嵌入式操作系统FreeRTOS的原理与实现

[日期：2006-6-6] 来源：单片机及嵌入式系统应用 作者：中国海洋大学 刘滨 王琦 刘丽

在嵌入式领域中，嵌入式实时操作系统正得到越来越广泛的应用。采用嵌入式实时操作系统(RTOS)可以更合理、更有效地利用CPU的资源，简化应用软件的设计，缩短系统开发时间，更好地保证系统的实时性和可靠性。由于RTOS需占用一定的系统资源(尤其是RAM资源)，只有μC／OS-II、embOS、salvo、FreeRTOS等少数实时操作系统能在小RAM单片机上运行。相对于C／OS-II、embOS等商业操作系统，FreeRTOS操作系统是完全免费的操作系统，具有源码公开、可移植、可裁减、调度策略灵活的特点，可以方便地移植到各种单片机上运行，其最新版本为2.6版。

1 FreeRTOS操作系统功能

作为一个轻量级的操作系统，FreeRTOS提供的功能包括：任务管理、时间管理、信号量、消息队列、内存管理、记录功能等，可基本满足较小系统的需要。FreeRTOS内核支持优先级调度算法，每个任务可根据重要程度的不同被赋予一定的优先级，CPU总是让处于就绪态的、优先级最高的任务先运行。FreeRTOS内核同时支持轮换调度算法，系统允许不同的任务使用相同的优先级，在没有更高优先级任务就绪的情况下，同一优先级的任务共享CPU的使用时间。

FreeRTOS的内核可根据用户需要设置为可剥夺型内核或不可剥夺型内核。当FreeRTOS被设置为可剥夺型内核时，处于就绪态的高优先级任务能剥夺低优先级任务的CPU使用权，这样可保证系统满足实时性的要求；当FreeRTOS被设置为不可剥夺型内核时，处于就绪态的高优先级任务只有等当前运行任务主动释放CPU的使用权后才能获得运行，这样可提高CPU的运行效率。

2 FreeRTOS操作系统的原理与实现

2．1任务调度机制的实现

任务调度机制是嵌入式实时操作系统的一个重要概念，也是其核心技术。对于可剥夺型内核，优先级高的任务一旦就绪就能剥夺优先级较低任务的CPU使用权，提高了系统的实时响应能力。不同于μC／OS-II，FreeRTOS对系统任务的数量没有限制，既支持优先级调度算法也支持轮换调度算法，因此FreeRTOS采用双向链表而不是采用查任务就绪表的方法来进行任务调度。系统定义的链表和链表节点数据结构如下所示：

```
typedef struct xLIST{  //定义链表结构

unsigned portSHORPT usNumberOfItems;

//usNumberOfItems为链表的长度，为0表示链表为空

volatile xListItem * pxHead;//pxHead为链表的头指针

volatile xListItem * pxIndex; //pxIndex指向链表当前结点的指针

volatile xListItem xListEnd;  //xListEnd为链表尾结点

}xList;

struct xLIST_ITEM {  //定义链表结点的结构

port Tick type xItem Value;

//xItem Value的值用于实现时间管理

//port Tick Type为时针节拍数据类型，

//可根据需要选择为16位或32位

volatile struct xLIST_ITEM * pxNext;

//指向链表的前一个结点
```

```
void * pvOwner;//指向此链表结点所在的任务控制块

void * pvContainer;//指向此链表结点所在的链表};
```

FreeRTOS中每个任务对应于一个任务控制块(TCB)，其定义如下所示：

```
typedef struct tskTaskControlBlock {

portSTACK_TYPE  * pxTopOfStack;

//指向任务堆栈结束处

portSTACK_TYPE  * pxStack;

//指向任务堆栈起始处

unsigned portSHORT  usStackDepth;  //定义堆栈深度

signed portCHAR  pcTaskName[tskMAX_TASK_NAME_LEN];//任务名称

unsigned portCHAR  ucPriority;  //任务优先级

xListItem xGenericListItem;

//用于把TCB插入就绪链表或等待链表

xListItem xEventListItem;

//用于把TCB插入事件链表（如消息队列）

unsigned portCHAR  ucTCBNumber;  //用于记录功能
```

}tskTCB;

FreeRTOS定义就绪任务链表数组为xList pxReady—TasksLists[portMAX_PRIORITIES]。其中portMAX_PRIORITIES为系统定义的最大优先级。若想使优先级为n的任务进入就绪态，需要把此任务对应的TCB中的结点xGenericListltem插入到链表pxReadyTasksLiStS[n]中，还要把xGenericListItem中的pvContainer指向pxReadyTasksLists[n]方可实现。

当进行任务调度时，调度算法首先实现优先级调度。系统按照优先级从高到低的顺序从就绪任务链表数组中寻找usNumberOfItems第一个不为0的优先级，此优先级即为当前最高就绪优先级，据此实现优先级调度。若此优先级下只有一个就绪任务，则此就绪任务进入运行态；若此优先级下有多个就绪任务，则需采用轮换调度算法实现多任务轮流执行。

若在优先级n下执行轮换调度算法，系统先通过执行(pxReadyTasksLists[n])→pxIndex=(pxReadyTasks-Lists[n])→pxlndex→pxNext语句得到当前结点所指向的下一个结点，再通过此结点的pvOwner指针得到对应的任务控制块，最后使此任务控制块对应的任务进入运行态。由此可见，在FreeRTOS中，相同优先级任务之间的切换时间为一个时钟节拍周期。

以图l为例，设系统的最大任务数为pottMAX_PRIORITIES，在某一时刻进行任务调度时，得到pxReadyTasksLists[i]．usNumberOfItems=O(i=2...portMAX_PRIORITIES)以及pxReadyTasksLists[1]。usNumberOfItems=3。由此内核可知当前最高就绪优先级为l，且此优先级下已有三个任务已进入就绪态．由于最高就绪优先级下有多个就绪任务，系统需执行轮换调度算法实现任务切换；通过指针pxlndex可知任务l为当前任务，而任务l的pxNext结点指向任务2，因此系统把pxIndex指向任务2并执行任务2来实现任务调度。当下一个时钟节拍到来时，若最高就绪优先级仍为1，由图l可见，系统会把pxIndex指向任务3并执行任务3。

为了加快任务调度的速度，FrecRTOS通过变量ucTopReadyPriotity跟踪当前就绪的最高优先级。当把一个任务加入就绪链表时，如果此任务的优先级高于ucTopReadyPriority，则把这个任务的优先级赋予ucTopReadyPriority。这样当进行优先级调度时，调度算法不是从portMAX_PRIORITIES而是从ucTopReady-Priority开始搜索。这就加快了搜索的速度，同时缩短了内核关断时间。

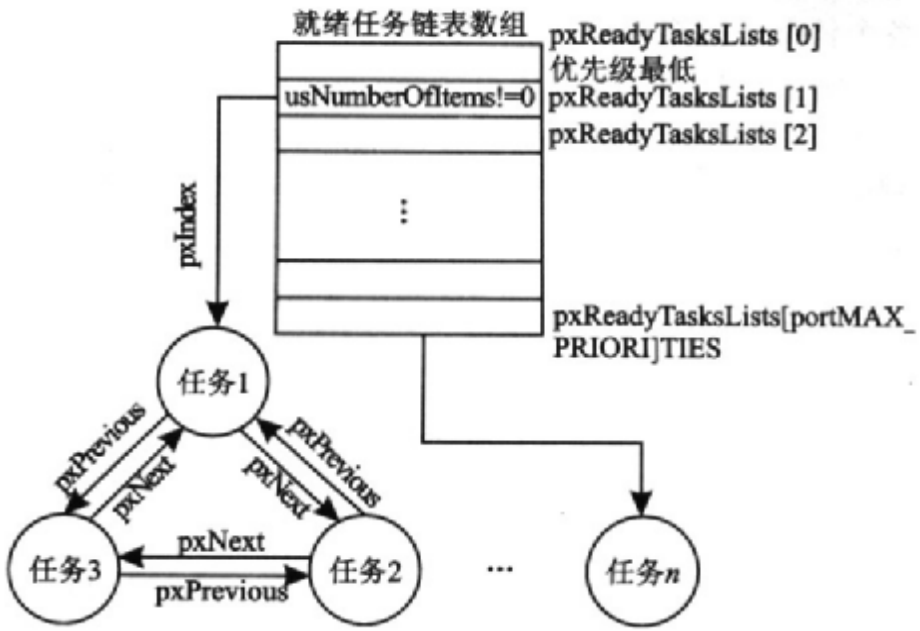

图 1 任务调度示意

2.2 任务管理的实现

实现多个任务的有效管理是操作系统的主要功能。FreeRTOS下可实现创建任务、删除任务、挂起任务、恢复任务、设定任务优先级、获得任务相关信息等功能。下面主要讨论FreeRTOS下任务创建和任务删除的实现。当调用sTaskCreate()函数创建一个新的任务时，FreeRTOS首先为新任务分配所需的内存。若内存分配成功，则初始化任务控制块的任务名称、堆栈深度和任务优先级，然后根据堆栈的增长方向初始化任务控制块的堆栈。接着，FreeRTOS把当前创建的任务加入到就绪任务链表。若当前此任务的优先级为最高，则把此优先级赋值给变量ucTopReadyPriorlty(其作用见2.1节)。若任务调度程序已经运行且当前创建的任务优先级为最高，则进行任务切换.

不同于μC / OS—II，FreeRTOS下任务删除分两步进行。当用户调用vTaskDelete()函数后，执行任务删除的第一步：FreeRTOS先把要删除的任务从就绪任务链表和事件等待链表中删除，然后把此任务添加到任务删除链表，若删除的任务是当前运行任务，系统就执行任务调度函数，至此完成任务删除的第一步。当系统空闲任务即prvldleTask()函数运行时，若发现任务删除链表中有等待删除的任务，则进行任务删除的第二步，即释放该任务占用的内存空间，并把该任务从任务删除链表中删除，这样才彻底删除了这个任务。值得注意的是，在FreeRTOS中，当系统被配置为不可剥夺内核时，空闲任务还有实现各个任务切换的功能。

通过比较μC / OS-II和FreeRTOS的具体代码发现，采用两步删除的策略有利于减少内核关断时间，减少任务删除函数的执行时

间，尤其是当删除多个任务的时候。

2.3 时间管理的实现

FreeRTOS提供的典型时间管理函数是vTaskDelay()，调用此函数可以实现将任务延时一段特定时间的功能。在FreeRTOS中，若一个任务要延时xTicksToDelay个时钟节拍，系统内核会把当前系统已运行的时钟节拍总数(定义为xTickCount，32位长度)加上xTicksToDelay得到任务下次唤醒时的时钟节拍数xTimeToWake。然后，内核把此任务的任务控制块从就绪链表中删除，把xTimeToWake作为结点值赋予任务的xItemValue，再根据xTimeToWake的值把任务控制块按照顺序插入不同的链表。若xTimeToWake>xTickCount，即计算中没有出现溢出，内核把任务控制块插入到pxDelayedTaskList链表；若xTimeToWake<xTickCount，即在计算过程中出现溢出，内核把任务控制块插入到pxOverflowDelayed-Taskust链表。

每发生一个时钟节拍，内核就会把当前的xTick-Count加1。若xTickCount的结果为0，即发生溢出，内核会把pxOverflowDelayedTaskList作为当前链表；否则，内核把pxDelaycdTaskList作为当前链表。内核依次比较xTickCotlrtt和链表各个结点的xTimcToWake。若xTick-Count等于或大于xTimeToWake，说明延时时间已到，应该把任务从等待链表中删除，加入就绪链表。

由此可见，不同于μC/OS—II，FreeRTOS采用“加”的方式实现时间管理。其优点是时间节拍函数的执行时间与任务数量基本无关，而μC/OS—II的OSTimcTick()的执行时间正比于应用程序中建立的任务数。因此当任务较多时，FreeRTOS采用的时间管理方式能有效加快时钟节拍中断程序的执行速度。

2.4 内存分配策略

每当任务、队列和信号量创建的时候，FreeRTOS要求分配一定的RAM。虽然采用malloc()和free()函数可以实现申请和释放内存的功能，但这两个函数存在以下缺点：并不是在所有的嵌入式系统中都可用，要占用不定的程序空间，可重人性欠缺以及执行时间具有不可确定性。为此，除了可采用malloc()和free()函数外，FreeRTOS还提供了另外两种内存分配的策略，用户可以根据实际需要选择不同的内存分配策略。

第1种方法是，按照需求内存的大小简单地把一大块内存分割为若干小块，每个小块的大小对应于所需求内存的大小。这样做的好处是比较简单，执行时间可严格确定，适用于任务和队列全部创建完毕后再进行内核调度的系统；这样做的缺点是，由于内存不能有效释放，系统运行时应用程序并不能实现删除任务或队列。

第2种方法是，采用链表分配内存，可实现动态的创建、删除任务或队列。系统根据空闲内存块的大小按从小到大的顺序组织空闲内存链表。当应用程序申请一块内存时，系统根据申请内存的大小按顺序搜索空闲内存链表，找到满足申请内存要求的最小空闲内存块。为了提高内存的使用效率，在空闲内存块比申请内存大的情况下，系统会把此空闲内存块一分为二。一块用于满足申请内存的要求，一块作为新的空闲内存块插入到链表中。

下面以图2为例介绍方法2的实现。假定用于动态分配的RAM共有8KB，系统首先初始化空闲内存块链表，把8KB RAM全部作为一个空闲内存块。当应用程序分别申请1KB和2KB内存后，空闲内存块的大小变为5KB3。2KB的内存使用完毕后，系统需要把2KB插入到现有的空闲内存块链表。由于2 KB<5KB，所以把这2 KB插入5KB的内存块之前。若应用程序又需要申请3 KB的内存，而在空闲内存块链表中能满足申请内存要求的最小空闲内存块为5KB，因此把5KB内存拆分为2部分，3KB部分用于满足申请内存的需要，2KB部分作为新的空闲内存块插入链表。随后1KB的内存使用完毕需要释放，系统会按顺序把1KB内存插入到空闲内存链表中。

方法2的优点是，能根据任务需要高效率地使用内存，尤其是当不同的任务需要不同大小的内存的时候。方法二的缺点是，不能

把应用程序释放的内存和原有的空闲内存混合为一体，因此，若应用程序频繁申请与释放“随机”大小的内存，就可能造成大量的内存碎片。这就要求应用程序申请与释放内存的大小为“有限个”固定的值(如图2中申请与释放内存的大小固定为1 KB、2 KB或3 KB)。方法2的另一个缺点是，程序执行时间具有一定的不确定性。

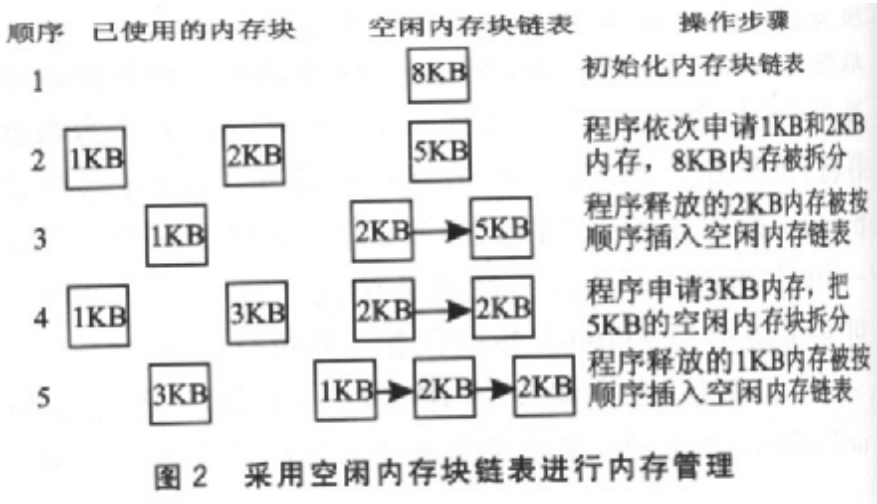

图2 采用空闲内存块链表进行内存管理

μC/OS—II提供的内存管理机制是把连续的大块内存按分区来管理，每个分区中包含整数个大小相同的内存块。由于每个分区的大小相同，即使频繁地申请和释放内存也不会产生内存碎片问题，但其缺点是内存的利用率相对不高。当申请和释放的内存大小均为一个固定值时(如均为2 KB)，FreeRTOS的方法2内存分配策略就可以实现类似μC/OS—II的内存管理效果。

2.5 FreeRTOS的移植

FreeRTOS操作系统可以被方便地移植到不同处理器上工作，现已提供了ARM、MSP430、AVR、PIC、C8051F等多款处理器的移植。FrceRTOS在不同处理器上的移植类似于μC/OS一II，故本文不再详述FreeRTOS的移植。此外，TCP/IP协议栈μIP已被移植到FreeRTOS上，具体代码可见FreeRTOS网站。

2.6 FreeRTOS的不足

相对于常见的μC/OS—II操作系统，FreeRTOS操作系统既有优点也存在不足。其不足之处，一方面体现在系统的服务功能上，如FreeRTOS只提供了消息队列和信号量的实现，无法以后进先出的顺序向消息队列发送消息；另一方面，FreeRTOS只是一个操作系统内核，需外扩第三方的GUI(图形用户界面)、TCP/IP协议栈、FS(文件系统)等才能实现一个较复杂的系统，不像μC/OS-II可以和μC/GUI、μC/FS、μC/TCP-IP等无缝结合。

3 结 论

作为一个源码公开的操作系统，学习FreeRTOS可以更好地掌握嵌入式实时操作系统的实现原理；作为一个免费的操作系统，采用FreeRTOS可在基本满足较小系统需要的情况下降低系统成本、简化开发难度。在实践中，采用FreeRTOS操作系统和MSP430单片机构成的温度控制系统稳定可靠，实现了较好的控制效果。相信随着时间的发展，FreeRTOS会不断完善其功能，以更好地满足人们对嵌入式操作系统实时性、可靠性、易用性的要求。

录入：录入员012

【>>>>>察看网友评论 ，或发表您对本文的看法】【 打印 】

上一篇：UC系列通用嵌入式软硬件平台
下一篇：浅谈JNI技术在嵌入式软件开发中的应用

相关新闻