# 基于小熊派IoT开发套件的LiteOS内核实战 - 任务管理

# 1、 LiteOS内核的任务管理

Huawei LiteOS 内核提供任务的创建、删除、延迟、挂起、恢复等功能，以及锁定和解锁任务调度，支持任务按优先级高低的抢占调度及同优先级时间片轮转调度。

## 1.1 任务

在 LiteOS 中，一个任务就是一个线程，多个任务按照优先级进行抢占式调度，达到多个任务“同时”运行的目的。

## 1.2 任务的状态

Huawei LiteOS 系统中的每个任务都有多种运行状态。当系统初始化完成并启动调度器后，系统中所有创建的任务就由内核进行调度，在不同运行状态之间切换，同时在系统中竞争一定的资源。

任务的状态有以下四种：

1） 就绪（Ready）：该任务在就绪列表中，只等待 CPU；

2） 运行（Running）：该任务正在执行；

3） 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；

4） 退出态（Dead）：该任务运行结束，等待系统回收资源。

## 1.3 任务ID

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。

用户可以通过任务ID对指定任务进行任务挂起、任务恢复、查询任务名等操作。

## 1.4 任务优先级

优先级表示任务执行的优先顺序。任务的优先级决定了在发生任务切换时即将要执行的任务，在就绪列表中的最高优先级的任务将得到执行。

Huawei LiteOS 的任务一共有 32 个优先级 (0-31)，最高优先级为 0，最低优先级为 31。

因为是LiteOS的内核是抢占式调度内核，所以：

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。

## 1.5 任务入口函数

任务入口函数是每个新任务得到调度后将执行的函数，该函数由用户实现，在任务创建时，通过任务创建结构体指定。

## 1.6 多任务运作背后的机制

在多任务操作系统的内核中，为了方便对每个任务进行管理，每一个任务都有一个任务控制块（TCB），其中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 相当于每个任务在内核中的身份证，可以反映出每个任务运行情况。

那么，操作系统中这么多的任务，它们依靠TCB被系统统一管理，那么又是如何被系统执行的呢？

其实，每个任务相当于一个裸机程序，每个任务之间相互独立，这个“独立”指的是每个任务都有自己的运行环境 —— 栈空间，称为任务栈，栈空间里保存的信息包含局部变量、寄存器、函数参数、函数返回地址等。

可是，系统中只有一个CPU，即使每个任务的任务栈是独立的，可是多个任务都需要被同一个CPU所执行，CPU的资源是共用的吧。

对的，CPU的资源是多个任务共用的，这些CPU的寄存器只有在任务执行的时候被使用，称为任务上下文，因此，内核在任务切换时会将切出任务的上下文信息保存在自身的任务栈空间里面，以便任务恢复时还原现场，从而在任务恢复后在切出点继续开始执行。

用户创建任务时，系统会先申请任务控制块需要的内存空间，申请成功后，系统会将任务栈进行初始化，预置上下文。此外，系统还会将“任务入口函数”地址放在相应位置。这样在任务第一次启动进入运行态时，将会执行“任务入口函数”。

# 2、 任务管理API

Huawei LiteOS 任务管理模块提供任务创建、任务删除、任务延时、任务挂起和任务恢复、更改任务优先级、锁任务调度和解锁任务调度、根据任务控制块查询任务 ID、根据 ID 查询任务控制块信息功能。

Huawei LiteOS 任务管理提供的 API 都是以 LOS 开头，但是这些 API 使用起来比较复杂，所以本文中我们使用 Huawei IoT Link SDK 提供的统一API接口进行实验，这些接口底层已经使用 LiteOS 提供的API实现，对用户而言更为简洁，API列表如下：

osal的api接口声明在中，使用相关的接口需要包含该头文件，关于函数的详细参数请参考该头文件的声明。

任务相关的接口定义在osal.c中，基于LiteOS的接口实现在 liteos\_imp.c文件中：

接口名 功能描述

osal\_task\_create 创建任务

osal\_task\_kill 删除任务（非自身）

osal\_task\_exit 任务退出

osal\_task\_sleep 任务休眠

## 2.1 osal\_task\_create

osal\_task\_create的接口用于创建一个任务，其接口原型如下：

void\* osal\_task\_create(const char \*name,int (\*task\_entry)(void \*args),\

void \*args,int stack\_size,void \*stack,int prior)

{

void \*ret = NULL;

if((NULL != s\_os\_cb) &&(NULL != s\_os\_cb->ops) &&(NULL != s\_os\_cb->ops->task\_create))

{

ret = s\_os\_cb->ops->task\_create(name, task\_entry,args,stack\_size,stack,prior);

}

return ret;

}

该接口的参数说明如下表：

参数名称 参数说明

name 任务名称

tsak\_entry 任务入口函数的函数指针

args 任务入口函数的参数列表

stack\_size 任务栈大小

stack 任务栈地址

prior 任务优先级

返回值 任务ID

## 2.2 osal\_task\_kill

osal\_task\_kill用于删除某个其他任务（非自身），其接口原型如下：

int osal\_task\_kill(void \*task)

{

int ret = -1;

if((NULL != s\_os\_cb) &&(NULL != s\_os\_cb->ops) &&(NULL != s\_os\_cb->ops->task\_kill))

{

ret = s\_os\_cb->ops->task\_kill(task);

}

return ret;

}

该接口的参数说明如下表：

参数名称 参数说明

task 任务ID

返回值 0-删除成功

返回值 -1-删除失败

## 2.3 osal\_task\_exit

osal\_task\_exit接口用于任务退出（自身），目前暂未支持，可以直接return退出。

## 2.4 osal\_task\_sleep

osal\_task\_sleep接口用于任务主动休眠，单位是ms,其接口原型如下：

void osal\_task\_sleep(int ms)

{

if((NULL != s\_os\_cb) &&(NULL != s\_os\_cb->ops) &&(NULL != s\_os\_cb->ops->task\_sleep))

{

s\_os\_cb->ops->task\_sleep(ms);

}

return ;

}

# 3、 动手实验 —— 体验任务的创建与切换

## 3.1 实验内容

本实验中将创建两个任务，一个低优先级任务task1，一个高优先级任务task2，两个任务都会每隔2s在串口打印自己的任务id号，在串口终端中观察两个任务的运行情况。

## 3.2 实验代码

首先打开之前创建的 HelloWorld 工程，基于此工程进行实验。

在Demo文件夹右击，选择新建文件夹：

新建osal\_kernel\_demo文件夹，用于存放内核的实验文件：

接下来在此osal\_kernel\_demo文件夹中新建第一个实验文件osal\_task\_demo.c文件，开始编写代码：

/\* 使用osal接口需要包含该头文件 \*/

#include <osal.h>

/\* 任务优先级宏定义（shell任务的优先级为10） \*/

#define USER\_TASK1\_PRI 12 //低优先级

#define USER\_TASK2\_PRI 11 //高优先级

/\* 任务ID \*/

uint32\_t user\_task1\_id = 0;

uint32\_t user\_task2\_id = 0;

/\* 任务task1入口函数 \*/

static int user\_task1\_entry()

{

int n = 0;

/\* 每隔2s在串口打印一次，打印5次后主动结束 \*/

for(n = 0; n < 5; n++)

{

printf("task1: my task id is %ld, n = %d!\r\n", user\_task1\_id, n);

/\* 任务主动挂起2s \*/

osal\_task\_sleep(2\*1000);

}

printf("user task 1 exit!\r\n");

/\* 任务结束 \*/

return 0;

}

/\* 任务task2入口函数 \*/

static int user\_task2\_entry()

{

/\* 每隔2s在串口打印一次，不结束 \*/

while (1)

{

printf("task 2: my task id is %ld!\r\n", user\_task2\_id);

/\* 任务主动挂起2s \*/

osal\_task\_sleep(2\*1000);

}

}

/\* 标准demo启动函数，函数名不要修改，否则会影响下一步实验 \*/

int standard\_app\_demo\_main()

{

/\* 创建任务task1 \*/

user\_task1\_id = osal\_task\_create("user\_task1",user\_task1\_entry,NULL,0x400,NULL,USER\_TASK1\_PRI);

/\* 创建任务task2 \*/

user\_task2\_id = osal\_task\_create("user\_task2",user\_task2\_entry,NULL,0x400,NULL,USER\_TASK2\_PRI);

return 0;

}

编写完成之后，要将我们编写的osal\_task\_demo.c文件添加到makefile中，加入整个工程的编译：

这里有个较为简单的方法，直接修改Demo文件夹下的user\_demo.mk配置文件，添加如下代码：

#example for osal\_task\_demo

ifeq ($(CONFIG\_USER\_DEMO), "osal\_task\_demo")

user\_demo\_src = ${wildcard $(TOP\_DIR)/targets/STM32L431\_BearPi/Demos/osal\_kernel\_demo/osal\_task\_demo.c}

user\_demo\_defs = -D CONFIG\_OSAL\_TASK\_DEMO\_ENABLE=1

endif

添加位置如图：

这段代码的意思是：

如果 CONFIG\_USER\_DEMO 宏定义的值是osal\_task\_demo，则将osal\_task\_demo.c文件加入到makefile中进行编译。

那么，如何配置 CONFIG\_USER\_DEMO 宏定义呢？在工程根目录下的.sdkconfig文件中的末尾即可配置：

因为我们修改了mk配置文件，所以点击重新编译按钮进行编译，编译完成后点击下载按钮烧录程序。

## 3.3 实验现象

程序烧录之后，即可看到程序已经开始运行，在串口终端中可看到实验的输出内容：

\*\*Link\*\*main:V1.2.1 AT 11:30:59 ON Nov 28 2019

WELCOME TO IOT\_LINK SHELL

LiteOS:/>task 2: my task id is 5!

task1: my task id is 4, n = 0!

task 2: my task id is 5!

task1: my task id is 4, n = 1!

task 2: my task id is 5!

task1: my task id is 4, n = 2!

task 2: my task id is 5!

task1: my task id is 4, n = 3!

task 2: my task id is 5!

task1: my task id is 4, n = 4!

task 2: my task id is 5!

user task 1 exit!

task 2: my task id is 5!

……

可以看到，系统启动后，首先打印版本号，串口shell的优先级为10，最先打印shell信息，接下来task1先创建，但是优先级较低，所以后创建的task2抢占执行，task2打印后主动挂起2s，这时task1开始执行，依次执行5次后task1结束，task2一直保持运行。