中图分类号: TN949.197 文献标识码: B 文章编号: 1007-7022(2006)03-0285-04

STLite/OS20嵌入式实时操作系统原理及其应用

□王瑞胡(重庆文理学院, 重庆 402160)

摘 要: STLite/OS20是用于数字电视机顶盒中的嵌入式操作系统, 具有实时、多任务等特点。介绍了该操作系统的内核、存储器管理和分区、分配策略、多任务调度及任务间通过信号量和消息处理进行通信的机制、中断管理等。

关键词: STLite/OS20; 实时系统; 内核; 存储器管理; 多任务

The Principle and Application of Embedded Real-time Operation System of STLite/OS20

□WANG Rui-hu
(Chongqing University of Art and Science, Chongqing 402160, China)

Abstract: As an embedded operation system in set top box of digital TV, STLite/OS20 has some characters as real-time, multi-task, etc. Introduce the kernel, memory management and partition, allocation strategy, multi-task scheduling and how to communicate by semaphore and message handling as well as interrupt management.

Key words: STLite/OS20; real-time system; kernel; memory management; multi-task

数字电视机顶盒具有复杂的软件系统, 在一定意义上讲, 就是一个复杂的计算机系统, 因此操作系统是必不可少的。相对通常意义上的计算机操作系统, 数字电视机顶盒操作系统应具有更高的稳定性和实时性。机顶盒采用的操作系统都是实时嵌入式操作系统, 目前常用的机顶盒操作系统主要有 STLite/OS20、VxWorks、PSOS、Hopen等。 STLite/OS20是一种高效率的实时多任务操作系统, 适用于所有 ST20系列处理器, 提供任务管理、内存管理、消息队列服务、信号量服务、时钟和定时器管理、中断管理等功能。

1 内核

为了实现多优先级调度, STLite/OS20采用了一个很小的调度内核, 调度内核是基于系统中任务的优先级来作出调度决策的一小段代码, 调度内核的任务是确保当前运行的任务总是系统中具有最高优先级的任务。

系统中运行的任务状态有运行态、就绪态、挂起态和休眠态。

内核维护两个重要的信息: 当前正在运行的任务及其优先级别、等待投入运行的任务队列。当一个任务被调度的时候, 调度程序判断新任务的优先级别是否比当前正在运行的任务的优先级别高, 如果新任务的优先级别高于当前运行任务的优先级别, 则保存后者的运行状态, 并启动新的任务投入运行, 这种方式称为"抢占式调度"。任务在运行时因申请资源如等待

作者简介: 王瑞胡(1975-), 男, 讲师, 硕士, 主要研究方向为嵌入式系统及网络数据库。

消息队列中的某个消息而挂起时，内核从当前已就绪的所有任务中选择一个具有最高优先级别的任务投入运行。当就绪队列中存在与当前正在运行任务的优先级别相同的任务时，内核采用时间片轮转的调度算法来完成任务间的调度。

在STLite/OS20内核上唯一能够实施的操作就是安装(installation)和启动(start)，这通过调用 kernel_initialize()和 kernel_start()来完成。

2 存储器和分区

嵌入式系统中的存储器资源非常有限，为了更有效地利用这些资源，存储器的管理就显得非常重要。

通常操作系统的内核，由于可供使用的系统资源相对比较充足，实时性能只需满足用户能忍耐的限度，系统考虑的是提供更好的性能和安全机制，所以操作系统通常都引入虚拟存储管理。在虚拟存储中经常要对页进行换入换出操作，因而内存中页命中率和换入换出所耗费的时间严重破坏了整个系统的确定性。这种存储机制不能提供实时系统所要求的时间确定性，所以在实时内核中不采用虚拟内存管理。

操作系统对于频繁的动态分配内存的请求，可能会把一整块的内存空间分割成许多相互间隔的内存碎片，通常操作系统解决内存碎片的方法是进行定期的内存清理，通过移动内存块来合并内存碎片，这不但增加了系统开销，使内存管理模块进一步复杂，而且也影响了系统的响应时间。

在大多数嵌入式实时系统中，不采用虚拟存储机制来实现对内存空间的直接管理，并且用分区与块的结合来避免出现内存碎片。

2.1 分配策略

内核把内存分成多个空间大小不等的分区，每个分区又分为许多大小相同的块，各分区的块的大小可以不同。对于应用程序的动态申请内存的要求，内存管理模块将比较每个分区中块的大小，从大于且最接近用户申请空间块大小的分区中选出一个未使用的块分配给用户，使用完后，用户释放内存时，仍把该内存块放入申请前的原分区中。

STLite/OS20支持3种不同的内存分区：堆分区(heap)、固定分区(fixed)、简单分区(simple)。

堆分区的分配策略类似于传统C中的 malloc和free函数，通过调用 memory_allocate()和 memory_deallocate()来完成。如果需要申请可变大小的空间时，只需指定所要求分配的存储空间的大小。当申请的内存块被释放的时候，如果在它的前面或后面存在有空闲块，它将与这些空闲块组合在一起形成一个更大的可供分配的存储空间。

虽然堆的分配方式比较灵活，但也存在一些缺点：

(1)不确定性，申请和释放存储空间的时间是可变的，它取决于前一次分配和释放操作的执行，同时还得对存储空间进行搜索。

(2)分配器自己所占用的高端存储空间较大，每一次申请都需要额外的空间来描述。

固定分区则解决了这些问题，在创建分区的时候，通过调用 partition_create_fixed()或 partition_init_fixed()，将分区中可供分配的块设为固定的大小，这样在申请或释放某一个内存块的时候所用的时间是确定的，并且高端所占据的存储空间非常小。

最后一种简单分区则显得不是那么重要，它只是将指针加1指向下一个可供分配的存储块，这就意味着不可能将释放后的存储块放回到该分区，但是在申请空间的时候不会有浪费的存储器。

2.2 预定义分区(Pre-defined Partitions)

STLite/OS20自身不需要分配任何动态存储空间，然而为了方便起见，在对象初始化函数 task_init()、semaphore_init_fifo()等被用到的时候，STLite/OS20需要为这些对象分配存储空间，这种情况下，STLite/OS20需要用到两种类型的预定义分区：

(1) system_partition 用于所有对象(object)存储空间的分配，包括信号量、消息队列、任务数据结构的静态部分，如任务堆栈区。

(2) internal_partition 用于任务数据结构动态部分的存储空间的分配。为了减少场景切换(context switch)时间，数据应该放在内部存储器当中，这部分作为简单分区来管理。

以上两种分区必须在对象创建函数调用之前定义。

3 任务

STLite/OS20内核的基本功能是提供一个多任务环境。多任务使得许多程序在表面上表现为并发执行，而事实上内核是根据基本的调度算法使它们分段执行。一个任务描述了应用程序的一个离散的、独立的、部分的行为。任务除了能与其他任务通信外，其他行为与独立的程序差不多，新的任务可以被已存在的任务动态创建。

一个任务由数据结构、堆栈和一段代码组成，其任务的数据结构被称作它的状态，具体的内容和结构与处理器无关，在STLite/OS20中它被分成两个部分：

动态状态(dynamic state)，在数据结构 tdesc_t中

定义, CPU利用它来执行任务。该结构中的域的改变取决于处理器的类型,该结构中最重要的元素是 machine registers,尤其是指令指针(Iptr)和工作区指针(Wptr)寄存器,在任务运行的时候, Iprt和 Wptr由 CPU保持,在任务不处于运行状态的时候,它们就存储在 tdesc_t结构中。

静态状态(static state),在数据结构 task_t中定义, STLite /OS20利用它来描述任务。 task_t包含任务的状态如创建、运行、终止和用于堆栈检测的堆栈范围(stack range)。一个任务通过 task_t数据结构来识别,指向该结构的指针称作该任务的 ID。

任务的数据结构既可以由 STLite /OS20来分配,也可以由用户通过声明 tdesc_t和 task_t数据结构来分配,任务的执行代码由用户函数提供,要创建一个任务, tdesc_t和 task_t必须被分配和初始化,并且要将堆栈和任务函数与它们关联起来,这通过调用 task_create( )或 task_init( )函数来实现。

3.1　任务优先级和时间片的实现

STLite /OS20实现 16级的任务优先级别,任务是作为目标硬件的最低优先级硬件进程而运行的,并且它具有用户指定的 STLite /OS20优先级。 STLite /OS20的任务建立在硬件实现的进程上,利用硬件的特性来保证高效率的实现。在 ST20-C2处理器上,硬件支持高优先级和低优先级两种进程,高优先级进程优先于低优先级进程,比如 STLite /OS20的任务。如图 1所示,在 ST20-C2上,对于临界区代码就有可能创建直接使用硬件高优先级进程的任务。

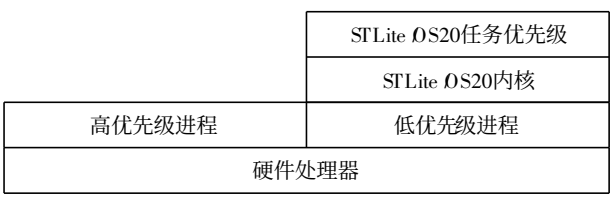

图 1　ST20-C2优先级

一个任务的初始优先级是调用任务创建函数时定义的,唯一不是用这种方法定义任务优先级的任务是根任务(root task),即调用 kernel_start( )来启动 STLite /OS20运行的任务,这个任务是以最大用户优先级(MAX_USER_PRIORITY)来运行的。

ST20-C2微处理器有两个时钟寄存器,一个是高优先级时钟寄存器,一个是低优先级时钟寄存器。在高优先级时钟超过一个指定数量的时钟周期后,一个时间片周期任务就结束了。当两个时间片周期结束而同一个任务(低优先级的硬件进程)还在继续运行时,处理器就要试图剥夺该任务的控制权。高优先级的进程不采用时间分片方式,它们一直要运行到完成或等待通信或等待资源。

一个任务一般运行一到两个时间片,为使延时最小并避免一个任务独占处理器时间,编译器会在适当的位置插入指令,使得可以进行时间分片。如果一个任务被一个更高优先级的任务剥夺了控制权,当低优先级任务恢复运行时,它的时间片周期将从时间片周期的开始处计时,但是如果一个任务被中断或被一个高优先级进程所剥夺,那么它将从中断或高优先级进程释放时开始计时,这样该任务可能会失去一些时间。

3.2　任务调度

一个活动(active)的任务要么处于运行状态,要么处于就绪状态。 STLite /OS20确保做到以下几点:

(1)当前正在运行的任务具有最高的优先级别

如果一个具有更高优先级的任务变为就绪状态准备占用处理器时, STLite /OS20调度程序将保存当前任务的状态并使具有更高优先级的任务成为当前任务,如果当前任务没有被其他具有更高优先级的任务剥夺的话,该任务将一直运行直到结束。

(2)相同优先级的任务采用时间分片的方式来进行调度

这样所有的任务都能获得占用处理器的机会。为使内核调度程序不对当前的任务采用时间分片或剥夺方式进行调度,可以通过调用 task_tick( )和 task_untick( )函数来实现。

4　信号量

信号量(semaphore)在多任务的实时内核中其主要作用是用作共享数据结构或共享资源的互斥机制,标志某个事件的发生以及同步两个任务。

信号量提供一种任务之间同步的简单而有效的方式,信号量可用于保证互斥、控制对共享资源的访问和任务间的同步。

描述信号量的数据结构 semaphore_t包含两项数据:

(1)信号量可以被使用的次数计数;

(2)等待使用信号量的任务队列。

信号量可用以下的几个函数来创建:

```
semaphore_t * semaphore_create_fifo( int value);
void semaphore_init_fifo( semaphore_t * sem, int value);
semaphore_t * semaphore_create_priority( int value);
void semaphore_init_priority( semaphore_t * sem, int value)
```

如果在等待一个信号量时要求有时间控制功能，可以使用上面函数的超时版本，如：

semaphore_t * semaphore_create_fifo_timeout(int value)

创建函数 Create将自动为信号量分配存储空间，而 init函数可以让用户利用 semaphore_t结构来定义一个指向信号量的指针。

信号量有两种：二进制信号量（binary）和计数信号量（counting）。

二进制信号量的取值只有 0和 1。使用二进制信号量时必须先初始化为 1。任务使用信号量（semaphore_wait或 P操作）时，如果信号量的值为 1，则对信号量减 1，并进行任务的操作；如果信号量的值为 0，则任务进入该信号量的等待队列，信号量的值保持不变。任务释放信号量（semaphore_signal或 V操作）时，如果等待队列不为空，则从等待队列中选出一个任务进入就绪状态，信号量的值不加 1，选择任务的算法可以基于优先级或 FIFO，如果等待队列为空，则信号量加 1。

计数型信号量可以取非负整数值。信号量的值可以表示实际的含义（如一个资源最多允许同时使用的任务数）。当一个任务需要一个设备时就进行 semaphore_wait操作，如果一个设备可用，对 semaphore_wait()的操作将立即返回，并将信号量的计数减 1；如果没有设备可用，那么该任务就被添加到等待该信号量的任务队列当中去。当一个任务使用完设备时，它就调用 semaphore_signal()来释放信号量。

信号量的使用应该有所节制，不能让所有的互斥处理都使用信号量机制来实现，因为信号量机制是有一定系统开销的。对于简单的数据共享，如果处理时间很短，使用开/关中断就可以了，不需要使用信号量，只有涉及系统消耗比较大的共享数据操作，才考虑使用信号量，因为，如果此时使用开/关中断，就可能会影响系统的中断响应时间。

5　消息处理

消息队列提供了一种任务间缓冲通信的方法，同时也提供了一种不用数据拷贝的通信方法。消息队列受到以下的限制：在中断处理程序中，只能使用带有超时设定的消息队列函数，并且超时值应该设置为 TIMEOUT_IMMEDIATE，这是为了防止中断处理函数在请求消息的时候被阻塞。

STLite/OS20消息队列包含两个队列：一个为当前没有被使用到的消息缓冲区（也称作“空闲”队列），另一个保存已经被发送但还没有被接收的消息的缓冲区（“发送”队列），用户调用不同消息函数的结果使消息缓冲区在这两个队列中移动，如图 2所示。

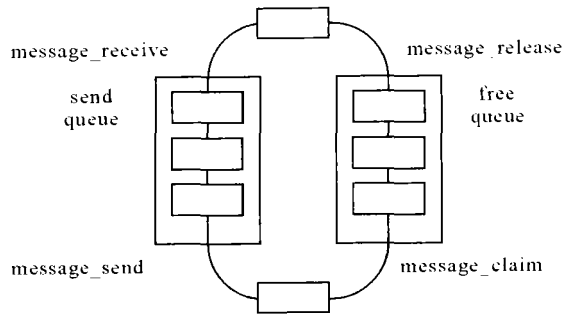

图 2　消息队列

可以用 message_create_queue()或 message_init_queue()来创建一个消息队列，初始化时，所有的消息缓冲区都在空闲队列中，用户调用 message_claim()或 message_claim_timeout()来申请消息缓冲区，这些函数返回后，用户就可以填充消息结构中相应的数据成员。

6　中断

实时内核的中断管理与一般操作系统内核的中断管理大体相同，中断管理负责中断的初始化安装、现场的保存和恢复、中断栈的嵌套管理等。

7　结语

STLite/OS20通常与嵌入式开发平台 ST20 Embedded Toolset结合起来使用。ST20 Embedded Toolset支持带有诊断控制器 DCU（Diagnostic Controller Unit）和 ST20－C2处理器核 ST20硅片设备的编程和调试。用于开发 C语言应用程序的 ST20 Embedded Toolset体系结构包括以下几个部分：st20cc，用于编译和连接源程序；st20run、st20sim，在开发平台下用于测试的调试工具；st20libr、st20list支持工具，如 librarian和 lister工具。

本文介绍的只是一款机顶盒系统专用的嵌入式实时操作系统，但其基本原理也适用于其他的通用型嵌入式系统。

参考文献：

[1]　杨　勇，饶　群. 数字电视机顶盒的嵌入式操作系统内核分析[J]. 电视技术，2002，(2)：50－53.

[2]　JEAN J LABROSSE. 嵌入式系统构件[M]. 袁勤勇，译. 北京：机械工业出版社，2002

[3]　杨　勇，周海山，刘少情. 嵌入式操作系统在HDTV机顶盒中的应用[J]. 电视技术，2002，(2)：18－19.

[收稿日期：2005-11-07]