凌智電子
LOCKZHINER Electronic

OpenHarmony内核开发

信号量

凌智電子
LOCKZHINER Electronic

目 录

CONTENTS

01 什么是信号量

■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。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。

■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。

02 信号量的运行机制

一般来说，信号量的运行机制依照工作流程可分为以下几个步骤：

（1）信号量初始化；

（2）信号量创建；

（3）信号量申请；

（4）信号量释放；

（5）信号量删除。

总而言之，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。

03 信号量的接口

信号量接口的头文件

/kernel/liteos_m/kernel/include/los_sem.h

OpenHarmony内核开发中，信号量接口可分为以下两大类：

（1）创建、删除信号量；

（2）申请、释放信号量。

03 信号量的接口

功能分类	接口名	功能描述
创建、删除信号量	LOS_SemCreate	创建信号量，返回信号量ID
	LOS_BinarySemCreate	创建二值信号量，其计数值最大为1
	LOS_SemDelete	删除指定的信号量
申请、释放信号量	LOS_SemPend	申请指定的信号量，并设置超时时间
	LOS_SemPost	释放指定的信号量

03 信号量的接口

```
UINT32 LOS_SemCreate(UINT16 count, UINT32 *semHandle);
```

该函数主要功能是创建信号量，返回信号量ID。

参数count表示初始化时信号量资源有多少个（该值取值范围为：0~OS_SEM_COUNTING_MAX_COUNT）；

参数semHandle则是信号量的句柄。

返回LOS_OK为成功，其余为失败。

03 信号量的接口

UINT32 LOS_BinarySemCreate(UINT16 count, UINT32 *semHandle);

该函数主要功能是创建二值信号量，其计数值最大为1。

参数count表示初始化时信号量资源有多少个（该值取值范围为：0~1）；

参数semHandle则是信号量的句柄。

返回LOS_OK为成功，其余为失败。

UINT32 LOS_SemDelete(UINT32 semHandle);

该函数主要功能是删除指定的任务。

参数taskID为任务的句柄。

返回LOS_OK为成功，其余为失败。

03 信号量的接口

UINT32 LOS_SemPend(UINT32 semHandle, UINT32 timeout);

该函数主要功能是申请指定的信号量，并设置超时时间。

参数sendHandle是信号量句柄；

参数timeout是设置超时时间，时间单位为Tick。

返回LOS_OK为成功，其余为失败。

03 信号量的接口

UINT32 LOS_SemPost(UINT32 semHandle);

该函数主要功能是释放指定的信号量。

参数sendHandle是信号量句柄。

返回LOS_OK为成功，其余为失败。

04 如何使用信号量

1、打开sdk下面路径的文件

vendor/lockzhiner/rk2206/samples/a2_kernel_semaphore/kernel_semaphore_example.c

2、创建任务

在semaphore_example()中，通过LOS_SemCreate()创建一个信号量m_sem。其中，MAX_COUNT为信号量的初始化资源数量，一共有4个资源。。

```
    ret = LOS_SemCreate(MAX_COUNT, &m_sem);
    if (ret != LOS_OK)
    {
        printf("Falied to create Semaphore\n");
        return;
    }
```

如何使用信号量

3、创建3个任务

在semaphore_example()中，通过LOS_TaskCreate()创建3个任务。

```
unsigned int thread_crtl;
    unsigned int thread_id1;
    unsigned int thread_id2;
    TSK_INIT_PARAM_S task1 = {0};
    TSK_INIT_PARAM_S task2 = {0};
    TSK_INIT_PARAM_S task3 = {0};
    unsigned int ret = LOS_OK;
```

如何使用信号量

```
    task1.pfnTaskEntry = (TSK_ENTRY_FUNC)control_thread;
    task1.uwStackSize = 2048;
    task1.pcName = "control_thread";
    task1.usTaskPrio = 24;
    ret = LOS_TaskCreate(&thread_crtl, &task1);
    if (ret != LOS_OK)
    {
        printf("Falied to create control_thread ret:0x%x\n", ret);
        return;
    }
```

如何使用信号量

```
    task2.pfnTaskEntry = (TSK_ENTRY_FUNC)sem_one_thread;
    task2.uwStackSize = 2048;
    task2.pcName = "sem_one_thread";
    task2.usTaskPrio = 24;
    ret = LOS_TaskCreate(&thread_id1, &task2);
    if (ret != LOS_OK)
    {
        printf("Falied to create sem_one_thread ret:0x%x\n", ret);
        return;
    }
```

```
    task3.pfnTaskEntry = (TSK_ENTRY_FUNC)sem_two_thread;
    task3.uwStackSize = 2048;
    task3.pcName = "sem_two_thread";
    task3.usTaskPrio = 24;
    ret = LOS_TaskCreate(&thread_id2, &task3);
    if (ret != LOS_OK)
    {
        printf("Falied to create sem_two_thread ret:0x%x\n", ret);
        return;
    }
```

如何使用信号量

其中，control_thread任务每隔一段时间通过LOS_SemPost()释放1~2个信号资源。具体代码如下：

```
void control_thread()
{
    unsigned int count = 0;

    while (1)
    {
        /*释放两次信号量，sem_one_thread和sem_two_thread同步执行;
        释放一次信号量，sem_one_thread和sem_two_thread交替执行*/
        if (count++%3)
        {
            LOS_SemPost(m_sem);
            printf("control_thread Release once Semaphore\n");
        }
        else
        {
            LOS_SemPost(m_sem);
            LOS_SemPost(m_sem);
            printf("control_thread Release twice Semaphore\n");
        }

        LOS_Msleep(1000);
    }
}
```

如何使用信号量

而sem_one_thread和sem_two_thread任务则每隔100msec通过LOS_SemPend申请抢占信号量资源。具体代码如下：

```
void sem_one_thread()
{
    while (1)
    {
        /*申请信号量*/
        LOS_SemPend(m_sem, LOS_WAIT_FOREVER);

        printf("sem_one_thread get Semaphore\n");
        LOS_Msleep(100);
    }
}
```

```
void sem_two_thread()
{
    while (1)
    {
        /*申请信号量*/
        LOS_SemPend(m_sem, LOS_WAIT_FOREVER);

        printf("sem_two_thread get Semaphore\n");
        LOS_Msleep(100);
    }
}
```

如何创建和删除任务

4、修改编译脚本

修改 vendor/lockzhiner/rk2206/sample 路径下 BUILD.gn 文件，指定 semaphore_example 参与编译。

```
"./a2_kernel_semaphore:semaphore_example",
```

修改 device/lockzhiner/rk2206/sdk_liteos 路径下 Makefile 文件，添加 -lsemaphore_example 参与编译。

```
hardware_LIBS = -lhal_iothardware -lhardware -lsemaphore_example
```

5、编译固件

```
hb set -root .
```

```
hb set
```

```
hb build -f
```

如何创建和删除任务

6、烧写固件

7、通过串口查看结果

运行结果

```
control_thread Release once Semaphore
sem_one_thread get Semaphore
control_thread Release once Semaphore
sem_two_thread get Semaphore
control_thread Release twice Semaphore
sem_two_thread get Semaphore
sem_one_thread get Semaphore
……
```

凌智電子
LOCKZHINER Electronic

谢谢聆听

单击此处添加副标题内容