

**操作系统实验报告**

**实验题目**  实验3 进程的创建

**学生姓名**  孙淼

**学 号**  2018211958

**专业班级**  计算机科学与技术18 -2班

**指导教师**  田卫东

**完成日期**  11.07

**合肥工业大学 计算机与信息学院**

1. **实验目的和任务要求**

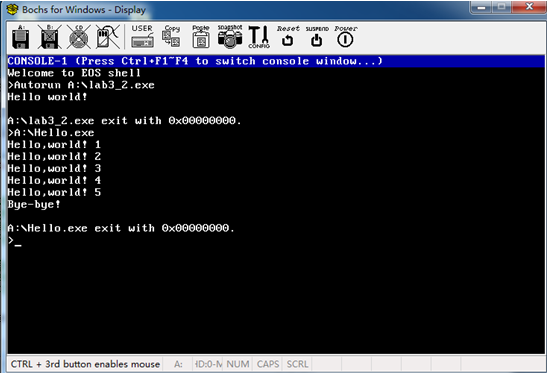
* 练习使用EOS API函数CreateProcess创建一个进程，掌握创建进程的方法,理解进程和程序的区别。
* 调试跟踪CreateProcess函数的执行过程，了解进程的创建过程，理解进程是资源分配的基本单位。
* 调试跟踪CreateThread函数的执行过程，了解线程的创建过程，理解线程是调度的基本单位。

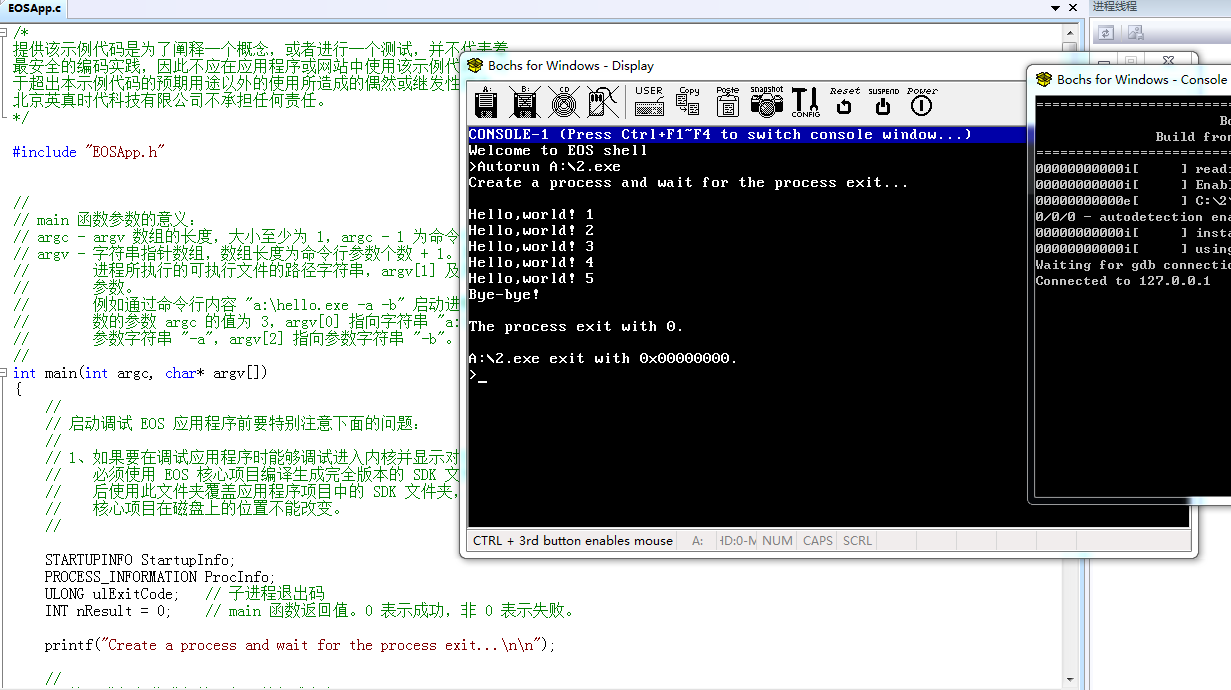
1. **实验原理**

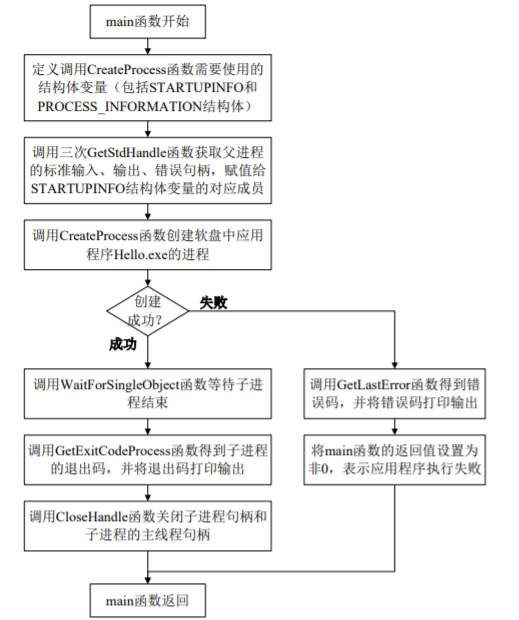
阅读本书第5.1节，重点理解程序和进程的关系，熟悉进程控制块结构体以及进程创建的过程。学习CreateProcess函数和其它与创建进程相关的函数的说明，注意理解这些函数的参数和返回值的含义。

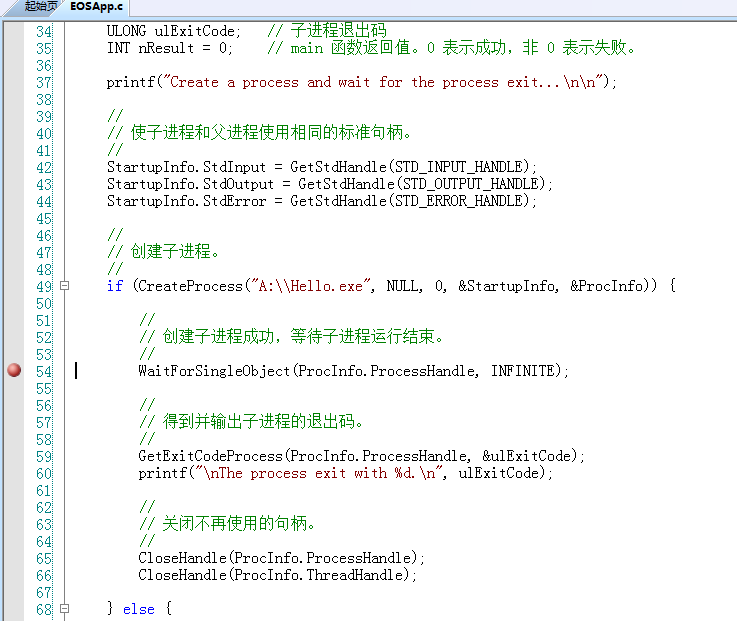
阅读本书第5.2节，熟悉线程控制块结构体以及线程创建的过程。学习 CreateThread 函数,注意理解该函数参数和返回值的含义。

1. **实验内容**

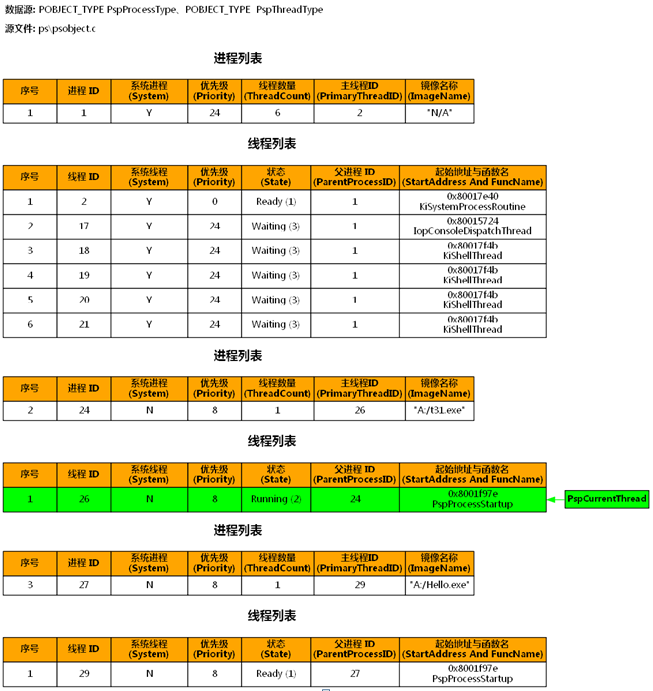
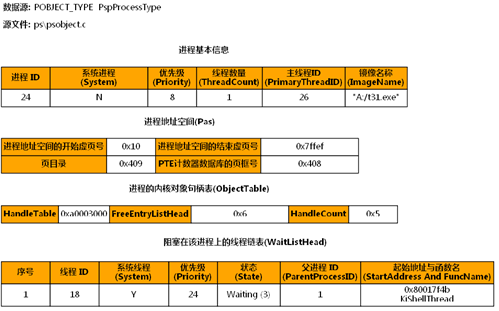
使用控制台命令创建EOS应用程序的进程，启动调试，Hello.exe 应用程序开始执行

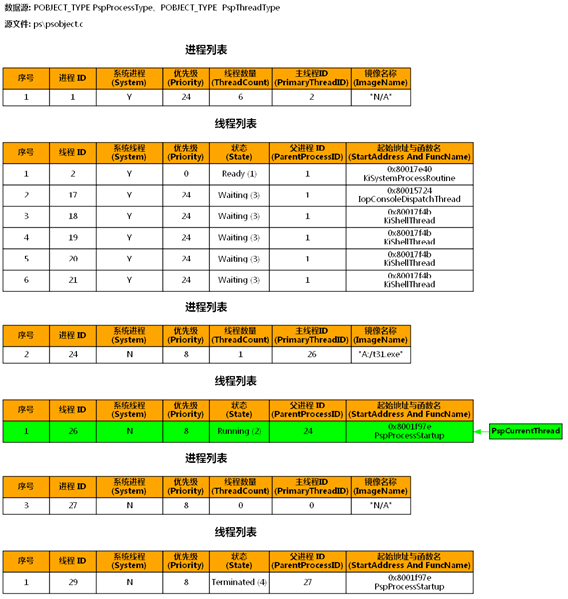
查看让一个应用程序创建另一个应用程序的进程的执行结果

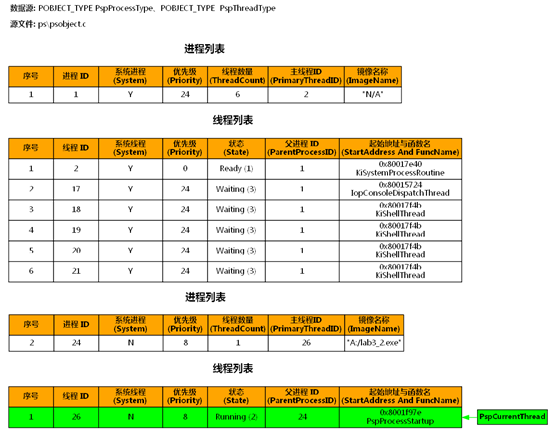
根据教程书本知识我们知道，该main函数流程图如下：

接下来我们从应用程序的角度理解进程的创建过程

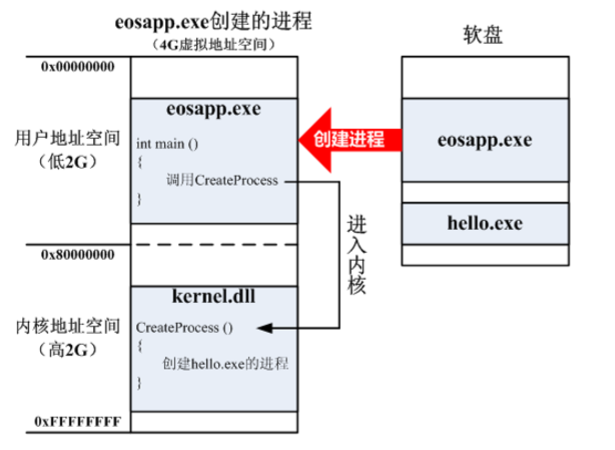
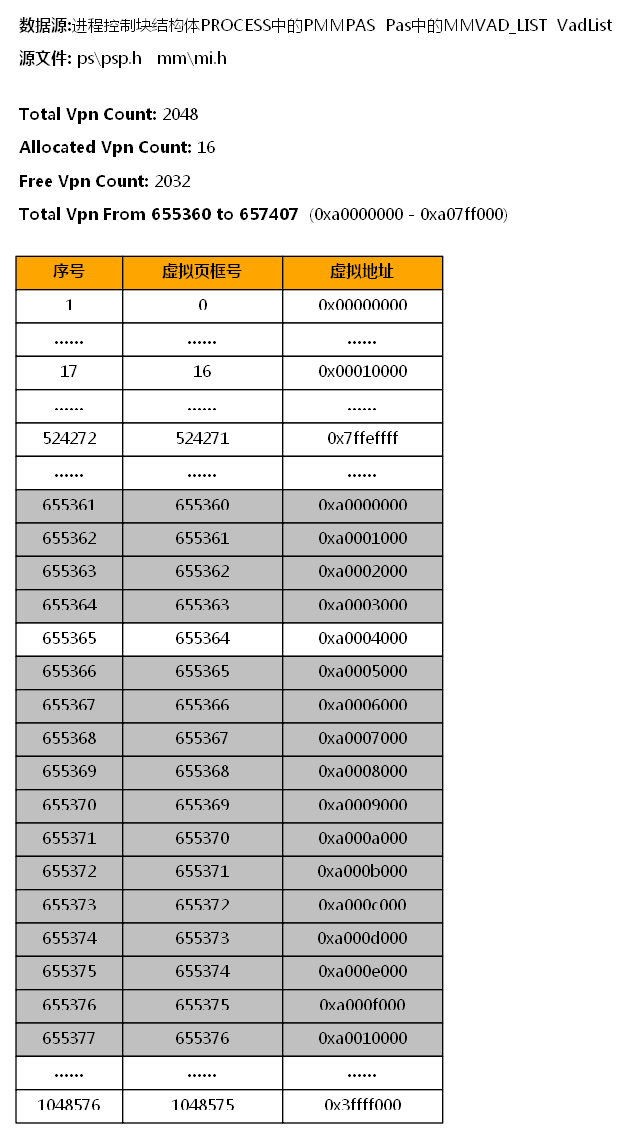
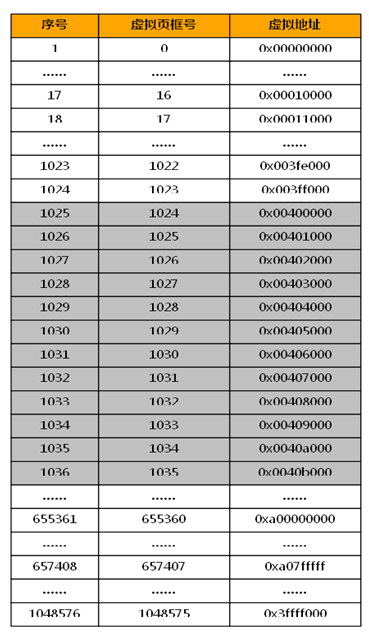
创建两个应用程序进程后的进程线程列表，可以看到当前系统中有三个进程，其中第一个是系统进程，镜像名称为空；另外两个是刚刚创建的应用程序的进程，镜像名称分别为“A:\EOSApp.exe”（会根据读者创建的项目名称变化）和“A:\Hello.exe”。其中，A:\EOSApp.exe进程是父进程，其主线程处于运行状态（正准备调用WaitForSingleObject函数，但是命中了断点）；A:\Hello.exe进程是子进程，其主线程处于就绪状态，当父进程的主线程通过调用WaitForSingleObject函数进入阻塞状态让出处理器后，这个处于就绪状态的线程就会占用处理器开始运行。这两个应用程序的进程和线程的优先级都为8。

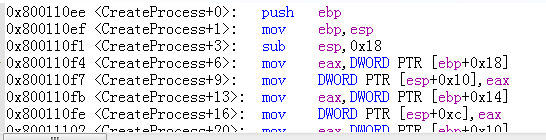
控制台线程阻塞在应用程序的进程上

按F10单步执行WaitForSingleObject函数，再次刷新进程线程窗口，确认子进程已经结束运行

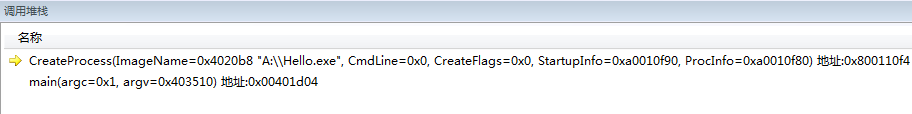
从内核的角度理解进程的创建过程，应用程序进程，创建了一个线程，并处于运行态，也就是在其准备调用 CreateProcess 函数时命中了断点。

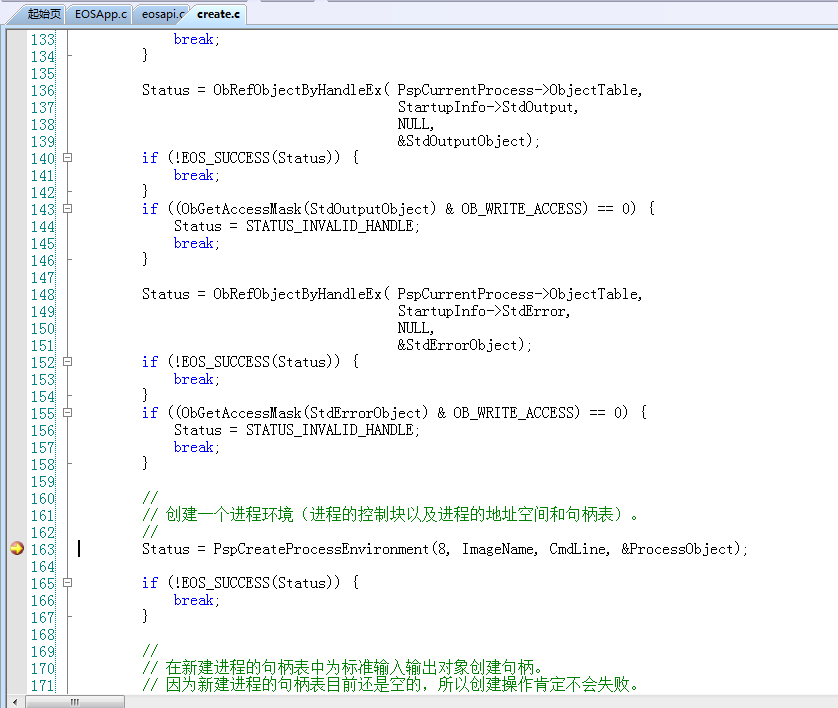
进入 CreateProcess 函数

验证一下图 11-7 所示的应用程序和操作系统内核在进程的 4G 虚拟地址空间中所处的位置：选择“调试”菜单“窗口”中的“虚拟地址描述符”菜单项，打开“虚拟地址描述符” 窗口，从该窗口的工具栏下拉框中选择“进程控制块 PID=1”选项，此时显示的是系统进程的虚拟地址描述符表（如图 11-8 所示），其描述的就是系统进程动态管理的存储空间，用于存储内核管理的数据。可以看到，已经使用的虚拟地址空间（底色为灰色的表示已使用）都大于 0x80000000。

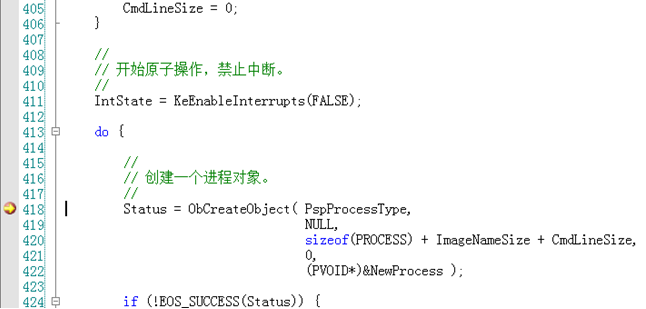
查看 EOS 应用程序项目链接器属性中基址的值

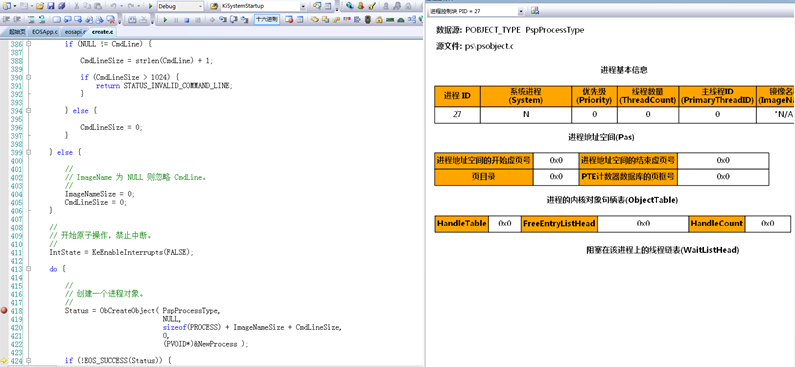
查看当前系统物理内存的使用情况

观察 eosapi.c 文件中 CreateProcess 函数的源代码，可以看到此函数只是调用了 EOS 内核函数 PsCreateProcess 并将创建进程所用到的参数传递给了此函数。所以，此时读者可以按 F11 调试进入create.c 文件中的 PsCreateProcess 函数，在此函数中才真正开始执行创建进程的各项操作。

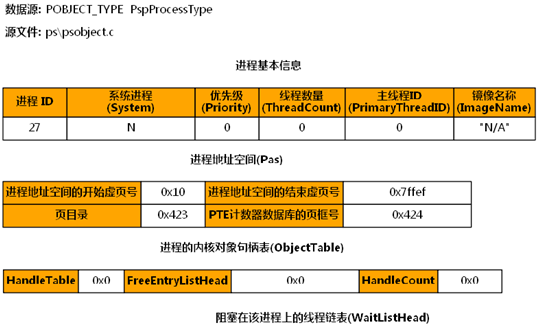
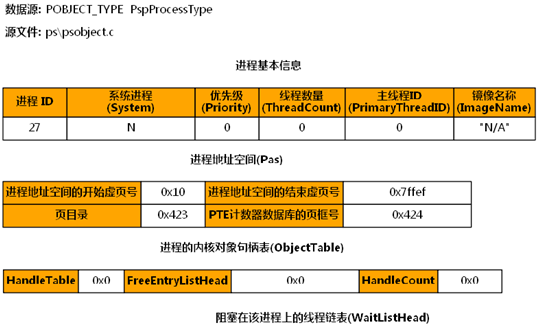
调试 PsCreateProcess 函数，在 PsCreateProcess 函数中找到调用 PspCreateProcessEnvironment 函数的代码行（create.c 文件的第 163 行），并在此行添加一个断点。

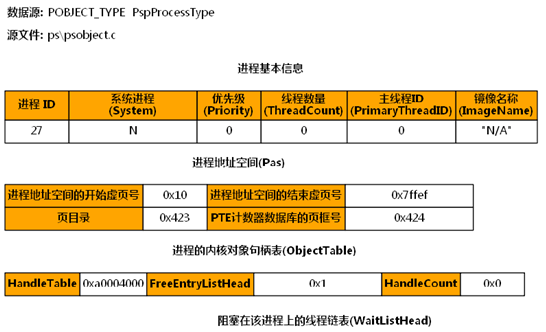
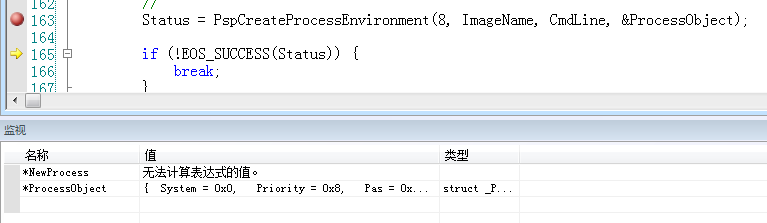
在此函数中查找到创建进程控制块的代码行

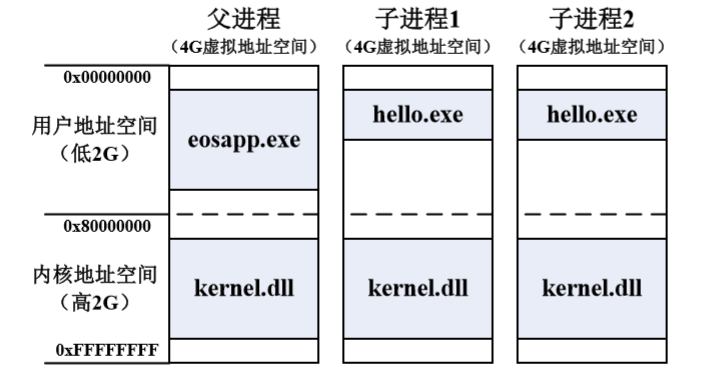
这里的 ObCreateObject 函数会在由 EOS 内核管理的内存中创建了一个新的进程控制块（也就是分配了一块内存），并由 NewProcess 返回进程控制块的指针（也就是返回所分配内存的起始地址）

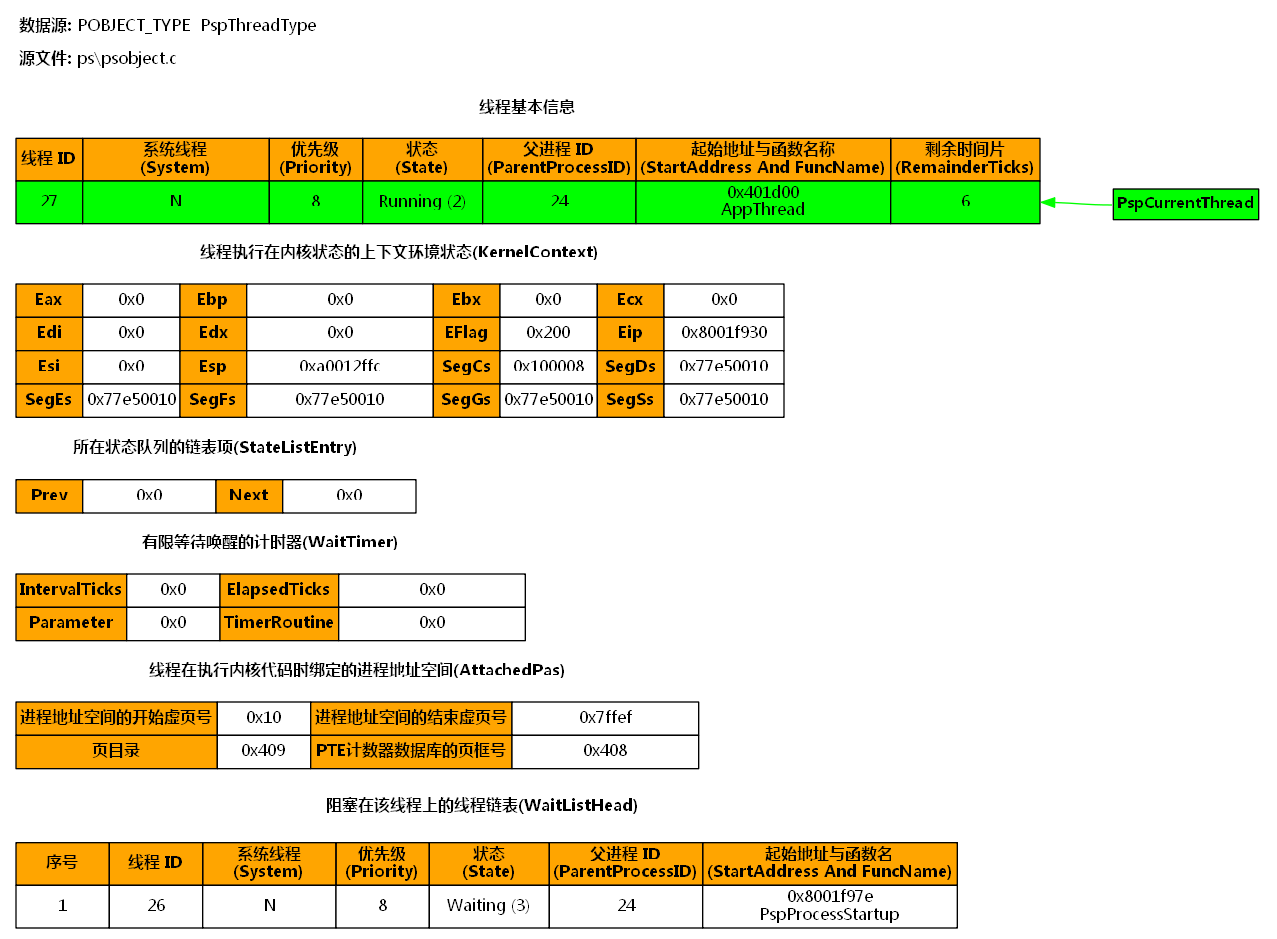
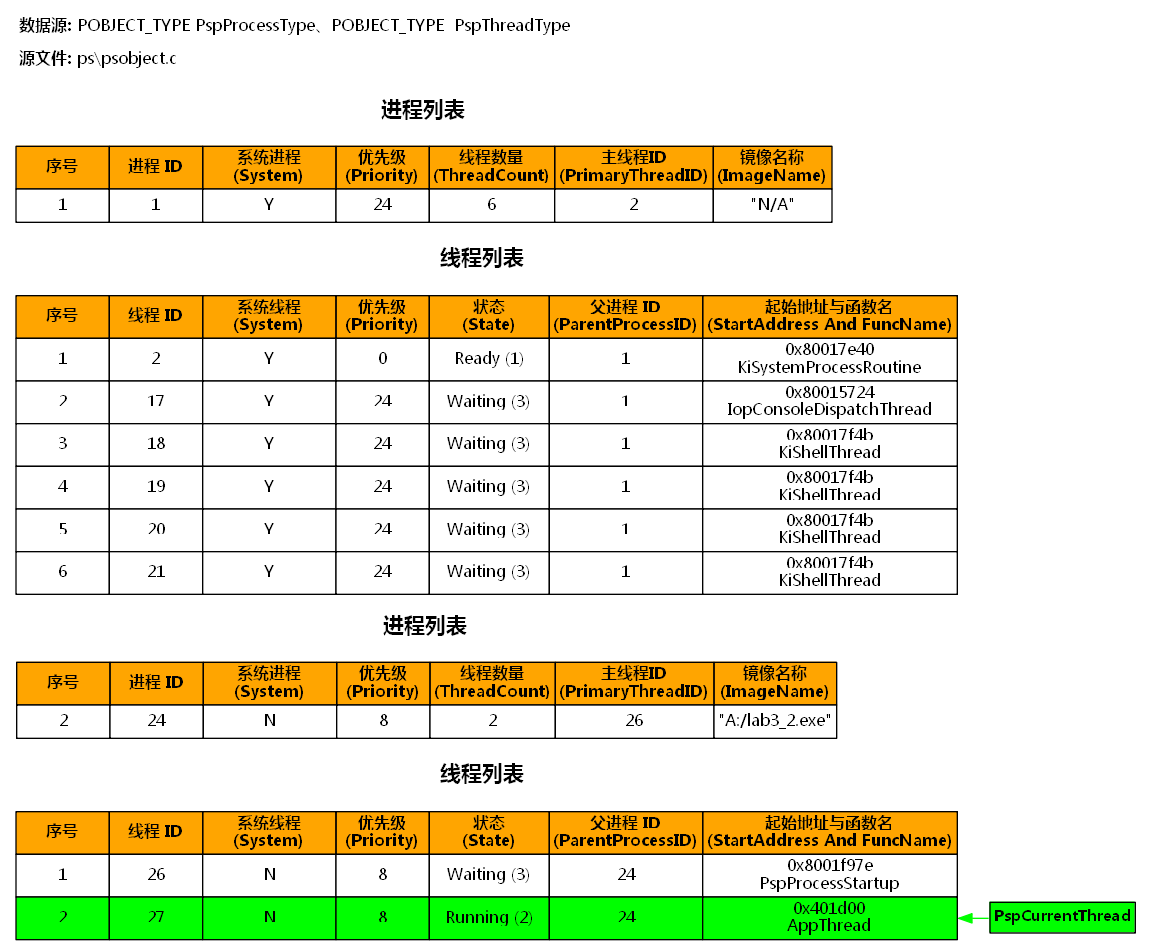
由于目前只是新建了进程控制块，还没有为其中的成员变量赋值，所以值都为0。

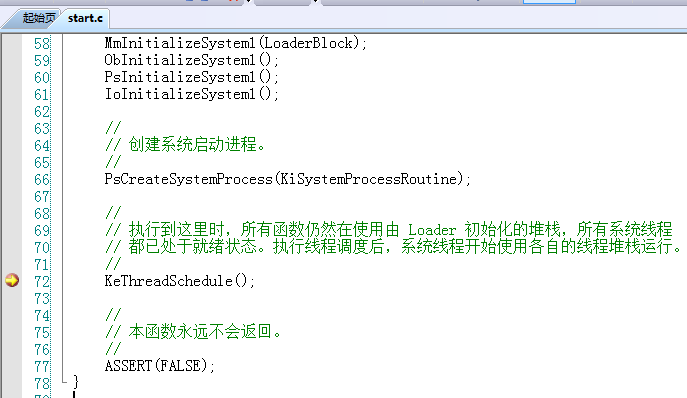
调试初始化进程控制块中各个成员变量的过程：可以看到该进程控制块的“进程地址空间”的值已经不再是 0。说明已经初始化了进程的 4G 虚拟地址空间。还可以在“监视”窗口中查看进程控制块的成员变量 Pas 的值已经不再是 0。

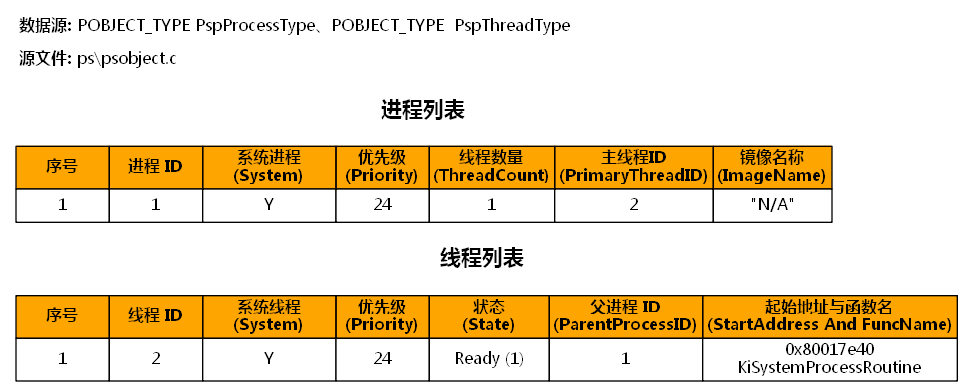
使用 F10 一步步调试 PspCreateProcessEnvironment 函数中后面的代码，在调试的过程中根据执行的源代码，可以在“进程控制块”窗口中查看“进程控制块 PID=27”的信息，或者在“监视” 窗口中查看\*NewProcess 表达式的值，观察进程控制块中哪些成员变量是被哪些代码初始化的， 哪些成员变量还没有被初始化。

将表达式\*ProcessObject 添加到“监视”窗口中就可以继续观察新建进程控制块中的信息。

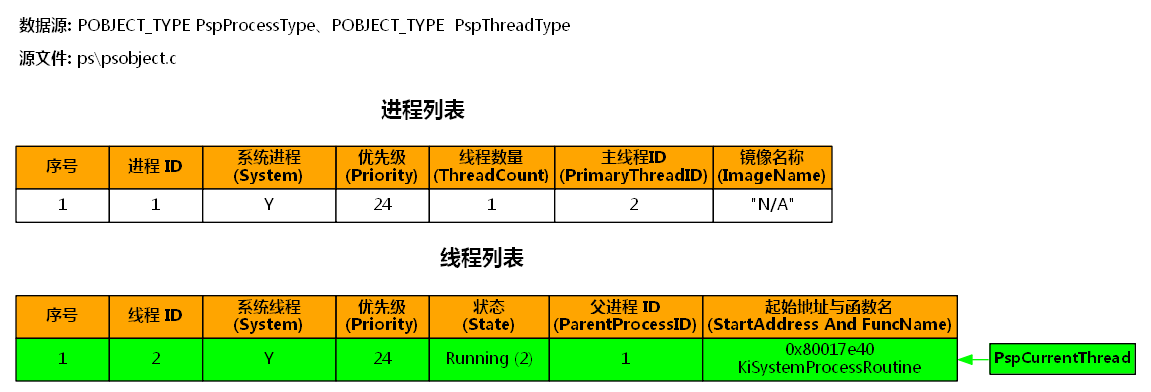
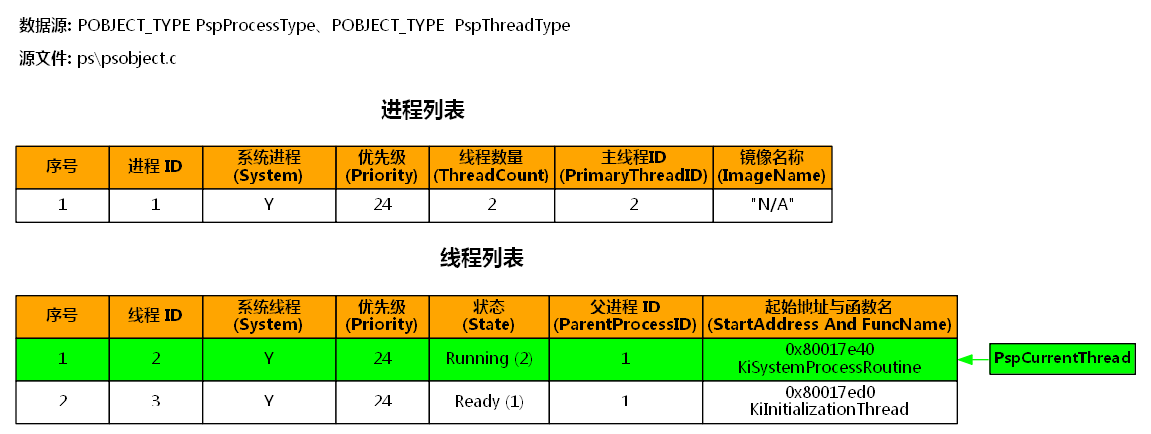
通过编程的方式创建一个应用程序的多个进程, 多个进程并发时，EOS 操作系统中运行的用户进程可以参见图 11-12，从而验证一个程序（hello.exe） 可以同时创建多个进程。

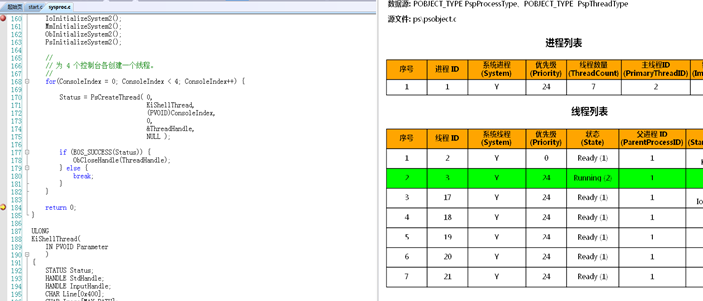
在一个应用程序进程中创建一个工作线程, 在“线程控制块”窗口工具栏的组合框中选择“线程控制块 TID=27”的选项，可以查看工作线程的详细信息，如图 11-14 所示，包括：线程的基本信息、线程执行在内核状态的上下文环境状态、所在状态队列的链表项（由于该线程处于运行态，也就不在任何状态队列中，所以 Prev 和 Next 指针的值都为 0）、有限等待唤醒的计时器、线程在执行内核代码时绑定的地址空间、阻塞在该线程上的线程链表。其中，阻塞在该线程上的线程链表中显示了应用程序的主线程阻塞在该工作线程上，也就是主线程在等待该工作线程结束。

系统线程的创建过程, 通过调试 EOS 内核的初始化过程来理解几个重要的系统线程的创建过程，这些线程包括：系统初始化线程、控制台派遣线程、控制台线程。

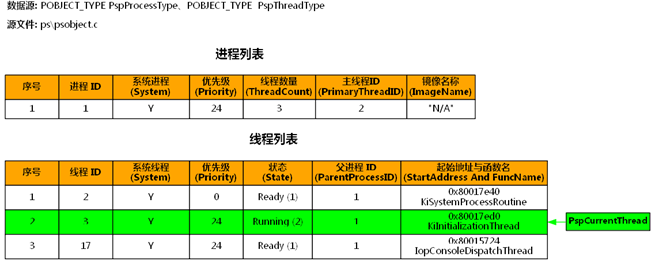
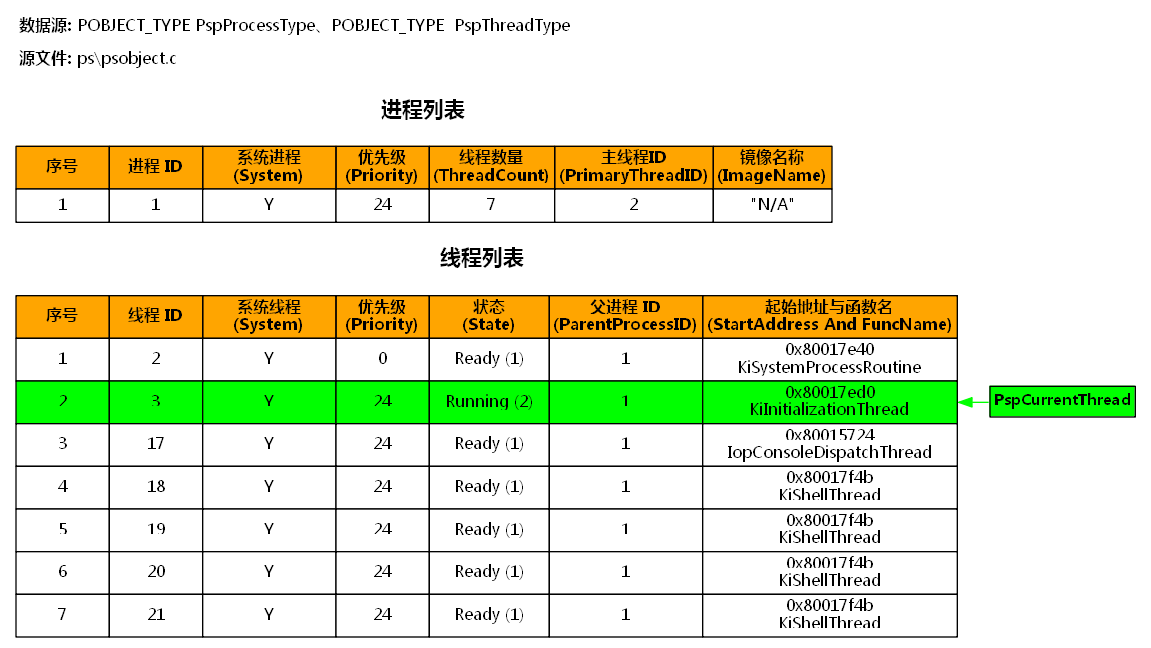
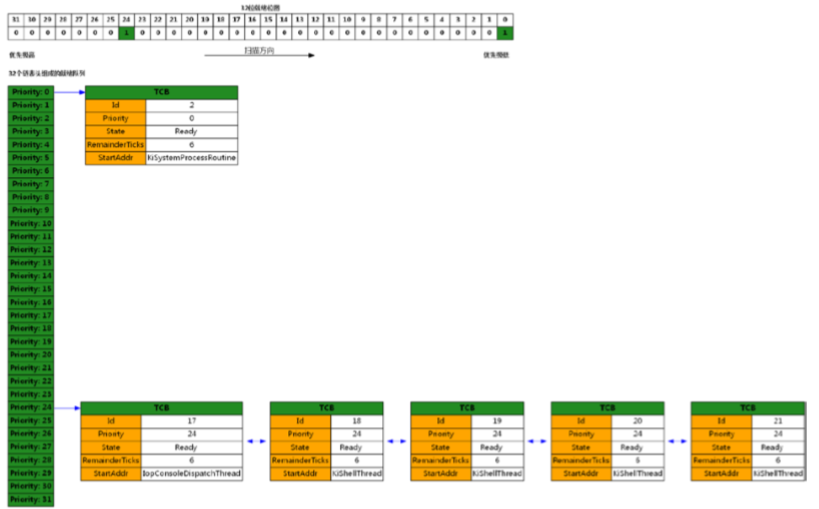
启动调试，在断点处中断。刷新“进程线程”窗口，显示如图 11-15 所示的内容。可以看到当前系统中，只创建了一个线程， 该线程就是由第 66 行的PsCreateSystemProcess 函数在创建系统进程后，为系统进程创建的第一个子线程，同时也是系统进程的主线程。

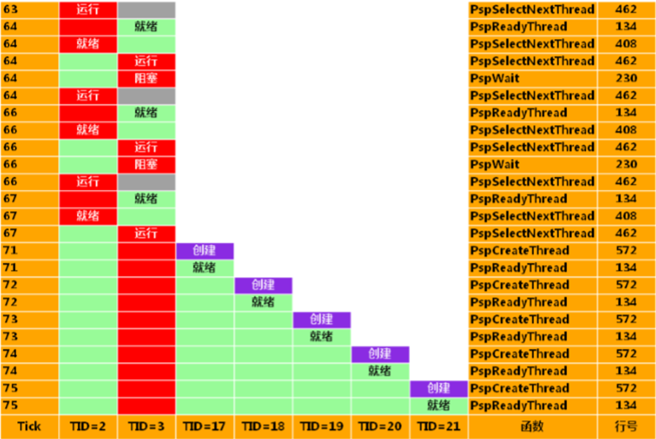
继续调试，在断点处中断。刷新“进程线程”窗口，可以看到系统中唯一的线程处于运行状态，说明该线程正在执行其线程函数并命中了断点。

当前中断位置处的代码行用于创建一个新的系统线程，并使用该新建线程继续进行操作系统的初始化工作。所以，按 F10 单步调试一次后，刷新“进程线程”窗口，会显示如图 11-16 所示的内容，可以看到当前系统中又多出了一个线程，并处于就绪状态。

由于第一个系统线程会使用第 140 行的代码将自己的优先级降为 0，退化为空闲线程。这就会让刚刚新建的还处于就绪状态的系统初始化线程抢占处理器开始运行。接下来，在初始化线程的线程函数中添加一个断点：打开 ke\sysproc.c 文件，在 KiInitializationThread 函数中（第 160 行代码处）添加一个断点。运行到断点处。

刷新“进程线程” 窗口，显示如图 11-17 所示的内容，可以看到当前系统中已经创建了控制台派遣线程，并处于就绪状态。请读者自行查找创建控制台派遣线程的代码位置，并尝试说明其函数调用层次。

创建控制台派遣线程，由于第 168 行的 for 语句会循环创建四个控制台线程，所以，读者可以在 184 行代码处添加一个断点，按 F5 继续调试。待命中断点后，刷新“进程线程”窗口，会显示如图 11-18 所示的内容， 可以看到当前系统已经完成了四个控制台线程的创建，并且所有的控制台线程都处于就绪状态。

打开“线程运行轨迹”窗口。点击此窗口工具栏上的“刷新”按钮，会显示如图 11-20 所示的内容，可以查看这些系统线程的状态转换过程和运行轨迹。注意，系统初始化线程（TID=3）在初始化的过程中会由于等待一些硬件设备的响应，从而频繁进入阻塞状态。

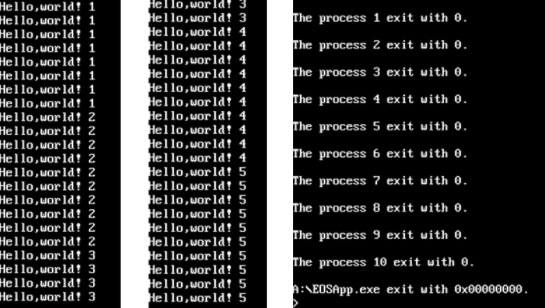
至此，系统初始化线程（TID=3）已经完成其所有的工作，在其结束运行后，处于就绪状态的控制台派遣线程和四个控制台线程会依次获得处理器，但是控制台派遣线程会在等待键盘事件时进入阻塞状态， 四个控制台线程会在等待控制台派遣线程分派任务时进入阻塞状态，当它们都进入阻塞状态后，低优先级的系统空闲线程（TID=2）就会占用处理器开始运行，直到有更高优先级的线程抢占处理器为止。读者可以在 ke/sysproc.c 文件的空闲线程的死循环中（第 143 行）添加一个断点，按 F5 继续调试，会在断点处中断。然后刷新“线程运行轨迹”窗口或其它感兴趣的可视化窗口，查看系统线程运行的情况。

1. **实验的思考与问题分析**

（1）在源代码文件NewTwoProc.c提供的源代码基础上进行修改，要求使用hello.exe同时创建10个进程。提示：可以使用PROCESS\_INFORMATION类型定义一个有10个元素的数组，每一个元素对应一个进程。使用一个循环创建10个子进程，然后再使用一个循环等待10个子进程结束，得到退出码后关闭句柄。

答：

修改代码如下：

1. **if** (CreateProcess("A:\\Hello.exe", NULL, 0, &StartupInfo, &ProcInfoOne)
2. && CreateProcess("A:\\Hello.exe", NULL, 0, &StartupInfo, &ProcInfoTwo)
3. && CreateProcess("A:\\Hello.exe", NULL, 0, &StartupInfo, &ProcInfo3)……
4. && CreateProcess("A:\\Hello.exe", NULL, 0, &StartupInfo, &ProcInfo10)) {
5. WaitForSingleObject(ProcInfoOne.ProcessHandle, INFINITE);
6. WaitForSingleObject(ProcInfoTwo.ProcessHandle, INFINITE);
7. WaitForSingleObject(ProcInfo3.ProcessHandle, INFINITE);……
8. WaitForSingleObject(ProcInfo10.ProcessHandle, INFINITE);
9. GetExitCodeProcess(ProcInfoOne.ProcessHandle, &ulExitCode);……
10. GetExitCodeProcess(ProcInfo3.ProcessHandle, &ulExitCode);
11. printf("\nThe process 3 exit with %d.\n", ulExitCode);……
12. GetExitCodeProcess(ProcInfo10.ProcessHandle, &ulExitCode);
13. printf("\nThe process 10 exit with %d.\n", ulExitCode);
14. CloseHandle(ProcInfoOne.ProcessHandle);……
15. CloseHandle(ProcInfoTwo.ThreadHandle);
16. CloseHandle(ProcInfo3.ProcessHandle);
17. CloseHandle(ProcInfo3.ThreadHandle);……
18. CloseHandle(ProcInfo10.ProcessHandle);
19. CloseHandle(ProcInfo10.ThreadHandle);

（2）学习本书第5章中的5.2节，了解关于线程的相关知识，然后尝试调试PspCreateThread函数，并在“线程控制块”窗口中观察线程控制块（TCB）初始化的过程。

答：

（3）sCreateProcess函数中调用了PspCreateProcessEnvironment函数后又先后调用了PspLoadProcessImage和PspCreateThread函数，学习这些函数的主要功能。能够交换这些函数被调用的顺序吗？思考其中的原因。

答:

PspCreateProcessEnvironment创建了地址空间和分配了句柄表。PspLoadProcessImage是将进程的可执行映像加载到了进程的地址空间中。PspCreateThread创建了进程的主线程。这三个函数知道自己要从哪里开始执行，执行哪些指令。这三个函数被调用的顺序是不能够改变的就向上面描述的加载可执行映像之前必须已经为进程创建了地址空间这样才能够确定可执行映像可以被加载到内存的什么位置在创建主线程之前必须已经加载了可执行映像这样主线程才能够知道自己要从哪里开始执行，执行哪些指令。因此不能交换他们的顺序。

1. **总结和感想体会**

本次实验主要练习使用EOSAPI函数CreateProcess创建一个进程，掌握创建进程的方法，理解进程和程序的区别。以及调试跟踪CreateProcess函数的执行过程，了解进程的创建过程，理解进程是资源分配的单位。通过本次实验了解了了解操作系统中进程与线程的区别。通过对进程以及线程进行不同的创建方式，观察启动之后的区别，从而理解操作系统启动后的工作方式。通过一步步的操作，加深了在操作系统上对进程与线程的创建、调试程序的掌握。

**参考文献**

[1]北京英真时代科技有限公司[DB/CD].http://www.engintime.com.

[2]汤子瀛，哲凤屏，汤小丹。计算机操作系统。西安：西安电子科技大学出版社，1996.