# 基于小熊派IoT开发套件的LiteOS内核实战-信号量

# 1、 LiteOS内核的信号量

## 1.1 信号量

在多任务操作系统中，不同的任务之间需要同步运行，信号量功能可以为用户提供这方面的支持。信号量（Semaphore）是一种实现任务间通信的机制，实现任务之间同步或临界资源的互斥访问。

## 1.2 信号量的使用方式

信号量可以被任务获取或者申请，不同的信号量通过信号量索引号来唯一确定，每个信号量都有一个计数值和任务队列。

通常一个信号量的计数值用于对应有效的资源数，表示剩下的可被占用的互斥资源数，其值的含义分两种情况：

0：表示没有积累下来的 Post 操作，且有可能有在此信号量上阻塞的任务；

正值：表示有一个或多个 Post 下来的释放操作；

当任务申请(Pend)信号量时，如果申请成功，则信号量的计数值递减，如若申请失败，则挂起在该信号量的等待任务队列上，一旦有任务释放该信号量，则等待任务队列中的任务被唤醒开始执行。

## 1.3 信号量的使用场景

信号量是一种非常灵活的同步方式，可以运用在多种场合中，实现锁、同步、资源计数等功能，也能方便的用于任务与任务，中断与任务的同步中。

互斥锁

用作互斥时，信号量创建后记数是满的，在需要使用临界资源时，先申请信号量，使其变空，这样其他任务需要使用临界资源时就会因为无法申请到信号量而阻塞，从而保证了临界资源的安全。

任务间同步

用作同步时，信号量在创建后被置为空，任务1申请信号量而阻塞，任务2在某种条件发生后，释放信号量，于是任务1得以进入 READY 或 RUNNING 态，从而达到了两个任务间的同步。

资源计数

用作资源计数时，信号量的作用是一个特殊的计数器，可以递增或者递减，但是值永远不能为负值，典型的应用场景是生产者与消费者的场景。

中断与任务的同步

用作中断与任务的同步时，可以在中断未触发时将信号量的值置为0，从而堵塞断服务处理任务，一旦中断被触发，则唤醒堵塞的中断服务处理任务进行中断处理。

# 2、 信号量API

Huawei LiteOS 系统中的信号量模块为用户提供创建/删除信号量、申请/释放信号量的功能。

Huawei LiteOS 系统中提供的信号量 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。

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\_semp\_create 信号量创建

osal\_semp\_del 信号量删除

osal\_semp\_pend 信号量申请

osal\_semp\_post 信号量释放

## 2.1 osal\_semp\_create

osal\_semp\_create接口用于创建一个信号量，其接口原型如下：

bool\_t osal\_semp\_create(osal\_semp\_t \*semp,int limit,int initvalue)

{

bool\_t ret = false;

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

{

ret = s\_os\_cb->ops->semp\_create(semp,limit,initvalue);

}

return ret;

}

该接口的参数说明如下表：

参数 描述

semp 信号量索引ID的地址

limit 信号量计数值的最大值

initvalue 信号量计数值的初始值

返回值 false - 创建失败

返回值 true - 创建成功

## 2.2 osal\_semp\_del

osal\_semp\_del接口用于删除一个信号量，其接口原型如下：

bool\_t osal\_semp\_del(osal\_semp\_t semp)

{

bool\_t ret = false;

if((NULL != s\_os\_cb) &&(NULL != s\_os\_cb->ops) &&(NULL != s\_os\_cb->ops->semp\_del))

{

ret = s\_os\_cb->ops->semp\_del(semp);

}

return ret;

}

该接口的参数说明如下表：

参数 描述

semp 信号量索引ID

返回值 false - 删除失败

返回值 true - 删除成功

## 2.3 osal\_semp\_pend

osal\_semp\_pend接口用于申请一个信号量，其接口原型如下：

bool\_t osal\_semp\_pend(osal\_semp\_t semp,int timeout)

{

bool\_t ret = false;

if((NULL != s\_os\_cb) &&(NULL != s\_os\_cb->ops) &&(NULL != s\_os\_cb->ops->semp\_pend))

{

ret = s\_os\_cb->ops->semp\_pend(semp,timeout);

}

return ret;

}

该接口的参数说明如下表：

参数 描述

semp 信号量索引ID

timeout 32位值，具体见下文

返回值 false - 申请失败

返回值 true - 申请成功

信号量有三种申请模式：非阻塞模式、永久阻塞模式、定时阻塞模式，用 timeout 参数的值选择。

非阻塞模式(0)：

任务需要申请信号量，若当前信号量的任务数没有到信号量设定的上限，则申请成功。否则，立即返回申请失败

永久阻塞模式(cn\_osal\_timeout\_forever或0xFFFFFFFF)：

任务需要申请信号量，若当前信号量的任务数没有到信号量设定的上限，则申请成功。否则，该任务进入阻塞态，系统切换到就绪任务中优先级最高者继续执行。任务进入阻塞态后，直到有其他任务释放该信号量，阻塞任务才会重新得以执行

定时阻塞模式(任意定时值，32bit)：

任务需要申请信号量，若当前信号量的任务数没有到信号量设定的上限，则申请成功。否则，该任务进入阻塞态，系统切换到就绪任务中优先级最高者继续执行。任务进入阻塞态后，指定时间超时前有其他任务释放该信号量，或者用户指定时间超时后，阻塞任务才会重新得以执行

由于中断不能被阻塞，因此在申请信号量时，阻塞模式不能在中断中使用。

## 2.4 osal\_semp\_post

osal\_semp\_post接口用于释放一个信号量，如果有任务阻塞于该信号量，则唤醒该信号量阻塞队列上的第一个任务，该任务进入就绪态，并进行调度。

其接口原型如下：

bool\_t osal\_semp\_post(osal\_semp\_t semp)

{

bool\_t ret = false;

if((NULL != s\_os\_cb) &&(NULL != s\_os\_cb->ops) &&(NULL != s\_os\_cb->ops->semp\_post))

{

ret = s\_os\_cb->ops->semp\_post(semp);

}

return ret;

}

该接口的参数说明如下表：

参数 描述

semp 信号量索引ID

返回值 false - 释放失败

返回值 true - 释放成功

# 3、 动手实验 —— 使用信号量进行任务间同步

## 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task2，两个任务之间使用信号量同步运行，在串口终端中观察两个任务的运行情况。

## 3.2 实验代码

首先打开上一篇使用的 HelloWorld 工程，基于此工程进行实验。

在Demo文件夹右击，新建文件夹osal\_kernel\_demo用于存放内核的实验文件（如果已有请忽略这一步）。

接下来在此文件夹中新建一个实验文件 osal\_semp\_demo.c，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

/\* 信号量索引ID \*/

osal\_semp\_t sync\_semp;

/\* 任务task1入口函数 \*/

static int user\_task1\_entry()

{

while(1)

{

/\* 在串口打印信息 \*/

printf("task 1 post a semp!\r\n");

/\* 打印完毕释放信号量 \*/

osal\_semp\_post(sync\_semp);

/\* 任务主动挂起2s \*/

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

}

}

/\* 任务task2入口函数 \*/

static int user\_task2\_entry()

{

while (1)

{

/\* 优先级高，抢占执行打印信息 \*/

printf("task2 is waiting for a semp...\r\n");

/\* 申请信号量，申请失败则挂起等待 \*/

osal\_semp\_pend(sync\_semp, cn\_osal\_timeout\_forever);

/\* 一旦申请到信号量，则恢复执行 \*/

printf("task 2 access a semp!\r\n");

}

}

/\* 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()

{

/\* 创建信号量sync\_semp \*/

osal\_semp\_create(&sync\_semp, 1, 0);

printf("sync\_semp semp create success.\r\n");

/\* 创建任务task1 \*/

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

/\* 创建任务task2 \*/

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\_semp\_demo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\_semp\_demo

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

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

user\_demo\_defs = -D CONFIG\_OSAL\_SEMP\_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\_semp\_demo，则将osal\_semp\_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。

编译之后会有警告，提示user\_task1\_entry和user\_task2\_entry函数没有返回值，这里使用了while(1)，不会返回，所以不用管。

## 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

sync\_semp semp create success.

WELCOME TO IOT\_LINK SHELL

LiteOS:/>task2 is waiting for a semp...

task 1 post a semp!

task 2 access a semp!

task2 is waiting for a semp...

task 1 post a semp!

task 2 access a semp!

task2 is waiting for a semp...

task 1 post a semp!

task 2 access a semp!

……

可以看到，系统启动后，首先打印版本号，由于串口shell的优先级为10，所以接下来打印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未等待到信号量，打印之后挂起，然后低优先级任务task1继续执行，打印信息之后释放信号量，释放之后task2任务被唤醒，恢复执行。