多任务同步和通讯

系统中运行的代码主要包括线程和 ISR，在系统运行过程中，它们的运行步骤有时需要同步，它们的访问资源有时需要互斥，在它们之间有时也要彼此交换数据。这些需求，有的是因为应用需求，有的是多任务编程模型带来的需求。因此内核必须提供相应的机制来完成这些功能。在这里把这些机制统称为进(线)程间通讯（IPC ,Internal Process Communication），常见的机制主要包括信号量、消息队列、邮箱、事件标记、管道、信号和条件变量等。

飞鸟 RTOS 目前支持的 IPC 机制包括信号量、互斥量、邮箱、消息队列和事件标记。在设计 IPC 机制时，对这几种 IPC 机制仔细分析，发现在线程的阻塞和数据的传输流程上，有很多相似的地方，可以设计一套比较完整通用的代码作为 IPC 模块的基础。本篇主要分析这些 IPC 机制的底层通用函数和数据结构。

线程阻塞队列

首先考虑一下线程队列的问题。前面有关线程的章节中，定义了两个内核线程队列：内核线程就绪队列和内核线程辅助队列。在 IPC 部分，我们又定义了一种类型的线程队列：线程阻塞队列。各类型的 IPC 对象，均带有线程阻塞特性，需要保存那些因 IPC 操作失败而不得不等待 IPC 条件满足的线程，线程阻塞队列就是用来实现这个功能的。

在飞鸟 RTOS 中，IPC 线程阻塞队列定义在文件 ipc.h 中。和其他内核相比，这个线程阻塞队列实现有自己的特点:线程阻塞队列中设计了两个线程阻塞分队列。这样做的理由是像邮箱、消息队列的功能里，会把数据分为紧急和一般的消息/邮件，所以需要分别保存到不同的队列中。如果同时有紧急数据和普通数据到达，内核优先考虑的是紧急消息。而同样类型的数据，比如都是普通消息或者都是紧急消息，那就既可以按照 FIFO 机制来处理，也可以按照发送数据线程的优先级机制来处理。线程阻塞队列结构如下所示：

代码清单 3-1：线程阻塞队列结构定义

```
1.  /* IPC 线程阻塞队列结构定义 */
2.  struct IPCBlockedQueueDef
3.  {
4.      TProperty* Property;          /* 队列宿主属性参数指针 */
5.      TLinkNode* PrimaryHandle;     /* 队列中基本线程分队列 */
6.      TLinkNode* AuxiliaryHandle;   /* 队列中辅助线程分队列 */
7.  };
```

- **Property 宿主属性参数指针**
  指向具体 IPC 对象的属性字段,在属性字段中,会有分队列的调度策略的参数,

指明两个分队列按照那种方式来调度：FIFO 或者线程优先级。

- **PrimaryHandle 队列中基本线程分队列**
  没有特殊要求的线程会阻塞在这个队列中

- **AuxiliaryHandle 队列中辅助线程分队列**
  在该队列中阻塞的线程会被特殊处理

通过这几个成员，针对线程的优先级、数据的特性，用户可以对每个 IPC 对象做出最符合应用的配置。下图基于信号量的结构演示 IPC 线程阻塞队列结构：

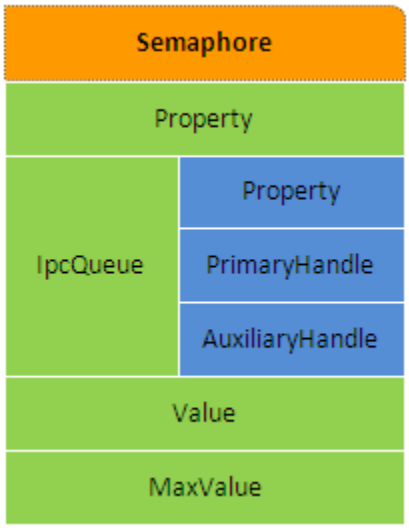

图 3-1 信号量的线程阻塞队列演示

线程阻塞记录

前面曾经提到过，在线程结构中有一个 IpcRecord 成员，当线程因操作 IPC 对象不能完成的时候，有时需要阻塞在 IPC 对象的线程阻塞队列上，而此时需要记录该线程在线程阻塞队列上的阻塞情况。包括被操作的 IPC 对象类型和地址，操作时的参数以及用于传递数据的指针。这些信息都保存在 IpcRecord 结构中。该成员的结构定义如下：

代码清单 3-2：IPC 阻塞记录结构定义

```
1.  /* 线程用于记录 IPC 对象的详细信息的记录结构 */
2.  struct IpcRecordDef
3.  {
4.      void*      Resource;     /* 指向 IPC 对象地址的指针              */
5.      TIpcQueue* Queue;        /* 线程所属 IPC 线程队列指针            */
6.      TIpcData   Data;         /* 和 IPC 对象操作相关的数据指针        */
7.      TOption    Option;       /* 访问 IPC 对象的操作参数              */
8.      TState     State;        /* IPC 对象操作的返回值                 */
9.      TErrno     Errno;        /* IPC 对象操作的错误代码               */
10. };
11. typedef struct IpcRecordDef TIpcRecord;
```

- Resource 被操作的 IPC 对象的地址
- Queue 线程所在的 IPC 的线程阻塞队列地址
- Data 如果线程需要交互数据，则这个变量指向线程待传输数据的地址
- Option 记录 IPC 操作的参数，比如是否有时限要求，是否在辅助阻塞队列中
- State 线程操作 IPC 的结果，可能是成功、失败、被取消、被取消初始化等
- Errno IPC 对象操作的错误代码

下图演示了线程访问信号量时的阻塞情况：

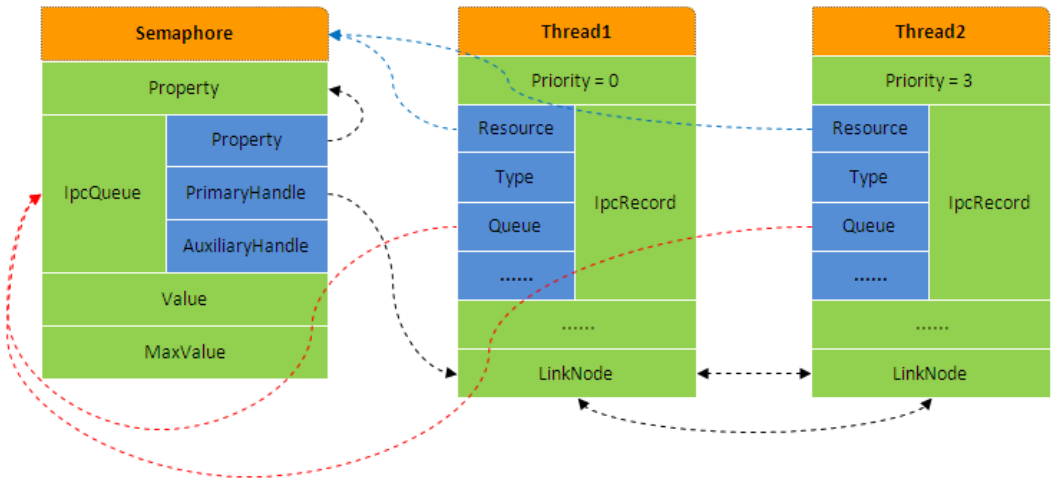

图 3-1 信号量的线程阻塞现场

从图中我们可以看到：

- 信号量的结构中有一个线程阻塞队列
- 线程阻塞队列的 Property 直接指向信号量的 Property 参数
- 线程阻塞队列中的基本队列中有两个线程被阻塞
- 被阻塞线程通过 IpcRecord 记录下当前操作的信号量的地址、阻塞队列的地址。所在阻塞队列的分队列由 IpcRecord 的 Option 成员记录。

底层支撑函数

当线程以阻塞方式访问一个 IPC 对象时，如果操作不能立刻成功，那么这个线程就要被阻塞了。访问每种类型的 IPC 对象的机制是不一样的，但阻塞时的流程却都差不多。需要几个函数配合来完成具体的阻塞操作。线程阻塞-解除阻塞的过程如下描述：

- 首先是线程发起 IPC 操作，但操作不能成功；
- 此时需要保存当前访问的 IPC 信息，如果是消息传递类型的 IPC 对象，还要保存被传递（发送和接收）的数据变量的地址；
- 然后当前线程的状态会被设置为阻塞态，并且会从内核线程就绪队列中挪到线程阻塞队列中。如果需要，还会立刻发起线程调度；
- 当 IPC 对象满足那些被阻塞的线程时，就需要选择其中一个最恰当的线程解除阻塞，如果需要，还要把数据传给那个刚被解除阻塞的线程；

- 等到该线程再次被调度时，它会检查本次 IPC 操作的结果并随后清除 IpcRecord 记录；

本模块的函数基本都是 IPC 机制的内部支持函数，这些函数只会被各 IPC 功能的代码来调用，并不直接暴露给用户，也不会被 API 层直接调用。主要包括以下几个函数：

编号	函数	作用
1	uIpcInitQueue	初始化线程阻塞队列
2	uIpcSaveRecord	保存 IPC 对象操作的信息到线程结构
3	uIpcCleanRecord	清除 IPC 对象操作的信息
4	uIpcReadState	获得 IPC 对象访问的结果
5	uIpcBlockThread	将当前线程阻塞在线程阻塞队列上
6	uIpcUnblockThread	将阻塞队列上的指定线程正常解除阻塞
7	uIpcUnblockOptimalThread	在阻塞队列上选择一个合适的线程并解除阻塞
8	uIpcUnblockAllThreads	将阻塞队列上的所有线程解除阻塞
9	uIpcAbortThread	将阻塞队列上的某个线程强制解除阻塞
10	uIpcDeActivate	将阻塞队列上的某个线程休眠
11	uIpcSetThreadPriority	修改阻塞队列上的某个线程的优先级

表 3-1 IPC 底层支持函数列表

本篇详细介绍了 trochili rtos 的多任务通讯机制的底层实现。下篇我们将仔细介绍信号量这个同步机制的原理、设计和实现。