软件与微电子学院
集成电路与智能系统系

《嵌入式与物联网操作系统》

代码分析报告

姓　名：冯轩

学　号：2001210240

日　期：2021/4/26

一、OSMutexCreate

1 函数功能及使用说明

用于创建一个互斥信号量。

2 参数说明

OSMutexCreate 参数如下图 1 所示：

```
void  OSMutexCreate (OS_MUTEX  *p_mutex,
                     CPU_CHAR  *p_name,
                     OS_ERR    *p_err)
{
```

图 1 OSMutexCreate 参数

- *p_mutex：指向互斥型信号量控制块，互斥型信号量必须有用户应用程序进行实际分配；
- *p_name：互斥信号量的名字；
- *p_err：调用此函数后返回的错误码。

3 代码分析

标记所创建对象的数据结构为互斥信号量，如下图 2 所示：

```
#if (OS_OBJ_TYPE_REQ == DEF_ENABLED)
    p_mutex->Type              =  OS_OBJ_TYPE_MUTEX;
#endif
```

图 2 type

把第二个参数赋值给这个互斥信号量，如下图 3 所示：

```
#if (OS_CFG_DBG_EN == DEF_ENABLED)
    p_mutex->NamePtr           =  p_name;
#else
```

图 3 name

把当前信号量标记为可用，即“开锁”状态，如下图 4 所示：

```
p_mutex->OwnerNestingCtr   =                0u;
```

图 4 设置初始状态为开锁状态

初始化该互斥信号量的等待列表，如下图 5 所示：

```
OS_PendListInit(&p_mutex->PendList);
```

图 5 初始化等待列表

将该信号量添加到互斥信号量双向调试链表，互斥信号量个数加 1，如下图 6 所示：

```
#if (OS_CFG_DBG_EN == DEF_ENABLED)
    OS_MutexDbgListAdd(p_mutex);
    OSMutexQty++;
#endif
```

图 6 dbg

二、OSMutexPend

1 函数功能及使用说明

用于请求一个互斥信号量，当一个任务需要对资源进行独占式访问的时候就可以使用函数 OSMutexPend，如果该互斥信号量正在被其他的任务使用，那么 UCOSIII 就会将请求这个互斥信号量的任务放置在这个互斥信号量的等待表中。任务会一直等待，直到这个互斥信号量被释放掉，或者设定的超时时间到达为止。如果在设定的超时时间到达之前信号量被释放，UCOSIII 将会恢复所有等待这个信号量的任务中优先级最高的任务。

2 参数说明

OSMutexPend 参数如下图 7 所示：

```
void  OSMutexPend (OS_MUTEX  *p_mutex,
                   OS_TICK    timeout,
                   OS_OPT     opt,
                   CPU_TS    *p_ts,
                   OS_ERR    *p_err)
{
```

图 7 OSMutexPend 参数

- *p_mutex：指向互斥型信号量控制块；
- timeout：指定等待互斥信号量的超时时间（时钟节拍数），如果在指定的时间内互斥信号量没有释放，则允许任务恢复执行。该值设置为 0 的话，表示任务将会一直等待下去，直到信号量被释放掉。

- opt：用于选择是否使用阻塞模式；
- *p_ts：指向一个时间戳，记录发送、终止或删除互斥信号量的时刻；
- *p_err：用于保存调用此函数后返回的错误码。

3 代码分析

判断请求的信号量是否可用，如果可用，让任务持有当前信号量，并开始嵌套，并return，不再继续执行，如下图 8 所示：

```
if (p_mutex->OwnerNestingCtr == 0u) {
    p_mutex->OwnerTCBPtr      = OSTCBCurPtr;
    p_mutex->OwnerNestingCtr = 1u;
```

图 8 嵌套

如果当前任务已经持有该信号量，信号量嵌套数加 1，设置错误类型为“任务已经持有该信号量”，返回，不再继续执行，如下图 9 所示：

```
if (OSTCBCurPtr == p_mutex->OwnerTCBPtr) {
    if (p_mutex->OwnerNestingCtr == (OS_NESTING_CTR)-1) {
        CPU_CRITICAL_EXIT();
        OS_TRACE_MUTEX_PEND_FAILED(p_mutex);
        OS_TRACE_MUTEX_PEND_EXIT(OS_ERR_MUTEX_OVF);
       *p_err = OS_ERR_MUTEX_OVF;
        return;
    }
    p_mutex->OwnerNestingCtr++;
f (OS_CFG_TS_EN == DEF_ENABLED)
    if (p_ts != (CPU_TS *)0) {
       *p_ts = p_mutex->TS;
    }
ndif
    CPU_CRITICAL_EXIT();
    OS_TRACE_MUTEX_PEND_FAILED(p_mutex);
    OS_TRACE_MUTEX_PEND_EXIT(OS_ERR_MUTEX_OWNER);
   *p_err = OS_ERR_MUTEX_OWNER;
    return;
}
```

图 9 已经持有该信号量

如果当前任务非持有该信号量，分为两种情况：选择了不阻塞任务和选择了阻塞任务且调度器被锁，分别设置相应的错误类型后，返回，不再继续执行，如下图 10 所示：

```
    if ((opt & OS_OPT_PEND_NON_BLOCKING) != 0u) {
        CPU_CRITICAL_EXIT();
#if (OS_CFG_TS_EN == DEF_ENABLED)
        if (p_ts != (CPU_TS *)0) {
            *p_ts = 0u;
        }
#endif
        OS_TRACE_MUTEX_PEND_FAILED(p_mutex);
        OS_TRACE_MUTEX_PEND_EXIT(OS_ERR_PEND_WOULD_BLOCK);
       *p_err = OS_ERR_PEND_WOULD_BLOCK;
        return;
    } else {
        if (OSSchedLockNestingCtr > 0u) {
            CPU_CRITICAL_EXIT();
#if (OS_CFG_TS_EN == DEF_ENABLED)
            if (p_ts != (CPU_TS *)0) {
                *p_ts = 0u;
            }
#endif
            OS_TRACE_MUTEX_PEND_FAILED(p_mutex);
            OS_TRACE_MUTEX_PEND_EXIT(OS_ERR_SCHED_LOCKED);
           *p_err = OS_ERR_SCHED_LOCKED;
            return;
        }
    }
```

图 10 非持有

如果调度器未被锁，锁住调度器后重开中断，获取信号量持有任务，如果持有任务优先级低于当前任务，堵塞任务，将当前任务脱离就绪列表，并插入到节拍列表和等待列表，然后进入 switch 块，根据持有任务的任务状态分类处理，如下图 11 所示：

```
p_tcb = p_mutex->OwnerTCBPtr;
if (p_tcb->Prio > OSTCBCurPtr->Prio) {
    OS_TaskChangePrio(p_tcb, OSTCBCurPtr->Prio);
    OS_TRACE_MUTEX_TASK_PRIO_INHERIT(p_tcb, p_tcb->Prio);
}

OS_Pend((OS_PEND_OBJ *)((void *)p_mutex),
        OS_TASK_PEND_ON_MUTEX,
        timeout);

CPU_CRITICAL_EXIT();
OS_TRACE_MUTEX_PEND_BLOCK(p_mutex);
OSSched();

CPU_CRITICAL_ENTER();
switch (OSTCBCurPtr->PendStatus) {
    case OS_STATUS_PEND_OK:
f (OS_CFG_TS_EN == DEF_ENABLED)
        if (p_ts != (CPU_TS *)0) {
```

图 11 进入 switch 块

三、OSMutexPost

1 函数功能及使用说明

用于发送互斥信号量。

2 参数说明

OSMutexPost 参数如下图 12 所示：

```
void  OSMutexPost (OS_MUTEX  *p_mutex,
                   OS_OPT     opt,
                   OS_ERR    *p_err)
{
```

图 12 OSMutexPost 参数

- *p_mutex：指向互斥型信号量控制块；
- opt：用来指定是否进行任务调度操作；
- *p_err：用于保存调用此函数后返回的错误码。

3 代码分析

如果当前运行任务不持有该信号量，开中断，设置错误类型为“任务不持有该信号量”，返回，如下图 13 所示：

```
if (OSTCBCurPtr != p_mutex->OwnerTCBPtr) {
    CPU_CRITICAL_EXIT();
    OS_TRACE_MUTEX_POST_FAILED(p_mutex);
    OS_TRACE_MUTEX_POST_EXIT(OS_ERR_MUTEX_NOT_OWNER);
   *p_err = OS_ERR_MUTEX_NOT_OWNER;
    return;
}
```

图 13 判断是否持有

信号量嵌套数减 1 后，如果信号量仍被嵌套，解锁调度器，将错误类型设置为“信号量被嵌套”返回，如下图 14 所示：

```
p_mutex->OwnerNestingCtr--;
if (p_mutex->OwnerNestingCtr > 0u) {
    CPU_CRITICAL_EXIT();
    OS_TRACE_MUTEX_POST_EXIT(OS_ERR_MUTEX_NESTING);
   *p_err = OS_ERR_MUTEX_NESTING;
    return;
}
```

图 14 减 1 后仍嵌套

如果信号量未被嵌套，已可用，获取信号量的等待列表，如果没有任务在等待该信号量，清空信号量持有者信息，解锁调度器，错误类型设置为“无错误”，返回，如下图 15 所示：

```
p_pend_list = &p_mutex->PendList;
if (p_pend_list->HeadPtr == (OS_TCB *)0) {
    p_mutex->OwnerTCBPtr      = (OS_TCB *)0;
    p_mutex->OwnerNestingCtr =           0u;
    CPU_CRITICAL_EXIT();
    OS_TRACE_MUTEX_POST_EXIT(OS_ERR_NONE);
   *p_err = OS_ERR_NONE;
    return;
}
```

图 15 无错误

如果有任务在等待该信号量，判断如果当前任务优先级被修改过，就从就绪列表中移除此任务，并还原当前任务的优先级，在优先级表格中插入这个优先级，再将当前任务插入就绪列表队尾，更改当前任务优先级变量的值，如下图 16 所示：

```
if (OSTCBCurPtr->Prio != OSTCBCurPtr->BasePrio) {           /* Has owner inherited a pr
    prio_new = OS_MutexGrpPrioFindHighest(OSTCBCurPtr);     /* Yes, find highest priori
    prio_new = (prio_new > OSTCBCurPtr->BasePrio) ? OSTCBCurPtr->BasePrio : prio_new;
    if (prio_new > OSTCBCurPtr->Prio) {
        OS_RdyListRemove(OSTCBCurPtr);
        OSTCBCurPtr->Prio = prio_new;                       /* Lower owner's priority b
        OS_TRACE_MUTEX_TASK_PRIO_DISINHERIT(OSTCBCurPtr, prio_new);
        OS_PrioInsert(prio_new);
        OS_RdyListInsertTail(OSTCBCurPtr);                  /* Insert owner in ready li
        OSPrioCur          = prio_new;
    }
}
```

图 16 优先级判断

获取等待列表的首端任务，将信号量交给该任务，开始嵌套，最后释放信号量给该任务，如下图 17 所示：

```
p_tcb                    = p_pend_list->HeadPtr;
p_mutex->OwnerTCBPtr     = p_tcb;
p_mutex->OwnerNestingCtr = 1u;
OS_MutexGrpAdd(p_tcb, p_mutex);

OS_Post((OS_PEND_OBJ *)((void *)p_mutex),
                       p_tcb,
                       (void *)0,
                       0u,
                       ts);
```

图 17 获取等待列表首端任务