凌智電子
LOCKZHINER Electronic

OpenHarmony内核开发

队列

凌智電子
LOCKZHINER Electronic

目 录

CONTENTS

01 什么是队列

OpenHarmony内核中，队列又称消息队列，是一种常用于**任务间通信**的数据结构。队列接收来自任务或中断的不固定长度消息，并根据不同的接口确定传递的消息是否存放在队列空间中。

任务能够从队列里面读取消息，当队列中的消息为空时，挂起读取任务；当队列中有新消息时，挂起的读取任务被唤醒并处理新消息。任务也能够往队列里写入消息，当队列已经写满消息时，挂起写入任务；当队列中有空闲消息节点时，挂起的写入任务被唤醒并写入消息。如果将读队列和写队列的超时时间设置为0，则不会挂起任务，接口会直接返回，这就是**非阻塞模式**。

消息队列提供了**异步处理**机制，允许将一个消息放入队列，但不立即处理。同时队列还有**缓冲消息**的作用。

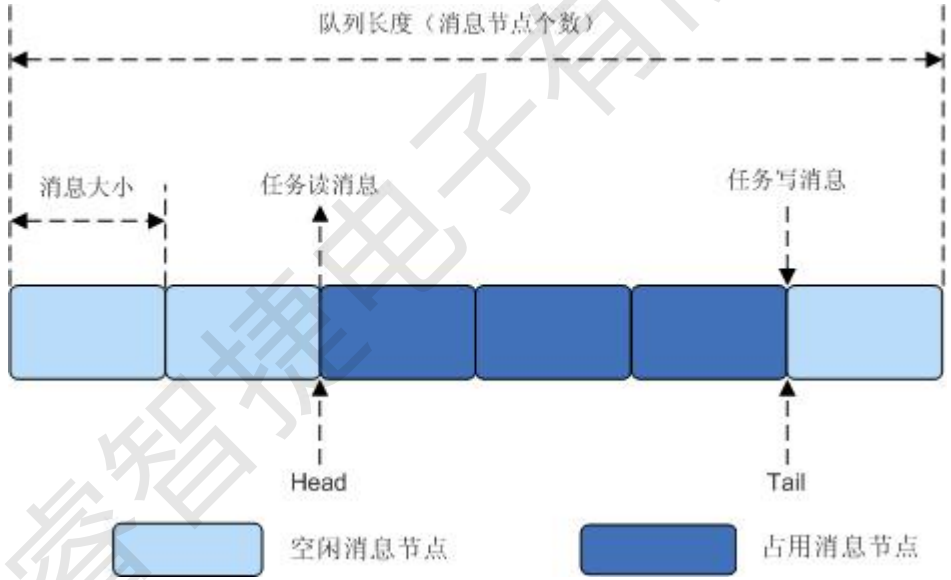
凌智電子
LOCKZHINER Electronic

01 什么是队列

LiteOS中使用队列实现任务异步通信，具有如下特性：

- ◆消息以先进先出的方式排队，支持异步读写。
- ◆读队列和写队列都支持超时机制。
- ◆每读取一条消息，就会将该消息节点设置为空闲。
- ◆发送消息类型由通信双方约定，可以允许不同长度的消息。
- ◆一个任务能够从任意一个消息队列接收和发送消息。
- ◆多个任务能够从同一个消息队列接收和发送消息。
- ◆创建队列时所需的队列空间，默认支持接口内系统自行动态申请内存的方式，同时也支持将用户分配的队列空间作为接口入参传入的方式。

02 队列机制



读队列时，判断队列是否有消息需要读取，对空闲的队列进行读操作会引起任务挂起。如果队列可以读取消息，则找到最先写入队列的消息节点进行读取。

写队列时，判断队列是否可以写入，不能对已满的队列进行写操作。写队列支持两种写入方式：向队列尾节点写入，也可以向队列头节点写入。尾节点写入时，找到起始空闲消息节点作为数据写入对象，如果是队列尾部则采用回卷方式。

凌智電子
LOCKZHINER Electronic

03 队列的接口

队列接口的头文件

/kernel/liteos_m/kernel/include/los_queue.h

OpenHarmony内核开发中，队列接口有很多，主要分为几大类：

（1）创建和删除队列；

（2）读/写队列（不带拷贝）；

（3）读/写队列（带拷贝）；

（4）获取队列信息；

凌智電子 LOCKZHINER Electronic

03 队列的接口

功能分类	接口名	功能描述
创建/删除队列	LOS_QueueCreate	创建一个消息队列，由系统动态申请队列空间。
	LOS_QueueCreateStatic	创建一个消息队列，由用户分配队列内存空间传入接口。
	LOS_QueueDelete	根据队列ID删除一个指定队列。
读/写队列（不带拷贝）	LOS_QueueRead	读取指定队列头节点中的数据（队列节点中的数据实际上是一个地址）。
	LOS_QueueWrite	向指定队列尾节点中写入入参bufferAddr的值（即buffer的地址）。
	LOS_QueueWriteHead	向指定队列头节点中写入入参bufferAddr的值（即buffer的地址）。
读/写队列（带拷贝）	LOS_QueueReadCopy	读取指定队列头节点中的数据。
	LOS_QueueWriteCopy	向指定队列尾节点中写入入参bufferAddr中保存的数据。
	LOS_QueueWriteHeadCopy	向指定队列头节点中写入入参bufferAddr中保存的数据。
获取队列信息	LOS_QueueInfoGet	获取指定队列的信息，包括队列ID、队列长度、消息节点大小、头节点、尾节点、可读节点数量、可写节点数量、等待读操作的任务、等待写操作的任务、等待mail操作的任务。

03 队列的接口

```
UINT32 LOS_QueueCreate(CHAR *queueName,
                       UINT16 len,
                       UINT32 *queueID,
                       UINT32 flags,
                       UINT16 maxMsgSize);
```

该函数主要功能是创建一个消息队列，由系统动态申请队列空间。其中，参数queueName为需要创建的队列名，参数len为需要创建队列的长度，参数queueID为生成队列的ID，参数flags为队列模式，参数maxMsgSize为队列节点大小。

执行函数返回LOS_OK为成功，其余为失败。

03 队列的接口

UINT32 LOS_QueueDelete(UINT32 queueID);

该函数主要功能是删除指定的队列。参数queueID为需要删除队列的ID。

接口执行完毕后，返回LOS_OK为成功，其余为失败。

03 队列的接口

```
UINT32 LOS_QueueRead(UINT32 queueID,
                     VOID *bufferAddr,
                     UINT32 bufferSize,
                     UINT32 timeOut);
```

该函数主要功能是读取指定队列头节点中的数据。其中，参数queueID为需要读取的队列ID；参数bufferAdder为缓冲区指针，该指数指向的是缓冲区数据的地址；参数bufferSize为缓冲区大小；参数timeout为超时时间。

接口执行完毕后，返回LOS_OK为成功，其余为失败。

03 队列的接口

```
UINT32 LOS_QueueWrite(UINT32 queueID,
                      VOID *bufferAddr,
                      UINT32 bufferSize,
                      UINT32 timeOut);
```

该函数主要功能是数据写入指定队列。其中，参数queueID为需要写入的队列ID；参数bufferAdder为待写入数据缓冲区指针；参数bufferSize为缓冲区大小；参数timeout为超时时间。

接口执行完毕后，返回LOS_OK为成功，其余为失败。

凌智電子
LOCKZHINER Electronic

04 如何使用队列

1、打开sdk下面路径的文件

vendor/lockzhiner/rk2206/samples/a5_kernal_queue/kernel_queue_example.c

2、创建队列

在queue_example函数中，通过LOS_QueueCreate函数创建队列，并通过LOS_TaskCreate

创建msg_write_thread和msg_read_thread两个任务。

```
void task_example()
{
    unsigned int ret = LOS_OK;
    ret = LOS_QueueCreate("queue", MSG_QUEUE_LENGTH, &m_msg_queue, 0, BUFFER_LEN);
    if (ret != LOS_OK)
    {
        printf("Falied to create Message Queue ret:0x%x\n", ret);
        return;
    }
}
```

如何使用队列

```
task1.pfnTaskEntry = (TSK_ENTRY_FUNC)msg_write_thread;
    task1.uwStackSize = 2048;
    task1.pcName = " msg_write_thread";
    task1.usTaskPrio = 24;
    ret = LOS_TaskCreate(&thread_id1, &task1);
    if (ret != LOS_OK)
    {
        printf("Falied to create msg_write_thread ret:0x%x\n", ret);
        return;
    }
```

```
task2.pfnTaskEntry = (TSK_ENTRY_FUNC)msg_read_thread;
    task2.uwStackSize = 2048;
    task2.pcName = " msg_read_thread";
    task2.usTaskPrio = 25;
    ret = LOS_TaskCreate(&thread_id2, &task2);
    if (ret != LOS_OK)
    {
        printf("Falied to create msg_read_thread ret:0x%x\n", ret);
        return;
    }
}
```

如何使用队列

```
void msg_write_thread(void *arg)
{
    unsigned int data = 0;
    unsigned int ret = LOS_OK;

    while (1)
    {
        data++;
        ret = LOS_QueueWrite(m_msg_queue, (void *)&data,
sizeof(data), LOS_WAIT_FOREVER);
        if (LOS_OK != ret)
        {
            printf("%s write Message Queue msg fail ret:0x%x\n",
__func__, ret);
        }
        else
        {
            printf("%s write Message Queue msg:%u\n", __func__,
data);
        }

        /*delay 1s*/
        LOS_Msleep(1000);
    }
}
```

```
void msg_read_thread(void *arg)
{
    unsigned int addr;
    unsigned int ret = LOS_OK;
    unsigned int *pData = NULL;

    while (1)
    {
        /*wait for message*/
        ret = LOS_QueueRead(m_msg_queue, (void *)&addr,
BUFFER_LEN, LOS_WAIT_FOREVER);
        if (ret == LOS_OK)
        {
            pData = addr;
            printf("%s read Message Queue msg:%u\n", __func__,
*pData);
        }
        else
        {
            printf("%s read Message Queue fail ret:0x%x\n",
  func__, ret);
        }
    }
}
```

如何使用队列

4.修改编译脚本

修改 vendor/lockzhiner/rk2206/sample 路径下 BUILD.gn 文件，指定 a5_kernal_queue 参与编译。

```
"./a5_kernal_queue:kernal_queue",
```

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

```
hardware_LIBS = -lhal_iothardware -lhardware -lkernal_queue
```

5.编译固件

```
hb set -root .
```

```
hb set
```

```
hb build -f
```

如何使用队列

6.烧写固件

7.通过串口查看结果

运行结果

```
msg_write_thread write Message Queue msg:1
msg_read_thread read Message Queue msg:1
msg_write_thread write Message Queue msg:2
msg_read_thread read Message Queue msg:2
msg_write_thread write Message Queue msg:3
msg_read_thread read Message Queue msg:3
msg_write_thread write Message Queue msg:4
msg_read_thread read Message Queue msg:4
……
```

凌智電子
LOCKZHINER Electronic

谢谢聆听

单击此处添加副标题内容