一：函数指针和指针函数

CH\_RTOS\_RESULT\_e CH\_RTOS\_CreateTask创建任务函数中

芯片厂家底层驱动函数BOOL mtos\_task\_create(u8 \*p\_taskname,

void (\*p\_taskproc)(void \*p\_param),//函数指针

void \*p\_param,

u32 nprio,

u32 \*pstk,

u32 nstksize);

形参：void (\*rpfn\_Function) (void \*)

函数指针，它的本质是一个指针，用来保存我们函数的入口地址。函数指针是指向函数的指针变量。

注：rpfn\_Function是一个指针变量，它指向的必定是rpfn\_Function函数的地址，rpfn\_Function函数的类型肯定是rpfn\_Function(void \*)类型。

函数指针变量rpfn\_Function，只能用来保存返回值是void，并且只有一个void \*参数的函数入口地址void rpfn\_Function(void \*)；

实参：test\_task\_prams

函数原型：static void test\_task\_prams(void \*p)，传参类型及个数、返回类型必须一一对应上面的函数指针

1.1回调函数是函数指针的一种应用，test\_task\_prams也是回调函数。

二：Linux任务调度和进程通信

1. 任务调度

1.1 任务（任务）

任务（task）指由软件完成的一个活动。一个任务既可以是一个进程,也可以是一个线程。例如,读取数据并将数据放入内存中。这个任务可以作为一个进程来实现,也可以作为一个线程（或作为一个中断任务）来实现。

任务的五种状态：

1>睡眠：任务只以代码的形式存在，没交给操作系统管理，即没有分配任务控制块和任务堆栈

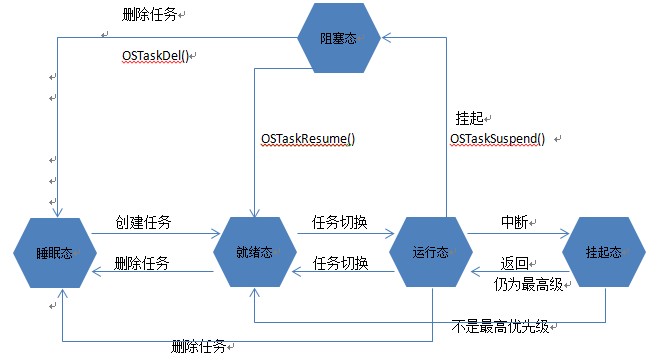
2>就绪：任务全副武装，并在就绪表中登了记

3>运行：获得了cup的使用权

4>等待（挂起）：正在运行的任务，需要等待一个事件的发生在运行，cpu使用权被剥夺，自己处于等待状态

5>中断服务：正在运行的任务一旦响应中断申请就会终止运行而去执行中断服务子程序（ISR），这时处于中断服务状态

任务转化图：



1.3 事件（event）

可以理解为对一个组件的某种同类型操作动作的集合。常见的如鼠标事件、键盘事件、窗口事件、选择事件等。

1.3.1 CH\_RTOS\_CreateEvent函数



传入参数：事件句柄

功能：创建一个事件 底层函数原型：





p\_event：事件指针

Retval：成功true，其他失败

1.3.2 CH\_RTOS\_DeleteEvent函数



传入参数：事件句柄

功能：删除一个事件 底层函数原型:





p\_event：事件指针

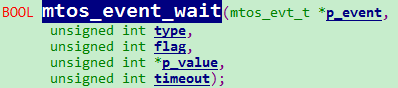
Retval：成功true，其他失败

1.3.3 CH\_RTOS\_WaitEvent函数



传入参数：事件句柄 要等待的事件，每一位代表一个事件 已经发生的事件 等待方式 超时时间

功能：等待事件 底层函数原型：





p\_event：事件指针

Type：当调用者指定多条件时，该标记用于确定是否需要满足所有条件

Flag：标记这是位掩码来声明一个或多个条件

P\_value：指向调用者的指针提供了空间，用于在条件满足时存储事件设置状态的快照

Timeout：用于定义最大时间的参数，在滴答声中，系统应该等待满足条件。使用mtoswait永远意味着永远等待

1.3 进程（process）

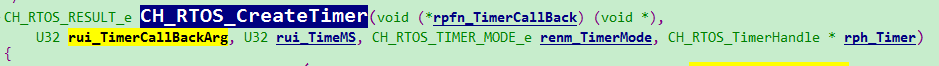
进程（process）常常被定义为程序的执行。可以把一个进程看成是一个独立的程序,在内存中有其完备的数据空间和代码空间。一个进程所拥有的数据和变量只属于它自己。

1.4 线程（pthread）

1.5 定时器（timer）

先创建create定时器，再开启start定时器

1.5.1 CH\_RTOS\_CreateTimer函数



传入参数：

rpfn\_TimerCallBack 定时器回调函数

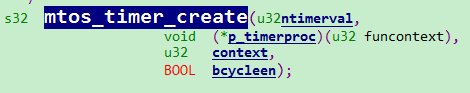
rui\_TimerCallBackArg 传递给回调函数的参数

rui\_TimeMS 定时器时间间隔,单位为ms

renm\_TimerMode 定时器工作模式(单次触发还是只要计时器没删除的情况下以指定的时间间隔触发)

rph\_Timer 创建成功后的定时器句柄

功能：创建一个定时器 底层函数原型：





Ntimerval：定时时间间隔

p\_timerproc：定时器回调函数

context the：回调函数的参数

Bcycleen：触发方式 FALSE: singal只触发一次, TRUE:repeat 每隔一段时间触发一次

创建成功，设置定时器触发模式，然后开启定时器，执行定时器回调函数。

1.5.2 CH\_RTOS\_StartTimer函数



传入参数：定时器句柄

功能：启动定时器 底层函数原型：





Ntimerid：定时器id号

1.5.3 CH\_RTOS\_StopTimer函数





功能：停止定时器

1.5.4 CH\_RTOS\_DeleteTimer函数





功能：删除定时器

2.进程通信：消息队列（message queue）

2.1 CH\_RTOS\_CreateMessageQuene函数

创建一个消息队列。芯片厂家提供的底层函数原型为：





Depth：可以容纳消息个数

P\_name：消息队列名称

Return value：一个的消息队列的标识符（非零整数）ui\_Id，失败时返回INVILD(~0).

2.2将数据缓冲区添加到消息队列中（消息队列支持额外的数据传输，但是必须首先附加外部缓冲区，is\_transition=1）





传入参数：

msgq\_id：消息队列标识符

p\_buf：存储数据的buffer缓冲区,大小为atom\_size\*depth（以字节为单位）

atom\_size：消息大小(字节为单位)

depth：可以容纳的消息个数

2.3 CH\_RTOS\_SendMessage函数



传入参数：消息队列句柄 要发送的消息 超时时间（单位ms）

功能：发送消息，底层函数原型如下：





msgq\_id：消息队列标识符

p\_msg：消息指针

Return value：成功true，错误fail

2.4 CH\_RTOS\_ReceiveMessage函数



传入参数：消息队列句柄 要发送的消息 超时时间（单位ms）

功能：接收消息，底层函数原型如下：





msgq\_id：消息队列标识符

p\_msg：消息指针

ms：超时时间（单位ms）

Return value：成功true，错误fail

三：semphore（信号量）

1. 信号量与互斥体的区别

1>根本区别：互斥量用于线程的互斥，信号量用于线程的同步。

互斥：是指某一资源同时只允许一个访问者对其进行访问，具有唯一性和排它性。但互斥无法限制访问者对资源的访问顺序，即访问是无序的。

同步：是指在互斥的基础上（大多数情况），通过其它机制实现访问者对资源的有序访问。在大多数情况下，同步已经实现了互斥，特别是所有写入资源的情况必定是互斥的。少数情况是指可以允许多个访问者同时访问资源。

note:信号量可以用来实现互斥量的功能

2>互斥量值只能为0/1，信号量值可以为非负整数。

也就是说，一个互斥量只能用于一个资源的互斥访问，它不能实现多个资源的多线程互斥问题。信号量可以实现多个同类资源的多线程互斥和同步。当信号量为单值信号量时，也可以完成一个资源的互斥访问。

3>互斥量的加锁和解锁必须由同一线程分别对应使用，信号量可以由一个线程释放，另一个线程得到。

注：二进制信号量(binary semaphore)：只允许信号量取0或1值，其同时只能被一个线程获取，这套代码就是使用二进制信号量。

3.1 CH\_RTOS\_CreateSemaphore函数



传入参数：信号量初始值 信号量句柄

功能：创建信号量 底层函数原型：





P\_sem：互斥体句柄

Mutex：true 互斥体，false 信号量

3.2 CH\_RTOS\_WaitSemaphore函数



传入参数：信号量句柄 超时时间（单位ms）

功能：等待信号量，成功时信号量减一 底层函数原型：



查询信号量计数，查询是否有任务挂起，是否有可用资源





p\_sem：互斥体句柄

ms：超时时间 0：永远等待

3.3 CH\_RTOS\_SignalSemaphore函数



传入参数：信号量句柄

功能：释放信号量,成功时信号量值加一 底层函数模型：





p\_sem：互斥体句柄

3.4 CH\_RTOS\_DeleteSemaphore函数



传入参数：信号量句柄

功能：删除信号量 底层函数原型：





P\_sem：信号量

Opt： opt == MTOS\_DEL\_NO\_PEND 只有在未完成任务时才删除信号量

opt == MTOS\_DEL\_ALWAYS 即使任务在等待，也要删除这个信号量，在这种情况下，所有正在等待的任务都将被重新执行。

四：mutex（互斥体）

1> 互斥体禁止多个任务同时访问受保护资源,在任意时刻，只有一个任务被允许进入受保护区域;

2> 当一个任务获取了某个资源的使用权限之后，除非该任务释放该权限，否则其他任务均无法获取该权限;

3> 统一任务已经对一个互斥体加锁后，可再次对其加锁，但多次加锁后需要多次释放才能释放该锁。

4> 资源释放后，其他等待该资源的任务的调度有两种方式:一种是按照先申请先得到的原则(FIFO)，另一种是按照任务的优先级进行调度(Priority)。

5> 可创建的最大互斥体的数量取决于平台及系统资源!

4.1 CH\_RTOS\_CreateMutex函数



传入参数：互斥体句柄 模式

功能：创建互斥量 底层函数原型：





P\_sem：互斥体

Mutex：true 互斥体，false 信号量

4.2 CH\_RTOS\_LockMutex函数



传入参数：互斥体句柄

功能：互斥体加锁 底层函数模型：





p\_sem：互斥体

ms：超时时间 0：永远等待

4.3 CH\_RTOS\_UnlockMutex函数



传入参数：互斥体句柄

功能：互斥体解锁 底层函数模型：





p\_sem：互斥体

4.4 CH\_RTOS\_TryLockMutex函数



4.5 CH\_RTOS\_DeleteMutex函数

