# OpenHarmony内核开发—信号量

# 前言

**本节主要介绍：**

- 信号量的相关概念
- 信号量的运作机制
- 如何利用信号量实现任务之间同步或临界资源的互斥访问

# 目录

# 信号量基本概念

## 信号量的概念

1、**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。常用于协助一组相互竞争的任务来访问临界资源。

2、在多任务系统中，各任务之间需要**同步或互斥实现临界资源的保护**，信号量功能可以为用户提供这方面的支持。

3、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：

1）0，表示没有积累下来的Post信号量操作，且有可能有在此信号量上阻塞的任务。

2）正值，表示有一个或多个Post信号量操作。

4、以同步为目的的信号量和以互斥为目的的信号量在使用有如下不同：

1）**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，先取信号量，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取到信号量而阻塞，从而保证了临界资源的安全。

2）**用作同步时**，信号量在创建后被置为空，任务1取信号量而阻塞，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。

# 信号量运作机制

**运作原理**

1、**信号量初始化**，为配置的N个信号量申请内存（N值可以由用户自行配置，受内存限制），并把所有的信号量初始化成未使用，并加入到未使用链表中供系统使用。

2、**信号量创建**，从未使用的信号量链表中获取一个信号量资源，并设定初值。

3、**信号量申请**，若其计数器值大于0，则直接减1返回成功。否则任务阻塞，等待其它任务释放该信号量，等待的超时时间可设定。当任务被一个信号量阻塞时，将该任务挂到信号量等待任务队列的队尾。

4、**信号量释放**，若没有任务等待该信号量，则直接将计数器加1返回。否则唤醒该信号量等待任务队列上的第一个任务。

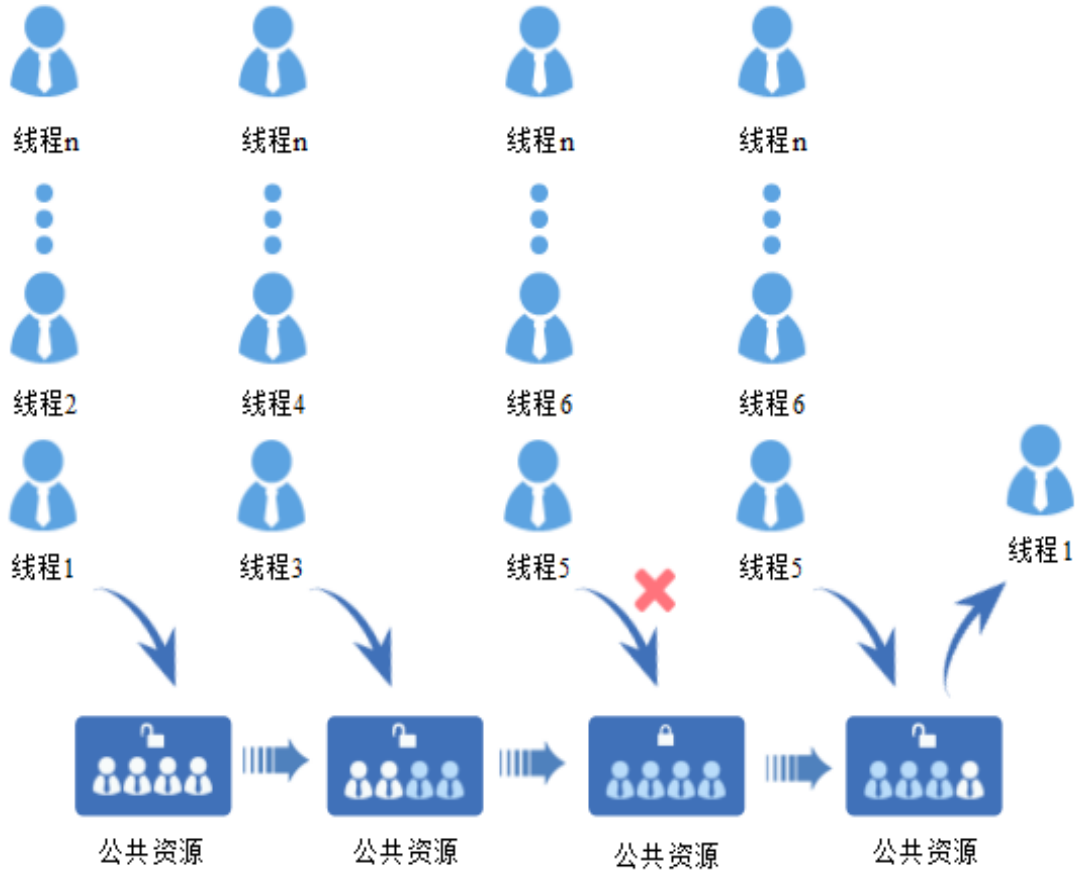
5、**信号量删除**，将正在使用的信号量置为未使用信号量，并挂回到未使用链表。

# 信号量运作机制

6、**信号量允许多个任务在同一时刻访问同一资源**，但会限制同一时刻访问此资源的最大任务数目。访问同一资源的任务数达到该资源的最大数量时，会阻塞其他试图获取该资源的任务，直到有任务释放该信号量。

**信号量运作示意图：**

公共资源有四个任务数，信号量都分别被线程1、2、3、4获取后，此时此资源就会锁定而不让线程5进入，线程5及后面的线程都进入阻塞模式，当线程1工作完成而释放出信号量，线程5立即获得信号而得到执行。如此往复。



# 实现信号量功能

## cmsis_os2的API信号量接口简介：

| 接口名 | 功能描述 |
| --- | --- |
| osSemaphoreNew | 创建信号量 |
| osSemaphoreAcquire | 获取信号量 |
| osSemaphoreRelease | 释放信号量 |
| osSemaphoreDelete | 删除信号量 |

**创建互斥锁：**osSemaphoreNew (uint32_t max_count, uint32_t initial_count, const osSemaphoreAttr_t *attr);

**获取互斥锁：**osSemaphoreAcquire (osSemaphoreId_t semaphore_id, uint32_t timeout);

**释放互斥锁：**osSemaphoreRelease (osSemaphoreId_t semaphore_id);

**删除互斥锁：**osMutexDelete (osMutexId_t mutex_id);

# 信号量扩展实验

## 扩展实验代码

```
void Thread_Semaphore1(void)
{
    osStatus_t status;
    while (1)
    {
        //释放两次sem1信号量，使得Thread_Semaphore2和Thread_Semaphore3能同步执行
        status = osSemaphoreRelease(sem1);
        if(status != osOK)
        {
            printf("Thread_Semaphore1 Release  Semap failed\n");
        }
        else
        {
            printf("Thread_Semaphore1 Release  Semap success\n");
        }
        // //此处若只释放一次信号量，则Thread_Semaphore2和Thread_Semaphore3会交替运行
。
        // osSemaphoreRelease(sem1);
        osDelay(100);
    }
}
```

```
void Thread_Semaphore2(void)
{
    osStatus_t status;
    while (1)
    {
        //申请sem1信号量
        status= osSemaphoreAcquire(sem1, 50U);
        if(status != osOK)
        {
            printf("Thread_Semaphore2 get Semap failed\n");
        }
        else
        {
            printf("Thread_Semaphore2 get Semap success\n");
        }

    }
}
```

```
void Thread_Semaphore3(void)
{
    osStatus_t status;
    while (1)
    {
        //申请sem1信号量
        status = osSemaphoreAcquire(sem1, osWaitForever);
        if(status != osOK)
        {
            printf("Thread_Semaphore3 get Semap failed\n");
        }
        else
        {
            printf("Thread_Semaphore3 get Semap success\n");
        }

        osDelay(1);
    }
}
```

# 本节小结

- 1、了解信号量的概念
- 2、掌握如何创建和删除信号量
- 3、如何利用信号量实现任务之间同步或临界资源的互斥访问

# 谢谢观看

开 源 从 小 熊 派 开 始

OPEN-SOURCE STARTED WITH THE BEARPI