



HarmonyOS内核开发—任务管理

本节主要介绍:

- 任务的相关概念
- 任务的调度机制
- 如何创建和删除任务

三 目录

1. 任务管理简介
2. 任务相关概念
3. 任务的调度机制
4. 实现任务的管理
5. 实验结果与扩展实验
6. 总结



任务管理简介

基本概念

- 1、从系统的角度看，任务是竞争系统资源的最小运行单元。任务可以使用或等待CPU、使用内存空间等系统资源，并独立于其它任务运行。
- 2、LiteOS的任务模块可以给用户提供多个任务，实现了任务之间的切换和通信，帮助用户管理业务程序流程。这样用户可以将更多的精力投入到业务功能的实现中。
- 3、LiteOS中的任务是抢占式调度机制，高优先级的任务可打断低优先级任务，低优先级任务必须在高优先级任务阻塞或结束后才能得到调度，同时支持时间片轮转调度方式。
- 4、LiteOS的任务默认有32个优先级(0-31)，最高优先级为0，最低优先级为31。



任务相关概念

任务状态

任务状态通常分为以下四种：

就绪 (Ready)：该任务在就绪列表中，只等待CPU。

运行 (Running)：该任务正在执行。

阻塞 (Blocked)：该任务不在就绪列表中。包含任务被挂起、任务被延时、任务正在等待信号量、读写队列或者等待读写事件等。

退出态 (Dead)：该任务运行结束，等待系统回收资源。



任务相关概念

任务ID：在任务创建时通过参数返回给用户，作为任务的一个非常重要的标识。

任务优先级：优先级表示任务执行的优先顺序。

任务入口函数：每个新任务得到调度后将执行的函数。

任务控制块TCB：每一个任务都含有一个任务控制块(TCB)。TCB包含了任务上下文栈指针 (stack pointer)、任务状态、任务优先级、任务ID、任务名、任务栈大小等信息。TCB可以反映出每个任务运行情况。

任务栈：每一个任务都拥有一个独立的栈空间，我们称为任务栈。

任务上下文：任务在运行过程中使用到的一些资源，如寄存器等，我们称为任务上下文。LiteOS在任务挂起的时候会将本任务的上下文信息，保存在自己的任务栈里面，以便任务恢复后，从栈空间中恢复挂起时的上下文信息，从而继续执行被挂起时被打断的代码。

任务切换：任务切换包含获取就绪列表中最高优先级任务、切出任务上下文保存、切入任务上下文恢复等动作。



任务的调度机制

任务状态迁移说明：

就绪态→运行态：任务创建后进入就绪态，发生任务切换时，就绪列表中最高优先级的任务被执行，从而进入运行态，但此刻该任务依旧在就绪列表中。

运行态→阻塞态：任务运行因挂起、读信号量等待等，在就绪列表中被删除进入阻塞。

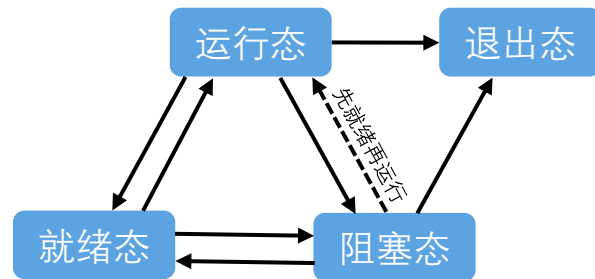
阻塞态→就绪态（阻塞态→运行态）：阻塞的任务被恢复后（任务恢复、延时时间超时、读信号量超时或读到信号量等），此时被恢复的任务会被加入就绪列表，从而由阻塞态变成就绪态；此时如果被恢复任务的优先级高于正在运行任务的优先级，则会发生任务切换，将该任务由就绪态变成运行态。

就绪态→阻塞态：任务也有可能就在就绪态时被阻塞（挂起）。

运行态→就绪态：有更高优先级任务创建或者恢复后，发生任务切换而进入就绪列表。

运行态→退出态：任务运行结束，内核自动将此任务删除，此时由运行态变为退出态。

阻塞态→退出态：阻塞的任务调用删除接口，任务状态由阻塞态变为退出态。





实现任务管理

cmsis_os2的API任务接口简介:

接口名	功能描述
osThreadNew	创建任务
osThreadTerminate	删除某个任务（一般是对非自任务操作）
osThreadSuspend	任务挂起
osThreadResume	任务恢复

创建任务: `osThreadNew(osThreadFunc_t func,void * argument,const osThreadAttr_t * attr)`

删除某个任务: `osThreadTerminate(osThreadId_t thread_id);`

任务挂起: `osThreadSuspend(osThreadId_t thread_id)`

任务恢复: `osThreadResume (osThreadId_t thread_id)`



实现任务的创建

创建任务接口详解：

`osThreadNew(osThreadFunc_t func, void * argument, const osThreadAttr_t * attr)`

名称	描述
func	任务函数.
argument	作为启动参数传递给任务函数的指针
attr	任务入口函数的参数列表
返回值	任务ID



实验结果与扩展实验

扩展实验代码

```
/******任务一*****/  
void threadHi(void)  
{  
    printf("enter threadHi\r\n");  
    osDelay(1);  
    printf("threadHi delay done\r\n");  
    osThreadSuspend(threadHiID);  
    printf("threadHi osThreadResume success\r\n");  
    osThreadTerminate(threadHiID);  
}
```

```
/******任务二*****/  
void threadLo(void)  
{  
    for(int i = 0; i < 10; i++)  
    {  
        printf("enter threadLo\r\n");  
    }  
    printf("threadHi osThreadSuspend success\r\n");  
    osThreadResume(threadHiID);  
    osThreadTerminate(threadLoID);  
}
```

```
osThreadId_t threadHiID ;  
osThreadId_t threadLoID ;  
/******任务创建*****/  
static void Thread_example(void)  
{  
    osThreadAttr_t attr;  
  
    attr.name = "threadHi";  
    attr.attr_bits = 0U;  
    attr.cb_mem = NULL;  
    attr.cb_size = 0U;  
    attr.stack_mem = NULL;  
    attr.stack_size = 1024 * 4;  
    attr.priority = 25;  
    threadHiID = osThreadNew((osThreadFunc_t)threadHi, NULL,  
&attr);  
    if ( threadHiID == NULL)  
    {  
        printf("Falied to create threadHi!\n");  
    }  
  
    attr.name = "threadLo";  
    attr.priority = 24;  
    threadLoID = osThreadNew((osThreadFunc_t)threadLo, NULL,  
&attr);  
    if (threadLoID == NULL)  
    {  
        printf("Falied to create threadLo!\n");  
    }  
}  
  
SYS_RUN(Thread_example);
```

本节小结

- 1、了解任务的概念
- 2、掌握如何创建任务
- 3、掌握如何管理好多个任务的运行



谢谢观看

开源从小熊派开始

OPEN-SOURCE STARTED WITH THE BEARPI