

在进行 mapping 的修改之前, 用 `g_file_mapping.lock` 进行了加锁, 此时所有其他尝试对同样的锁进行加锁操作的 Task, 都会进入阻塞状态, 这样就保证了当前 Task 对 mapping 修改的独占性和完整性。

在完成相关的 mapping 修改后, 释放 `g_file_mapping.lock`。

6.6.3 队列 (queue)

队列提供了一种跨 Task 的按照约定方式读写数据的机制, 队列有头有尾, 常用于读写不同步的消息传递。

LiteOS 的队列控制块定义如下:

```
typedef struct {
    UINT8 *queueHandle;    /** 队列句柄*/
    UINT16 queueState;     /** 队列状态, 启用/未启用*/
    UINT16 queueLen;       /** 队列长度*/
    UINT16 queueSize;      /** 队列节点的大小*/
    UINT32 queueID;        /** 队列 ID*/
    UINT16 queueHead;      /** 队头*/
    UINT16 queueTail;      /** 队尾*/
    UINT16 readWriteableCnt[OS_QUEUE_N_RW]; /** 可读和可写资源的数量, 第一个元素是可读, 第二个元素是可写*/
    LOS_DL_LIST readWriteList[OS_QUEUE_N_RW]; /** 独写的双向链表, 第一个是读双向链表, 第二个是写双向链表*/
    LOS_DL_LIST memList;   /** 内存元素双向链表*/
} LosQueueCB;
```

LiteOS 的队列定义了如下的几种操作方法:

```
typedef enum {
    OS_QUEUE_READ = 0,
    OS_QUEUE_WRITE = 1,
    OS_QUEUE_N_RW = 2
} QueueReadWrite;
```

```
typedef enum {
    OS_QUEUE_HEAD = 0,
    OS_QUEUE_TAIL = 1
} QueueHeadTail;
```

`OS_QUEUE_READ` 表示可读, `OS_QUEUE_WRITE` 表示可写, `OS_QUEUE_N_RW` 表示统计的可读可写的数量, `OS_QUEUE_HEAD` 表示读/写的位置为队头, `OS_QUEUE_TAIL` 表示读/写的位置为队尾。

队列应用的场景并不多见, OpenHarmony 提供队列机制, 更多是为了兼容一些第三方代码所需要的基于 Linux 的 mbox 等机制。

6.6.4 信号灯 (semaphore)

在计算机科学中, 信号量用于控制并发系统/多任务操作系统中多个 Task 对公共资源的访问。信号量只是一个变量, 此变量用于解决临界区问题并在多处理器环境中实现过程同步。信号量根据程序员定义的条件进行更改 (例如递增、递减或切换)。