阿熊的FreeRTOS教程系列！

哈喽大家好！我是你们的老朋友阿熊！STM32教程系列更新完结已经有一段时间了，视频反馈还是不错的，从今天开始我们将会更新我们的FreeRTOS的教程

由于东西真的太多了，也纠结了很久要不要讲这个系列，毕竟难度真的很大，怕在难以做到那么通俗易懂，经过一段时间的考虑，还是决定好了给大家做一个入门级的讲解使用，由于FreeRTOS的内容真的很多，作为还是学生的我使用的也相对较少，操作系统层面的东西，我会用最大的能力去让大家理解，主要讲述主要功能，学完以后保证大伙可以理解80%以上的FreeRTOS的使用场景，好了废话不多说，开始我们的课程吧！

第八章：互斥量（**Mutex**）

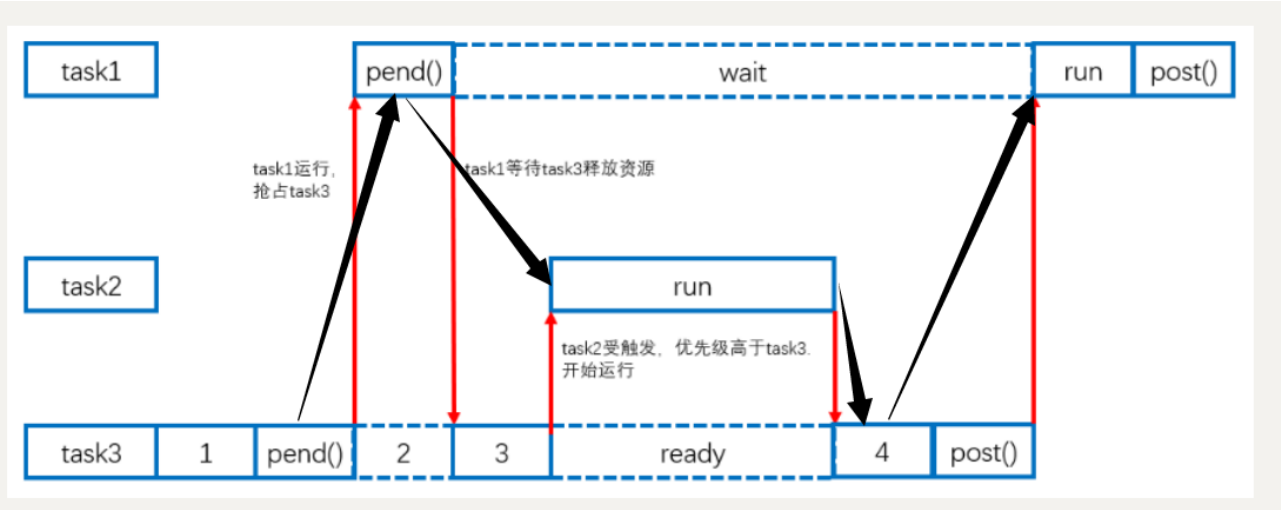
在我们的项目开发中，可能会有多个任务，共用一个资源的情况，为了避免其发生冲突，我们前面介绍了，我们的二值信号量可以一定程度上避免这种事情，但是它多数还是用于同步的状况，如果我们的任务之间出现优先级的差异，就会出现优先级反转的问题

壹：二值信号量的优先级反转现象

实验：

创建三个任务优先级从高到低，名字分别为H_Task、M_Task、L_Task

在H_Task和L_Task中加入二值信号量，观察任务状况



分析：当出现中间层优先级任务，我们的任务一等待时间将会大大增长，使得任务一明明优先级最高，任务还是最后执行的

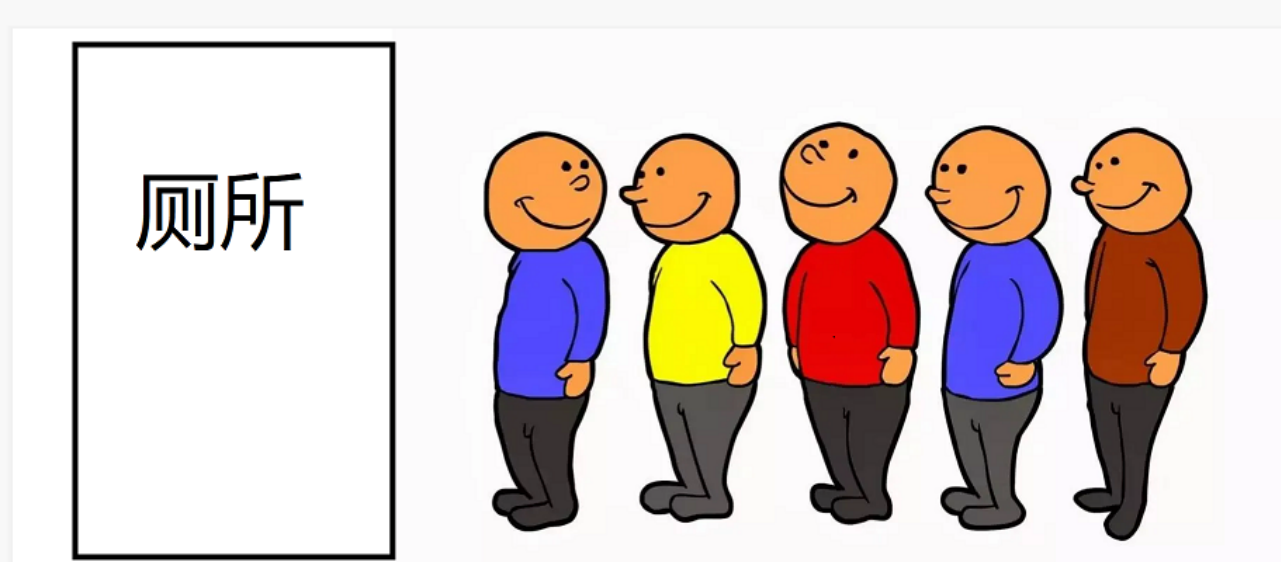
然后为了避免这种情况的出现，就有了互斥量的这个概念

贰：互斥量

互斥量的上锁机制

由于上面的那种情况导致我们的二值信号量，通常都是用来进行数据的同步一个负责发送数据，然后一个负责同步数据

而我们的互斥量，其实它更像一个上锁的机制，两个任务需要公用一个资源，所以我们可以把互斥量又叫做互斥锁



这里为了通俗易懂就找了一个上厕所的例子，给大家解释一下

厕所：任务公用的资源

小人：使用资源的任务

上锁：“厕所处于使用状态”

开锁：使用完了，处于空闲状态

然后我们去上卫生间的话，我们肯定会有一个上锁的操作，这样的话别人就不会闯进来了，当我们上完厕所之后，我们再将锁打开，这样的话才会有下一个人进入

互斥量的优先级继承机制

二值信号量：

我们高优先级的任务和低优先级的任务，他们两个有一个特殊的联系，就是他俩只有一个人带了“卫生纸”，所以尽管他优先级很高，但是没有卫生纸，他也还是上不了厕所，但是在FreeRTOS的世界里，低优先级的任务又会被高优先级的任务插队，只能等比他优先级高的任务上完厕所，自己才能进去，这导致明明优先级最高的“大哥”，等的时间更久

互斥量：

互斥量的出现就是去弥补二值信号量它的缺点，你想想你作为大哥，有人插你小弟的队，这不就等于是插你的队吗？大哥气不过，就和所有人宣布，我的小弟就是我，谁要敢插他的队我就打谁

叁：互斥量的基本函数：

互斥量系统默认是开启的也就是#define configUSE_MUTEXES 1

创建：

动态创建：

```
SemaphoreHandle_t    xSemaphoreCreateMutex( void );
//创建互斥量，成功的话返回句柄，失败返回NULL
```

静态创建：

```
SemaphoreHandle_t xSemaphoreCreateMutexStatic( StaticSemaphore_t
*pxMutexBuffer );
```

删除：

```
void vSemaphoreDelete( SemaphoreHandle_t xSemaphore );
//传入句柄即可删除
```

Give\Take:

！Take和Give需要成对使用

正常任务中使用：

```
BaseType_t xSemaphoreGive( SemaphoreHandle_t xSemaphore );
//传入句柄
BaseType_t xSemaphoreTake(SemaphoreHandle_t xSemaphore,TickType_t
xTicksToWait);
//传入句柄以及等待时间
```

中断中使用：

```
BaseType_t  xSemaphoreGiveFromISR(SemaphoreHandle_t xSemaphore,
                                  BaseType_t
*pxHigherPriorityTaskWoken);
BaseType_t  xSemaphoreGiveFromISR(SemaphoreHandle_t xSemaphore,
                                  BaseType_t
*pxHigherPriorityTaskWoken);
```

只有函数的创建是不一样的，删除以及其他操作基本上都是完全相同的，这里就不去，过多赘述，我们直接往下面看它的实战吧

肆：互斥量的使用

实验：上厕所模拟试验

同样是三个任务优先级，分别是从高到低，然后在任务一和任务三中使用我们的互斥量，然后观察它的执行顺序

现象：

我们的最高优先级和最低优先级的任务将会“连续执行”中间优先级的任务反而是最后执行

分析：

互斥量具有优先级继承的机制，在上锁期间其他的任务无法进行抢占