数据的安全访问和避免随机错误技术

一.临界代码段

在多任务系统中 CPU 执行这段代码期间不会被中断的代码区域叫临界代码端，也叫临界区。当我们的某一操作不想被中断时应进入临界区，例如两个任务都会改写某一共享内存时这两个任务的改写部分应在临界区操作，以避免写操作发生冲突。临界区的操作也叫原子操作，形象的说明此段程序是在处理器不可分割的时间段内得到执行。

在 avr-gcc 中进入和退出临界区的一般方法为 cli()和 sei()这两个函数，通常它们是配对使用的，如：

```
TaskLow(void) //低优先级任务
{
   …… //非临界代码
   cli();
   …. //临界代码
   sei();
   ……  //非临界代码
}
```

二．任务(Task)

任务看似专属于操作系统的概念，实际上一般前后台系统中主程序的中断分别是个不同的任务，为此临界代码段的概念仍适用于前后台系统。

任务通过进入临界区来暂时提升自身的优先级避免当前的操作被更高优先级的任务所打断而造成数据的访问错误，为此进入临界区往往是低优先级任务的行为。进入临界区的低优先级任务应尽快的退出临界区，否则会加长最大中断延迟时间，使系统的实时响应能力降低。下面举例一个例子：

有一个全局变量 uint8_t g_Flag; ，主程序根据某信号对其低一位进行设置，操作代码为： g_Flag|=_BV(0); ，而中断程序中根据另一条件对其进行这样的操作：g_Flag|=_BV(1)，即设置其第二位。而这样的“读－修改－写”操作是无法在一个指令完成的。当主程序检测到信号改写变量，先读出值修改后还没有执行写操作时产生中断，中断也执行了“读－修改－写”操作，中断退出后主程序继续执行刚才的写操作，覆盖了中断中操作的内容，从而本次中断失效。为了避免这样的情况主程序应在执行“读－修改－写”操作之前进入临界区，使操作期间关闭中断。

三．中断嵌套

在 avr-gcc 中默认情况下进入中断程序后先关闭全局中断，使得整个中断程序变成一个临界区，这意味着中断是不能嵌套的。尽管通过某些设置可以改变这样的情况，但建议你的程序适应这种结构，即不要使用中断嵌套。因为中断嵌套使程序变得更复杂，并具有堆栈溢出的风险。

但要注意的是，在中断不能嵌套的系统中，中断程序中应避免进入和退出临界区，因为整个中断程序都已经是个临界代码，在这里调用进入临界区的代码并不可怕，可怕的是配对使用的退出临界区操作往往会导致允许中断嵌套。

四．一种不需要进入临界区的情况

当任务间共享的数据为 8 位数据，并且在高优先级任务（如中断）中执行写操作，在低优先级任务中执行读操作的情况下不需要进入临界段。这是由于 8 位变量的读写操作在 AVR 上是单指令就能完成，便不存在被中断的情况。

然而数据为 16（或更高）位时情况就不同了，它们的读操作需要两个或两个以上的指令周期，当低优先级任务读到一半时可能发生中断并改变另一半的值，使得读出来的数据错误，便有可能导致系统的误动作。最可怕的是这种情况往往不是每次发生，这便是在调试时正常的系统在长时间运行时出现故障的原因之一。

五．该使用操作系统提供的信号量还是使用临界区？

操作系统（例如 AVRX）可以提供不同任务同步访问共享内存的机制，例如互斥信号量就可以保证不同任务访问数据时的安全性，这种情况下用户代码的执行不需要进入临界区。下面表格列出了信号量和临界区两种机制的优缺点：

安全机制	互斥信号量	临界区
优点	用户代码不在临界区执行	CPU 资源消耗较少
不足	CPU 资源消耗较大	有可能影响中断延迟时间

根据以上特点，使用互斥信号量的场合应该是大批数据的同步访问情况，而使用临界区代码的场合是少量（通常一个或几个变量）数据的同步访问情况。

六．结束语

数据的安全访问不仅涉及基于操作系统的应用，同样涉及主程序/中断结构的应用。程序初步勾勒时应当充分考虑数据安全访问问题，使编出来的程序严禁有序，不易产生错误。以上观点是作者编制 AVR 程序时总结的心得，并不一定完全符合其它系统（如 ARM,或其它 CPU 平台）,但仍然相信具有参考价值。

芯艺

2009-5-17