emwin 2-day quick tutorial 005_emWin 的工作原理以及 GUI_Exec 函数

看这篇文章,需要 emWin 一定的基础,如果你还不会用 emWin,请先看前面的教程.

emWin 整个工作流程,其实就和几个东西有关:GUI_Init()、GUI_Delay()、GUI_Exec()、回调函数、窗体创建函数、窗体/控件的子父关系、PID 输入设备(比如键盘、鼠标、触摸屏等等).

GUI_Init()是 emWin 的初始化,在初始化之前都不能执行任何和 emWin 相关的操作.在 STemWin 中,由于 ST 在 emWin 中加入了加密校验,限制 STemWin 只能用于 STM32,因此在 GUI_Init()之前必须使能 CRC,否则初始化失败.

可以说 emWin 所有动作都靠循环执行 GUI_Delay()或 GUI_Exec(),两个函数没有本质区别,GUI_Exec()包含在 GUI_Delay()中,即 GUI_Delay()在查询 OS_TimeMS 变量(通过一个硬件定时器每 1ms 中断加 1)实现延时的时候不断的去调用 GUI_Exec(),区别在于 GUI_Delay()带延时 1ms 功能,而 GUI_Exec()不带延时,但两个函数作用是一样的.emWin 的应用程序主函数一般是这样的:

```
void MainTask(void)
{
  GUI_Init();
  创建各个窗体;
  while(1) { GUI_Delay(xx);或 GUI_Exec();}
}
```

即利用不断的循环执行 GUI_Delay(xx);或 GUI_Exec();带动 emWin 所有的东西,包括:界面有改变时刷新界面,获取触摸屏、鼠标、键盘等 PID 输入设备,执行用户程序中的回调函数,反正就是带动了所有的事情,其实就是 emWin 的动作引擎.

GUI_Exec()首先会去查询界面有没有改变,如果什么都没有改变,是不会去刷界面,比如一个静态界面放在那里不去动它,这个时候 emWin 并没有执行和显示相关的东西.这就是为什么我们要在窗口上显示一些文字(是单纯显示文字而不是用 Text 控件或控件上的文字)或绘制一条直线,当我们在程序的另一处改变文字内容,但改变永远不会显示出来,因为这些是 2D 绘图而不是控件,emWin 是不会感知到改变的(控件就会,emWin 内部自动管理),所以要用 WM_InvalidateArea()/ WM_InvalidateRect()/ WM_InvalidateWindow()函数使那个区域/或整个控件无效之后 emWin 才感知到那个区域有变化而去重绘那个区域.还有就是我们在程序里改变了界面某个东西,而这个改变是不会马上显示出来的,要等 emWin 下一次执行 GUI_Exec()才会显示出来.

在每次执行 GUI_Exec()时,都会调用用户界面程序中的回调函数,包括所有窗体(显示和隐藏)的回调函数都会进入一遍,看到这里你应该知道在回调函数里能做什么事情了吧?所有事情都能做.

窗体创建时 emWin 会在 RAM 中开辟空间储存这个窗体所有控件的信息,比如位置、大小、ID、文字信息等待反正一大堆乱七八糟的东西,不过没有储存像素信息,因此都是小数据,像素显示是在界面刷新时才由这些被储存信息绘制出来的.还有创建窗体之后,会返回一个句柄,其实就是一个 ID,有了这个句柄,我们就可以找到这个窗体以及该窗体中任何的东西.

窗体创建时,其中一个参数是指定回调函数,所以每个窗体都会对应有一个回调函数,其实不单单是窗体才可以有回调函数,具体到任何的控件(比如按钮)都可以给它指定一个回调函数,在这个回调函数当中可以做一些和该控件相关的事情,只是一般没必要而已.

窗体创建时有些控件占用 RAM 控件是固定的,比如按钮等,但有些控件占用 RAM 是动态的,比如 ListView 列表控件,你每增加一行 emWin 会另开销一点 RAM 空间来储存这一行的信息,所以在做界面程序的时候实时监控 RAM 空间是很有必要的,怎么做?请看源码.

到这里你应该明白一个界面程序控件越多,RAM 的需求越大,当你一个界面有几十个几百个窗口时怎么办?这个时候你就要考虑一个窗口需要时才创建,不需要时用 WM_DeleteWindow()把它删除,经过这样处理之后,其实任何时刻同时存在的窗体也就只有几个而已(你不可能某一时刻要同时用到几十个窗口吧?),如果这样做的话,对于 MCU 来说,一般界面 emWin 部分分配 36Kbytes RAM 就完全足够.只是有三点一定一定要注意:(1)创建窗体前先检查句柄是否为 0 (2)GUI_EndDialog()结束(删除)窗体后将句柄请 0;为什么要这样做?有时候由于某些情况(疏忽或客观因素)会在窗体没有删除的情况下又重复去执行该窗体创建函数.(3)密切关注 RAM 的占用情况,一旦发现删除窗体时没有完全释放 RAM,或在某种操作下 RAM 不断增加,可以及时改正回来.

在 emWin 中所有的窗口都是连带关系的(没有单独独立的窗口),如果一定要说一个窗口是独立的,那只能说它的父窗

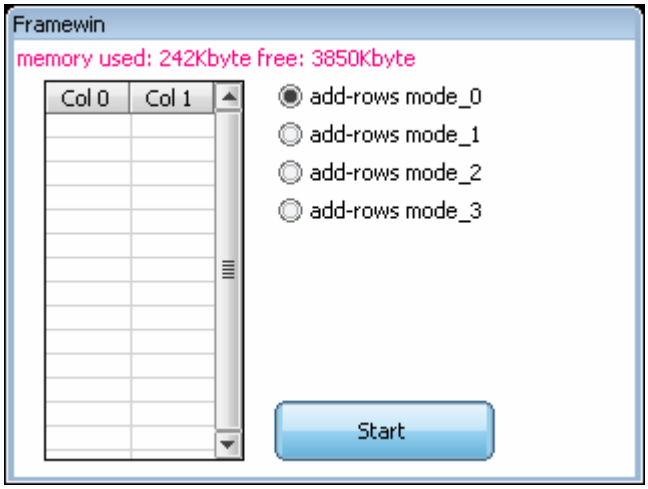
口就是桌面背景(如果两个窗体的父窗口都是桌面背景,那么他们是没有联系的).有了这个连带关系,我们可以找到任何子窗口或父窗口,还有点击界面时也不会乱套掉,比如不用担心点了这个窗口,另一个窗口跑到下面去了,因为子窗口永远显示在父窗口之上,隐藏或删除父窗口会连同子窗口一起隐藏或删除的,管理起来是不是很方便?

PID 输入设备(比如键盘、鼠标、触摸屏等等)是怎样和 emWin 关联起来的呢?其实就是在调用 GUI_Exec()时,在某个地方调用了 GUI_StoreKeyMsg()、GUI_TOUCH_StoreState()函数把键值、触摸坐标储存给 emWin 内部就可以了,emWin 内部底层怎么处理是它的事,跟我们没有关系.还有 emWin 已经集成了触摸屏程序和触摸校准程序,其实就是方便用户提供了一个触摸屏程序模板而已,你也可以用它现成的也可以自己写,用的话就按照它的方法修改下怎样获取触摸 AD 值,但它的程序跑来跑去最终还是离不开 GUI_TOUCH_StoreState()函数.

说了那么多,大家看的有点累,在以下实例中,通过程序演示更加深刻的说明 emWin 引擎函数 GUI_Exec()的作用和注意事项:

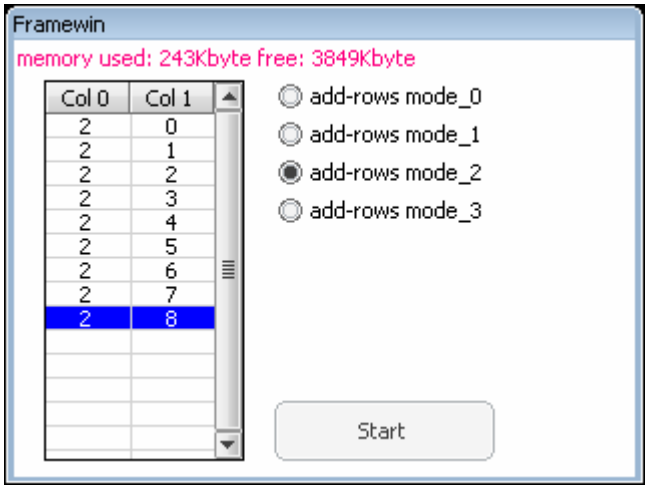
总共做了 4 种方式去更新列表控件(为什么用列表控件做例子?因为直观),在程序中还用 user_code_delay()函数去模拟有时候用户程序遇到大延时的情况(比如读 FLASH 数据):

(1)mode_0:是规范的做法,在回调函数中模拟大延时的情况,然后增加一行列表,如此重复 100 次完毕;在实际应用中,这种方式显示体验较差,因为等待时没有调用 GUI_Exec()函数及时刷新界面,按下的按键没有及时弹起(看下图),不知道的人还以为是死机了;还有一个致命的问题是,如果触摸屏是用中断而不是查询的方式处理的,在等待的时候不断去点击其他东西,这些点击是有效的,只是界面还没反应过来而已.

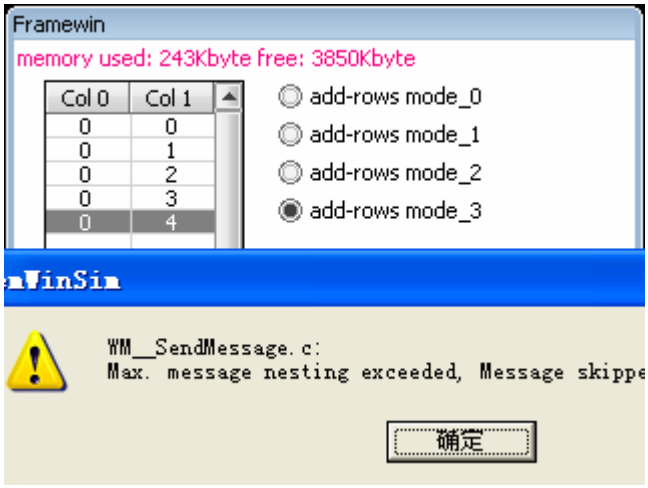


(2)mode_1:加入了 GUI_Exec()函数,列表每增加一行用 GUI_Exec()去更新(刷新)一次界面,这样在等待的过程中界面是实时更新的,但由于在回调函数中调用了 GUI_Exec(),而在 GUI_Exec()中又去重复执行该回调函数,什么风险?搞不好会无限嵌套!那怎么办?想办法不让程序又跑到该消息中去(回调函数是重复执行了,但没重复进入该消息),即让程序不会重复再跑到该点;怎么实现?调用 GUI_Exec()之前禁止掉该窗口的所有控件.再者,还会出现 mode_1 中程序等待时乱点屏幕而点击有效吗?不会,因为不断调用了 GUI_Exec()函数去处理(消除)这些触摸事件,而处理(消除)时所有东西都是禁止无效的.

(3)mode_2:和 mode_1 基本一样,唯一区别是每添加完 8 行列表之后才 GUI_Exec()刷新屏幕,笔者觉得这种方式是最好的,为什么?记住有这个思想就行,以后碰到一行刷一次反应慢时就会想起.



(4)mode_3:和 mode_1 有点相似,做个实验而已.在调用 GUI_Exec()之前没有禁止掉所有控件,这样在执行等待的时候不断的去点击 Start 按钮,程序很快就会因为嵌套而死掉.



总结:

当你发现由于某些阻塞事件造成了界面没有实时更新(坏处是显示体验差,界面会记住你的乱点击),这个时候就在阻塞事件里面(是里面)插入 GUI_Exec()以保证界面实时刷新,以及及时处理(消除)掉触摸事件,甚至还可以在程序等待时做个提示窗口(因为有 GUI_Exec(),该提示是可以实时显示的),只不过在 GUI_Exec()之前记得禁止掉该画面所有能点击的东西.

还有,我们作为程序员,要尽量避免大的阻塞事件,比如我们需要读 10Mbytes 的 Flash 数据,是否可以一次一个轮回读 1024Kbytes 呢?所以我们程序员的编程思想也很重要!