第3章 嵌入式 Linux 操作系统

Linux 操作系统的诸多优势，完全符合嵌入式系统对操作系统的“高度简练、界面友善、质量可靠、应用广泛、易开发、多任务，并且价格低”的要求，为嵌入式操作系统提供了一个极有吸引力的选择。近年来，由于价格低廉、功能强大又易于移植，嵌入式 Linux 操作系统被广泛采用，众多商家纷纷转向了嵌入式 Linux 开发和应用。嵌入式 Linux 成为嵌入式操作系统领域的主流。

本章将首先对三个主流的嵌入式 Linux 进行简介，然后从内存管理、进程管理和文件系统进一步详细介绍嵌入式 Linux 操作系统。

通过本章的学习，读者可以获得以下知识点。

(1) 常见的嵌入式 Linux 操作系统；

(2) 嵌入式 Linux 的内存管理；

(3) 嵌入式 Linux 的进程管理；

(4) 嵌入式 Linux 的文件系统。

3.1 嵌入式 Linux 简介

由于嵌入式 Linux 有着广阔的发展前景，国际上和国内的一些研究机构和知名企业都投入大量的人力和物力，力争在嵌入式 Linux 上有所为。目前，国内外主流的嵌入式 Linux 操作系统主要有以下几种。

3.1.1 μCLinux

在 μCLinux 这个英文单词中 μ 表示 Micro，小的意思，C 表示 Control，控制的意思，所以 μCLinux 代表着 Micro-Control-Linux，字面上的意思是“针对微控制领域而设计的 Linux 系统”。μCLinux 是 Lineo 公司的主打产品，同时也是开放源码的嵌入式 Linux 的典范之作。

μCLinux 秉承了标准 Linux 的优良特性，经过各方面的小型化改造，形成了一个高度优化的、代码紧凑的嵌入式 Linux。虽然它的体积很小，却仍然保留了 Linux 的大多数的优点：稳定、良好的移植性、优秀的网络功能、对各种文件系统完备的支持和标准丰富的 API。

最初的 μCLinux 仅支持 Palm 硬件系统，基于 Linux 2.0 内核。随着系统的日益改进，支持的内核版本从 2.0、2.2、2.4 一直到现在最新版本的 2.6(目前最新的 Linux 内核为 4.7.2)。编译后目标文件可控制在几百 KB 数量级，并已经被成功地移植到很多平台上。

大部分嵌入式系统为了减少系统复杂程度、降低硬件及开发成本和运行功耗，在硬件设计中取消了内存管理单元(MMU)模块。μCLinux 是专门针对没有 MMU 的处理器而设计的，即 μCLinux 无法使用处理器的虚拟内存管理技术。μCLinux 采用实存储器管理策略，通过地

址总线对物理内存进行直接访问。所有程序中访问的地址都是实际的物理地址,所有的进程都在一个运行空间中运行(包括内核进程),这样的运行机制给程序员带来了不小的挑战,在操作系统不提供保护的情况下必须小心设计程序和数据空间,以免引起应用程序进程甚至是内核的崩溃。

MMU的省略虽然带来了系统及应用程序开发的限制,但对于成本和体积敏感的嵌入式设备而言,其应用环境和应用需求并不要求复杂和相对昂贵的硬件体系,对于功能简单的专用嵌入式设备,内存的分配和管理完全可以由开发人员考虑。

3.1.2 RT-Linux

RT-Linux就是Real-time Linux的简写,RT-Linux是源代码开放的具有硬实时特性的多任务操作系统,它部分支持POSIX.1b标准。RT-Linux是美国新墨西哥州大学计算机科学系Victor Yodaiken和Micae Brannanov开发的嵌入式Linux操作系统。

RT-Linux开发者并没有针对实时操作系统的特性而重写Linux的内核,因为这样做的工作量非常大,而且要保证兼容性也非常困难。而是通过底层对Linux实施改造的产物。通过在Linux内核与硬件中断之间增加一个精巧的可抢先的实时内核,把标准的Linux内核作为实时内核的一个进程与用户进程一起调度,标准的Linux内核的优先级最低,可以被实时进程抢断。正常的Linux进程仍可以在Linux内核上运行,这样既可以使用标准分时操作系统即Linux的各种服务,又能提供低延时的实时环境。

到目前为止,RT-Linux已经成功地应用于航天飞机的空间数据采集、科学仪器测控和电影特技图像处理等广泛领域。

3.1.3 红旗嵌入式Linux

红旗嵌入式Linux是由北京中科红旗软件技术有限公司推出的,是国内做得较好的一款嵌入式Linux操作系统。这款嵌入式Linux具有以下特点。

(1) 精简内核,适用于多种常见的嵌入式CPU;

(2) 提供完善的嵌入式GUI和嵌入式X-Windows;

(3) 提供嵌入式浏览器、邮件程序和多媒体播放程序;

(4) 提供完善的开发工具和平台。

3.2 内存管理

3.2.1 内存管理和MMU

存储管理包含地址映射、内存空间的分配,有时候还包括地址访问的限制(即保护机制);如果将I/O也放在内存地址空间中,则还要包括I/O地址的映射;另外,像代码段、数据段、堆栈段空间的分配等都属于内存管理。对内核来讲,存储管理机制的实现和具体的CPU以及MMU的结构关系非常紧密,所以存储管理,特别是地址映射,是操作系统内核中比较复杂的一个成分。甚至可以说操作系统内核的复杂性相当程度上来自内存管理,对整个系统的结构有着根本性的深远影响。

MMU,也就是"内存管理单元",其主要作用是两个方面:一是地址映射;二是对地址访

问的保护和限制。简单地说，MMU 就是提供一组寄存器，依靠这组寄存器来实现地址映射和访问保护。MMU 可以做在芯片中，也可以作为协处理器。

由于地址映射是通过 MMU 实现的，因此不采用地址映射就不需要 MMU。但是严格地说，内存的管理总是存在的，只是方式和复杂程度不同而已。

3.2.2　标准 Linux 的内存管理

标准 Linux 使用虚拟存储器技术，提供比计算机系统中实际使用的物理内存大得多的内存空间，从而使得编程人员再不用考虑计算机中的物理内存容量。

为了支持虚拟存储管理器的管理，Linux 系统采用分页的方式来载入进程。所谓分页即是把实际的存储器分割为相同大小的段，例如每个段 1024B，这样 1024B 大小的段便称为一个页面。

虚拟存储器由存储器管理机制及一个大容量的快速硬盘存储器支持。它的实现基于局部性原理，当一个程序在运行之前，没有必要全部装入内存，而是仅将那些当前要运行的部分页面或段装入内存运行(Copy-on-Write)。其余暂时留在硬盘上，程序运行时如果它所要访问的页(段)已存在，则程序继续运行，如果发现不存在的页(段)，操作系统将产生一个页错误，这个错误导致操作系统把需要运行的部分加载到内存中。必要时操作系统还可以把不需要的内存页(段)交换到磁盘上。利用这样的方式管理存储器，便可把一个进程所需要用到的存储器以化整为零的方式，视需求分批载入，而核心程序则凭借属于每个页面的页码来完成寻址各个存储器区段的工作。

标准 Linux 是针对有内存管理单元的处理器设计的。在这种处理器上，虚拟地址被送到 MMU，把虚拟地址映射为物理地址。

通过赋予每个任务不同的虚拟-物理地址转换映射，支持不同任务之间的保护。地址转换函数在每一个任务中定义，在一个任务中的虚拟地址空间映射到物理内存的一个部分，而另一个任务的虚拟地址空间映射到物理存储器中的另外区域。计算机的存储管理单元(MMU)一般有一组寄存器来标识当前运行的进程的转换表。在当前进程将 CPU 放弃给另一个进程时(一次上下文切换)，内核通过指向新进程地址转换表的指针加载这些寄存器。MMU 寄存器是有特权的，只能在内核态才能访问。这就保证了一个进程只能访问自己用户空间内的地址，而不会访问和修改其他进程的空间。当可执行文件被加载时，加载器根据默认的 ld 文件，把程序加载到虚拟内存的一个空间，因为这个原因实际上很多程序的虚拟地址空间是相同的，但是由于转换函数不同，所以实际所处的内存区域也不同。而对于多进程管理，当处理器进行进程切换并执行一个新任务时，一个重要部分就是为新任务切换任务转换表。

标准 Linux 操作系统的内存管理至少实现了以下功能。

(1) 运行比内存还要大的程序。

(2) 先加载部分程序运行，缩短了程序启动的时间。

(3) 可以使多个程序同时驻留在内存中提高 CPU 的利用率。

(4) 可以运行重定位程序。即程序可以放于内存中的任何一处，且可以在执行过程中移动。

(5) 写机器无关的代码。程序不必事先约定机器的配置情况。

(6) 减轻程序员分配和管理内存资源的负担。

(7) 可以进行内存共享。

(8) 提供内存保护，进程不能以非授权方式访问或修改页面，内核保护单个进程的数据和代

码以防止其他进程修改它们。否则,用户程序可能会偶然(或恶意)地破坏内核或其他用户程序。

当然,虚存系统并不是没有代价的。内存管理需要地址转换表和其他一些数据结构,留给程序的内存减少了。地址转换增加了每一条指令的执行时间,而对于有额外内存操作的指令会更严重。当进程访问不在内存的页面时,系统发生失效。系统处理该失效,并将页面加载到内存中,这需要极耗时间的磁盘I/O操作。内存管理活动占用了相当一部分CPU时间。

3.2.3 μCLinux的内存管理

对于μCLinux来说,其设计针对没有MMU的处理器,即μCLinux不能使用处理器的虚拟内存管理技术。在μCLinux中,系统为进程分配的内存区域是连续的,代码段、数据段和栈段间没任何空隙。为节省内存,进程的私有堆被取消,所有进程共享一个由操作系统管理的堆空间。标准Linux和μCLinux的内存映射区别如图3-1所示。

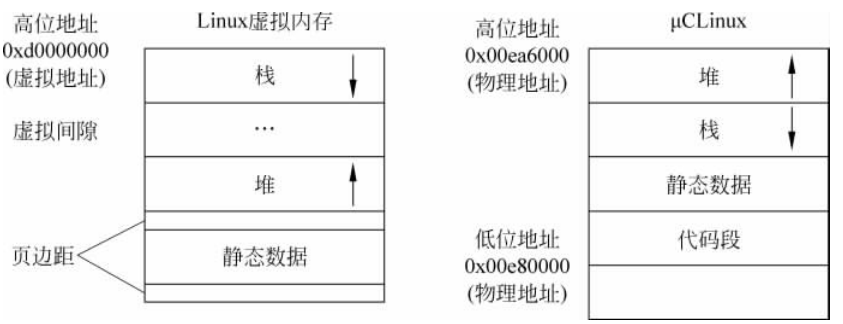

图3-1 标准Linux和μCLinux的内存映射

μCLinux不能使用处理器的虚拟内存管理技术,它仍然采用存储器的分页管理。系统启动时对存储器分页,加载应用程序对程序分页加载。由于没有MMU管理,所以μCLinux采用实存储器管理(Real Memory Management)。μCLinux系统对内存的访问是直接的(它对地址的访问不经MMU,而是直接送到地址线上输出),所有程序访问的地址是物理地址。那些比物理内存还大的程序将无法执行。

μCLinux将整个物理内存划分成为4KB的页面。由数据结构page管理,有多少页面就有多少page结构,它们又作为元素组成数组mem_map[]。物理页面可作为进程代码、数据和堆栈的一部分,还可存储装入的文件,也可作缓冲区。跟很多嵌入式操作系统一样,μCLinux操作系统对内存空间没有保护,各个进程没有独立的地址转换表,实际上共享一个运行空间。

一个进程在执行前,系统必须为进程分配足够的连续物理地址空间,然后全部载入主存储器的连续空间中。另外一个方面,程序加载地址与预期(ld文件中指出的)通常都不相同,这样Relocation过程就是必需的。此外,磁盘交换空间也是无法使用的,系统执行时如果缺少内存将无法通过磁盘交换来得到改善。

从易用性这一点来说,μCLinux的内存管理是一种倒退,退回到了UNIX早期或是DOS系统时代。开发人员不得不参与系统的内存管理。从编译内核开始,开发人员必须告诉系统这块开发板到底拥有多少的内存,从而系统将在启动的初始化阶段对内存进行分页,并且标记已使用的和未使用的内存。系统将在运行应用时使用这些分页内存。

由于应用程序加载时必须分配连续的地址空间,而针对不同硬件平台的可一次成块(连续地址)分配内存大小限制是不同的,所以开发人员在开发应用程序时必须考虑内存的分配情况

并关注应用程序需要运行空间的大小。另外,由于采用实存储器管理策略,用户程序同内核以及其他用户程序在一个地址空间,程序开发时要保证不侵犯其他程序的地址空间,以使得程序不至于破坏系统的正常工作,或导致其他程序的运行异常。

从内存的访问角度来看,开发人员的权力增大了(开发人员在编程时可以访问任意的地址空间),但与此同时系统的安全性也大为下降。

虽然 μCLinux 的内存管理与标准 Linux 系统相比功能相差很多,但应该说这是嵌入式设备的选择。在嵌入式设备中,由于成本等敏感因素的影响,普遍采用不带有 MMU 的处理器,这决定了系统没有足够的硬件支持实现虚拟存储管理技术。从嵌入式设备实现的功能来看,嵌入式设备通常在某一特定的环境下运行,只要实现特定的功能,其功能相对简单,内存管理的要求完全可以由开发人员考虑。

3.3 进 程 管 理

3.3.1 进程和进程管理

进程是一个运行程序并为其提供执行环境的实体,它包括一个地址空间和至少一个控制点,进程在这个地址空间上执行单一指令序列。进程地址空间包括可以访问或引用的内存单元的集合,进程控制点通过一个一般称为程序计数器(Program Counter,PC)的硬件寄存器控制和跟踪进程指令序列。

任务调度主要是协调任务对计算机系统内资源(如内存、I/O 设备、CPU)的争夺使用。进程调度又称为 CPU 调度,其根本任务是按照某种原则为处于就绪状态的进程分配 CPU。由于嵌入式系统中内存和 I/O 设备一般都和 CPU 同时归属于某进程,因此任务调度和进程调度概念相近,很多场合下,两者的概念是一致的。本文统一以进程来进行说明。

下面先来看几个进程管理相关的概念。

上下文(Context):上下文是指进程运行的环境。例如,针对 x86 的 CPU,进程上下文可包括程序计数器、堆栈指针、通用寄存器的内容。

上下文切换(Context Switching):多任务系统中,上下文切换是指 CPU 的控制权由运行进程转移到另外一个就绪进程时所发生的事件,当前运行进程转为就绪(或者挂起、删除)状态,另一个被选定的就绪进程成为当前进程。上下文切换包括保存当前进程的运行环境,恢复将要运行进程的运行环境。上下文的内容依赖于具体的 CPU。

抢占(Preemptive):抢占是指当系统处于核心态运行时,允许进程的重新调度。换句话说就是指正在执行的进程可以被打断,让另一个进程运行。抢占提高了应用对异步事件的响应能力。操作系统内核可抢占,并不是说任务调度在任何时候都可以发生。例如,当一个任务正在通过一个系统调用访问共享数据时,重新调度和中断都被禁止。

优先抢占(Preemptive Priority):每一个进程都有一个优先级,系统核心保证优先级最高的进程运行于 CPU。如果有进程优先级高于当前的进程优先级,系统立刻保存当前进程的上下文,切换到优先级高的进程的上下文。

轮转调度(Round-Robin Scheduling):使所有相同优先级,状态为就绪的进程公平分享 CPU(分配一定的时间间隔,使各进程轮流享有 CPU)。

进程调度策略可分为“抢占式调度”和“非抢占式调度”两种基本方式。所谓“非抢占式调

度”是指：一旦某个进程被调度执行，则该进程一直执行下去直至该进程结束，或由于某种原因自行放弃CPU进入等待状态，才将CPU重新分配给其他进程。所谓“抢占式调度”是指：一旦就绪状态中出现优先权更高的进程，或者运行的进程已用满了规定的时间片时，便立即抢占当前进程的运行(将其放回就绪状态)，把CPU分配给其他进程。

3.3.2 RT-Linux的进程管理

RT-Linux有两种中断：硬中断和软中断。软中断是常规Linux内核中断。它的优点在于可无限制地使用Linux内核调用。硬中断是实现实时Linux的前提。依赖于不同的系统，实时Linux下硬中断的延迟低于15μs。RT-Linux通过一个高效的、可抢占的实时调度核心来全面接管中断，并把Linux作为此实时核心的一个优先级最低的进程运行。当有实时任务需要处理时，RT-Linux运行实时任务；无实时任务时，RT-Linux运行Linux的非实时进程。其系统结构如图3-2所示。

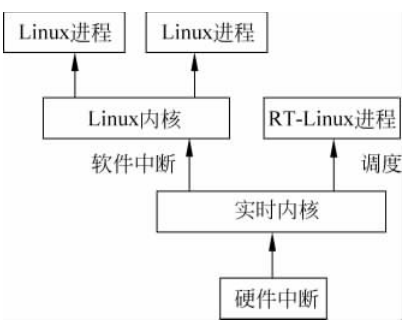

图3-2 RT-Linux的系统结构

在Linux进程和硬件中断之间，本来由Linux内核完全控制，现在Linux内核和硬件中断的地方加上了一个RT-Linux内核的控制。Linux的控制信号都要先交给RT-Linux内核进行处理。在RT-Linux内核中实现了一个虚拟中断机制，Linux本身永远不能屏蔽中断，它发出的中断屏蔽信号和打开中断信号都修改成向RT-Linux发送一个信号。如在Linux里面使用“SI”和“CLI”宏指令，让RT-Linux里面的某些标记做了修改。也就是说将所有的中断分成Linux中断和实时中断两类。如果RT-Linux内核接收到的中断信号是普通Linux中断，那就设置一个标志位；如果是实时中断，就继续向硬件发出中断。在RT-Linux中执行STI将中断打开之后，那些设置了标志位表示的Linux中断就继续执行，因此CLI并不能禁止RT-Linux内核的运行，却可以用来中断Linux。Linux不能中断自己，而RT-Linux可以。总之，RT-Linux将标准Linux内核作为实时操作系统里优先权最低的线程来运行，从而避开了Linux内核性能的问题。RT-Linux仿真了Linux内核所看到的中断控制器。这样即使在被CPU中断，同时Linux内核请求被取消的情况下，关键的实时中断也能够保持激活。

RT-Linux在默认的情况下采用优先级的调度策略，系统进程调度器根据各个实时任务的优先级来确定执行的先后次序。优先级高的先执行，优先级低的后执行，这样就保证了实时进程的迅速调度。同时RT-Linux也支持其他的调度策略，如最短时限最先调度(EDP)、确定周期调度(RM)(周期段的实时任务具有高的优先级)。RT-Linux将任务调度器本身设计成一个可装载的内核模块，用户可以根据自己的实际需要，编写适合自己的调度算法。

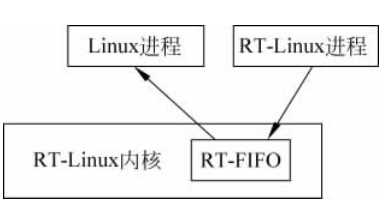

图3-3 带RT-FIFO的RT-Linux系统运行图

为保证RT-Linux实时进程与非实时Linux进程进行数据交换，RT-Linux引入了RT-FIFO队列。RT-FIFO被Linux视为字符设备，分别命名为/der/rtf0、/dev/rtf1，一直到/dev/rtf63。最大的RT-FIFO数量在系统内核编译时设定，最多可达150个。带RT-FIFO的RT-Linux系统运行图如图3-3所示。

RT-Linux 程序运行于用户空间和内核态两个空间。RT-Linux 提供了应用程序接口。借助这些 API 函数将实时处理部分编写成内核模块，并装载到 RT-Linux 内核中，运行于 RT-Linux 的内核态。非实时部分的应用程序则在 Linux 下的用户空间中执行。这样可以发挥 Linux 对网络和数据库的强大支持功能。

3.3.3　标准 Linux 的进程管理

1. Linux 进程

进程是由进程标识符(PID)表示的。从用户的角度来看，一个 PID 是一个数字值，可唯一标识一个进程。一个 PID 在进程的整个生命期间不会更改，但 PID 可以在进程销毁后被重新使用。进程的其他重要属性还有以下几个。

(1) 父进程和父进程的 ID(PPID)。

(2) 启动进程的用户 ID(UID)和所归属的组(GID)。

(3) 进程状态：状态分为运行 R、休眠 S、僵尸 Z。

(4) 进程执行的优先级。

(5) 进程所连接的终端名。

(6) 进程资源占用：比如占用资源大小(内存、CPU 占用量)。

由于进程为执行程序的环境，因此在执行程序前必须先建立这个能运行程序的环境。Linux 系统提供系统调用复制现行进程的内容，以产生新的进程，调用 fork 的进程称为父进程；而所产生的新进程则称为子进程。子进程会承袭父进程的一切特性，但是它有自己的数据段，也就是说，尽管子进程改变了所属的变量，却不会影响到父进程的变量值。父进程和子进程共享一个程序段，但是各自拥有自己的堆栈、数据段、用户空间以及进程控制块。

当内核收到 fork 请求时，它首先检查存储器是不是足够；其次是进程表是否仍有空缺；最后是看看用户是否建立了太多的子进程。如果上述三个条件满足，那么操作系统会给子进程一个进程识别码，并且设定 CPU 时间，接着设定与父进程共享的段，同时将父进程的 inode 复制一份给子进程应用，最终子进程会返回数值 0 以表示它是子进程，至于父进程，它可能等待子进程执行结束，或与子进程各做各的。

Linux 进程还可以通过 exec 系统调用产生。该系统调用提供一个进程去执行另一个进程的能力，exec 系统调用是采用覆盖旧进程存储器内容的方式，所以原来程序的堆栈、数据段与程序段都会被修改，只有用户区维持不变。

由于在使用 fork 时，内核会将父进程复制一份给子进程，但是这样做相当浪费时间，因为大多数情形都是程序在调用 fork 后就立即调用 exec，这样刚复制的进程区域又立即被新的数据覆盖掉。因此 Linux 系统提供一个系统调用 vfork，vfork 假定系统在调用完成 vfork 后会马上执行 exec，因此 vfork 不复制父进程的页面，只是初始化私有的数据结构与准备足够的分页表。这样实际在 vfork 调用完成后父子进程事实上共享同一块存储器(在子进程调用 exec 或是 exit 之前)，因此子进程可以更改父进程的数据及堆栈信息，因此 vfork 系统调用完成后，父进程进入睡眠，直到子进程执行 exec。当子进程执行 exec 时，由于 exec 要使用被执行程序的数据，代码覆盖子进程的存储区域，这样将产生写保护错误(do_wp_page)(这个时候子进程写的实际上是父进程的存储区域)，这个错误导致内核为子进程重新分配存储空间。当子进程正确开始执行后，将唤醒父进程，使得父进程继续往后执行。

2. Linux 进程的调度

Linux 操作系统有以下三种进程调度策略。

(1) 分时调度策略(SCHED_OTHER)。SCHED_OTHER 是面向普通进程的时间片轮转策略。采用该策略时,系统为处于 TASK_RUNNING 状态的每个进程分配一个时间片。当时间片用完时,进程调度程序再选择下一个优先级相对较高的进程,并授予 CPU 使用权。

(2) 先到先服务的实时调度策略(SCHED_FIFO)。SCHED_FIFO 策略适用于对响应时间要求比较高,运行所需时间比较短的实时进程。采用该策略时,各实时进程按其进入可运行队列的顺序依次获得 CPU。除了因等待某个事件主动放弃 CPU,或者出现优先级更高的进程而剥夺其 CPU 之外,该进程将一直占用 CPU 运行。

(3) 时间片轮转的实时调度策略(SCHED_RR)。SCHED_RR 策略适用于对响应时间要求比较高,运行所需时间比较长的实时进程。采用该策略时,各实时进程按时间片轮流使用 CPU。当一个运行进程的时间片用完后,进程调度程序停止其运行并将其置于可运行队列的末尾。

实时进程将得到优先调用,实时进程根据实时优先级决定调度权值。分时进程则通过 nice 和 counter 值决定权值,nice 越小,counter 越大,被调度的概率越大,也就是曾经使用了 CPU 最少的进程将会得到优先调度。

在 SCHED_OTHER 调度策略中,调度器总是选择那个 priority+counter 值最大的进程来调度执行。从逻辑上分析,SCHED_OTHER 调度策略存在着调度周期,在每一个调度周期中,一个进程的 priority 和 counter 值的大小影响了当前时刻应该调度哪一个进程来执行,其中 priority 是一个固定不变的值,在进程创建时就已经确定,它代表了该进程的优先级,也代表着该进程在每一个调度周期中能够得到的时间片的多少; counter 是一个动态变化的值,它反映了一个进程在当前的调度周期中还剩下的时间片。在每一个调度周期的开始,priority 的值被赋给 counter,然后每次该进程被调度执行时,counter 值都减少。当 counter 值为零时,该进程用完自己在本调度周期中的时间片,不再参与本调度周期的进程调度。当所有进程的时间片都用完时,一个调度周期结束,然后周而复始。

当采用 SHCED_RR 策略的进程的时间片用完,系统将重新分配时间片,并置于就绪队列尾。放在队列尾保证了所有具有相同优先级的 RR 任务的调度公平。

SCHED_FIFO 一旦占用 CPU 则一直运行。直到有更高优先级任务到达或自己放弃。如果有相同优先级的实时进程(根据优先级计算的调度权值是一样的)已经准备好,FIFO 时必须等待该进程主动放弃后才可以运行这个优先级相同的任务。而 RR 可以让每个任务都执行一段时间。

3.3.4 μCLinux 的进程管理

μCLinux 的进程调度沿用了 Linux 的传统,系统每隔一定时间挂起进程,同时系统产生快速和周期性的时钟计时中断,并通过调度函数(定时器处理函数)决定进程什么时候拥有它的时间片。然后进行相关进程切换,这是通过父进程调用 fork 函数生成子进程来实现的。在 μCLinux 下,由于 μCLinux 没有 MMU 管理存储器,在实现多个进程时需要实现数据保护。由于没有 MMU,系统虽然支持 fork 系统调用,但其实质上就是 vfork。μClinux 系统 fork 调用完成后,要么子进程代替父进程执行(此时父进程已经 sleep),直到子进程调用 exit 退出;要么调用 exec 执行一个新的进程,这个时候产生可执行文件的加载,即使这个进程只是父进

程的拷贝，这个过程也不可避免。当子进程执行 exit 或 exec 后，子进程使用 wakeup 把父进程唤醒，使父进程继续往下执行。

在 μCLinux 下，启动新的应用程序时系统必须为应用程序分配存储空间，并立即把应用程序加载到内存。缺少了 MMU 的内存重映射机制，μCLinux 必须在可执行文件加载阶段对可执行文件 Relocation 处理，使得程序执行时能够直接使用物理内存。

μCLinux 由于没有内存映射机制，因此其对内存的访问是直接的，所有程序中访问的地址都是实际的物理地址。操作系统对内存空间没有保护，各个进程实际上共享一个运行空间。这就需要实现多进程时进行数据保护，也导致了用户程序使用的空间可能占用到系统内核空间，这些问题在编程时都需要多加注意，否则容易导致系统崩溃。

3.4　文 件 系 统

3.4.1　文件系统定义

文件系统的定义如下：包含在磁盘驱动器或者磁盘分区的目录结构，整个磁盘空间可以给一个或者多个文件系统使用。在对某个文件系统做在某一个挂载点的挂载(Mount)操作后，就可以使用该文件系统了。

3.4.2　Linux 文件系统

Linux 支持许多种文件系统。Linux 初期的基本文件系统是 Minix，但其适用范围和功能都很有限。其文件名最长不能超过 14 个字符并且最大的文件不超过 64MB。因此于 1992 年开发了 Linux 专用的文件系统 ext(Extended File System)，解决了很多的问题。但 ext 的功能也并不是非常优秀，最终于 1993 年增加了 ext2(Extended File System 2)。Linux 还支持 ext3、JFS2、XFS 和 ReiserFS 等新的日志型文件系统。另外，Linux 支持加密文件系统(如 CFS)和虚拟文件系统(如/proc)。

下面对 ext2 和 ext3 文件系统做个简单介绍。

1. ext2 文件系统

ext2 是 Linux 事实上的标准文件系统，它已经取代了它的前任——ext 文件系统。ext 文件系统支持的文件大小最大为 2GB，支持的最大文件名称大小为 255 个字符，而且它不支持索引节点(包括数据修改时间标记)。ext2 文件系统做得更好，它的优点如下。

(1) 支持达 4TB 的内存。

(2) 文件名称最长可以到 1012 个字符。

(3) 当创建文件系统时，管理员可以选择逻辑块的大小(通常大小可选择 1024、2048 和 4096 字节)。

(4) 实现快速符号链接：不需要为此目的而分配数据块，并且将目标名称直接存储在索引节点(inode)表中。这使性能有所提高，特别是在速度上。

(5) 因为 ext2 文件系统的稳定性、可靠性和健壮性，所以几乎在所有基于 Linux 的系统(包括台式计算机、服务器和工作站，甚至一些嵌入式设备)上都使用 ext2 文件系统。然而，当在嵌入式设备中使用 ext2 时，它有一些缺点。

(6) ext2 是为像 IDE 设备那样的块设备设计的，这些设备的逻辑块大小是 512B、1KB 等

这样的倍数。这不太适合于扇区大小因设备不同而不同的闪存设备。

(7) ext2 文件系统没有提供对基于扇区的擦除/写操作的良好管理。在 ext2 文件系统中,为了在一个扇区中擦除单个字节,必须将整个扇区复制到 RAM,然后擦除,然后重写入。考虑到闪存设备具有有限的擦除寿命(大约能进行 100 000 次擦除),在此之后就不能使用它们,所以这不是一个特别好的方法。

(8) 在出现电源故障时,ext2 不是防崩溃的。

(9) ext2 文件系统不支持损耗平衡,因此缩短了扇区/闪存的寿命(损耗平衡确保将地址范围的不同区域轮流用于写/擦除操作以延长闪存设备的寿命)。

(10) ext2 没有特别完美的扇区管理,这使设计块驱动程序十分困难。

2. ext3 文件系统

在 ext2 文件系统尚未关闭之前就关机的话,很可能会造成文件系统的异常,在系统重新开机后,就能检测到内容的不一致性,此时,文件系统需要做检查工作,将不一致和错误的地方进行修复。这个检查和整理工作是比较耗时的,特别在文件系统容量比较大的时候,而且不能保证 100%的内容能完全修复。为了克服这个问题,日志式文件系统(Journal File System)便应运而生。这种文件系统的做法是:它会将整个磁盘的写入动作完整记录在磁盘的某个区域上,以便有需要时可以回溯追踪。

ext3 便是一种日志式文件系统,是对 ext2 系统的扩展,它兼容 ext2。在 ext3 文件系统中,由于详细记录了每个细节,故当在某个过程中被中断时,系统可以根据这些记录直接回溯并重整被中断的部分,而不必花时间去检查其他的部分,故重整的工作速度相当快,几乎不需要花时间。

ext3 相比较 ext2 文件系统而言,具有以下特点。

1) 具有高可用性

系统使用了 ext3 文件系统后,即使在非正常关机后,文件系统的恢复时间只要数十秒钟。

2) 具有很好的数据完整性

ext3 文件系统能够极大地提高文件系统的完整性,避免了意外关机对文件系统的破坏。

3) 访问速度快

尽管使用 ext3 文件系统时,有时在存储数据时可能要多次写数据,但是 ext3 的日志功能对磁盘的驱动器读写头进行了优化,因此从总体上看来,ext3 比 ext2 的性能还要好一些。

4) 兼容性好

由于 ext3 兼容 ext2,因此将 ext2 文件系统转换成 ext3 文件系统非常容易,只要简单地调用一个小工具即可完成整个转换过程,用户不用花时间备份、恢复、格式化分区等。另外,ext3 文件系统可以不经任何更改,而直接加载成为 ext2 文件系统。

5) 多种日志模式

ext3 有多种日志模式,一种工作模式是对所有的文件数据及 metadata 进行日志记录(data=journal 模式);另一种工作模式则是只对 metadata 记录日志,而不对数据进行日志记录,也即所谓 data=ordered 或者 data=writeback 模式。系统管理人员可以根据系统的实际工作要求,在系统的工作速度与文件数据的一致性之间做出选择。

3.4.3 嵌入式 Linux 文件系统

嵌入式文件系统就是在嵌入式系统中应用的文件系统。嵌入式文件系统是嵌入式系统的

一个重要组成部分，随着嵌入式系统硬件设备的广泛应用和价格的不断降低以及嵌入式系统应用范围的不断扩大，嵌入式文件系统的重要性显得更加突出。

由于系统体系结构的不同，嵌入式文件系统在很多方面与桌面文件系统有较大区别。例如在普通桌面操作系统中，文件系统不仅要管理文件，提供文件系统API，还要管理各种设备，支持对设备和文件操作的一致性（像操作文件一样操作各种I/O设备）。

嵌入式设备的自身特点决定了它很少使用大容量的IDE硬盘等常见的PC上的主要存储设备。嵌入式设备往往选用ROM、Flash Memory等作为它的主要存储设备。因此，学习嵌入式文件系统的目标就是找到最适合在这些存储设备上运行的文件系统。首先说到的是Rootfs（根文件系统），嵌入式系统一般从Flash启动，最简单的方法是将Rootfs装载到RAM的RamDisk，较复杂的是直接从Flash读取Cramfs，更复杂的是在Flash上分区，然后构建JFFS2等文件系统。首先来介绍Flash Memory。

1. Flash Memory简介

Flash Memory是近年来发展迅速的内存，属于Non-Volatile内存（Non-Volatile即断电数据也能保存），它具有EEPROM（Electrically EProM）电擦除的特点，还具有低功耗、密度高、体积小、可靠性高、可擦除、可重写、可重复编程等优点。Flash Memory由Toshiba（东芝）于1980年申请专利，并在1984年的国际半导体学术会议上首先发表，然后Intel和SEEQ大力发展芯片，到现在可以说，Flash Memory已经成为应用最广的移动微存储介质。实际上，不但在新型数码产品上广泛应用，连传统的EPROM、EEPROM等市场也开始被Flash Memory取代，像主板上的BIOS已经越来越多地采用Flash Memory为存储器。

2. Flash Memory上的两种技术NAND和NOR

Flash Memory主要有两种技术：NAND和NOR。NAND型的单元排列是串行的，而NOR型则是并行的。在NAND型Flash Memory中，存储单元被分成页，由页组成块。根据容量不同，块和页的大小有所不同，而组成块的页的数量也会不同，如8MB的模块，页大小为(512+16)B、块大小为(8K+256)B；而2MB模块，页大小为(256+8)B、块大小为(4K+128)B。NAND型存储单元的读写是以块和页为单位来进行的，像硬盘以及内存。实际上，NAND型的Flash Memory可以看作是顺序读取的设备，它仅用8比特的I/O端口就可以存取按页为单位的数据。正因为这样，它在读和擦文件、特别是连续的大文件时，与NOR型的Flash Memory相比速度相当的快。但NAND型的不足在于随机存取速度较慢，而且没有办法按字节写；这些方面就恰好是NOR型的优点所在：NOR型随机存取速度较快，而且可以随机按字节写。正因为这些特点，所以NAND型的Flash Memory适合用在大容量的多媒体应用中，而NOR型适合应用在数据/程序存储应用中。

NOR和NAND是现在市场上两种主要的非易失闪存技术。Intel于1988年首先开发出NOR Flash技术，彻底改变了原先由EPROM和EEPROM一统天下的局面。紧接着，1989年，东芝公司发表了NAND Flash结构，强调降低每比特的成本，更高的性能，并且像磁盘一样可以通过接口轻松升级。但是经过了十多年之后，仍然有相当多的硬件工程师分不清NOR和NAND闪存。

NOR的特点是芯片内执行（eXecute In Place，XIP），这样应用程序可以直接在Flash闪存内运行，不必再把代码读到系统RAM中。NOR的传输效率很高，在1～4MB的小容量时具有很高的成本效益，但是很低的写入和擦除速度大大影响了它的性能。

NAND结构能提供极高的单元密度，可以达到高存储密度，并且写入和擦除的速度也很

快。应用 NAND 的困难在于 Flash 的管理和需要特殊的系统接口。

在 NOR 器件上运行代码不需要任何的软件支持,在 NAND 器件上进行同样操作时,通常需要驱动程序,也就是内存技术驱动程序(MTD),NAND 和 NOR 器件在进行写入和擦除操作时都需要 MTD。

使用 NOR 器件时所需要的 MTD 要相对少一些,许多厂商都提供用于 NOR 器件的更高级软件,这其中包括 M-System 的 TrueFFS 驱动,该驱动被 Wind River System、Microsoft、QNX Software System、Symbian 和 Intel 等厂商所采用。

驱动还用于对 DiskOnChip 产品进行仿真和 NAND 闪存的管理,包括纠错、坏块处理和损耗平衡。

3. 嵌入式文件系统分类

不同的文件系统类型有不同的特点,因而根据存储设备的硬件特性、系统需求等有不同的应用场合。在嵌入式 Linux 应用中,主要的存储设备为 RAM(DRAM,SDRAM)和 ROM(常采用 Flash 存储器),常用的基于存储设备的文件系统类型包括:JFFS2、YAFFS、Cramfs、Romfs、RamDisk、Ramfs/Tmpfs 等。

1) 基于 Flash 的文件系统

Flash 作为嵌入式系统的主要存储媒介,有其自身的特性。Flash 的写入操作只能把对应位置的 1 修改为 0,而不能把 0 修改为 1(擦除 Flash 就是把对应存储块的内容恢复为 1),因此,一般情况下,向 Flash 写入内容时,需要先擦除对应的存储区间,这种擦除是以块(block)为单位进行的。

Flash 存储器的擦写次数是有限的,NAND 闪存还有特殊的硬件接口和读写时序。因此,必须针对 Flash 的硬件特性设计符合应用要求的文件系统;传统的文件系统如 ext2 等,用作 Flash 的文件系统会有诸多弊端。

在嵌入式 Linux 下,MTD 为底层硬件(闪存)和上层(文件系统)之间提供一个统一的抽象接口,即 Flash 的文件系统都是基于 MTD 驱动层的。使用 MTD 驱动程序的主要优点在于,它是专门针对各种非易失性存储器(以闪存为主)而设计的。

(1) JFFS2 文件系统

随着技术的发展,近年来日志文件系统在嵌入式系统上得到了较多的应用,其中以支持 NOR Flash 的 JFFS、JFFS2 文件系统和支持 NAND Flash 的 YAFFS 最为流行。这些文件系统都支持掉电文件保护,同时支持标准的 MTD 驱动。

JFFS 文件系统是瑞典 Axis 通信公司开发的一种基于 Flash 的日志文件系统,它在设计时充分考虑了 Flash 的读写特性和用电池供电的嵌入式系统的特点,在这类系统中必须确保在读取文件时,如果系统突然掉电,其文件的可靠性不受到影响。对 Red Hat 的 Davie Woodhouse 进行改进后,形成了 JFFS2。主要改善了存取策略以提高 Flash 的抗疲劳性,同时也优化了碎片整理性能,增加了数据压缩功能。需要注意的是,当文件系统已满或接近满时,JFFS2 会大大放慢运行速度。这是因为垃圾收集的问题。

JFFS2 的底层驱动主要完成文件系统对 Flash 芯片的访问控制,如读、写、擦除操作。在 Linux 中这部分功能是通过调用 MTD 驱动实现的。相对于常规块设备驱动程序,使用 MTD 驱动程序的主要优点在于 MTD 驱动程序是专门为基于闪存的设备所设计的,所以它们通常有更好的支持、更好的管理和更好的基于扇区的擦除和读写操作的接口。MTD 相当于在硬件和上层之间提供了一个抽象的接口,可以把它理解为 Flash 的设备驱动程序,它主要向上提

供两个接口：MTD字符设备和MTD块设备。通过这两个接口，就可以像读写普通文件一样对Flash设备进行读写操作。经过简单的配置后，MTD在系统启动以后可以自动识别支持CFI或JEDEC接口的Flash芯片，并自动采用适当的命令参数对Flash进行读写或擦除。

(2) YAFFS2文件系统

YAFFS(Yet Another Flash File System)是一种和JFFS类似的闪存文件系统。主要针对NAND Flash设计，和JFFS相比它减少了一些功能，所以速度更快，而且对内存的占用比较小。此外YAFFS自带NAND芯片驱动，并且为嵌入式系统提供了直接访问文件系统的API，用户可以不使用Linux中的MTD与VFS，直接对文件进行操作。在其他嵌入式系统中也可以直接使用这些API实现对文件的操作。

YAFFS2是YAFFS的改进版本，在速度、内存的使用上，对NAND设备的支持上都有所改善。YAFFS2还支持大页面的NAND设备，并且对大页面的NAND设备做了优化。

(3) Cramfs

Cramfs是Linux的创始人Linus Torvalds参与开发的一种只读的压缩文件系统。它也基于MTD驱动程序。

在Cramfs文件系统中，每一页(4KB)被单独压缩，可以随机页访问，其压缩比高达2∶1，为嵌入式系统节省大量的Flash存储空间，使系统可通过更低容量的Flash存储相同的文件，从而降低系统成本。

Cramfs文件系统以压缩方式存储，在运行时解压缩，所以不支持应用程序以XIP方式运行，所有的应用程序要求被复制到RAM里去运行，但这并不代表比Ramfs需求的RAM空间要大一点儿，因为Cramfs是采用分页压缩的方式存放档案，在读取档案时，不会一下子就耗用过多的内存空间，只针对目前实际读取的部分分配内存，尚未读取的部分不分配内存空间，当我们读取的档案不在内存时，Cramfs文件系统自动计算压缩后的资料所存的位置，再即时解压缩到RAM中。另外，它的速度快，效率高，其只读的特点有利于保护文件系统免受破坏，提高了系统的可靠性。

由于以上特性，Cramfs在嵌入式系统中应用广泛。但是它的只读属性同时又是它的一大缺陷，使得用户无法对其内容进行扩充。

Cramfs映像通常放在Flash中，但是也能放在别的文件系统里，使用loopback设备可以把它安装在别的文件系统里。

(4) Romfs

传统型的Romfs文件系统是一种简单的、紧凑的、只读的文件系统，不支持动态擦写保存，按顺序存放数据，因而支持应用程序以XIP方式运行，在系统运行时，节省RAM空间。μCLinux系统通常采用Romfs文件系统。

(5) 其他文件系统

FAT/FAT32也可用于实际嵌入式系统的扩展存储器(例如PDA、Smartphone、数码相机等的SD卡)，这主要是为了更好地与最流行的Windows桌面操作系统相兼容。ext2也可以作为嵌入式Linux的文件系统，不过将它用于Flash闪存会有诸多弊端。

2) 基于RAM的文件系统

(1) RamDisk

RamDisk将部分固定大小的内存当作分区来使用。它将实际的文件系统装入内存，所以并非一个实际的文件系统，可以作为根文件系统。将一些经常被访问而又不会更改的文件通

过 RamDisk 放在内存中,可以明显地提高系统的性能。在 Linux 的启动阶段,initrd 提供了一套机制,可以将内核映像和根文件系统一起载入内存。

(2) Ramfs/Tmpfs

Ramfs 是 Linus Torvalds 开发的一种基于内存的文件系统,工作于虚拟文件系统(VFS)层,不能格式化,可以创建多个,在创建时可以指定其最大能使用的内存大小(实际上 VFS 本质上可看成一种内存文件系统,它统一了文件在内核中的表示方式,并对磁盘文件系统进行缓冲)。由于文件系统把所有的文件都放在 RAM 中,所以读/写操作发生在 RAM 中,可以用 Ramfs/Tmpfs 来存储一些临时性或经常要修改的数据,例如/tmp 和/var 目录,这样既避免了对 Flash 存储器的读写损耗,也提高了数据读写速度。相对于传统的 RamDisk 的不同之处主要在于:不能格式化,文件系统大小可随所含文件内容大小变化。Tmpfs 的一个缺点是当系统重新引导时会丢失所有数据。

3) 网络文件系统

NFS 是 Network File System 的简写,即网络文件系统,由 Sun Microsystems 公司开发,是一种网络操作系统,并且是 UNIX 操作系统的协议。

网络文件系统是 FreeBSD 支持的文件系统中的一种,也被称为 NFS。NFS 允许一个系统在网络上与他人共享目录和文件。通过使用 NFS,用户和程序可以像访问本地文件一样访问远端系统上的文件。

以下是 NFS 最显而易见的好处。

① 本地工作站使用更少的磁盘空间,因为通常的数据可以存放在一台机器上而且可以通过网络访问到。

② 用户不必在每个网络上的机器里都有一个 Home 目录。Home 目录可以被放在 NFS 服务器上并且在网络上处处可用。

③ 诸如软驱,CDROM 之类的存储设备可以在网络上被别的机器使用。这可以减少整个网络上的可移动介质设备的数量。

NFS 至少有两个主要部分:一台服务器和一台(或者更多)客户机。客户机远程访问存放在服务器上的数据。为了正常工作,一些进程需要被配置并运行。

NFS 有很多实际应用。下面是比较常见的一些。

① 多个机器共享一台 CDROM 或者其他设备。这对于在多台机器中安装软件来说更加便宜与方便。

② 在大型网络中,配置一台中心 NFS 服务器用来放置所有用户的 Home 目录可能会带来便利。这些目录能被输出到网络以便用户不管在哪台工作站上登录,总能得到相同的 Home 目录。

③ 几台机器可以有通用的/usr/ports/distfiles 目录。这样的话,当需要在几台机器上安装 port 时,可以无须在每台设备上下载而快速访问源码。

服务器必须运行以下服务。

① nfsd,为来自 NFS 客户端的请求服务。

② mountd,FS 挂载服务,处理 NFSD 递交过来的请求

③ rpvbind,此服务允许 NFS 客户程序查询正在被 NFS 服务使用的端口。

小　　结

本章首先介绍了几种常见的嵌入式 Linux 操作系统，接着针对是否支持 MMU，介绍了嵌入式 Linux 的不同处理方法，然后针对不同的进程调度模式，分别介绍了实时操作系统和分时操作系统的进程管理，最后在标准 Linux 文件系统基础上，介绍了嵌入式 Linux 文件系统的种类和特点。

进一步探索

在本章内容的基础上，进一步了解各种嵌入式 Linux 操作系统的特点，以及它们的应用领域以及典型设备。