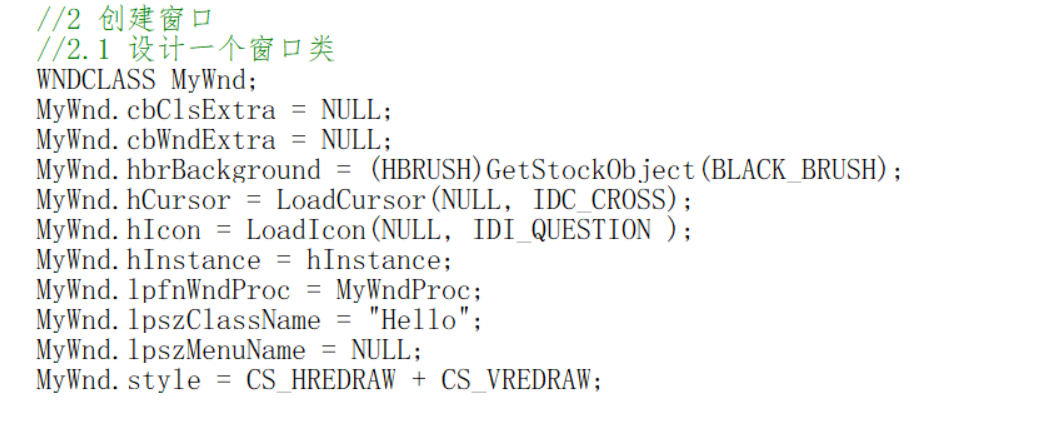
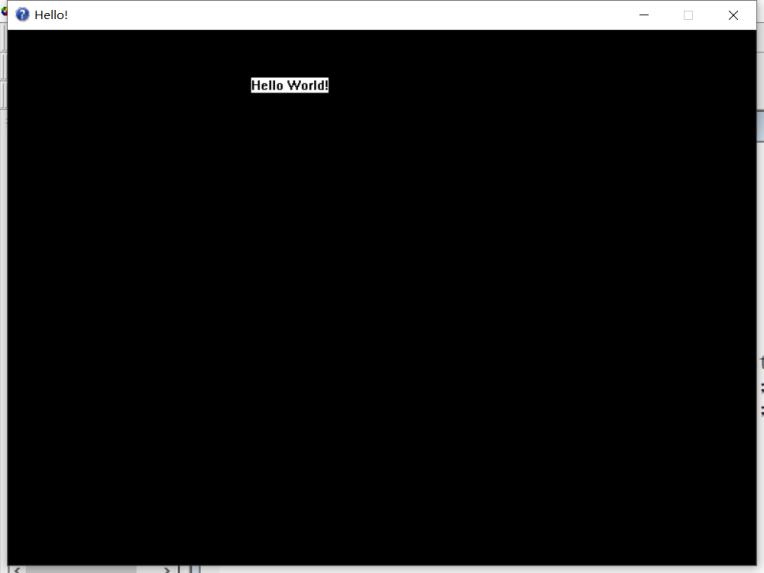
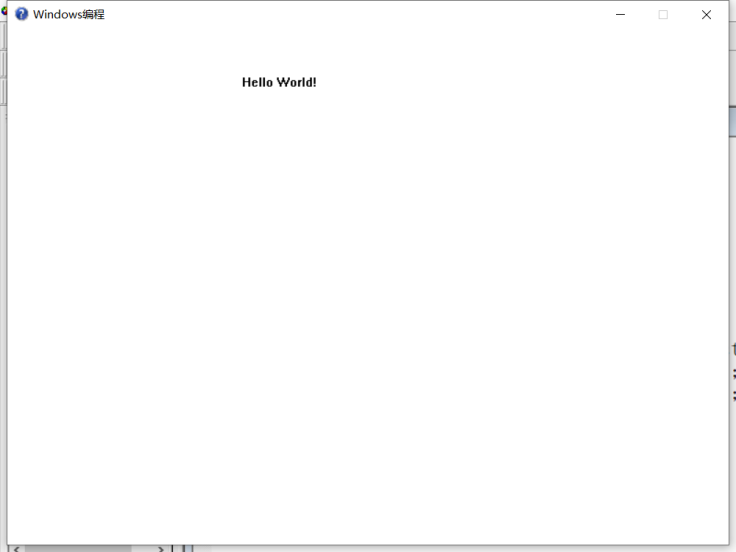
1. 创建窗口

·更改设计窗口类代码



（1）更改background的颜色，窗口背景从黑色变为白色

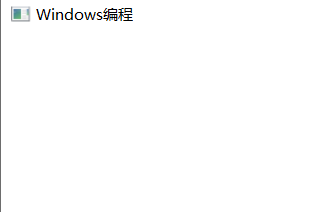
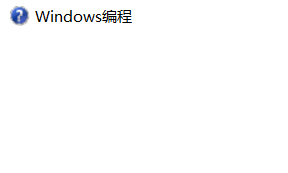




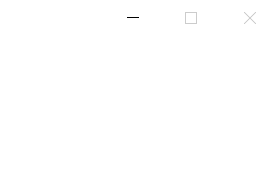
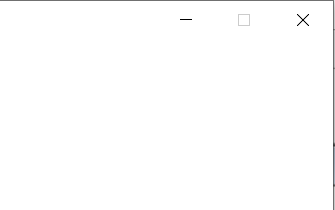
1. 更改hCursor的样式，鼠标从十字形变为箭头



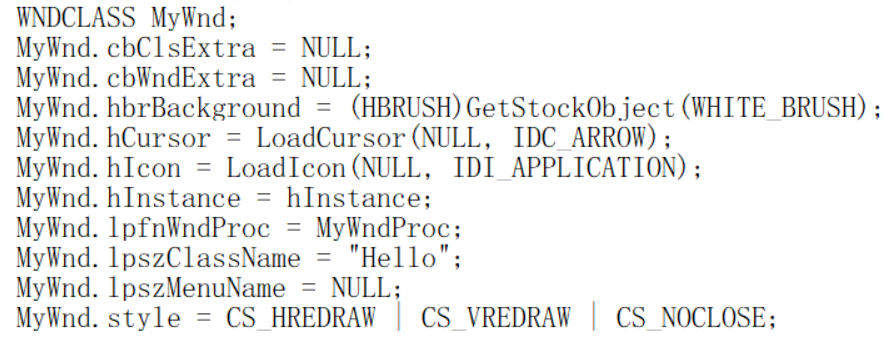
（3）更改hIcon的样式，窗口图标从问号变成应用图标



1. 改变窗口样式，窗口右上角的关闭的❌号变为不可选中



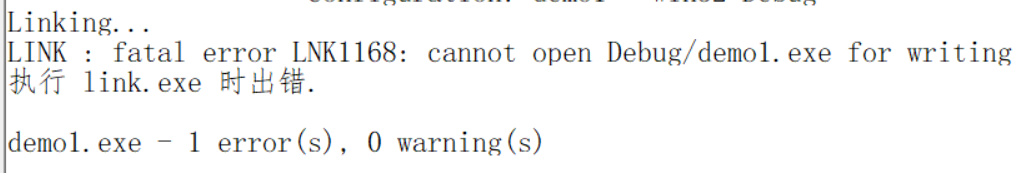
修改后的代码



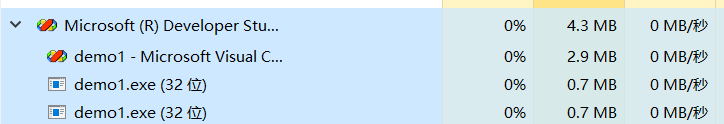
·更改注册窗口类的代码



运行一次发现没有反应，编译运行第二次发现报错



打开任务管理器

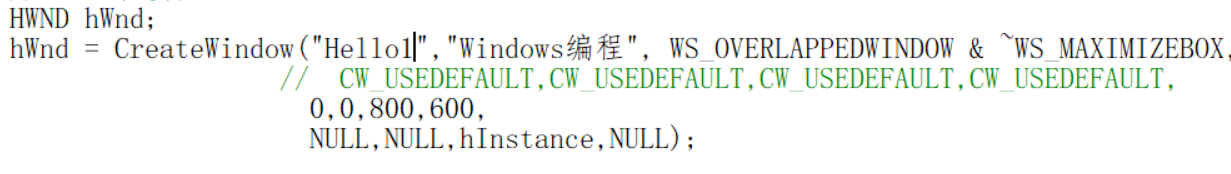


发现刚刚启动的程序仍在运行，自然就无法再次编译运行了

分析：没有注册窗口类，CreateWindow就无法创建窗口，我们无法

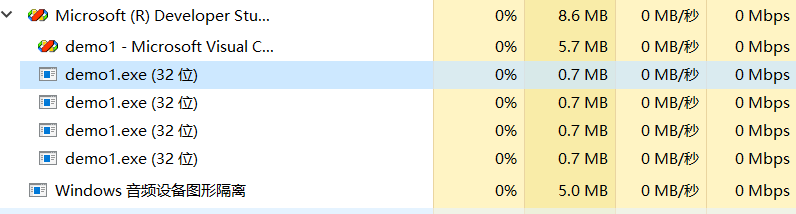
与窗口进行交互，无法产生更多的消息到消息队列中，消息队列很快就为空了；同时我们也无法发送quit消息，那么GetMessage就会处于阻塞状态，该程序不会停止，只占用极少的cpu和内存资源。

·更改创建窗口的代码



（1）在这里我故意将Hello改为Hello1，这样的话，CreateWindow就不能创建出一个窗口，因为根本没有名称Hello1这样的窗口类

点击几次运行，发现没有反应，打开任务管理器

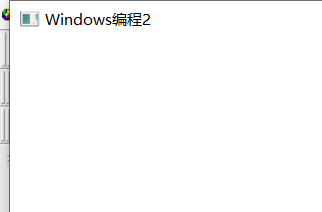
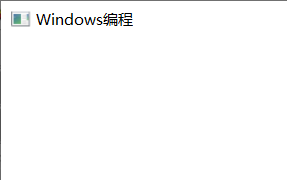


发现已经有好几个同样类型的进程在里面了，这个原因应该和刚刚没有注册窗口类的原因一样。

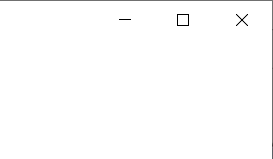
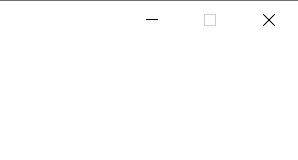
在这里，我们可以得出一个结论，当在创建窗口CreateWindow这一步错误，导致不能创建窗口时，这个进程最后都会进入阻塞状态。

GetMessage由于没有新的消息进来会一直等待。我们无法和窗口进行交互产生消息，无法产生close消息导致无法产生destroy消息最终导致无法产生quit消息，导致了这个进程一直都不能结束。

（2）改变窗口的名称，名称从”Windows编程”变为”windows编程2

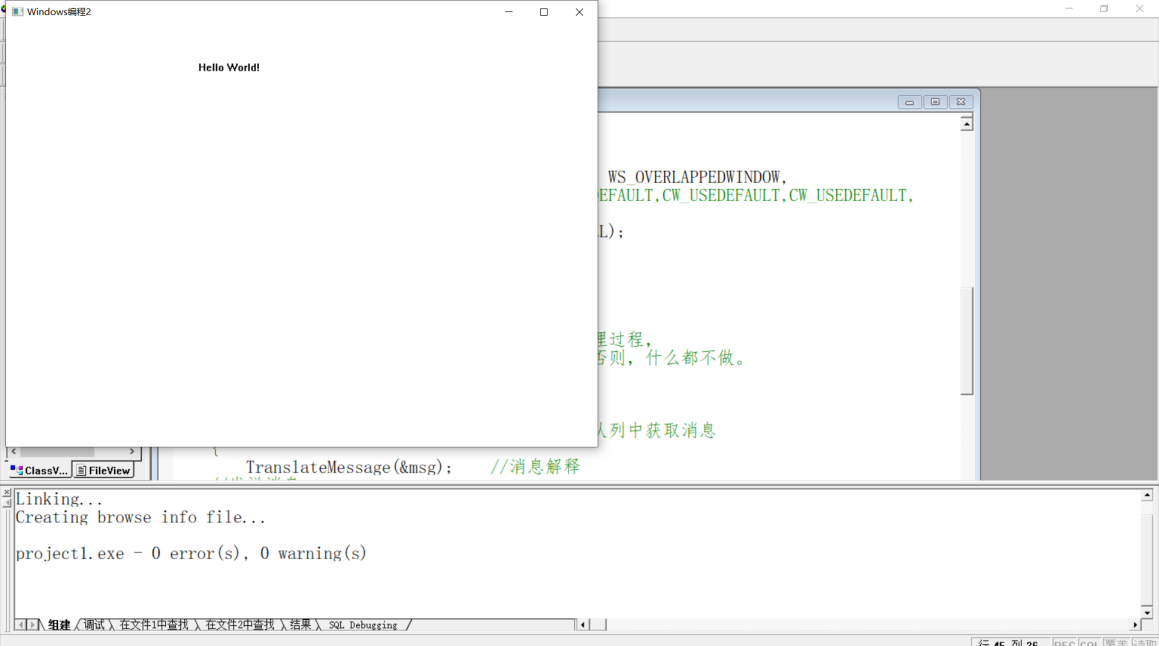


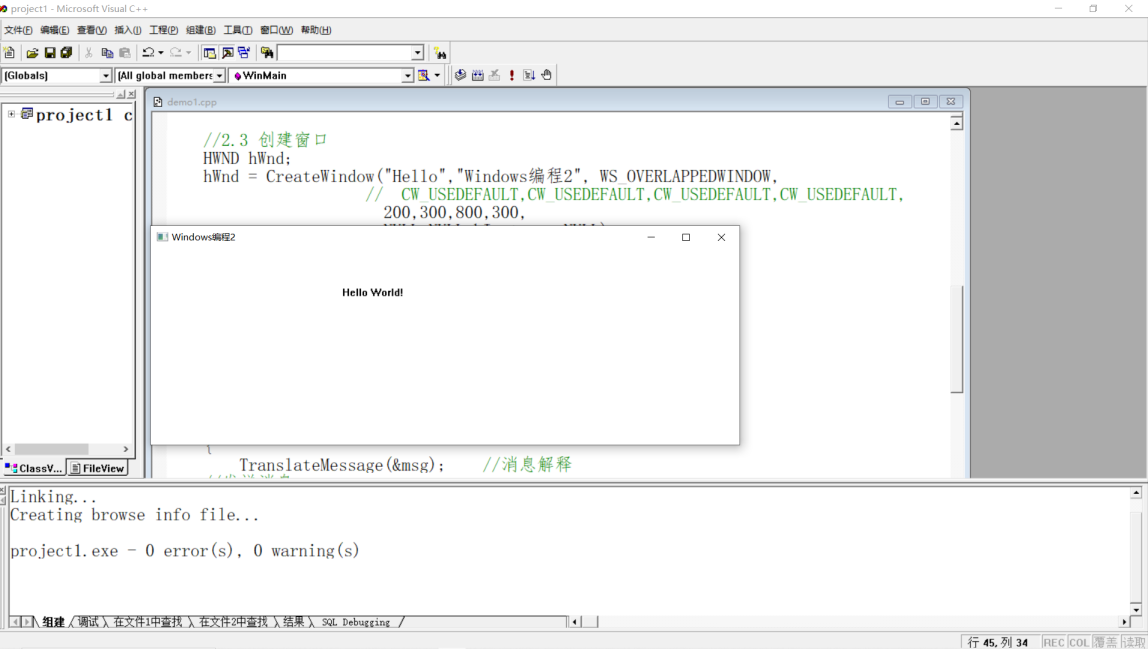
1. 更改窗口样式，最大化的按钮从无效变为有效



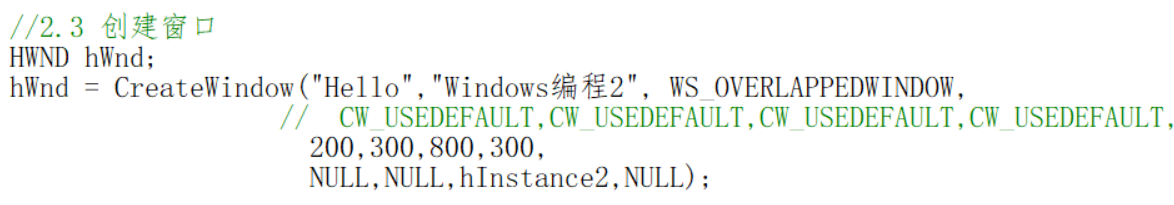
对窗口类样式和窗口样式的区分：窗口类样式是对这类窗口总的一个设置，而窗口样式则是窗口之间的个性化之分。

（3）更改窗口坐标和大小，这里需要注意的是窗口坐标的原点是在我们桌面的左上角，窗口大小的单位是像素

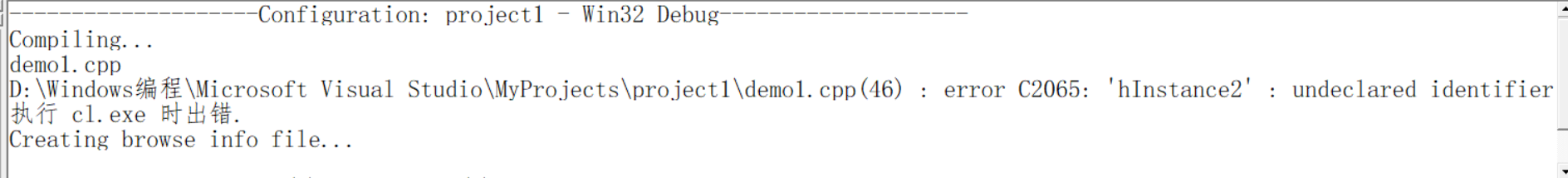




1. 修改窗口程序的实例句柄，将hInstance改成hInstance2



发现直接报错



这和我们刚刚的结论不一致，进程没有进入阻塞状态，而是直接报错不能运行了。我们分析，窗口是依附在这个窗口实例上的，当窗口句柄错误了，窗口找不到这个实例，窗口即使可以创建出来，但是实例不存在那也没有任何意义。也就是说，窗口不存在，但是实例存在并且找到了，进程进入阻塞；窗口其他创建信息正确，实例句柄错误，找不到实例，那会直接不能运行，进程也会结束了。

UpdateWindow(hWnd);

//如果有无效区，则马上发送WM\_PAINT到窗口处理过程，

//不进消息队列进行排队等待，立即刷新窗口，否则，什么都不做。

1. 消息循环

GetMessage(&msg,NULL,0,0)，对于GetMessage()的介绍

·第一个参数指向一个消息结构体，GetMessage从线程的消息队列中获取的消息信息将保存在这个结构体中。

·第二个参数指定接收属于哪一个窗口的消息。通常设置为NULL，表示接收属于调用进程的所有窗口的窗口信息。

·第三个参数指定获取消息的最小值

·第四个参数表示获取消息的最大值

·如果第三个参数和第四个参数都设置为0，表示接收所有消息。

·GetMessage从消息队列中获取一个消息。

·当消息队列为空时，GetMessage函数进入阻塞态。

·当正确获得消息(除了quit消息），返回1

·当获取了quit消息，返回0，结束消息循环

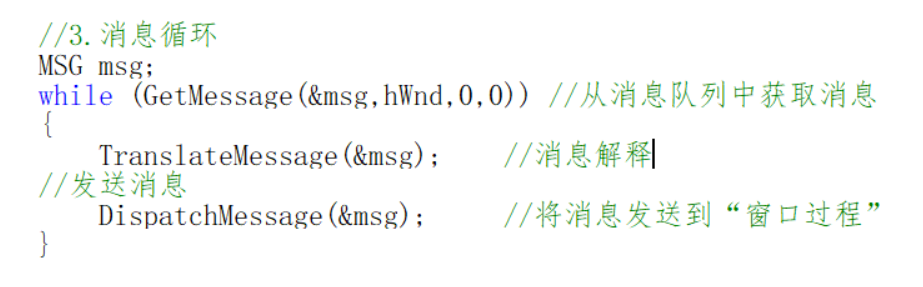
·当不能正确获取消息，比如第二个参数为无效的窗口句柄时，

或者&Msg为一个无效的指针，则返回-1

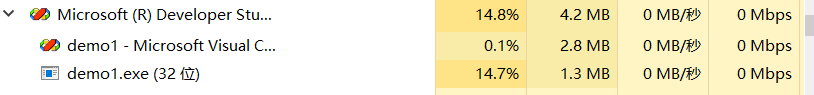
TranslateMessage函数用于将虚拟键消息转换为字符消息。

DispatchMessage函数分派一个函数到窗口过程。

将GetMessage函数的第二个参数改成是指向当前窗口的句柄



关闭窗口以后，打开任务管理器，发现进程没有结束



原因分析：当关闭窗口后，hwnd变为无效的窗口句柄，GetMessage函数一直返回-1，进入死循环，cpu占用率变高。那么为什么设置为NULL的时候就可以在关闭窗口后结束进程呢，因为在关闭窗口后，我们后续的代码会发送一个quit消息，但是这个消息是由这个应用程序的进程发送的，不是由窗口发送的。换句话来说，quit不属于窗口消息，GetMessage就不会得到这个quit消息，就无法退出循环了。

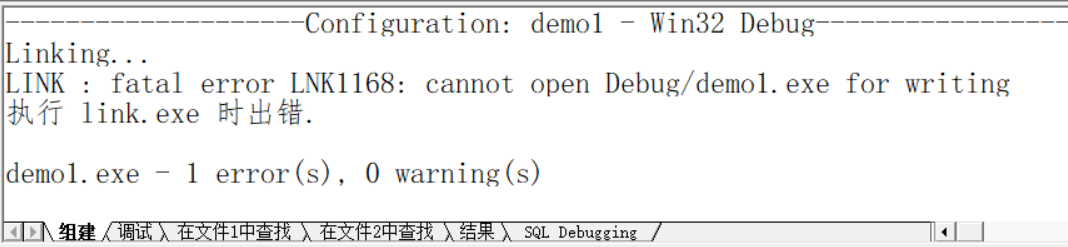
1. 窗口过程函数

将窗口过程函数指定返回0，也就是没有所谓的“大脑”，窗口过程函数中只有一句return 0;



第一次编译运行，没有窗口产生，没有任何反应；

第二次编译运行，直接报错；



打开任务管理器，发现这个程序还在运行

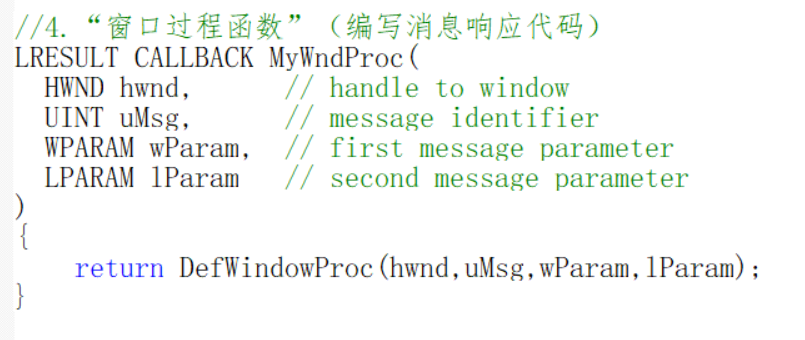


程序还在运行，自然无法再次编译运行了。

原因分析：

CreateWindow函数创建窗口时，会发送一个Create消息，如果窗口过程函数没有正确响应它，返回的句柄为空那么窗口就无法创建出来；这里的窗口过程函数直接return 0，这个响应不是正确的。但是应用程序的进程仍在继续，和前面一样，这个进程会卡在GetMessage函数中，一直处于阻塞态，我们要在任务管理器中才能关闭它。

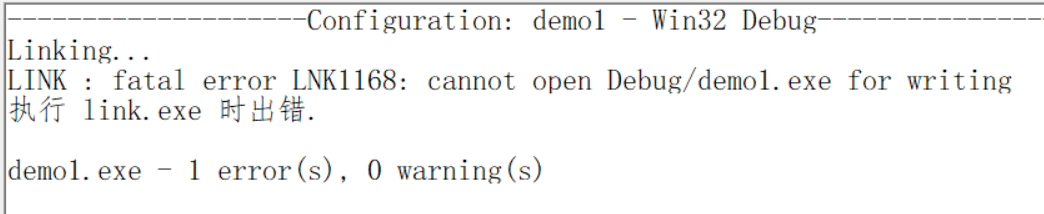
尝试只使用默认的大脑，窗口过程函数使用默认的函数



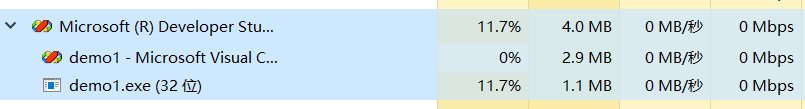
运行产生了一个空白的窗口



将窗口关闭后，再次编译运行，发生错误



老错误了，打开任务管理器，发现该进程没有结束。

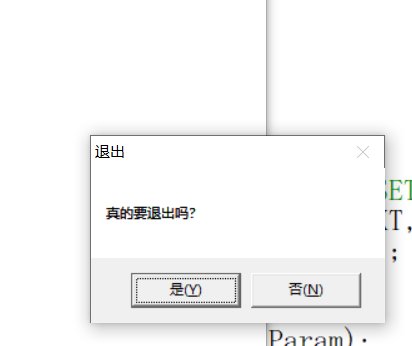
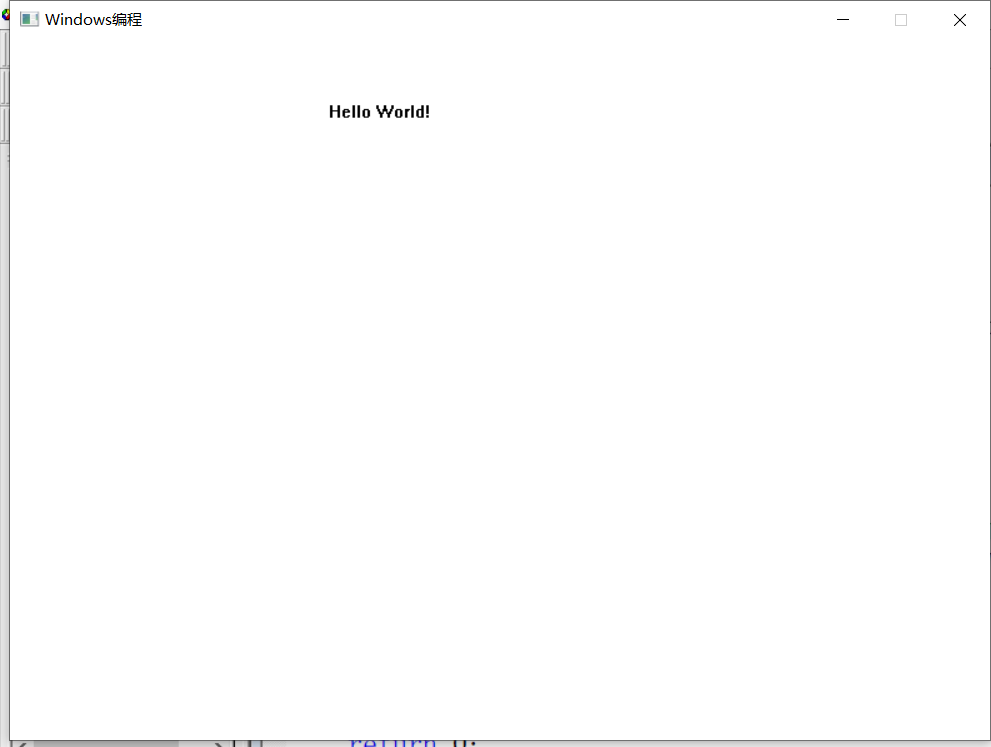


原因分析：这次的情况和之前的情况的原因是有相似性的。

都是由于没有收到quit消息而导致GetMessage函数一直在运行或者是处于阻塞态，导致进程无法结束。

但不同的是：

之前的情况是由于我们连窗口都没创建出来，导致我们不能和窗口进行交互从而无法产生更多的消息来导致quit消息的产生。而这次我们是将窗口创建出来的，但是由于默认的大脑，DefWindowProc函数是无法发送quit消息的，这也合情合理，一个应用程序的结束应该由程序员自己决定的，而不是由其他人决定的。

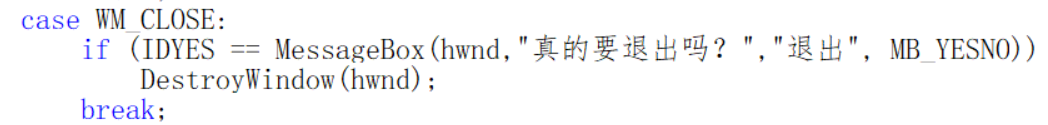
将窗口过程函数更改为正常的代码，创建窗口

关闭窗口再次编译运行，依然能够正常运行，这说明关闭窗口后，

我们刚刚的那个进程是的的确确关闭的，分析一下这个过程

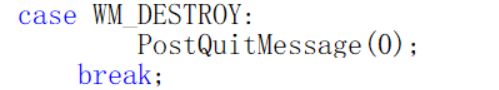
1. 按下窗口的关闭按钮，发送一条close消息到消息队列，

GetMessage函数会取到这条消息，并在窗口过程函数中处理它。

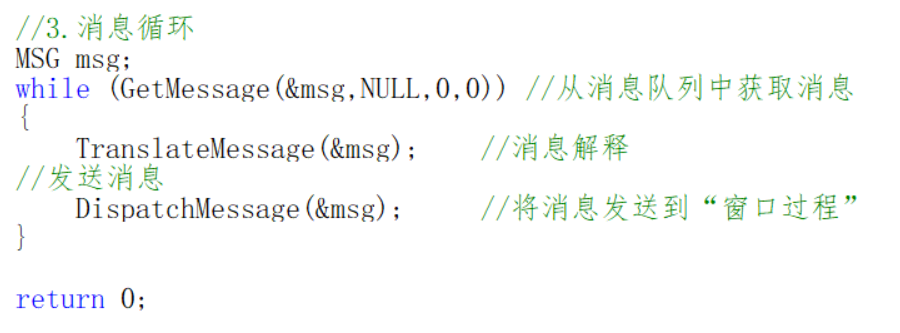


1. 确认关闭窗口后，在这个窗口过程函数中会销毁窗口，也就是

DestroyWindow(hwnd)，销毁窗口后，会发送一条destroy消息到消息队列中，GetMessage函数会取到这条消息，并在窗口过程函数中处理它。



1. 在窗口过程函数中，收到destroy消息后，会发送一条quit消息到消息队列中。



1. GetMessage函数收到quit消息后，会返回0，这个消息循环就会终止，接着执行下面的代码，return 0后，整个程序也就结束了。

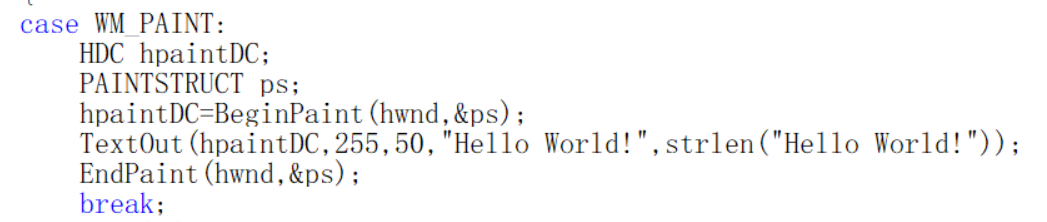
对于WM\_PAINT消息的分析

当窗口出现无效区域的时候，就会产生paint消息，出现无效区域的情况有以下几种。

1. 最初创建窗口的时候，updatewindow会让整个窗口变为无效。
2. 改变窗口尺寸的时候，水平改变或者是垂直改变，具体的无效区域是由设计窗口类的时候的窗口样式指定的。
3. 当窗口最小化再恢复或者最大化再恢复的时候。
4. 当移动窗口到屏幕外。
5. 和其他窗口重叠被覆盖了。
6. 拖动滚动条或者调用scroll函数。
7. 调用invalidate、invalidateRect函数。

具体产生的无效区域大小还需要具体情况分析。

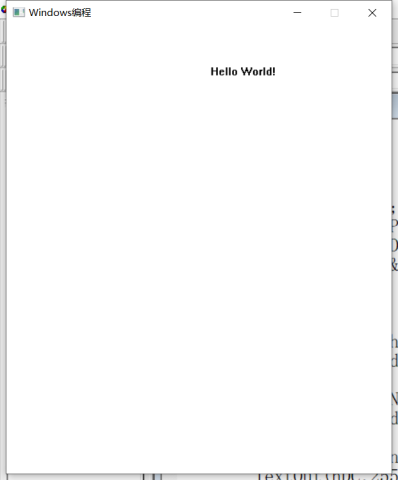
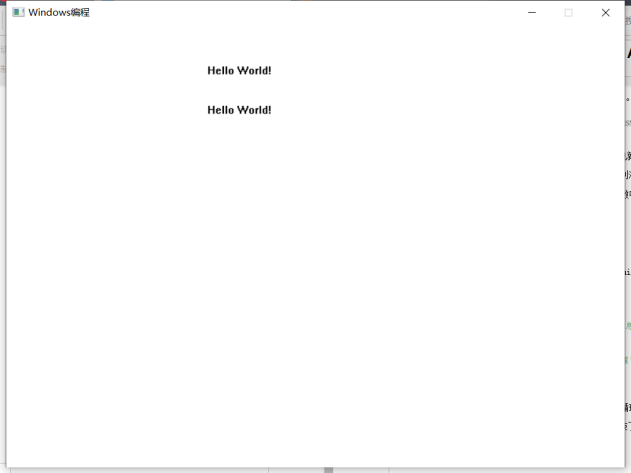
窗口过程函数中的响应paint消息的代码。



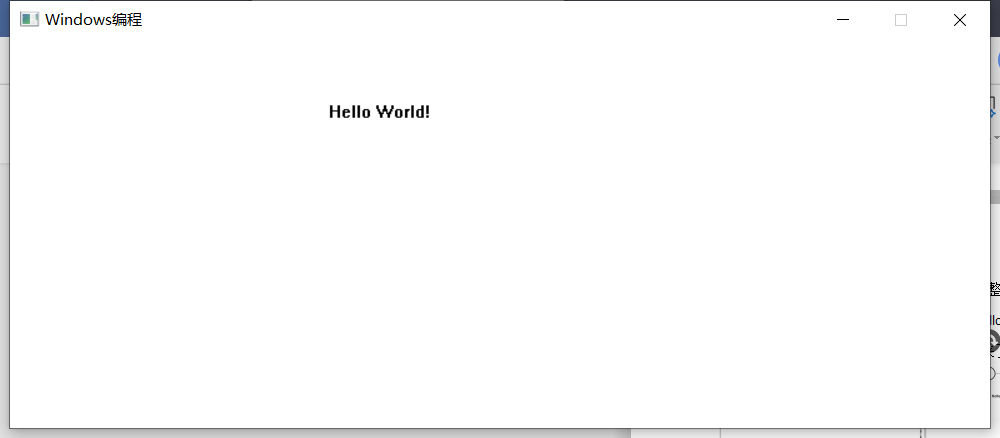
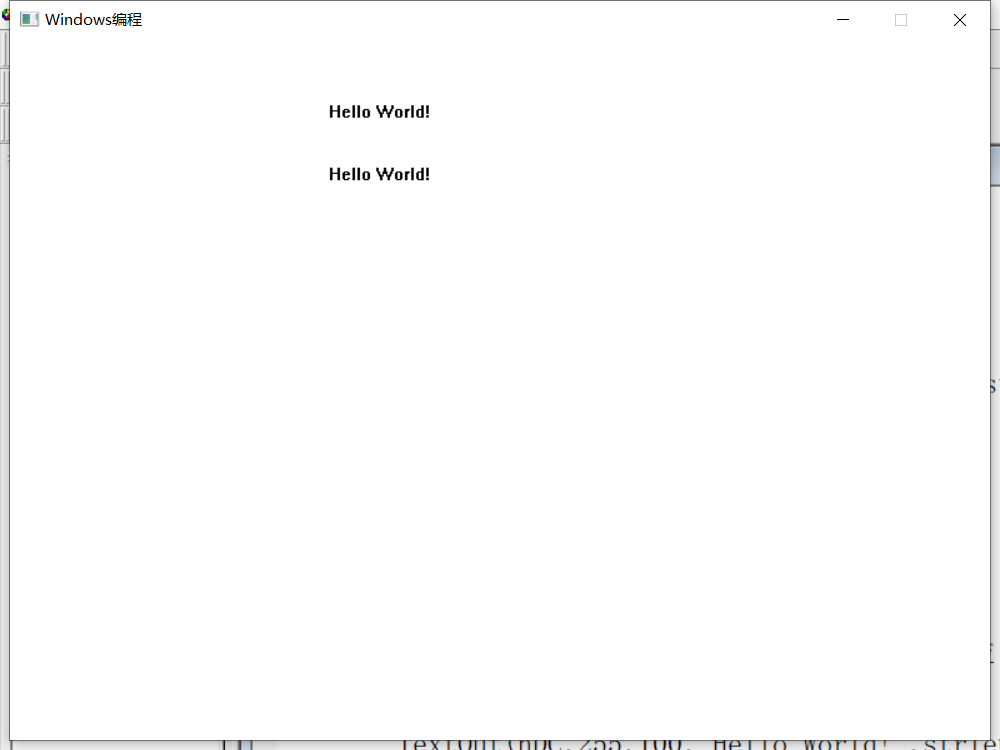
这里会将无效区域使用我们在设计窗口类中指定的背景笔刷进行重绘，并且还会在(255,50)的位置显示一句话。

测试一下

改变窗口水平宽度，下面的Hello World!由于整个窗口变为无效区，在重绘整个窗口时被刷掉了，而上面一个Hello World!其实也被刷掉了，但是我们在代码区又将它重新显示了出来了。

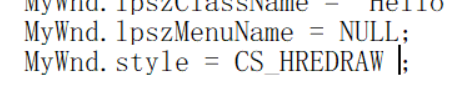


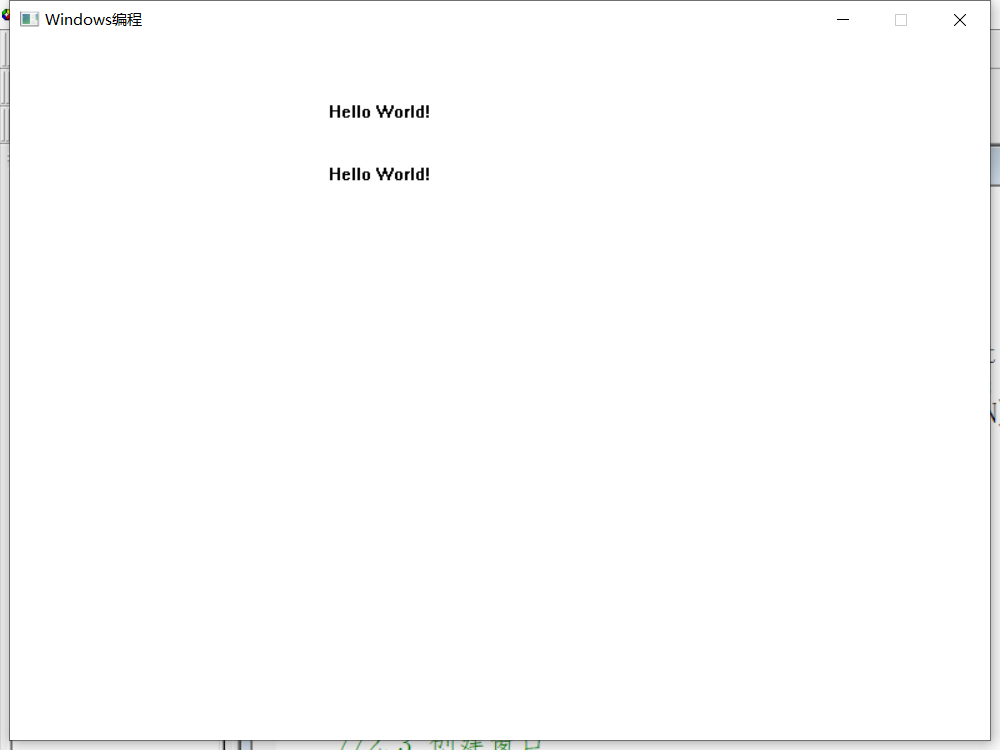
改变窗口的垂直高度，也是发生了重绘

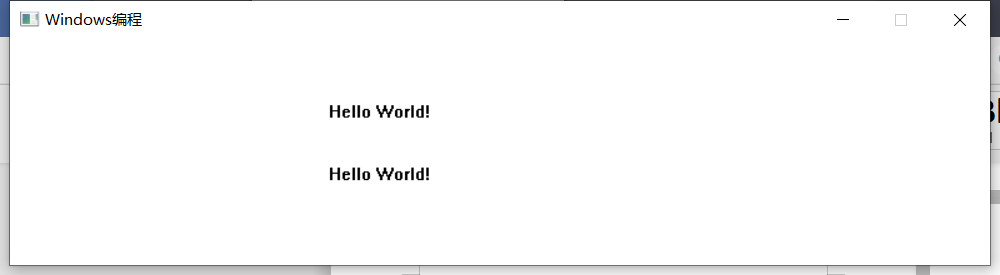


假如将设计窗口类代码中的窗口样式中的当窗口垂直发生变化时，整个窗口将变为无效进行重绘的样式去掉的话，就会出现以下的情况。









这是由于将这个样式去掉的话，无效区域就只是在这个窗口改变的区域，而窗口没有改变的区域，依然是有效区，不会发生重绘。

这里的无效区域是垂直方向上增加或者减少的区域。

其他使窗口由有效区变为无效区的方法与上面类似。

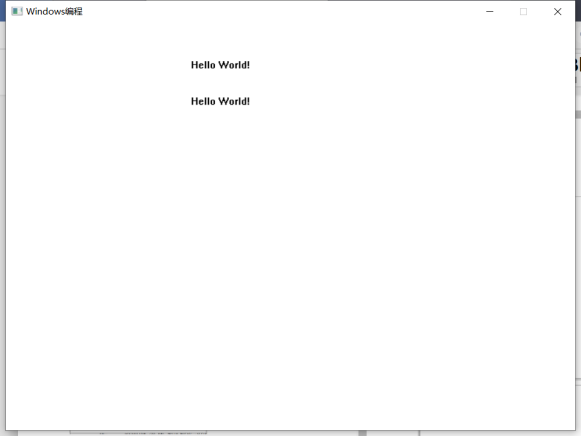
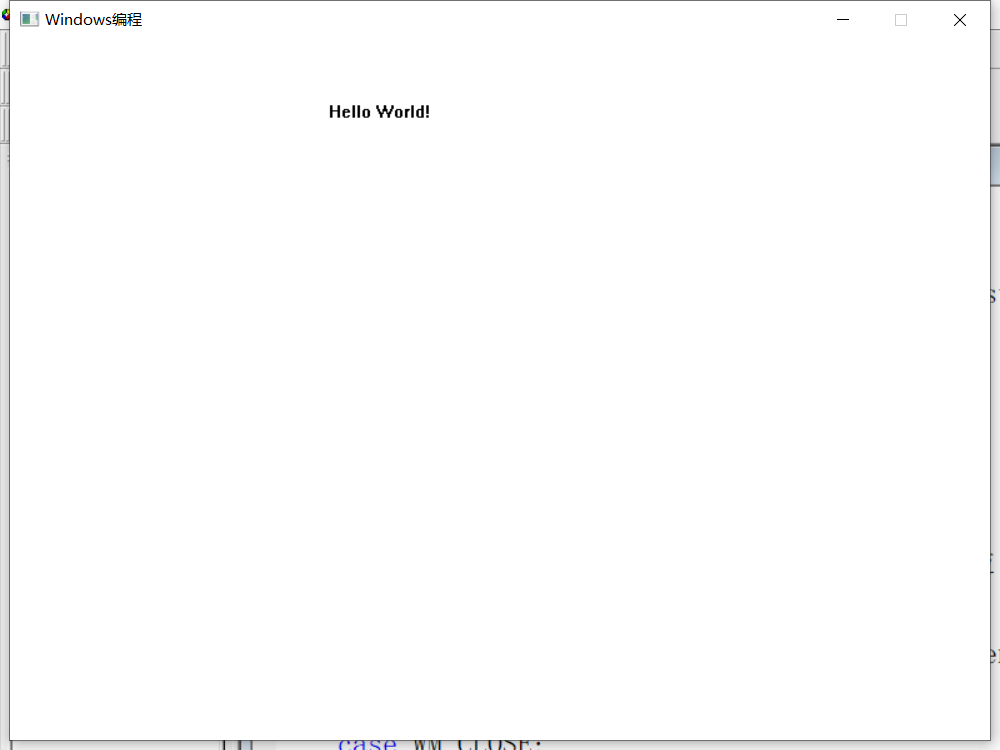
Begpaint和GetDC的区别

我们使用GetDC获取一个窗口的设备描述表(绘图工具箱），用于

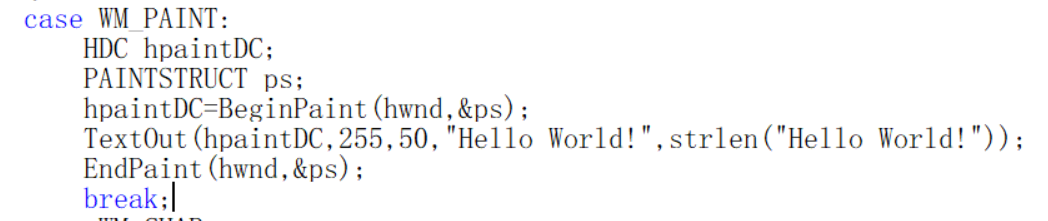
在这个窗口显示一些文字等操作



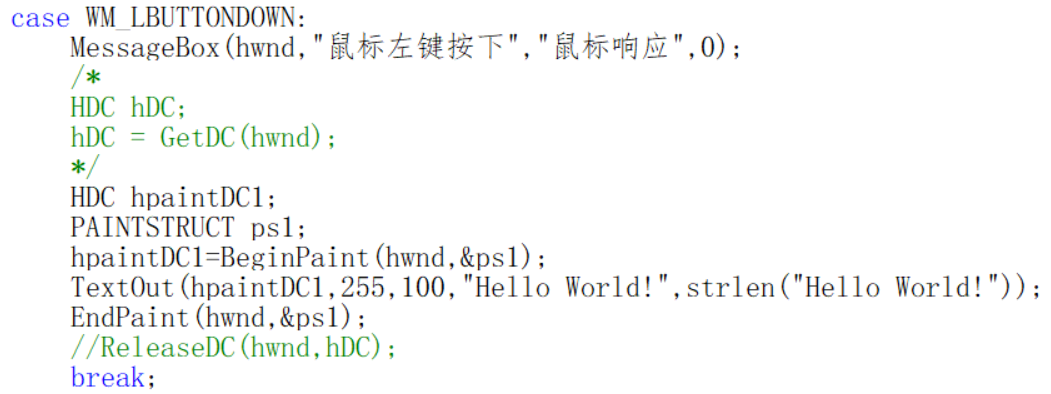
如按下鼠标左键，显示出一句Hello world!



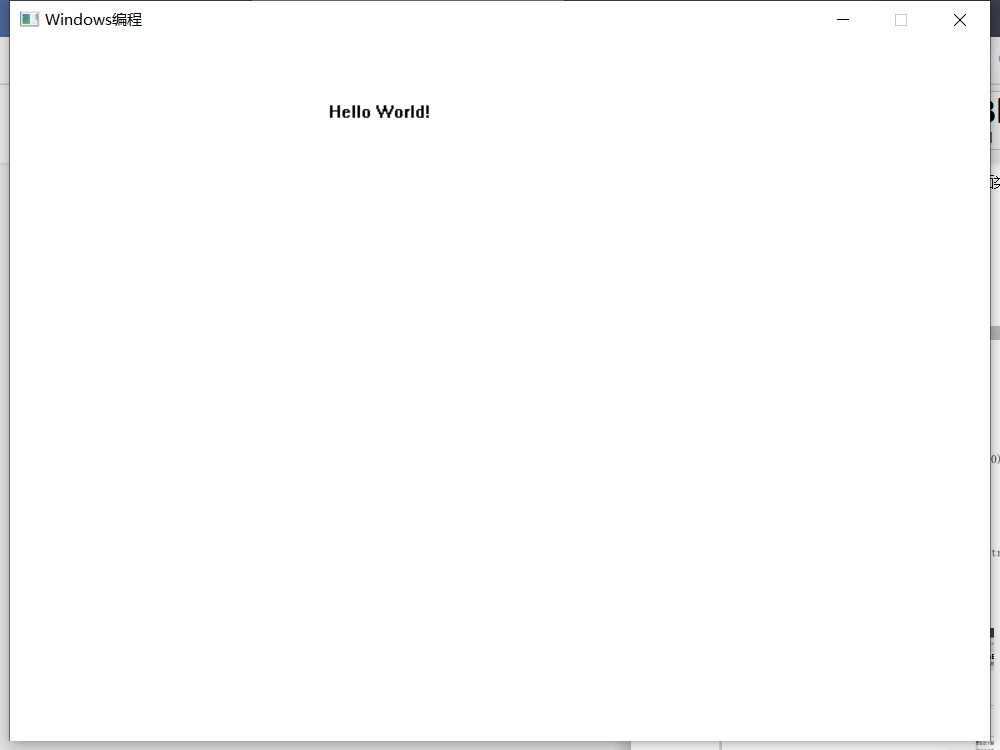
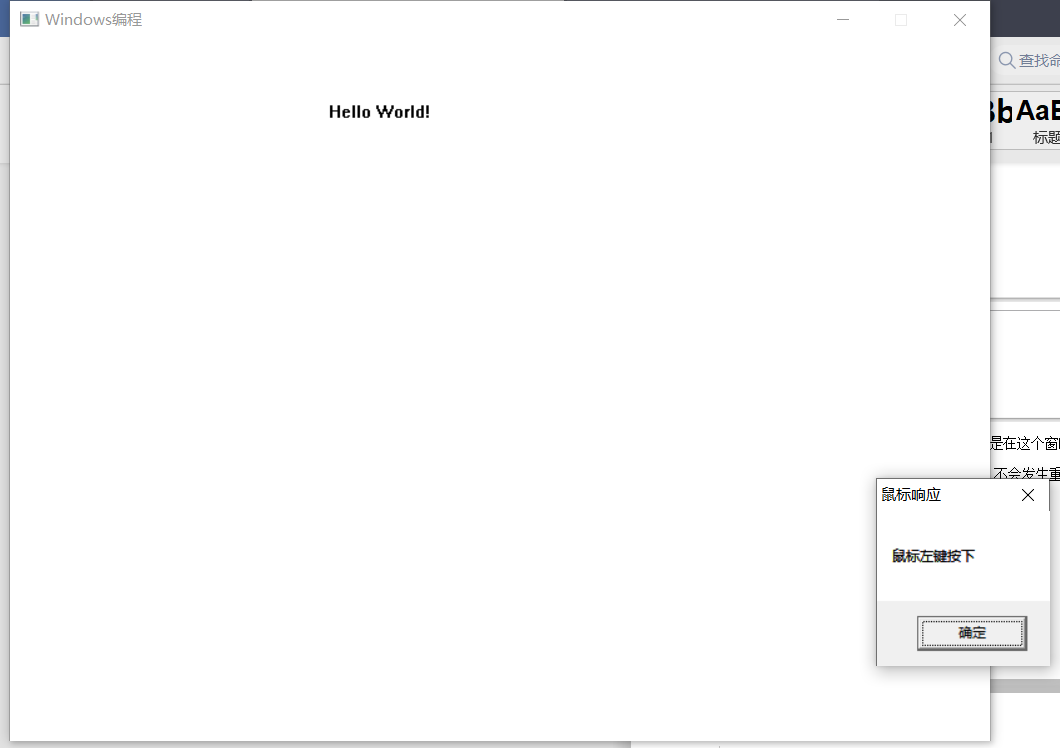
我们使用Begpaint获取一个窗口的设备描述表，也用于在这个窗口显示一些文字等操作。



使用Begpaint替换GetDC的尝试

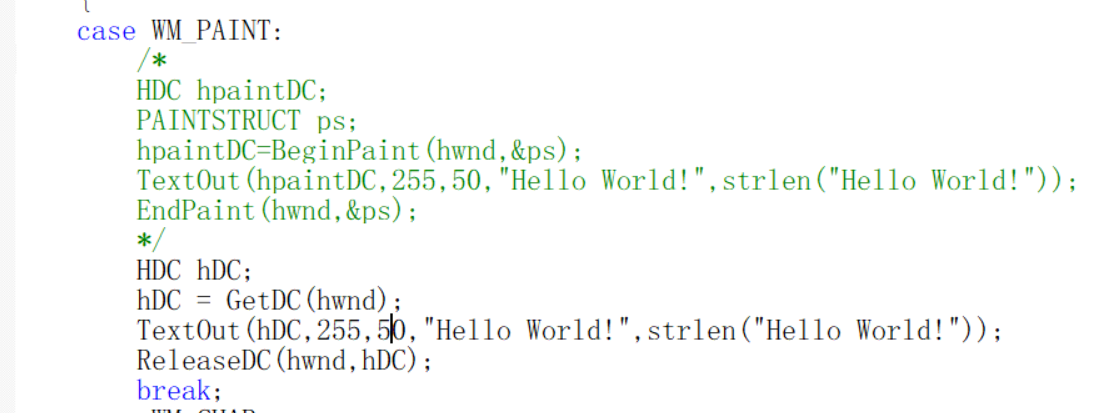


按下鼠标左键，发现没有Hello World!输出



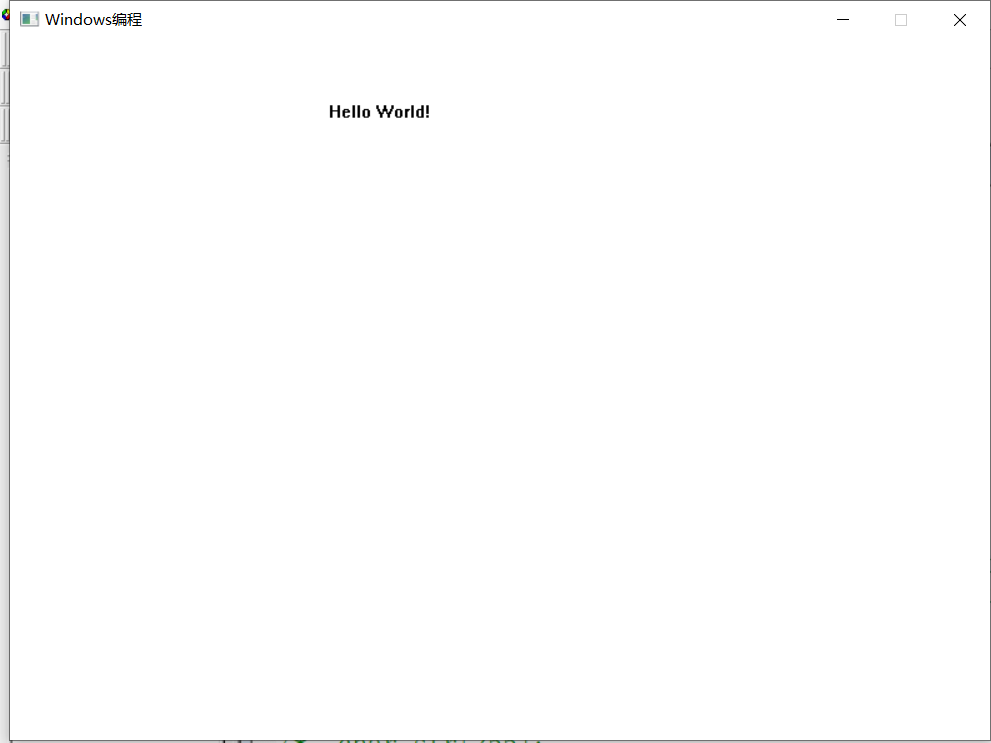
原因：Begpaint返回的绘图工具箱只在无效区域中会重绘窗口，我们按下鼠标左键时，整个窗口都处于有效，就不会输出Hello World!了。

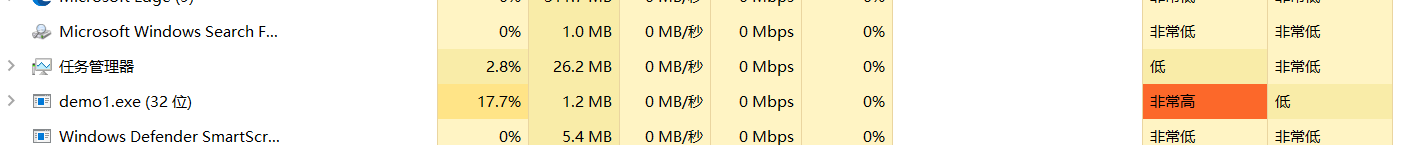
使用GetDC替换Begpaint的尝试



运行后发现无法关闭这个窗口，打开任务管理器，发现这个程序的

资源占有率非常高。





原因分析：

GetMessage函数在获取消息时，一般会将消息从队列中删除，但是

WM\_PAINT消息除外，begpaint消息才能将WM\_PAINT消息从消息队列中删除，并且将窗口的无效区域变为有效区域。

这里在处理WM\_PAINT消息的使用GetDC会导致两个后果：

1. 无效区域不会变为有效，并且不断发送WM\_PAINT消息到消息队列中。
2. WM\_PAINT消息不会从消息队列中被删除。

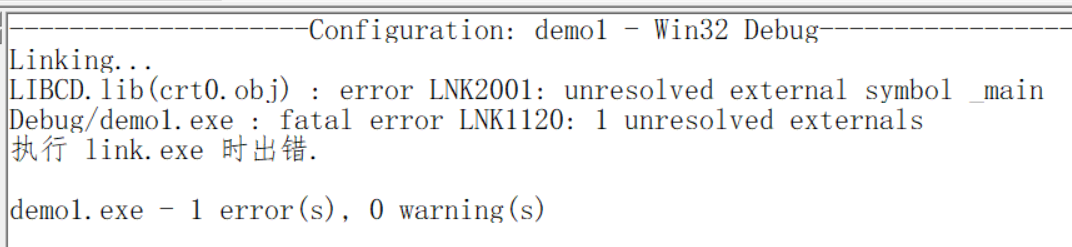
这样，程序就会一直在处理WM\_PAINT消息，而不会处理其他的消息比如close消息等等。由于在不断产生消息，处理WM\_PAINT消息进行窗口重绘，自然会消耗巨大的计算机资源。

对于Begpaint的总结

1. 返回设备描述表（绘图工具箱）。
2. 在无效区域内重绘窗口。
3. 把无效区域变为有效区域。
4. 把WM\_PAINT消息从消息队列中删除(GetMessage不会删除）。
5. 一定要在WM\_PAINT消息下使用。
6. 不要在WM\_PAINT消息外的任何地方使用begpaint消息，因为只在无效区域内进行重绘。

实验中遇到的一些问题：

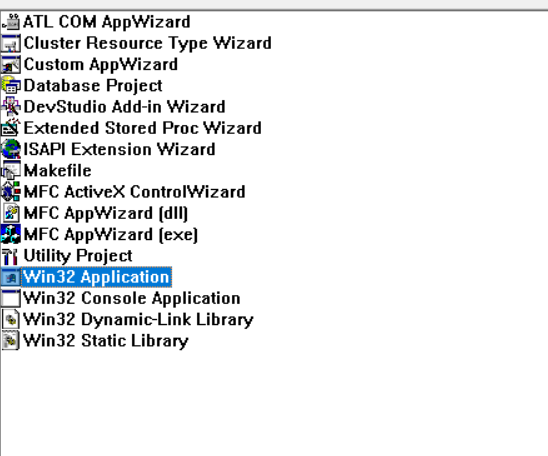
链接时出错



解决方法：

1. 创建项目工程时，要选择win32 Application而不是win32 console

Application



1. 打开工程-设置-连接



将console改为windows

实验总结：

1. 需要深刻理解windows编程内部的运行机制，操作系统创建消息队列，WinMain函数是身体，消息循环是心脏，窗口过程函数是大脑，我们大部分编程都在设计这个大脑。
2. 编程时申请的资源要进行释放。我们这一节的代码的底层都是C语言，一不小心就会造成内存泄漏或者是死循环之类的，这需要我们时刻注意自己代码的正确性。
3. 这一节的内容是非常基础的，这是很多前端编程的底层原理，学好windows编程，有助于我们学习其他前端编程。