Pixhawk源码笔记一：APM代码基本结构

Pixhawk源码笔记二：APM线程

[](http://photo.blog.sina.com.cn/showpic.html#blogid=402c071e0102v5br&url=http://album.sina.com.cn/pic/001aRpVkgy6OgWhm7vnd5)

        Pixhawk源码笔记一：APM代码基本结构，参见：

<http://blog.sina.com.cn/s/blog_402c071e0102v59r.html>

        这里，我们对 APM 线程进行讲解。如有问题，可以交流[30175224@qq.com](mailto:30175224@qq.com)。新浪@WalkAnt，转载本博客文章，请注明出处，以便更大范围的交流，谢谢。

**第三部分 APM线程**

        详细参考：<http://dev.ardupilot.com/wiki/learning-ardupilot-threading/>

        对于APM1、APM2硬件板，不支持多线程，所以只能通过简单的定时器加回调函数来实现。类似PX4和Linux硬件板支持Posix标准的多线程。线程一般是指基于多任务操作系统的并行任务，我们首先要明白的几个概念如下：

        1、 定时回调

        2、 HAL专属线程

        3、 驱动专属线程

        4、 APM驱动与板级驱动

        5、 板级专属线程、任务

        6、 AP\_Scheduler任务调度系统

        7、 信号灯（任务队列互锁用）

        8、 lockless data structures

        如果你对操作系统运行机制比较了解，那就很好理解了。

**1、定时回调The timer callbacks**

        每个飞控平台都提供一个1kHz的定时器（见AP\_HAL），通过“注册”一个定时器函数来获取1kHz定时功能。所有注册的定时器将被顺序调用。调用形式如下：

        hal.scheduler->**register\_timer\_process**(AP\_HAL\_MEMBERPROC(&AP\_Baro\_MS5611::\_update));

        定时器优先级为181，高于主进程的180。上面代码是以MS5611气压计驱动为例，其中 AP\_HAL\_MEMBERPROC() 宏，主要作用是**将一个C++成员函数包装起来，作为一个回调参数**。其定义在**AP\_HAL\_Namespace.h**文件中，如下：

        // macro to hide the details of AP\_HAL::MemberProc

        #define **AP\_HAL\_MEMBERPROC**(func) fastdelegate::MakeDelegate(this, func)

        使用hal.scheduler->millis() and hal.scheduler->micros() 可以记录时间。

        好了，你可以试着自己边一个简单的sketch，在setup()和loop()函数中练习一下1秒钟向USB终端输出一个时间或字符。

**2、HAL专属线程**

        以PX4为例，HAL专属线程有：

        1、 UART线程，用于读、写串行接口数据（包括USB）；

        2、 定时器线程，支持1kHz定时功能；

        3、 IO线程，支持写microSD、EEPROM、FRAM等。

        对于Pixhawk，请准备一条调试电缆，连接到nsh console（serial 5 端口），波特率57600。如果已经连接，试下”ps”命令，你会得到如下信息：

        PID PRI SCHD TYPE NP STATE NAME

        0 0 FIFO TASK READY Idle Task()

        1 192 FIFO KTHREAD WAITSIG hpwork()

        2 50 FIFO KTHREAD WAITSIG lpwork()

        3 100 FIFO TASK RUNNING init()

        37 180 FIFO TASK WAITSEM AHRS\_Test()                                 AHRS线程

        38 181 FIFO PTHREAD WAITSEM (20005400)         定时器线程

        39 60 FIFO PTHREAD READY (20005400)                UART线程

        40 59 FIFO PTHREAD WAITSEM (20005400)           IO线程

        10 240 FIFO TASK WAITSEM px4io()

        13 100 FIFO TASK WAITSEM fmuservo()

        30 240 FIFO TASK WAITSEM uavcan()

        上面的线程为定时器线程（优先级181），UART线程(60)， IO线程(59)，以及其他线程诸如：px4io, fmuservo, uavcan, lpwork, hpwork and idle tasks

        线程的主要目的是在不干扰主进程的情况下，在后台处理一些低优先级任务。例如 AP\_Terrain library，需要向microSD卡写地形文件，它的实现方式如下：

        hal.scheduler->**register\_io\_process**(AP\_HAL\_MEMBERPROC(&AP\_Terrain::io\_timer));

        注意：IO线程优先级59，相比定时器181优先级慢了很多。

**3、Driver专属线程**

        没什么好说的，请参考英文原版，需要提的一点是，我们可以利用register\_io\_process() 和register\_timer\_process()来处理驱动的访问。

**4、APM驱动与板级（原生）驱动**

        我们可以看到MPU6000驱动有两个版本：一个是APM版本，在libraries/AP\_InertalSensor/AP\_InertialSensor\_MPU6000.cpp,另一个为原生代码版本，在PX4Firmware/src/drivers/mpu6000。

        注意，对于Pixhawk，APM代码使用的是Pixhawk原生驱动，因为原生驱动已经做得很好了。libraries/AP\_InertialSensor/AP\_InertialSensor\_PX4.cpp中可以查看详情。

        在非PX4平台上，我们使用AP\_InertialSensor\_MPU6000.cpp驱动，在PX4平台上，我们就用PX4原生驱动AP\_InertialSensor\_PX4.cpp

**5、板级专属线程、任务**

        在上面第2节“HAL专属线程”讲到”ps”命令显示的线程。很多都不是 AP\_HAL\_PX4 Schedule启动的线程，这些线程列举如下：

* idle task – called when there is nothing else to run
* init – used to start up the system
* px4io – handle the communication with the PX4IO co-processor
* hpwork – PX4稍低优先级驱动线程。handle thread based PX4 drivers (mainly I2C drivers)
* lpwork –PX4非常低优先级驱动线程。handle thread based low priority work (eg. IO)
* fmuservo – AUX输出。handle talking to the auxillary PWM outputs on the FMU
* uavcan – handle the uavcan CANBUS protocol

        这些任务的启动，由**rc.APM**脚本文件(ardupilot\mk\PX4\ROMFS\init.d\**rc.APM**)指定。PX4启动时，会读取该文件。rc.APM属于nsh 类型脚本。作为练习，你可以修改rc.APM脚本文件，增加一些sleep和echo命令，那么当PX4启动时，通过debug console（也就是serial 5）可以显示出来。

        更多内容，可以参考英文原版。

        原生线程的启动代码如下：

**hrt\_call\_every(&\_call, 1000, \_call\_interval, (hrt\_callout)&MPU6000::measure\_trampoline, this);**

        等同于AP\_HAL中的hal.scheduler->register\_timer\_process()。上述代码的意思是，HRT （high resolution timer）高精度定时器，以1000微妙的周期调用MPU6000::measure\_trampoline函数。这些操作是禁止中断的，最多占用数十微妙的时间。

        上面的优先级非常高。下面的方法，是稍低优先级。

        work\_queue(**HPWORK**, &\_work, (worker\_t)&HMC5883::cycle\_trampoline, this, 1);

        用于处理I2C设备。大概花几百微妙的操作时间。是可以被中断的任务。如果是最低优先级，那么参数改为**LPWORK**，这样的任务一般需要花费更长的时间。

**6、AP\_Scheduler任务调度系统**

        用于飞行器主线程，提供了简单的机制控制每个操作花费了多少时间。例如：1、等待一个新IMU采样；2、在每一个IMU采样周期之间调用一系列其他任务。

        每一个飞行器都有一个AP\_Scheduler::Task table任务列表，参考代码（ardupilot\libraries\AP\_Scheduler\ [Scheduler\_test.pde](https://github.com/diydrones/ardupilot/blob/master/libraries/AP_Scheduler/examples/Scheduler_test/Scheduler_test.pde) ）类似如下：

                static const AP\_Scheduler::Task scheduler\_tasks[] PROGMEM = {

                { ins\_update, 1, 1000 },

                { one\_hz\_print, 50, 1000 },

                { five\_second\_call, 250, 1800 },

                };

        结构体第1列，循环调用的任务函数。第2列，调用频率（也叫tick，一个tick，就是一个最小时间单元，pixhawk为2.5ms）。第3列为最大可能占用的操作时间，scheduler.run()会传递当前可用的时间（微秒）,如果时间不够，那么这个任务就pass掉了，不执行。

        注意，AP\_Scheduler::Task table列表必须具备以下条件：

        1、 他们不能被阻塞。

        2、 在飞行时，他们不能调用sleep function

        3、 他们必须有可预估的最坏的运行时间。

        你可以修改[Scheduler\_test.pde](https://github.com/diydrones/ardupilot/blob/master/libraries/AP_Scheduler/examples/Scheduler_test/Scheduler_test.pde)，加入自己的代码来读取气压计、罗盘、GPS、更新AHRS输出roll/pitch。

**7、信号灯**

        有3种方法可以避免多线程访问冲突：1、信号灯；2、lockless data；3、PX4 ORB。

        例如：I2C驱动可以通过信号灯，确保同一时间，只有一个I2C设备被使用。可以查看ardupilot\libraries\AP\_Compass\AP\_Compass\_HMC5843.cpp了解：

        获得信号灯：\_i2c\_sem->take(1);

        释放信号灯：\_i2c\_sem->give();

**8、Lockless Data Structures**

        Lockless Data Structures比信号灯要方便，例子见：

* the \_shared\_data structure in libraries/AP\_InertialSensor/AP\_InertialSensor\_MPU9250.cpp
* the ring buffers used in numerous places. A good example is libraries/DataFlash/DataFlash\_File.cpp

        Go and have a look at these two examples, and prove to yourself that they are safe for concurrent access. For DataFlash\_File look at the use of the \_writebuf\_head and \_writebuf\_tail variables.

**9、PX4 ORB**

        ORB(Object Request Broker)是PX4的互斥机制。

        另外两种PX4驱动通信机制，列举如下：

* **ioctl** calls (see the examples in AP\_HAL\_PX4/RCOutput.cpp)
* /dev/xxx **read/write** calls (see \_timer\_tick in AP\_HAL\_PX4/RCOutput.cpp)

        想要学习Pixhawk源码的朋友有福了，后边我会陆续的将Pixhawk的源码学习笔记整理出来分享给大家。敬请关注：新浪微博@WalkAnt，[3017224@qq.com](mailto:3017224@qq.com)。欢迎交流。