系统调用复制自己，然后子进程用找到的可执行的二进制文件的内容替换正在执行的 shell二进制文件。一般情况下， shell要等待命令运行结束或子进程的退出。你也可以按 CTRL+Z键以产生一个 SIGSTOP信号来停止子进程，以重新恢复 shell的运行。之后，可以使用bg命令将进程推入后台运行。

可执行文件可以有各种形式，甚至可能只是一个脚本文件。可执行的目标文件包括可执行代码和数据，以便操作系统装入和执行。 Linux系统最为常用的目标文件格式是 EFL，但它还可以处理大多数格式的可执行目标文件。

9.8.1 ELF文件

ELF（Executable and Linkable Format）文件格式是各种Unix系统中最为常用的格式。虽然EFL文件和其他格式的文件 (例如ECOFF 和a.out)相比系统开销稍大，但EFL文件更为灵活。EFL可执行文件包括可执行代码和数据。可执行镜像中的表格描述了程序应该装入到进程虚拟内存的位置。可执行镜像还指定了镜像在内存中的分布和镜像中第一条执行代码的地址。

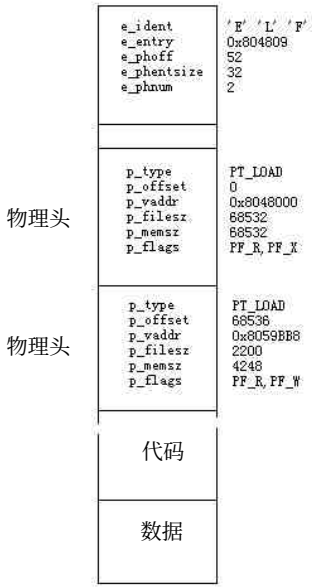

图9-3 ELF可执行文件镜像示意图

图9-3是一个静态链接的EFL可执行镜像的布局。

这是一个显示hello world的简单C语言程序。文件头表示这是一个 EFL镜像，带有两个物理文件头 (e_phnum的值为 2)，物理文件头从镜像开头的第 52个字节开始(e_phoff的值为52)。第一个物理文件头描述了镜像中的可执行代码。可执行代码从虚拟内存的0x8048000地址开始，一共有 65532字节。这是因为此镜像是一个静态链接镜像，它包括库函数 printf()的所有代码。镜像的入口点，也就是程序的第一条指令，是在虚拟内存地址 0x8048090 (e_entry的值为0x8048090)。可执行代码紧跟在第二个物理文件头的后面。第二个物理文件头用来描述程序的数据。程序的数据部分被装入到虚拟内存的地址 0x8059BB8。这些数据是可读写的（p_flags的值为PF_R、PF_w)。

9.8.2 脚本文件

脚本文件需要一个解释器以便运行。 Linux系统中包括很多的解释器，例如 perl 和不同的 shell程序。Linux脚本文件的第一行是运行此脚本文件的解释器的名称，

Linux系统在运行脚本文件时，先试图打开文件第一行中的可执行文件。如果可执行文件可以打开，则它获得一个指向VFS索引节点的指针，并解释执行脚本文件。