

**操作系统实验报告**

**实验题目** 实验4 线程的状态和转换

**学生姓名**  余梓俊

**学 号**  2018211991

**专业班级** 计算机科学与技术18-3班

**指导教师**  田卫东

**完成日期**  2020年11月30日

**合肥工业大学 计算机与信息学院**

* 1. **实验目的和任务要求**

调试线程在各种状态间的转换过程，熟悉线程的状态和转换。

通过为线程增加挂起状态，加深对线程状态的理解。

* 1. **实验原理**

在EOS中，线程是处理器调度的基本单位。当一个进程被创建时，系统首先会为该进程分配一些资源（包括内存，内核对象，以及指令和数据等），然后系统会为该进程创建一个默认线程，做为该进程的主线程。进程的主线程开始执行后，就可以认为是进程开始执行了。多数情况下，进程只需要在主线程运行的过程中就可以完成工作。但是，随着单个处理器中内核数量的增加，越来越多的软件要求使用多线程进行并行处理，从而提高硬件资源的利用率，以及软件执行的效率。EOS支持多线程并发执行，除了在前一节提到的多个进程（每个进程都有一个主线程）并发执行的情况外，还可以在一个进程中创建多个线程。例如，在一个进程的主线程中，可以调用API函数CreateThread来创建一个新线程，这个新线程与主线程共享该进程的所有资源，例如访问进程的地址空间、执行进程的代码、读写进程打开的文件等。而且，EOS中的线程是属于内核级的，所有线程会一起竞争处理机的使用权，不会区分线程属于哪个进程。本节的主要内容就是向读者详细讲解以上的各种概念，帮助读者理解线程的本质。

线程在其整个生命周期中（从创建到终止）会在多个不同的状态间进行转换。EOS线程的状态由线程控制块TCB中的State域保存，EOS线程的状态和其转换过程可以参见图2-1。在椭圆圈内的是线程的状态，箭头表示状态的转换过程。EOS被设计为运行在单处理器上的多任务操作系统，所以，在任意时刻，最多只能有一个处于运行状态的线程占用处理器，而处于其他状态的线程数量可以为0个或多个。注意，由于Zero状态是线程状态转换过程中的中间状态，所以在图2-1中并没有出现。

* 1. **实验内容**

1、调试线程状态的转换过程

在本练习中，会在与线程状态转换相关的函数中添加若干个断点，并引导读者单步调试这些函数，使读者对EOS中的下列线程状态转换过程有一个全面的认识：

1.线程由阻塞状态进入就绪状态。

2.线程由运行状态进入就绪状态。

3.线程由就绪状态进入运行状态。

4.线程由运行状态进入阻塞状态。

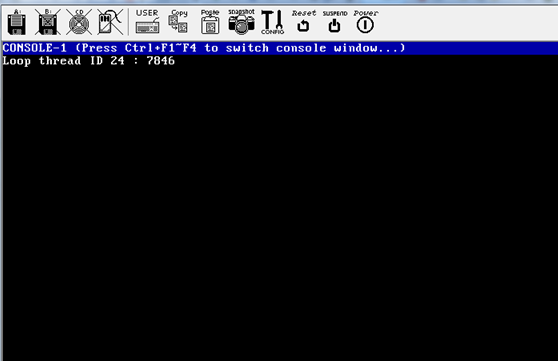
为了完成这个练习，EOS准备了一个控制台命令“loop”，这个命令的命令函数是ke/sysproc.c文件中的ConsoleCmdLoop函数（第797行），在此函数中使用LoopThreadFunction函数（第755行）创建了一个优先级为8的线程（后面简称为“loop线程”），该线程会在控制台中不停的（死循环）输出该线程的ID和执行计数，执行计数会不停的增长以表示该线程在不停的运行。可以按照下面的步骤查看一下loop命令执行的效果：

1. 按F7生成在本实验3.1中创建的EOS Kernel项目。

2. 按F5启动调试。

3. 待EOS启动完毕，在EOS控制台中输入命令“loop”后按回车。

4. 结束此次调试。



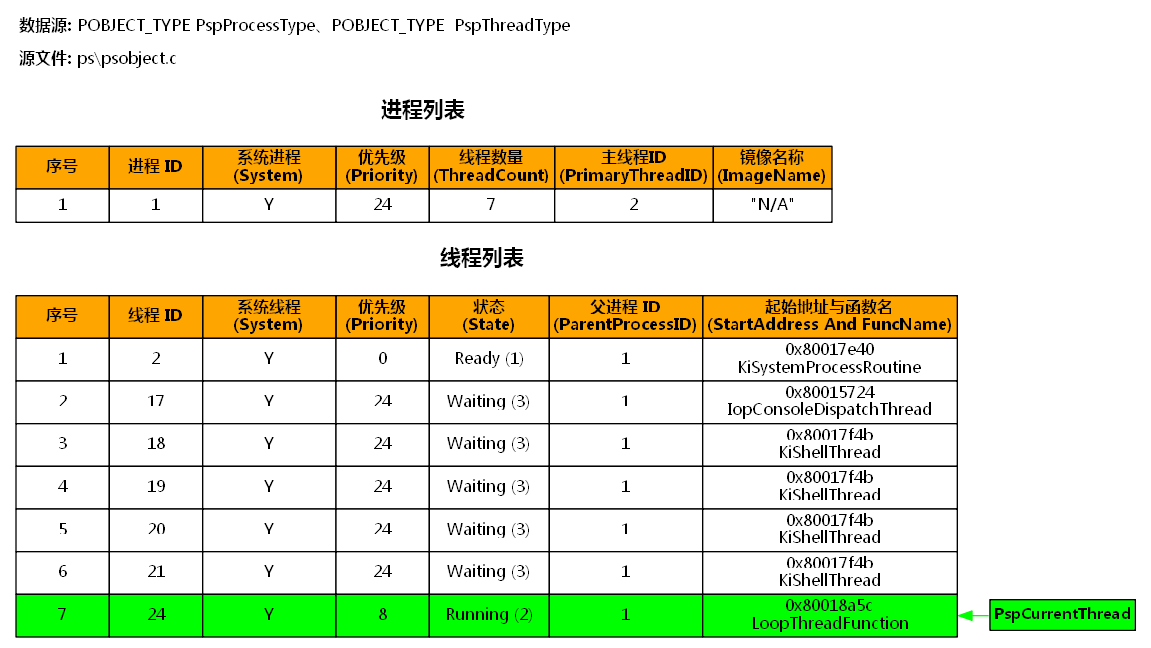
接下来按照下面的步骤调试线程状态转换的过程：

1. 在ke/sysproc.c文件的LoopThreadFunction函数中，开始死循环的代码行（第787行）添加一个断点。

2. 按F5启动调试。

3. 待EOS启动完毕，在EOS控制台中输入命令“loop”后按回车。

EOS会在断点处中断执行，表明loop线程已经开始死循环了。此时，EOS中所有的系统线程要么处于就绪状态（其优先级一定小于8，例如系统空闲线程），要么就处于阻塞状态（例如控制台派遣线程或控制台线程），所以，只有优先级为8的loop线程能够在处理器上执行。



接下来按照下面的步骤对断点进行一些调整：

1. 删除所有断点。

2. 打开ps/sched.c文件，在与线程状态转换相关的函数中添加断点，这样，一旦有线程的状态发生改变，EOS会中断执行，就可以观察线程状态转换的详细过程了。需要添加的断点有：

* 在PspReadyThread函数体中添加一个断点（第130行）。
* 在PspUnreadyThread函数体中添加一个断点（第158行）。
* 在PspWait函数体中添加一个断点（第223行）。
* 在PspUnwaitThread函数体中添加一个断点（第282行）。
* 在PspSelectNextThread函数体中添加一个断点（第395行）。

3. 按F5继续执行，然后激活虚拟机窗口。

此时在虚拟机窗口中会看到loop线程在不停执行，而之前添加的断点都没有被命中，说明此时还没有任何线程的状态发生改变。具体实验效果如图所示。

在开始观察线程状态转换过程之前还有必要做一个说明。在后面的练习中，会在loop线程执行的过程中按一次空格键，这会导致EOS依次执行下面的操作：

1. 控制台派遣线程被唤醒，由阻塞状态进入就绪状态。

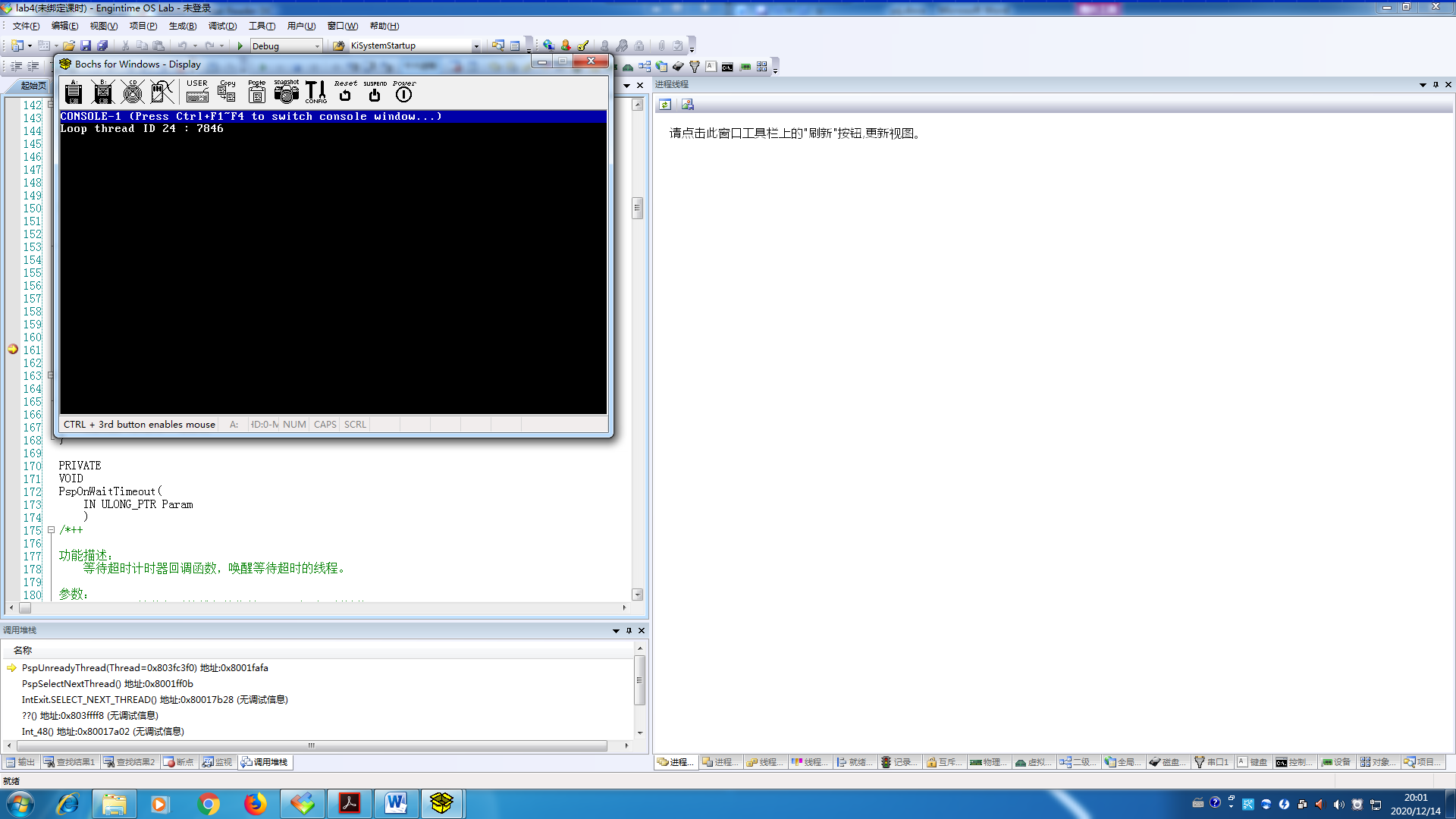
2. loop线程由运行状态进入就绪状态。

3. 控制台派遣线程由就绪状态进入运行状态。

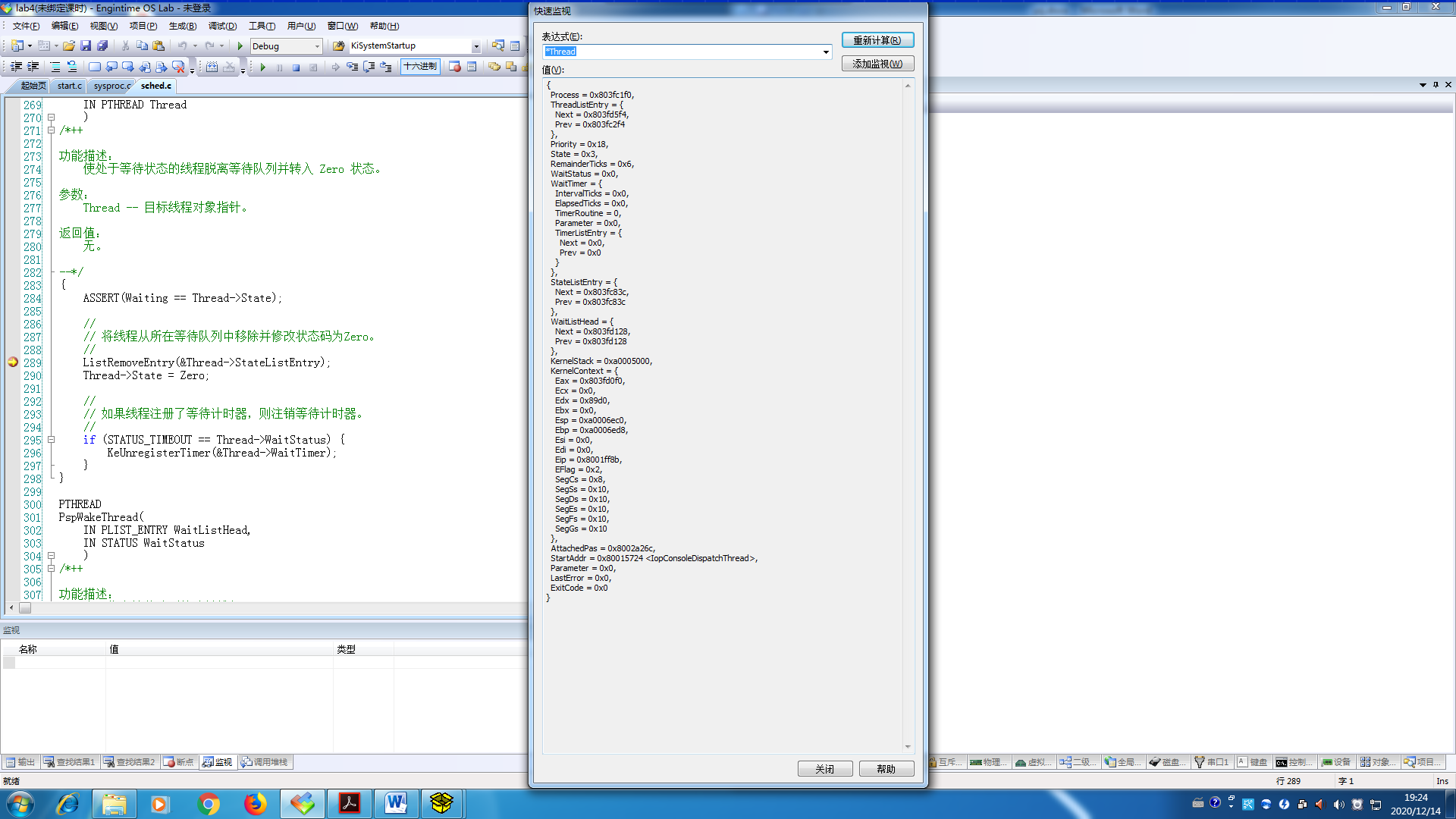
4. 待控制台派遣线程处理完毕由于空格键被按下而产生的键盘事件后，派遣线程会由运行状态重新进入阻塞状态，开始等待下一个键盘事件到来。

5. loop线程由就绪状态进入运行状态，继续执行死循环。

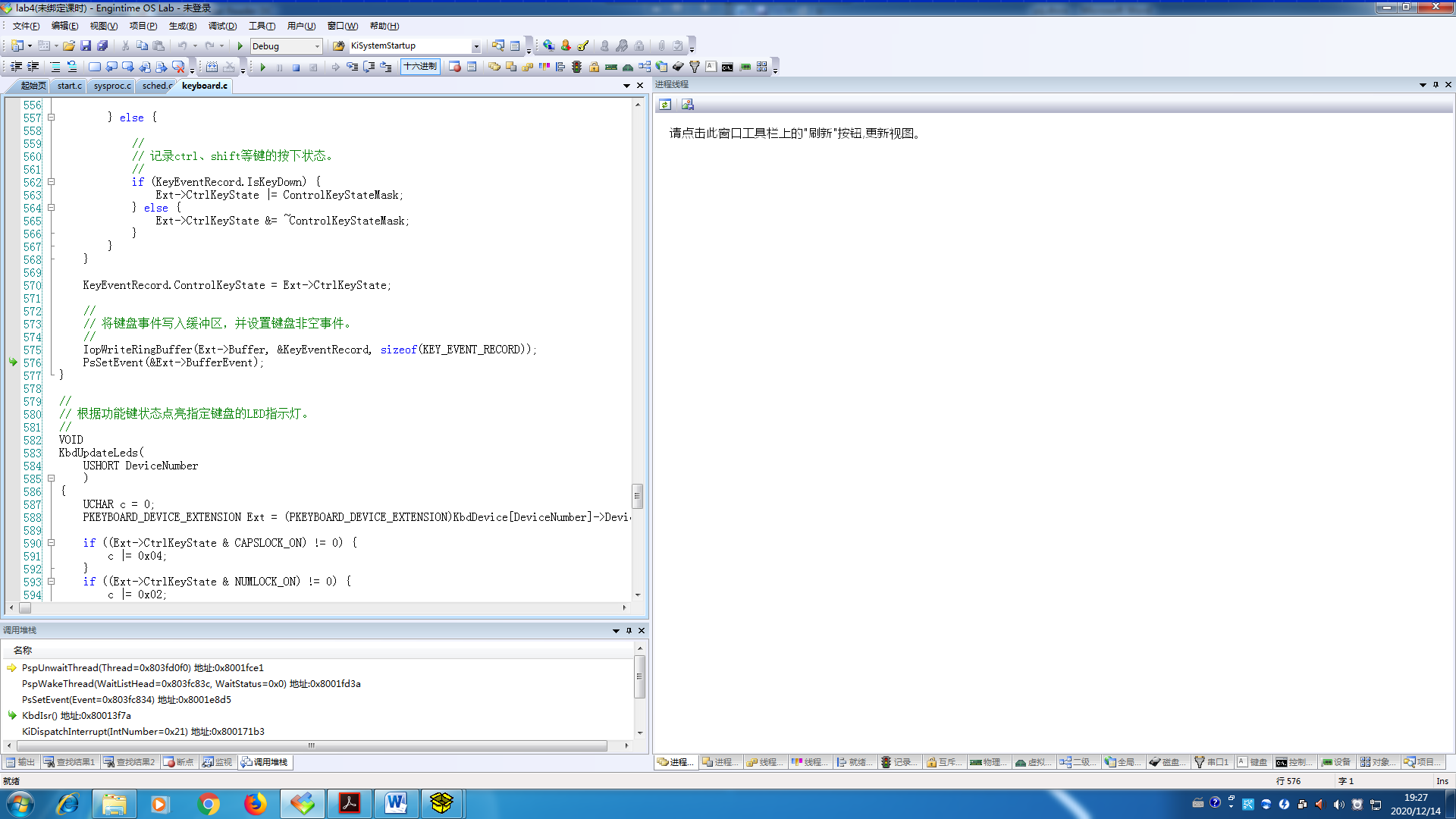
（1）控制台派遣线程由阻塞状态进入就绪状态



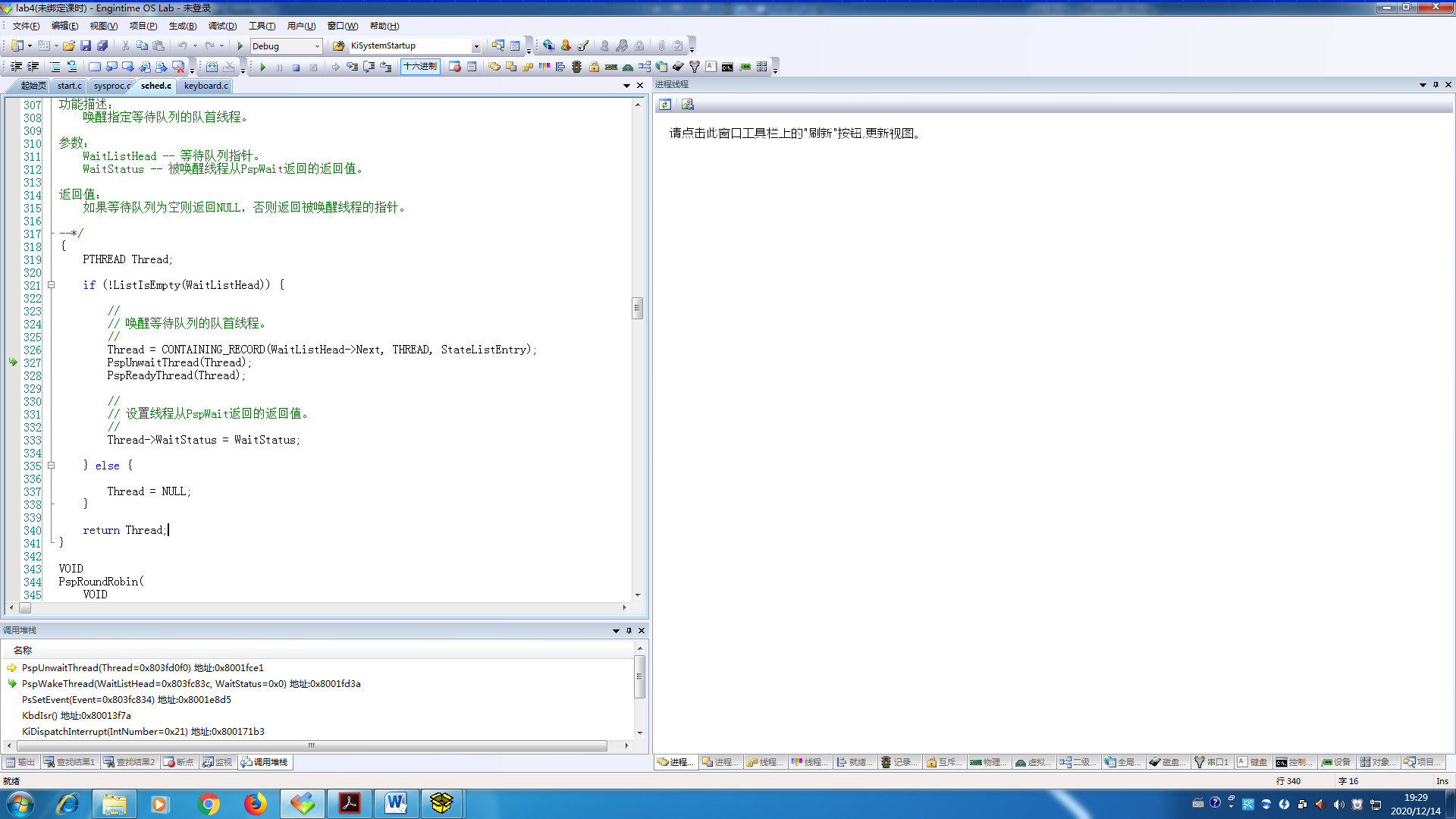
在虚拟机窗口中按下一次空格键。此时EOS会在PspUnwaitThread函数中的断点处中断。在“调试”菜单中选择“快速监视”，在快速监视对话框的表达式编辑框中输入表达式“\*Thread”，然后点击“重新计算”按钮，即可查看线程控制块（TCB）中的信息。其中State域的值为3（Waiting），双向链表项StateListEntry的Next和Prev指针的值都不为0，说明这个线程还处于阻塞状态，并在某个同步对象的等待队列中；StartAddr域的值为IopConsoleDispatchThread，说明在PspUnwaitThread函数中要处理的线程就是控制台派遣线程。



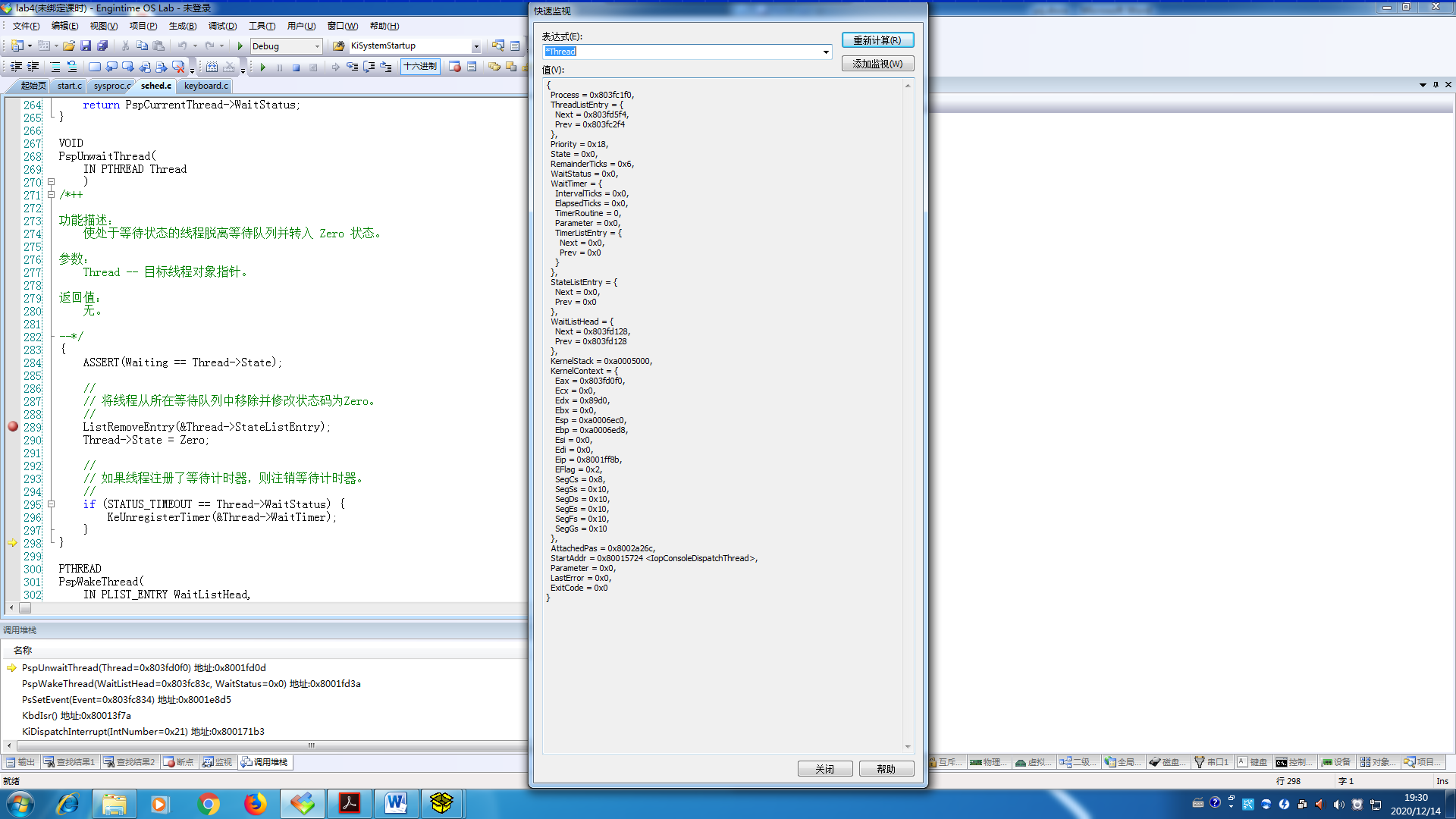
激活“调用堆栈”窗口。根据当前的调用堆栈，可以看到是由键盘中断服务程序（KdbIsr 函数）进入的。当按下空格键后，就会发生键盘中断，从而触发键盘中断服务程序。在“调用堆栈”窗口中双击 KdbIsr 函数对应的堆栈项,可以看到在该服务程序的最后会将键盘事件放入缓冲区，然后唤醒控制台派遣线程，由控制台派遣线程将键盘事件派遣到活动的控制台。



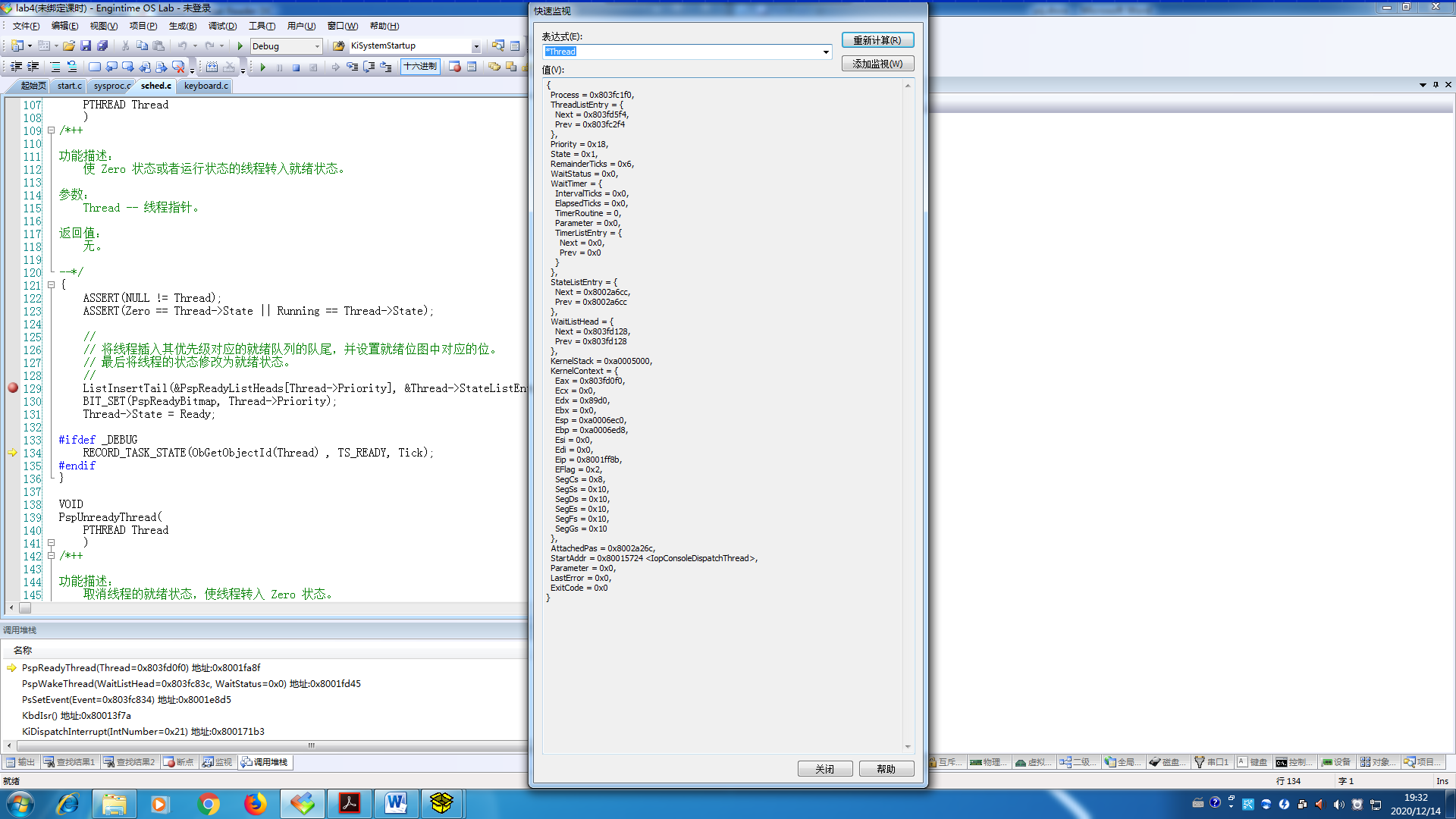
在“调用堆栈”窗口中双击 PspWakeThread 函数对应的堆栈项。可以看到在此函数中连续调用了PspUnwaitThread 函数和 PspReadyThread 函数，从而使处于阻塞状态的控制台派遣线程先退出阻塞状态，然后再进入就绪状态。



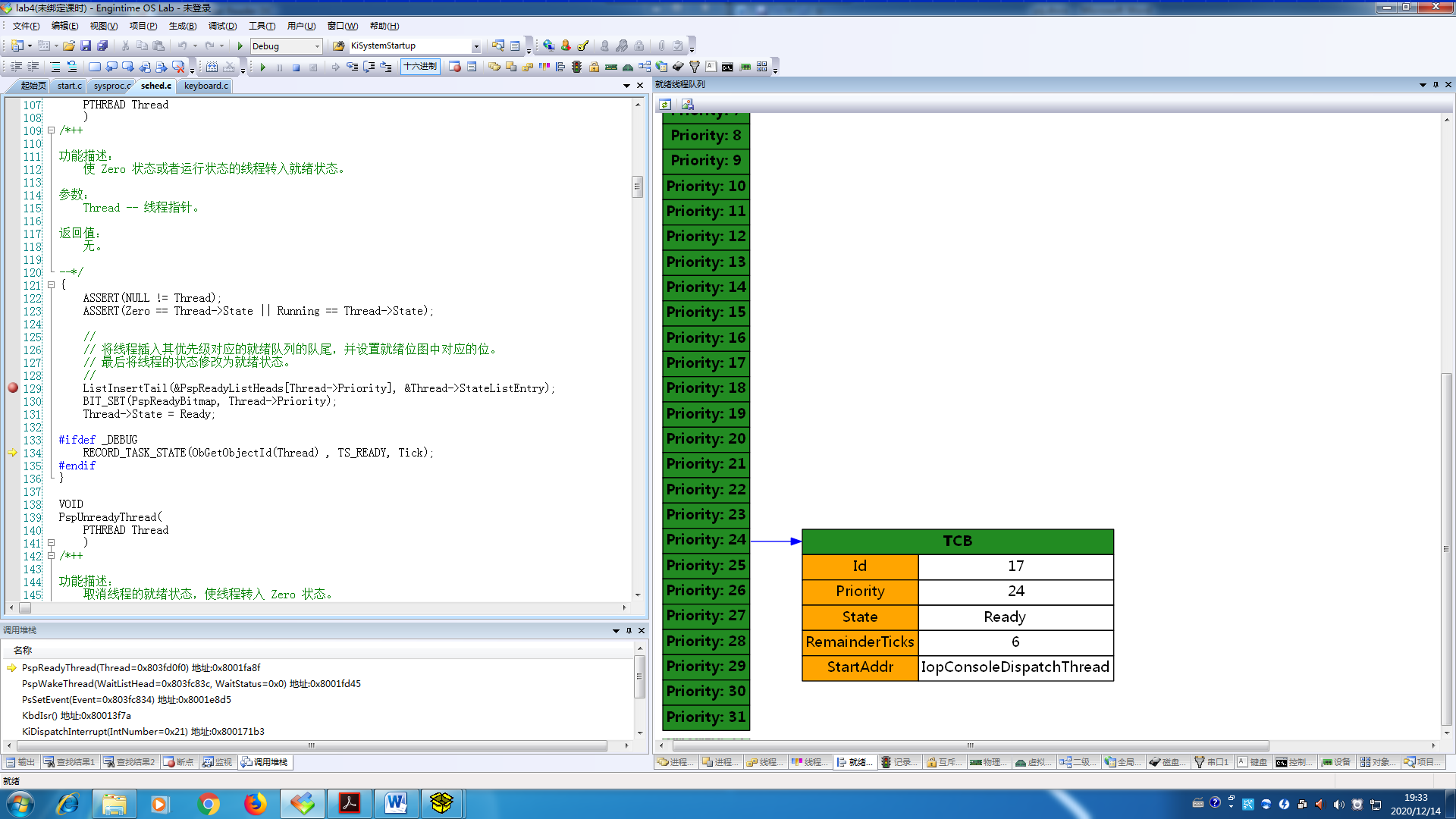
在“调用堆栈”窗口中双击 PspUnwaitThread 函数对应的堆栈项，再来看看此函数是如何改变线程状态的。按 F10 单步调试直到此函数的最后，然后再从快速监视对话框中观察“\*Thread”表达式的值。此时 State 域的值为 0（Zero），双向链表项 StateListEntry 的 Next 和 Prev 指针的值都为 0，说明这个线程已经处于游离状态，并已不在任何线程状态的队列中。

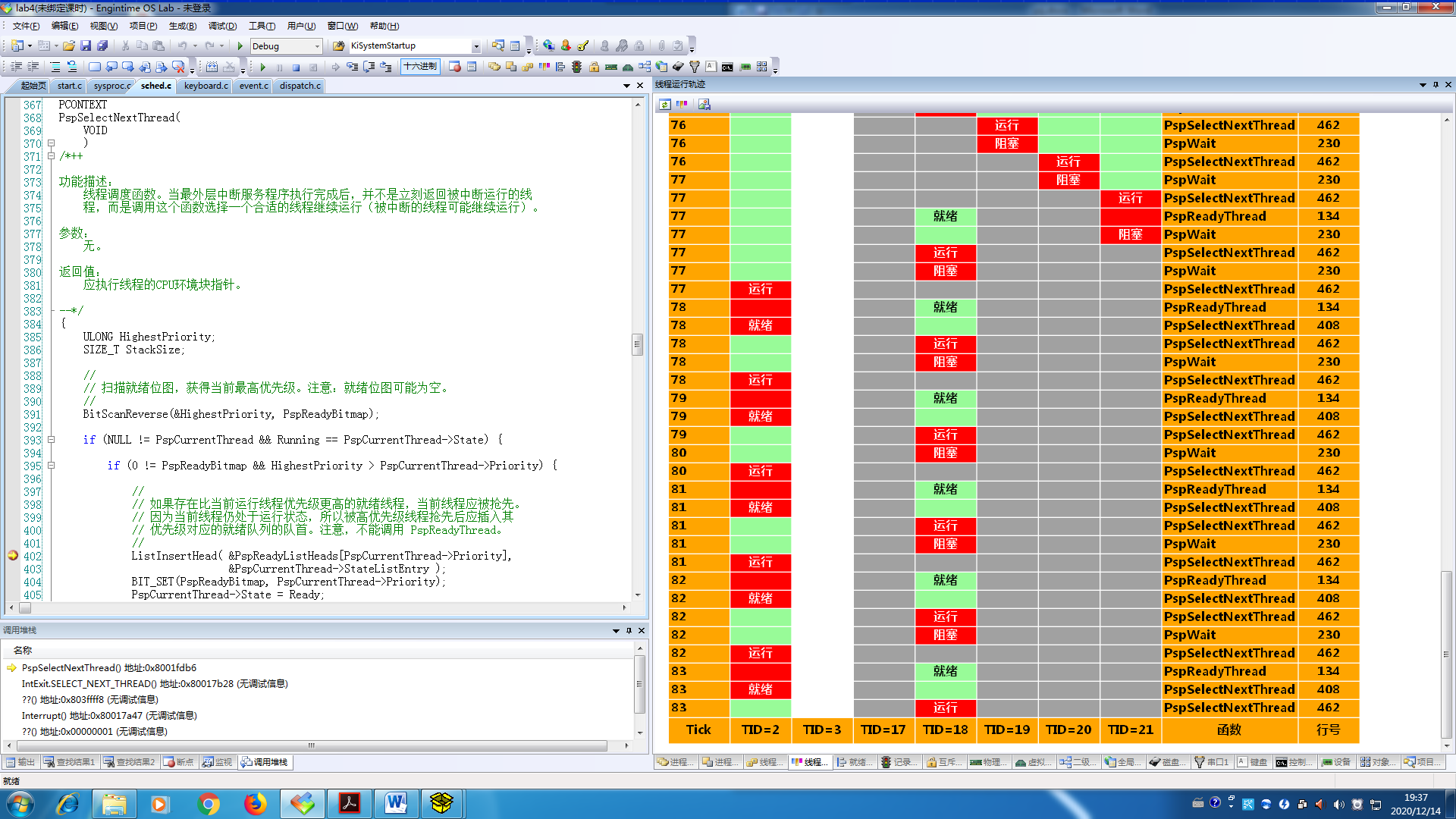


按F5继续执行，在PspReadyThread函数中的断点处中断。按F10单步调试直到此函数的最后，然后再从快速监视对话框中观察“\*Thread”表达式的值。此时State域的值为1（Ready），双向链表项StateListEntry的Next和Prev指针的值都不为0，说明这个线程已经处于就绪状态，并已经被放入优先级为24的就绪队列中。仔细阅读PspReadyThread函数中的源代码，理解这些源代码是如何改变线程状态的。



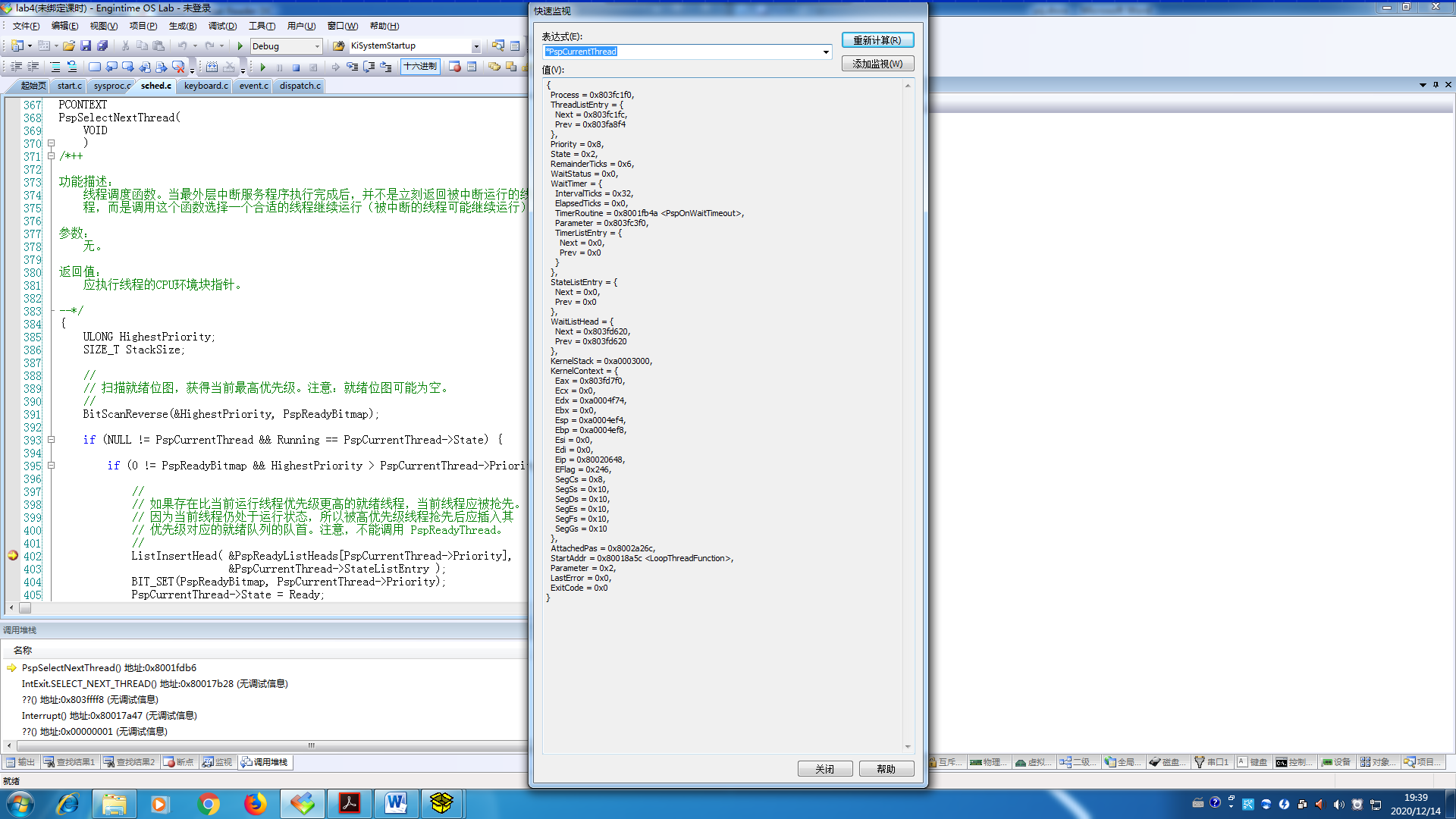
选择“调试”菜单“窗口”菜单中的“就绪线程队列”菜单项，打开“就绪线程队列”窗口，在该窗口的工具栏上点击“刷新”按钮，可以看到在优先级为 24 的就绪队列中已插入线程 ID 为 17的线程控制块，如图 12-3 所示。然后，在“线程运行轨迹”窗口，查看 ID 为 17 的线程的运行轨迹，可以看到该线程已由“阻塞”状态转化为“就绪”状态了。



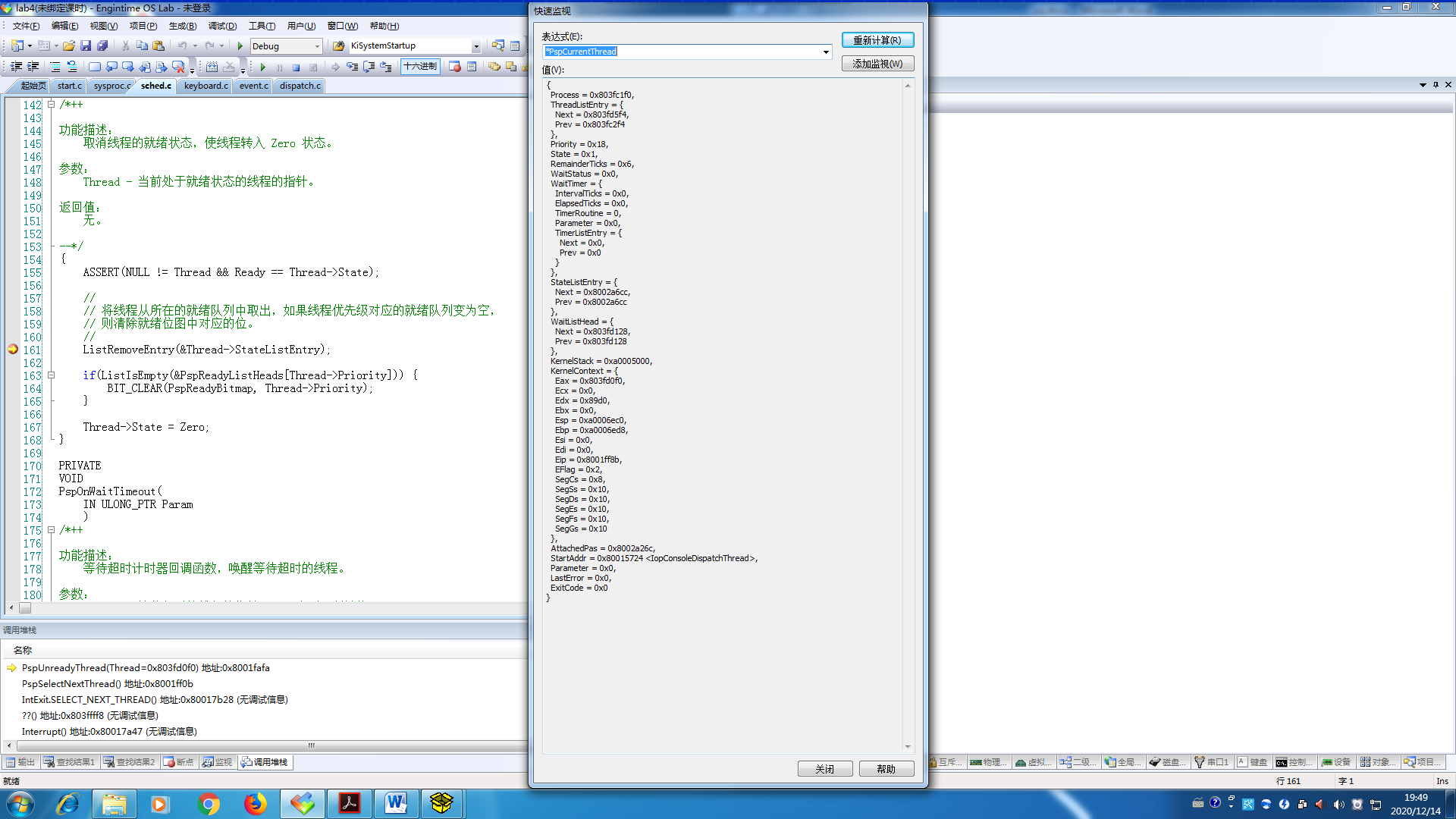


（2）loop 线程由运行状态进入就绪状态

按 F5 继续执行，在 PspSelectNextThread 函数中的断点处中断。在“快速监视”对话框中查看“\*PspCurrentThread”表达式的值，观察当前线程的信息。其中 State 域的值为 2（Running），双向链表项 StateListEntry 的 Next 和 Prev 指针的值都为 0，说明这个线程仍然处于运行状态，由于只能有一个处于运行状态的线程，所以这个线程不在任何线程状态的队列中；StartAddr 域的值为 LoopThreadFunction，说明这个线程就是 loop 线程。注意，在本次断点被命中之前，loop线程就已经被中断执行了，并且其上下文已经保存在线程控制块的 KernelContext 中。



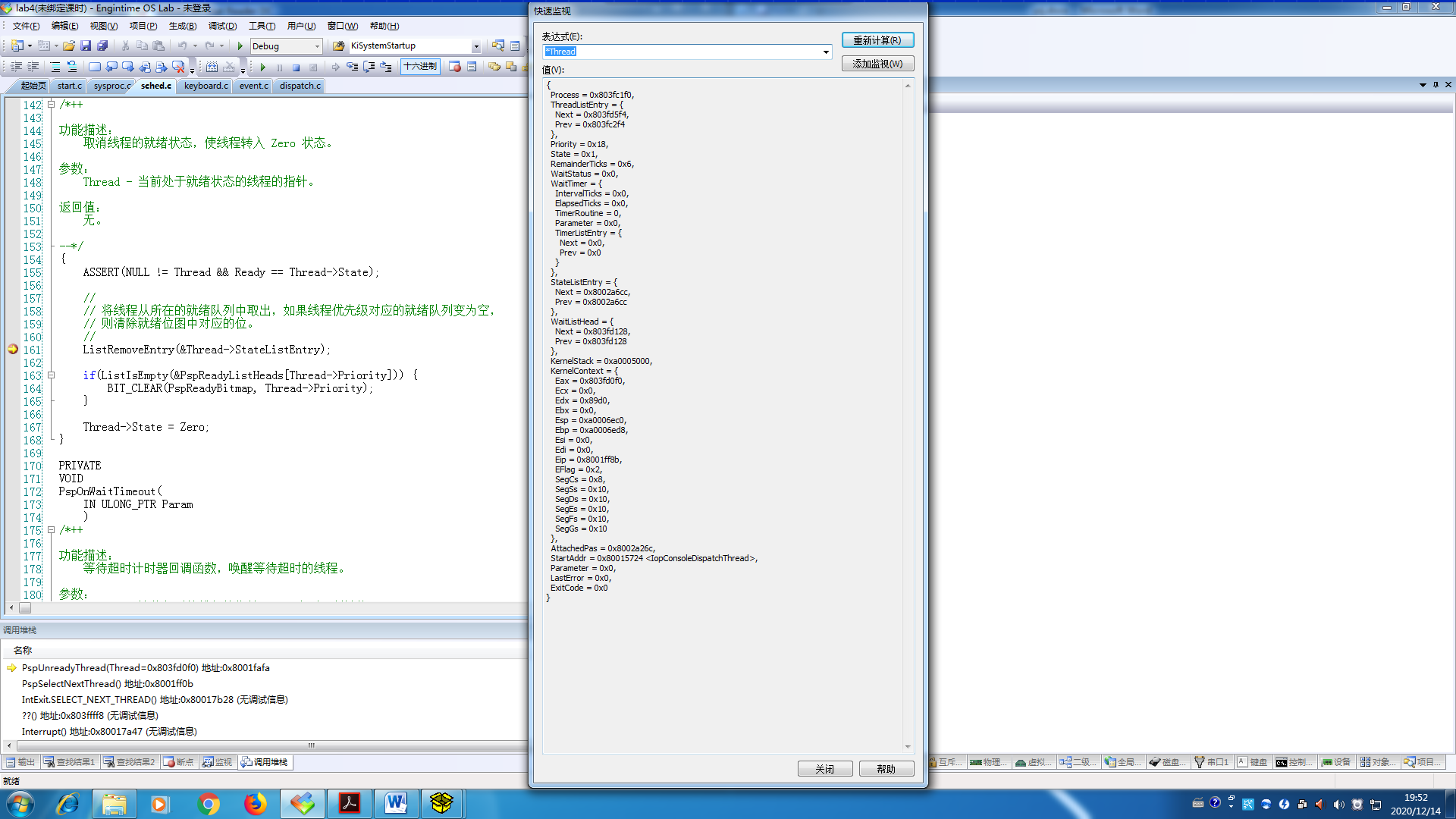
按 F10 单步调试，直到对当前线程的操作完成（也就是花括号中的操作完成）。在“快速监视”对话框中查看“\*PspCurrentThread”表达式的值。其中 State 域的值为 1（Ready），双向链表项StateListEntry 的 Next 和 Prev 指针的值都不为 0，说明 loop 线程已经进入了就绪状态，并已经被放入优先级为 8 的就绪队列中。刷新“就绪线程队列”窗口中，可以看到在优先级为 8 的就绪队列中已插入线程 ID 为 24 的线程控制块。仔细阅读 PspSelectNextThread 函数这个花括号中的源代码，理解这些源代码是如何改变线程状态的，并与 PspReadyThread 函数中的源代码进行比较，说明这两段源代码的异同，体会为什么在这里不能直接调用 PspReadyThread 数。



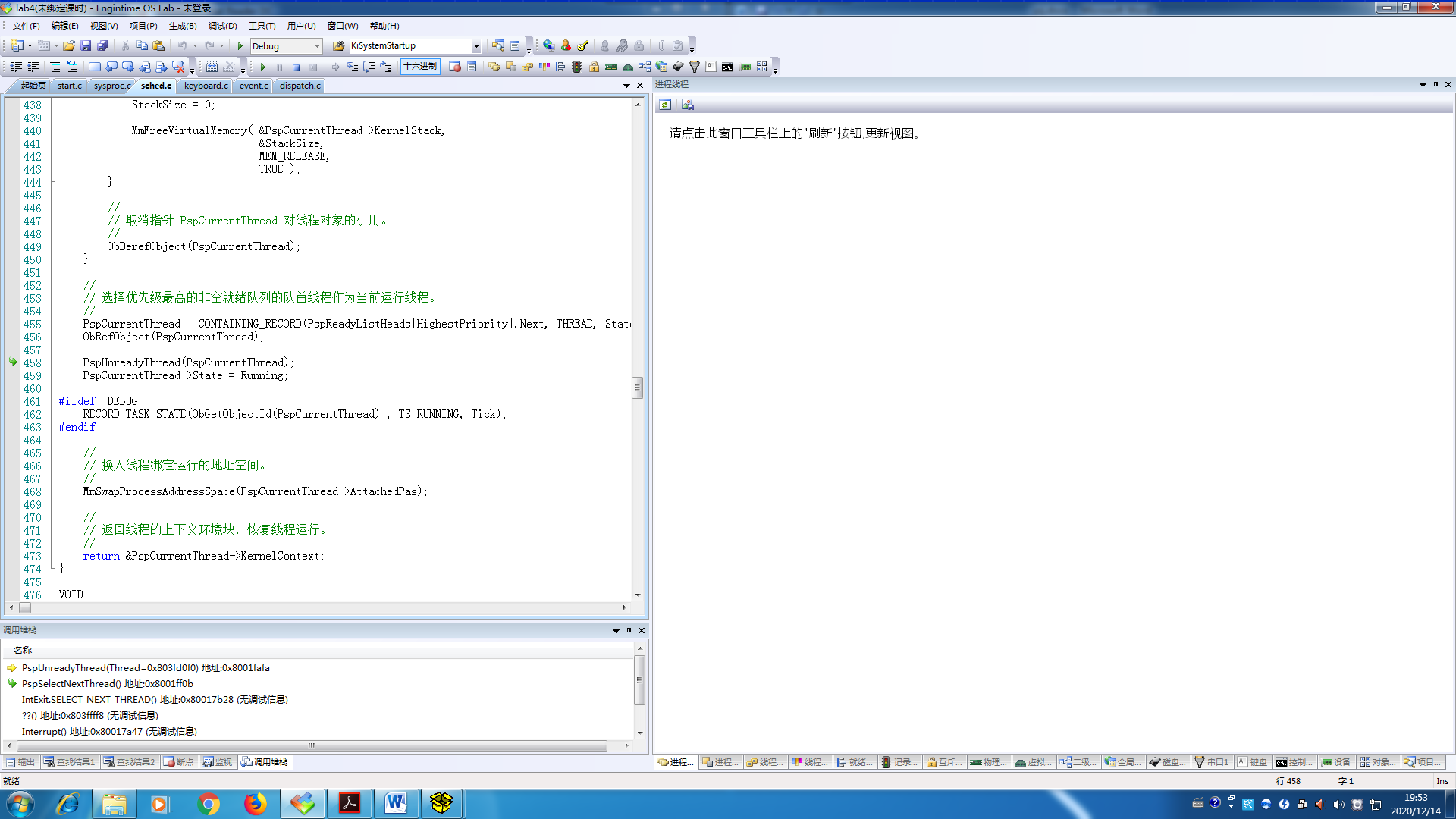


（3）控制台派遣线程由就绪状态进入运行状态

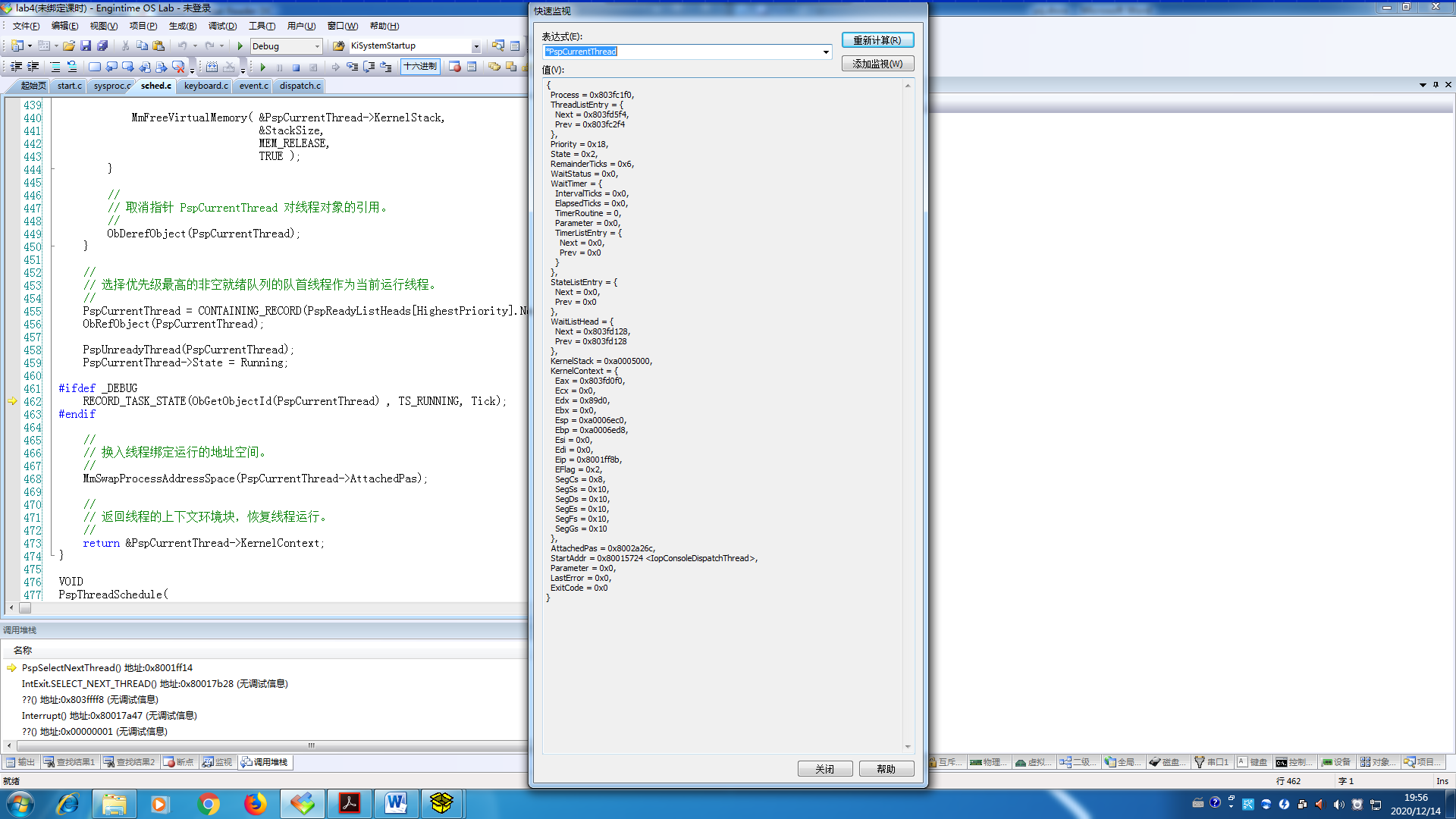
按 F5 继续执行，在 PspUnreadyThread 函数中的断点处中断。在快速监视对话框中查看“\*Thread”表达式的值。其中 State 域的值为 1（Ready），双向链表项 StateListEntry 的 Next 和 Prev 指针的值都不为 0，说明这个线程处于就绪状态，并在优先级为 24 的就绪队列中；StartAddr 域的值为 IopConsoleDispatchThread，说明这个线程就是控制台派遣线程。仔细阅读 PspUnreadyThread函数中的源代码，理解这些源代码是如何改变线程状态的。



关闭快速监视对话框后，在“调用堆栈”窗口中激活 PspSelectNextThread 函数对应的堆栈项，可以看到在 PspSelectNextThread 函数中已经将 PspCurrentThread 全局指针指向了控制台派遣线程，并在调用 PspUnreadyThread 函数后，将当前线程的状态改成了 Running。

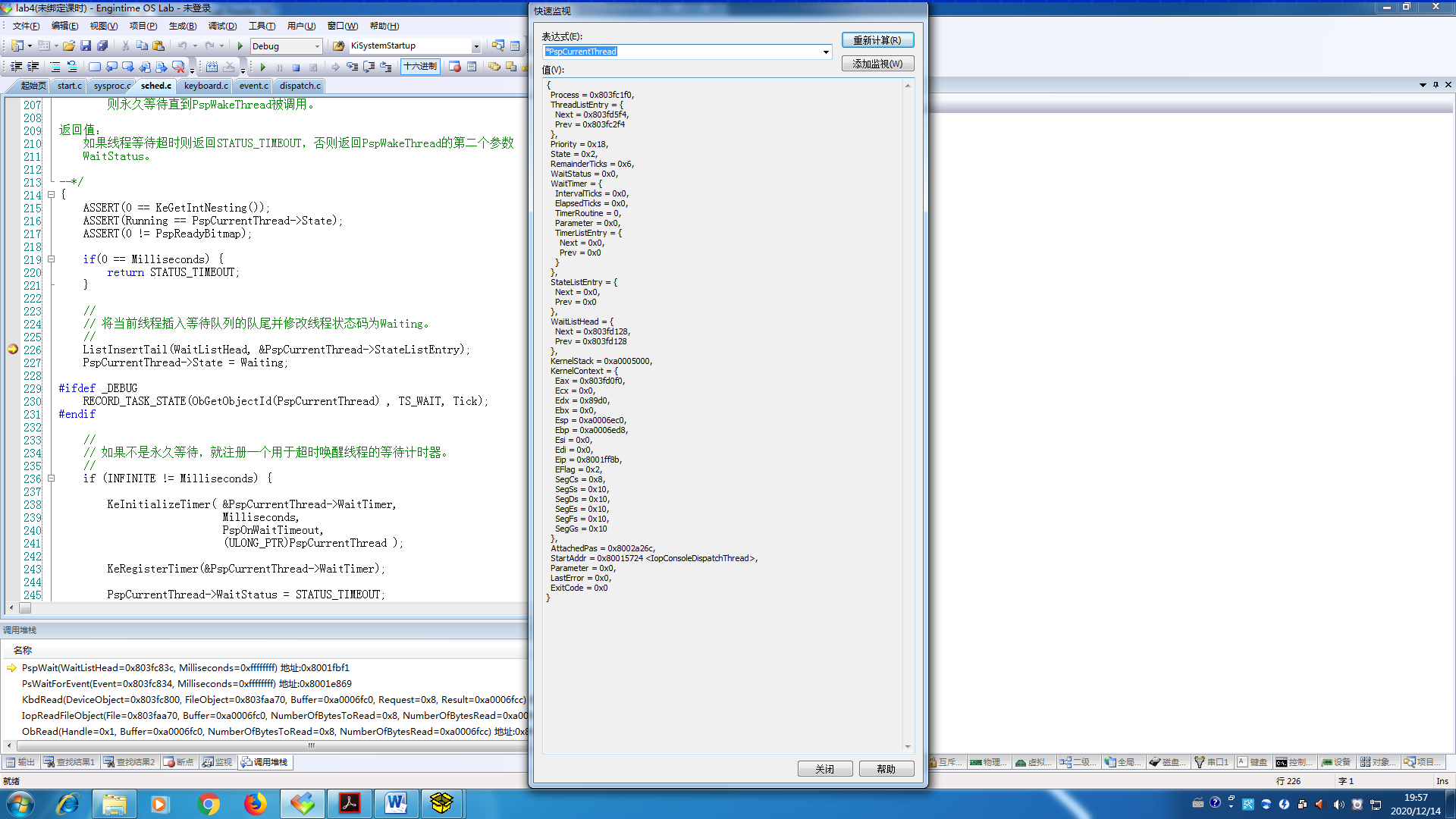


在“调用堆栈”窗口中激活 PspUnreadyThread 函数对应的堆栈项，然后按 F10 单步调试，直到返回 PspSelectNextThread 函数并将线程状态修改为 Running。再从快速监视对话框中查看“\*PspCurrentThread”表达式的值，观察当前占用处理器的线程的情况。其中 State 域的值为 2 （Running），双向链表项 StateListEntry 的 Next 和 Prev 指针的值都为 0，说明控制台派遣线程已经处于运行状态了。接下来，会将该线程的上下文从线程控制块（TCB）复制到处理器的各个寄存器中，处理器就可以从该线程上次停止运行的位置继续运行了。

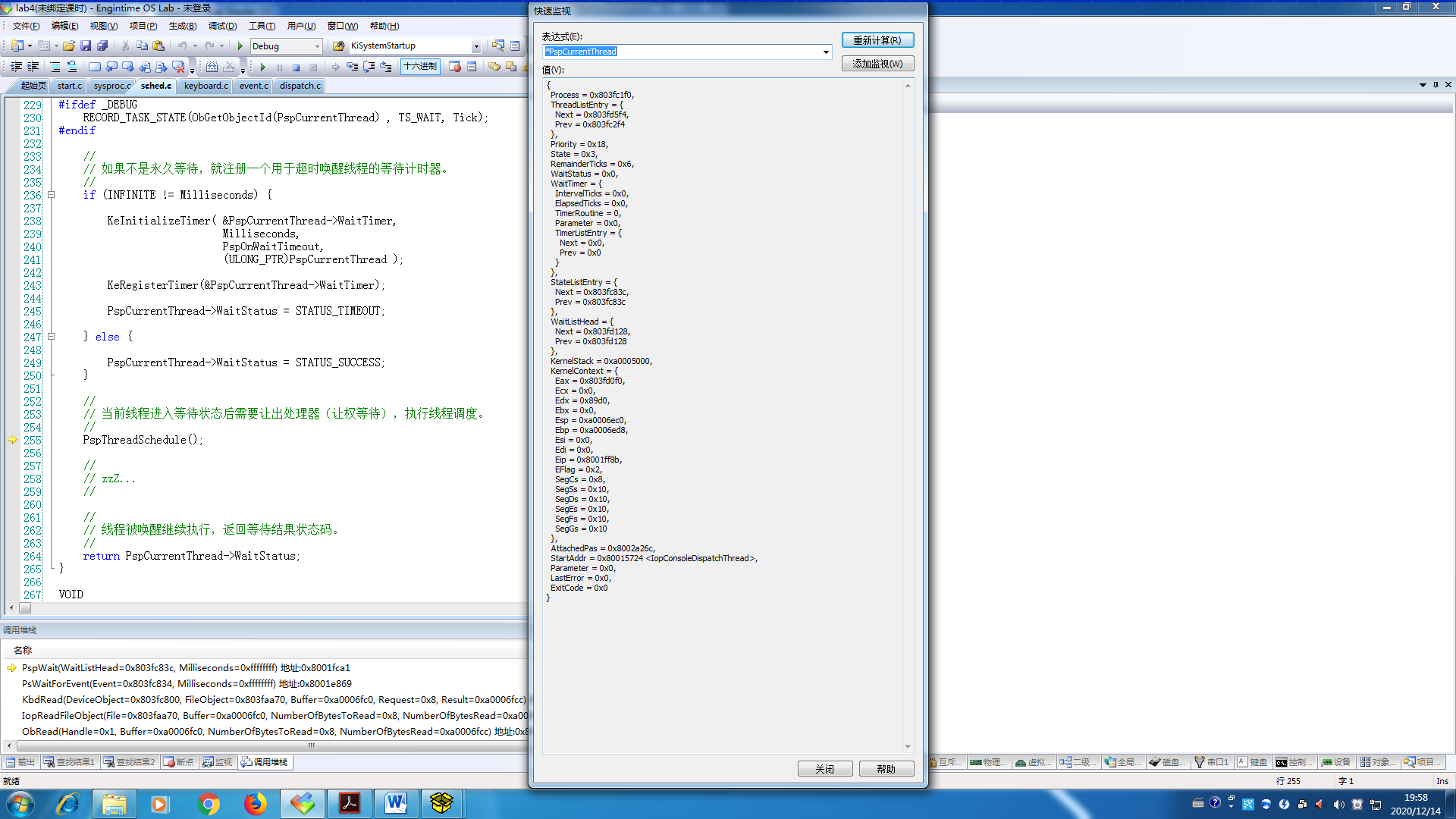


（4）控制台派遣线程由运行状态进入阻塞状态

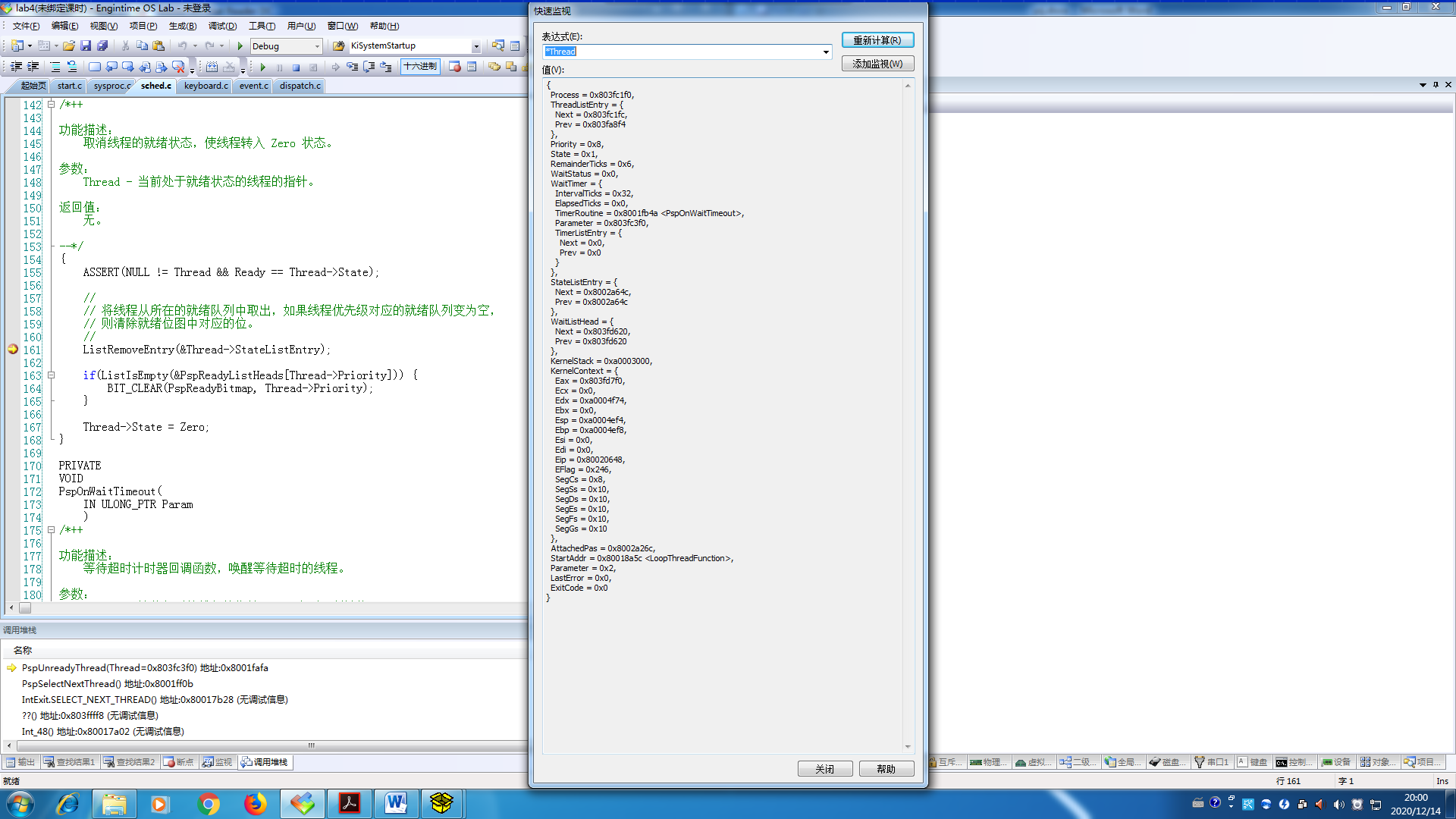
按 F5 继续执行，在 PspWait 函数中的断点处中断。在快速监视对话框中查看“\*PspCurrentThread”表达式的值，观察当前占用处理器的线程的情况。其中 State 域的值为 2（Running），双向链表项 StateListEntry 的 Next 和 Prev 指针的值都为 0，说明这个线程仍然处于运行状态；StartAddr域的值为 IopConsoleDispatchThread，说明这个线程就是控制台派遣线程。



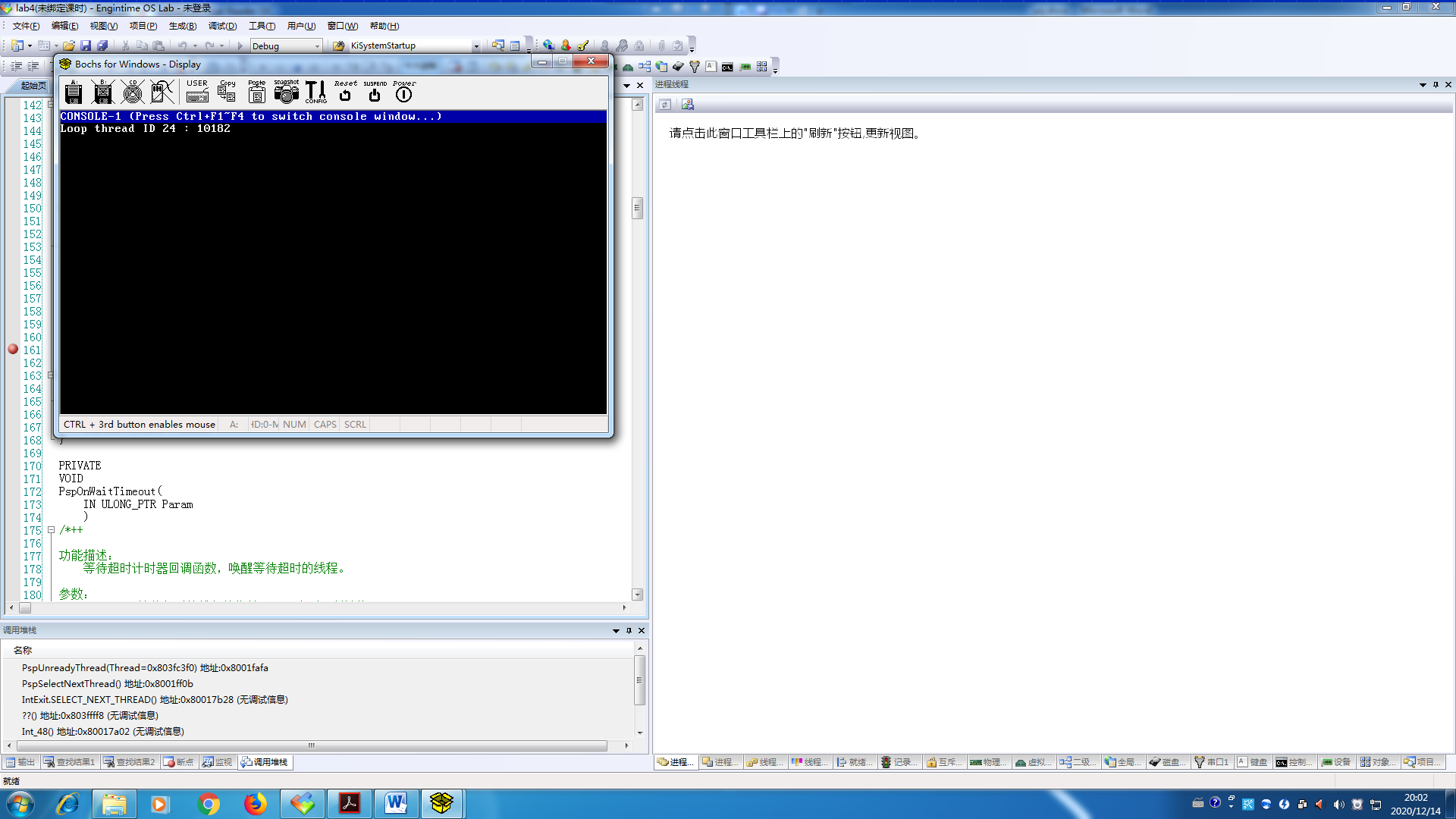
按F10 单步调试，直到左侧的黄色箭头指向代码第 255 行。再从快速监视对话框中查看“\*PspCurrentThread”表达式的值。其中 State 域的值为 3（Waiting），双向链表项StateListEntry 的 Next 和 Prev 指针的值都不为 0，说明控制台派遣线程已经处于阻塞状态了，并在某个同步对象的等待队列中。第 255 行代码可以触发线程调度功能，会中断执行当前已经处于阻塞状态的控制台派遣线程，并将处理器上下文保存到该线程的线程控制块中。



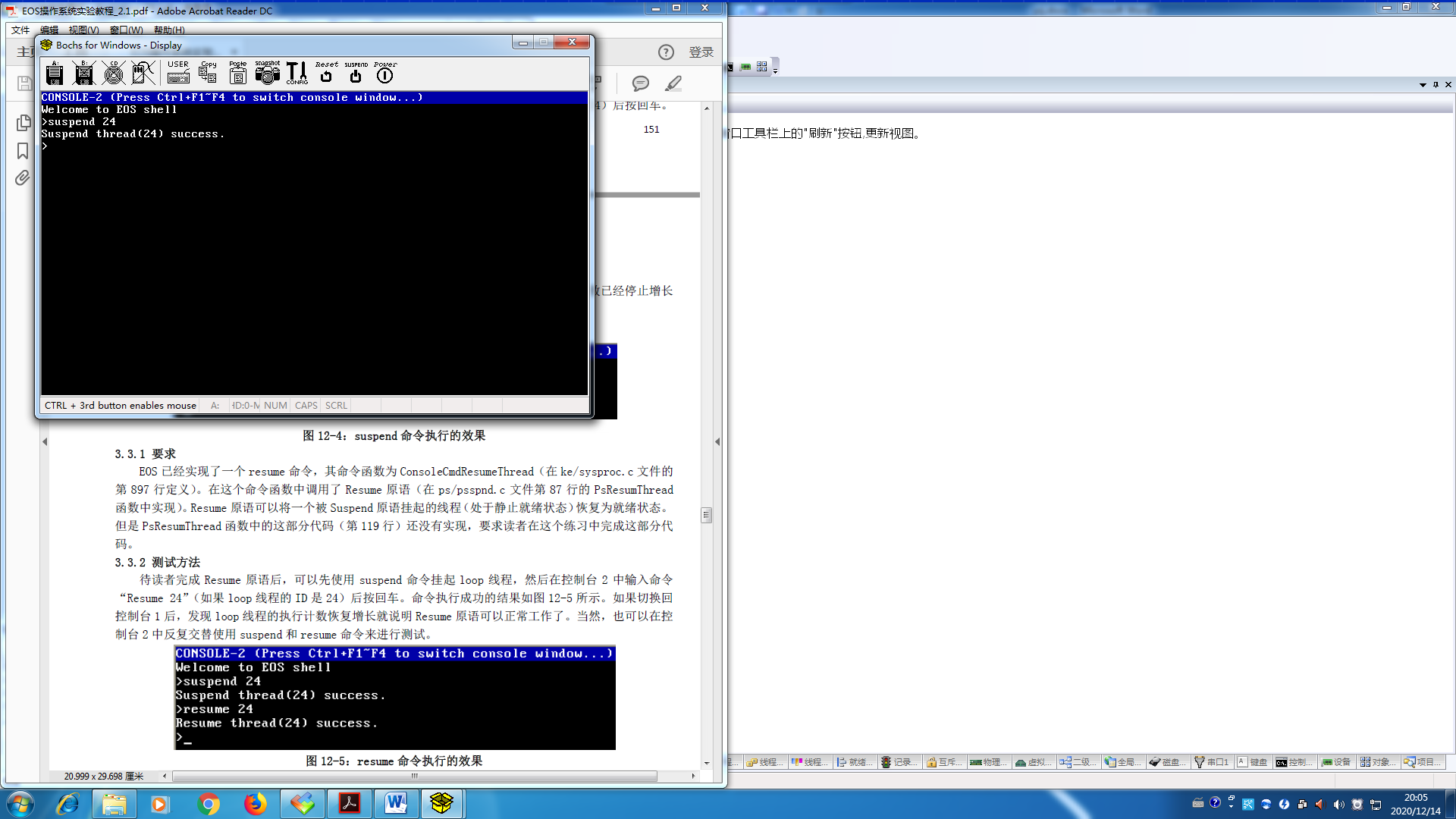
按 F5 继续执行，与本实验 3.2.3 节中的情况相同，只不过这次变为 loop 线程由就绪状态进入运行状态。

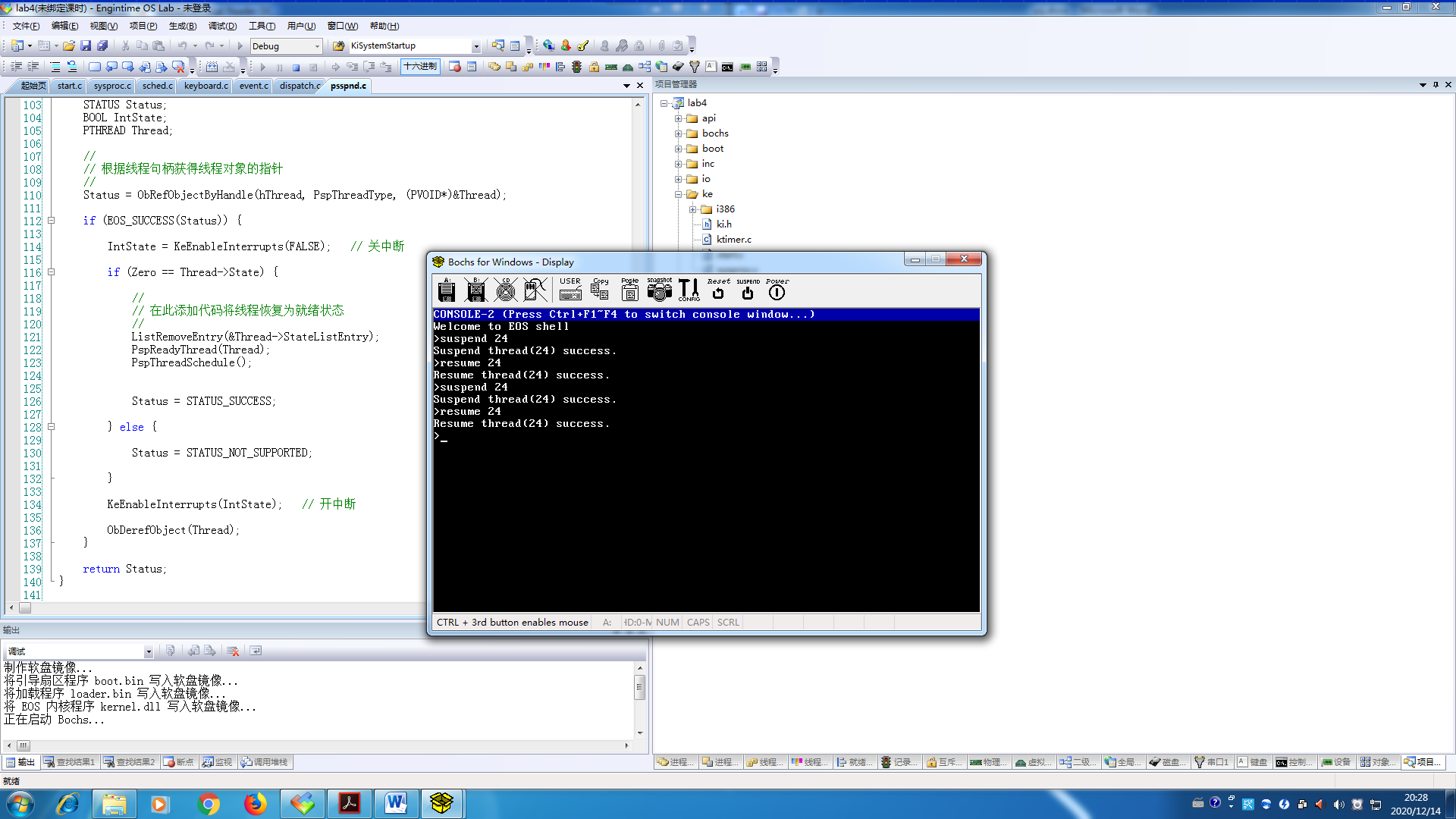


再按 F5 继续执行，EOS 不会再被断点中断。激活虚拟机窗口，可以看到 loop 线程又开始不停的执行死循环了。



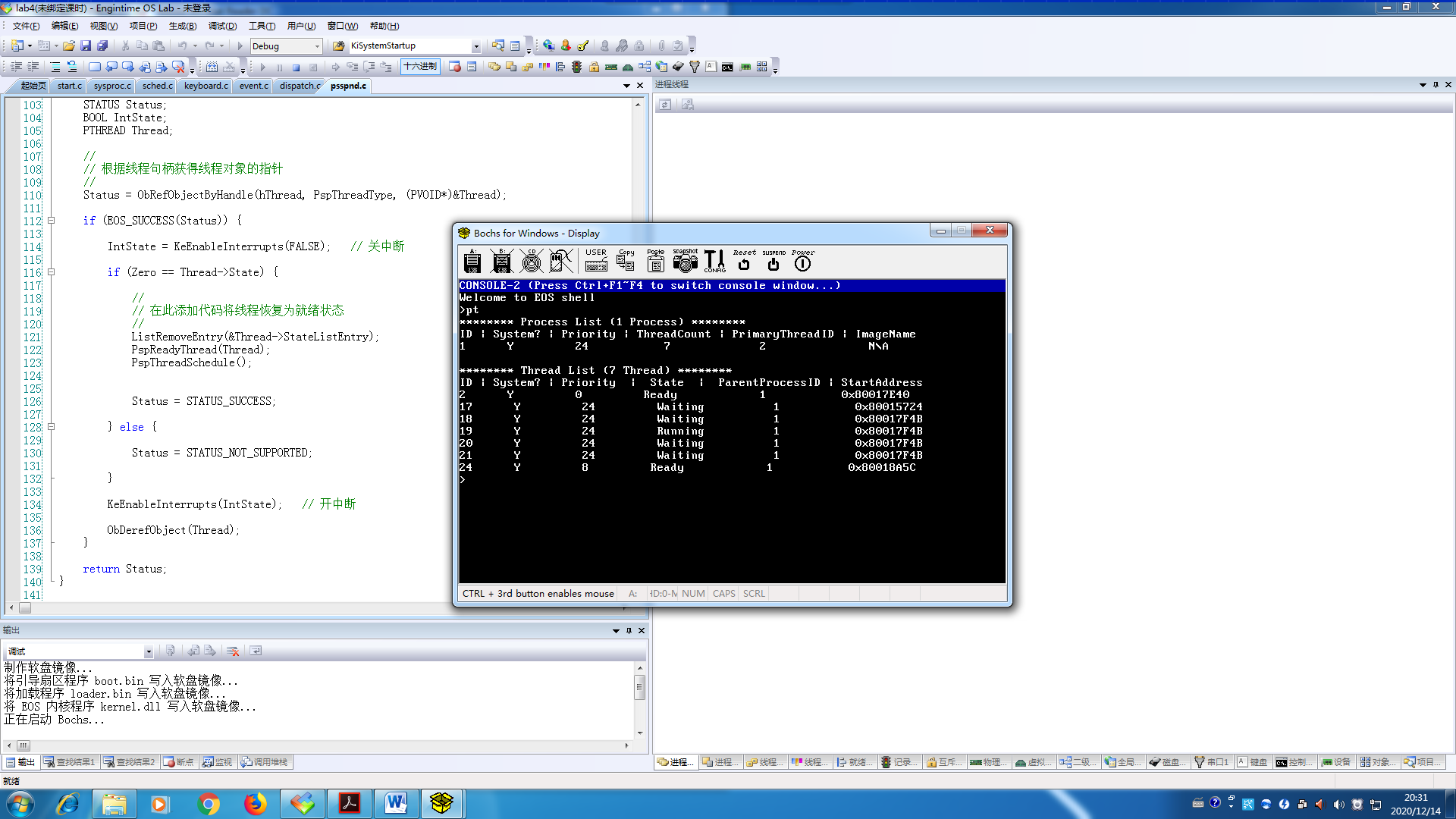
2、为线程增加挂起状态





* 1. **实验的思考与问题分析**

1. 思考一下，在本实验中，当 loop 线程处于运行状态时，EOS 中还有哪些线程，它们分别处于什么状态。可以使用控制台命令 pt 查看线程的状态或者在“进程线程”窗口中查看线程的状态。



1. 当 loop 线程在控制台 1 中执行，并且在控制台 2 中执行 suspend 命令时，为什么控制台 1 中的 loop线程处于就绪状态而不是运行状态？

答:

当在控制台2 中执行suspend命令时，实质上是优先级为24的控制台2线程抢占了处理器，也就是控制台2线程处于运行状态，所以此时loop线程处于就绪状态了。

1. 总结一下在图 5-3 中显示的转换过程，哪些需要使用线程控制块中的上下文（将线程控制块中的上下文恢复到处理器中，或者将处理器的状态复制到线程控制块的上下文中），哪些不需要使用，并说明原因。

答:

一个进程在运行过程中或执行系统调用，或产生了一个中断事件，处理器都进行一次模式切换，操作系统接收控制权，有关系统例程完成必须的操作后，或恢复被中断进程或切换到新进程。当系统调度新进程占有处理器时，新老进程随之发生上下文切换，因此，进程的运行被认为是在进程的上下文中执行，这时的控制权在操作系统手中，它在完成必要的操作后，可以恢复被中断的进程或切换到别的进程。

1. 在本实验 3.2 节中总结的所有转换过程都是分步骤进行的，为了确保完整性，显然这些转换过程是不应该被打断的，也就是说这些转换过程都是原语操作（参见本书第 2.6 节）。请读者找出这些转换过程的原语操作（关中断和开中断）是在哪些代码中完成的。（提示，重新调试这些转换过程，可以在调用堆栈窗口列出的各个函数中逐级查找关中断和开中断的代码。）

答:

IntState=KeEnableInterrupts (FALSE);//关中断

KeEnableInterrupts(IntState);//开中断