

合肥工业大学

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

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。

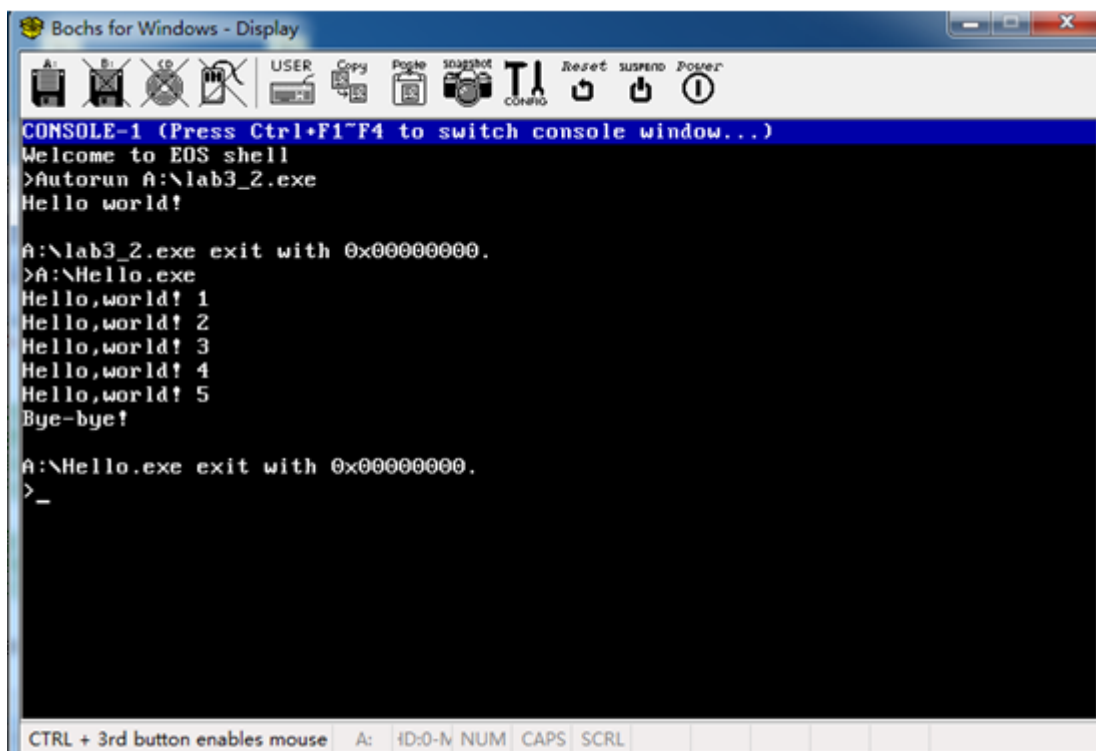
2. 实验原理

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。

3. 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

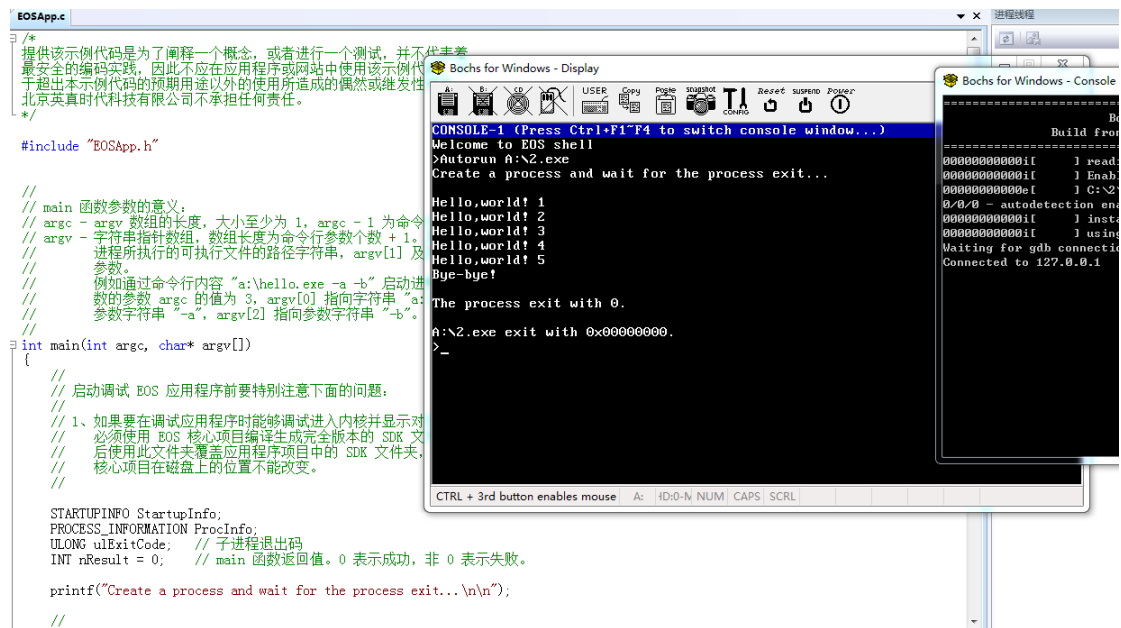


```
Bochs for Windows - Display
A: B: CD USER Copy Paste Snapshot T1 Reset suspend Power
CONFIG
CONSOLE-1 (Press Ctrl+F1~F4 to switch console window...)
Welcome to EOS shell
>Autorun A:\lab3_2.exe
Hello world!

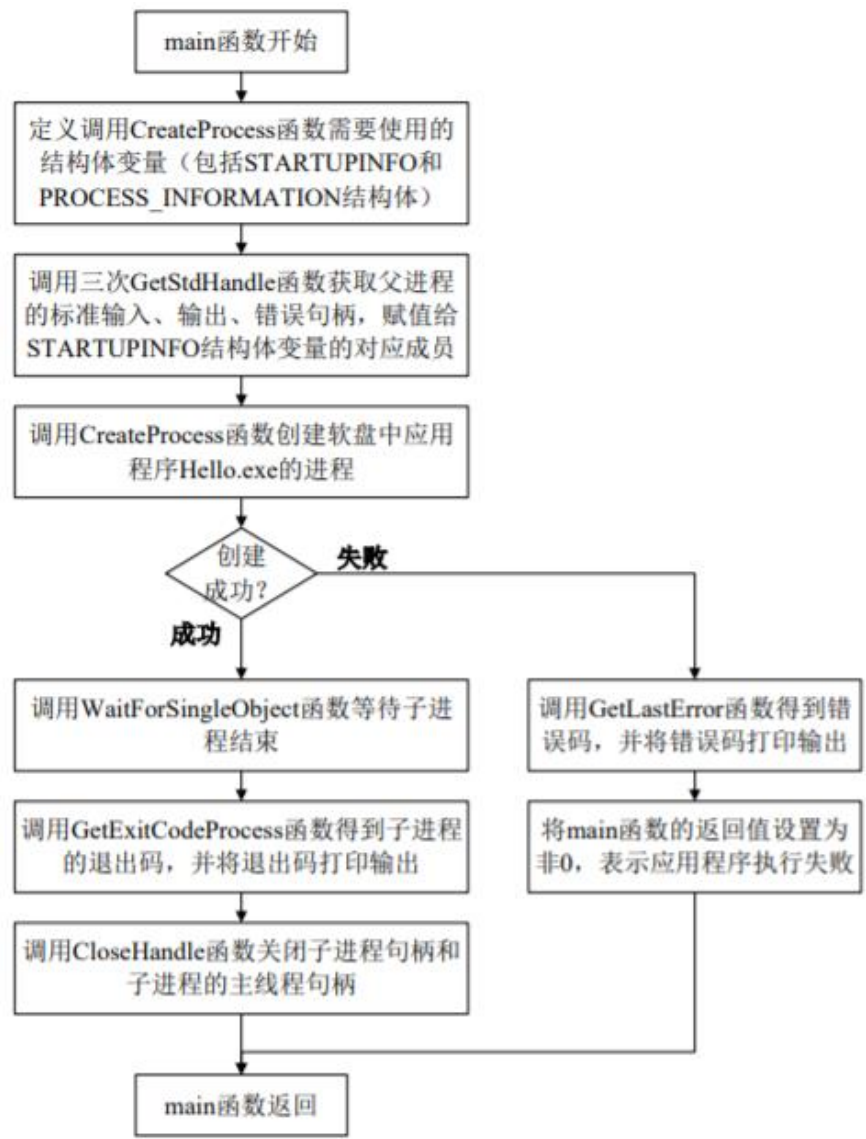
A:\lab3_2.exe exit with 0x00000000.
>A:\Hello.exe
Hello,world! 1
Hello,world! 2
Hello,world! 3
Hello,world! 4
Hello,world! 5
Bye-bye!

A:\Hello.exe exit with 0x00000000.
>_
CTRL + 3rd button enables mouse  A: 4D:0-N NUM CAPS SCRL
```

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

```

34  ULONG ulExitCode; // 子进程退出码
35  INT nResult = 0; // main 函数返回值。0 表示成功，非 0 表示失败。
36
37  printf("Create a process and wait for the process exit...\n\n");
38
39  //
40  // 使子进程和父进程使用相同的标准句柄。
41  //
42  StartupInfo.StdInput = GetStdHandle(STD_INPUT_HANDLE);
43  StartupInfo.StdOutput = GetStdHandle(STD_OUTPUT_HANDLE);
44  StartupInfo.Stderr = GetStdHandle(STD_ERROR_HANDLE);
45
46  //
47  // 创建子进程。
48  //
49  if (CreateProcess("A:\\Hello.exe", NULL, 0, &StartupInfo, &ProcInfo)) {
50
51      //
52      // 创建子进程成功，等待子进程运行结束。
53      //
54      WaitForSingleObject(ProcInfo.ProcessHandle, INFINITE);
55
56      //
57      // 得到并输出子进程的退出码。
58      //
59      GetExitCodeProcess(ProcInfo.ProcessHandle, &ulExitCode);
60      printf("\nThe process exit with %d.\n", ulExitCode);
61
62      //
63      // 关闭不再使用的句柄。
64      //
65      CloseHandle(ProcInfo.ProcessHandle);
66      CloseHandle(ProcInfo.ThreadHandle);
67
68  } else {

```

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。

数据源: POBJECT_TYPE PspProcessType, POBJECT_TYPE PspThreadType
源文件: ps\psobject.c

进程列表

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
1	1	Y	24	6	2	"N/A"

线程列表

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	2	Y	0	Ready (1)	1	0x80017e40 KiSystemProcessRoutine
2	17	Y	24	Waiting (3)	1	0x80015724 IopConsoleDispatchThread
3	18	Y	24	Waiting (3)	1	0x80017f4b KiShellThread
4	19	Y	24	Waiting (3)	1	0x80017f4b KiShellThread
5	20	Y	24	Waiting (3)	1	0x80017f4b KiShellThread
6	21	Y	24	Waiting (3)	1	0x80017f4b KiShellThread

进程列表

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
2	24	N	8	1	26	"A/t31.exe"

线程列表

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	26	N	8	Running (2)	24	0x8001f97e PspProcessStartup

PspCurrentThread

进程列表

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
3	27	N	8	1	29	"A/Hello.exe"

线程列表

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	29	N	8	Ready (1)	27	0x8001f97e PspProcessStartup

控制台线程阻塞在应用程序的进程上

数据源: POBJECT_TYPE PspProcessType
源文件: ps\psobject.c

进程基本信息

进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
24	N	8	1	26	"A/t31.exe"

进程地址空间(Pas)

进程地址空间的开始虚页号	0xc10	进程地址空间的结束虚页号	0xc7ffe1
页目录	0xc409	PTE计数器数据库的页框号	0xc408

进程的内核对象句柄表(ObjectTable)

HandleTable	0xc0003000	FreeEntryListHead	0xc6	HandleCount	0xc5
-------------	------------	-------------------	------	-------------	------

阻塞在该进程上的线程链表(WaitListHead)

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	18	Y	24	Waiting (3)	1	0x80017f4b KiShellThread

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

数据源: POBJECT_TYPE PspProcessType, POBJECT_TYPE PspThreadType
源文件: ps.psoject.c

进程列表

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	映像名称 (ImageName)
1	1	Y	24	6	2	"N/A"

线程列表

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	2	Y	0	Ready (1)	1	0x80017e40 KiSystemProcessRoutine
2	17	Y	24	Waiting (3)	1	0x80015724 IopConsoleDispatchThread
3	18	Y	24	Waiting (3)	1	0x80017f4b KfShellThread
4	19	Y	24	Waiting (3)	1	0x80017f4b KfShellThread
5	20	Y	24	Waiting (3)	1	0x80017f4b KfShellThread
6	21	Y	24	Waiting (3)	1	0x80017f4b KfShellThread

进程列表

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	映像名称 (ImageName)
2	24	N	8	1	26	"A/t31.exe"

线程列表

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	26	N	8	Running (2)	24	0x8001f97e PspProcessStartup

PspCurrentThread

进程列表

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	映像名称 (ImageName)
3	27	N	8	0	0	"N/A"

线程列表

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	29	N	8	Terminated (4)	27	0x8001f97e PspProcessStartup

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。

数据源: POBJECT_TYPE PspProcessType, POBJECT_TYPE PspThreadType
源文件: ps.psoject.c

进程列表

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	映像名称 (ImageName)
1	1	Y	24	6	2	"N/A"

线程列表

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	2	Y	0	Ready (1)	1	0x80017e40 KiSystemProcessRoutine
2	17	Y	24	Waiting (3)	1	0x80015724 IopConsoleDispatchThread
3	18	Y	24	Waiting (3)	1	0x80017f4b KfShellThread
4	19	Y	24	Waiting (3)	1	0x80017f4b KfShellThread
5	20	Y	24	Waiting (3)	1	0x80017f4b KfShellThread
6	21	Y	24	Waiting (3)	1	0x80017f4b KfShellThread

进程列表

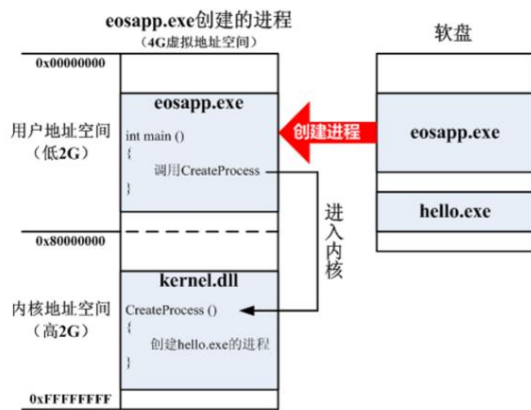
序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	映像名称 (ImageName)
2	24	N	8	1	26	"A/lab3_2.exe"

线程列表

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	26	N	8	Running (2)	24	0x8001f97e PspProcessStartup

PspCurrentThread

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

数据源:进程控制块结构体PROCESS中的PMMPAS Pas中的MMVAD_LIST VadList
源文件: ps\psp.h mm\mi.h

Total Vpn Count: 2048
Allocated Vpn Count: 16
Free Vpn Count: 2032
Total Vpn From 655360 to 657407 (0xa0000000 - 0xa07ff000)

序号	虚拟页框号	虚拟地址
1	0	0x00000000
.....
17	16	0x00010000
.....
524272	524271	0x7ffeffff
.....
655361	655360	0xa0000000
655362	655361	0xa0001000
655363	655362	0xa0002000
655364	655363	0xa0003000
655365	655364	0xa0004000
655366	655365	0xa0005000
655367	655366	0xa0006000
655368	655367	0xa0007000
655369	655368	0xa0008000
655370	655369	0xa0009000
655371	655370	0xa000a000
655372	655371	0xa000b000
655373	655372	0xa000c000
655374	655373	0xa000d000
655375	655374	0xa000e000
655376	655375	0xa000f000
655377	655376	0xa0010000
.....
1048576	1048575	0x3ffff000

用) 都大于 0x80000000。

序号	虚拟页框号	虚拟地址
1	0	0x00000000
.....
17	16	0x00010000
18	17	0x00011000
.....
1023	1022	0x003fe000
1024	1023	0x003ff000
1025	1024	0x00400000
1026	1025	0x00401000
1027	1026	0x00402000
1028	1027	0x00403000
1029	1028	0x00404000
1030	1029	0x00405000
1031	1030	0x00406000
1032	1031	0x00407000
1033	1032	0x00408000
1034	1033	0x00409000
1035	1034	0x0040a000
1036	1035	0x0040b000
.....
655361	655360	0xa0000000
.....
657408	657407	0xa07fffff
.....
1048576	1048575	0x3ffff000

查看 EOS 应用程序项目链接器属性中基址的值

```
0x800110ee <CreateProcess+0>: push    ebp
0x800110ef <CreateProcess+1>: mov     ebp, esp
0x800110f1 <CreateProcess+3>: sub     esp, 0x18
0x800110f4 <CreateProcess+6>: mov     eax, DWORD PTR [ebp+0x18]
0x800110f7 <CreateProcess+9>: mov     DWORD PTR [esp+0x10], eax
0x800110fb <CreateProcess+13>: mov     eax, DWORD PTR [ebp+0x14]
0x800110fe <CreateProcess+16>: mov     DWORD PTR [esp+0xc], eax
```

查看当前系统物理内存的使用情况

数据源: ULONG_PTR MiFreePageListHead

源文件: mm\pfnlist.c

物理页的数量: 8176

物理内存的大小: 8176 * 4096 = 33488896 Byte

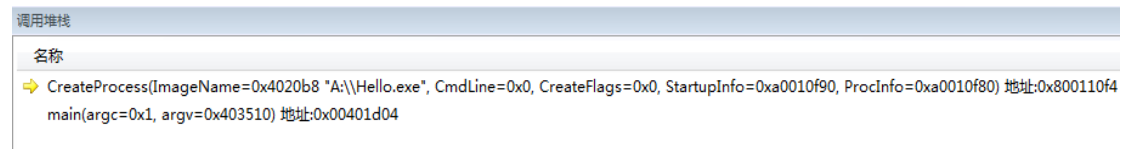
零页的数量: 0

空闲页的数量: 7117

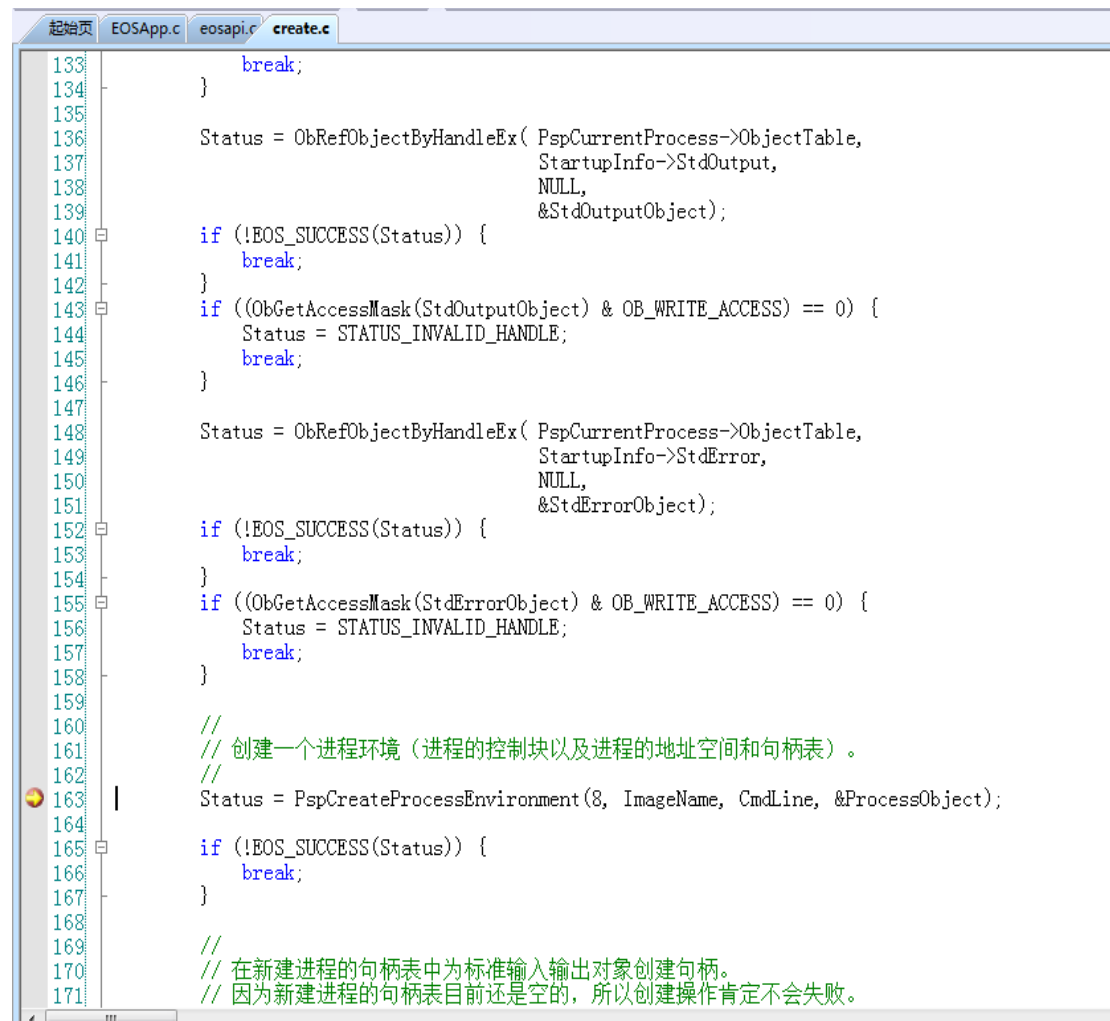
已使用页的数量: 1059

物理页框号	页框号数据库项	状态
0x0	0x80100000	BUSY
0x1	0x80100001	BUSY
.....
0x421	0x80100421	BUSY
0x422	0x80100422	BUSY
0x423	0x80100423	FREE
0x424	0x80100424	FREE
.....
0x1fee	0x80101fee	FREE
0x1fef	0x80101fef	FREE

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

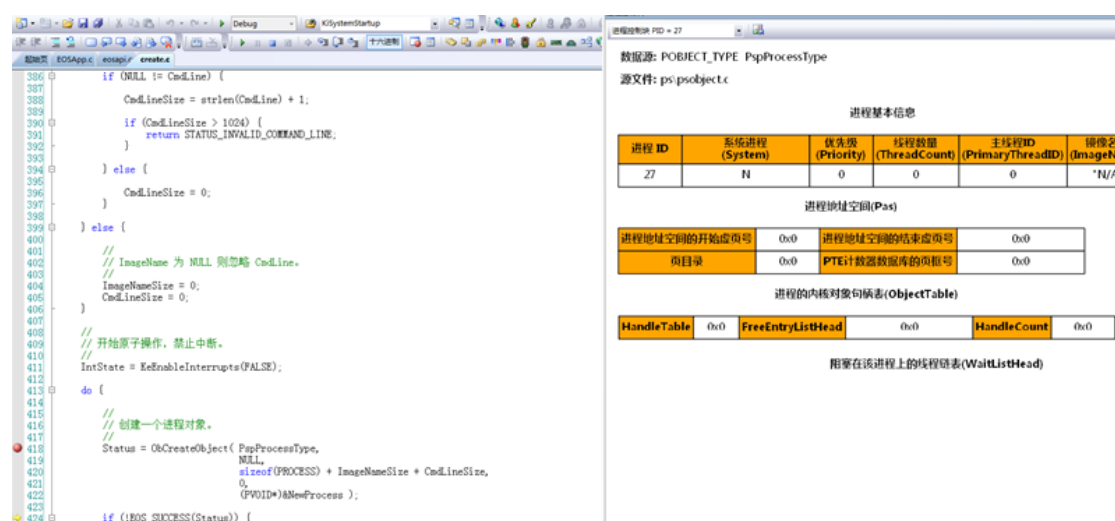
```

405 CmdLineSize = 0;
406 }
407
408 //
409 // 开始原子操作，禁止中断。
410 //
411 IntState = KeEnableInterrupts(FALSE);
412
413 do {
414
415     //
416     // 创建一个进程对象。
417     //
418     Status = ObCreateObject( PspProcessType,
419                             NULL,
420                             sizeof(PROCESS) + ImageNameSize + CmdLineSize,
421                             0,
422                             (PVOID*)&NewProcess );
423
424     if (!EOS_SUCCESS(Status)) {

```

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。



数据源: POBJECT_TYPE PspProcessType
源文件: pspobject.c

进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	映像名 (ImageName)
27	N	0	0	0	*N/A

进程地址空间的开始虚司	进程地址空间的结束虚司
0x0	0x0

HandleTable	FreeEntryListHead	HandleCount
0x0	0x0	0x0

阻塞在该进程上的线程链表(WaitListHead)

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。

数据源: POBJECT_TYPE PspProcessType

源文件: ps\psobject.c

进程基本信息

进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
27	N	0	0	0	"N/A"

进程地址空间(Pas)

进程地址空间的开始虚页号	0x10	进程地址空间的结束虚页号	0x7ffef
页目录	0x423	PTE计数器数据库的页框号	0x424

进程的内核对象句柄表(ObjectTable)

HandleTable	0x0	FreeEntryListHead	0x0	HandleCount	0x0
-------------	-----	-------------------	-----	-------------	-----

阻塞在该进程上的线程链表(WaitListHead)

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。

数据源: POBJECT_TYPE PspProcessType

源文件: ps\psobject.c

进程基本信息

进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
27	N	0	0	0	"N/A"

进程地址空间(Pas)

进程地址空间的开始虚页号	0x10	进程地址空间的结束虚页号	0x7ffef
页目录	0x423	PTE计数器数据库的页框号	0x424

进程的内核对象句柄表(ObjectTable)

HandleTable	0x0	FreeEntryListHead	0x0	HandleCount	0x0
-------------	-----	-------------------	-----	-------------	-----

阻塞在该进程上的线程链表(WaitListHead)

数据源: POBJECT_Type PspProcessType
源文件: ps\psobject.c

进程基本信息

进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
27	N	0	0	0	"N/A"

进程地址空间(Pas)

进程地址空间的开始虚页号	0x10	进程地址空间的结束虚页号	0x7ffef
页目录	0x423	PTE计数器数据库的页框号	0x424

进程的内核对象句柄表(ObjectTable)

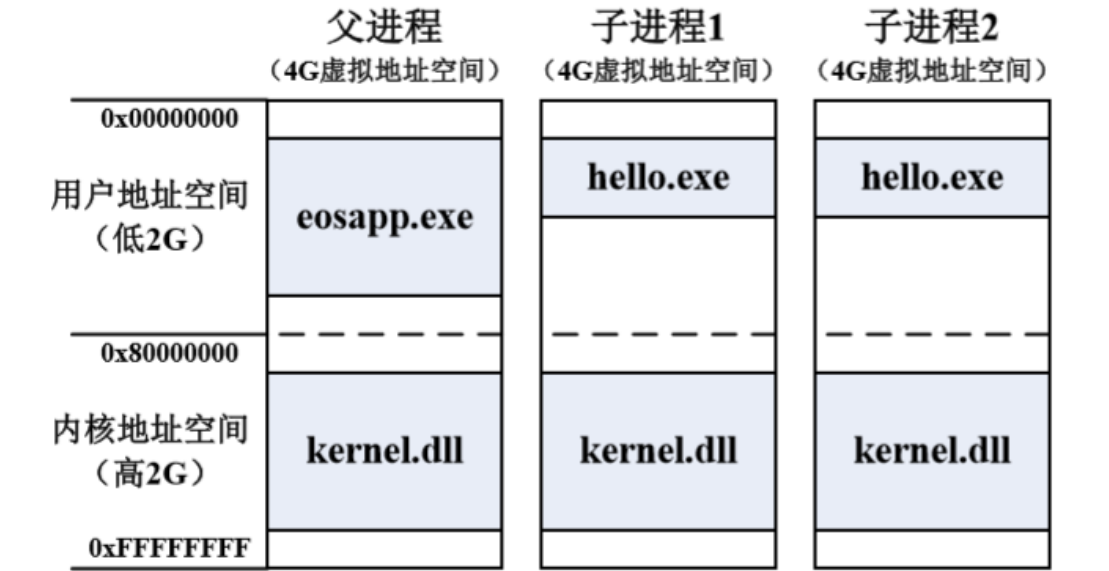
HandleTable	0xa0004000	FreeEntryListHead	0x1	HandleCount	0x0
-------------	------------	-------------------	-----	-------------	-----

阻塞在该进程上的线程链表(WaitListHead)

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。

监视			
名称	值	类型	
*NewProcess	无法计算表达式的值。		
*ProcessObject	{ System = 0x0, Priority = 0x8, Pas = 0x...	struct _P...	

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。

数据源: POBJECT_TYPE PspProcessType、POBJECT_TYPE PspThreadType
源文件: ps\psobject.c

进程列表

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
1	1	Y	24	6	2	"N/A"

线程列表

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	2	Y	0	Ready (1)	1	0x80017e40 KiSystemProcessRoutine
2	17	Y	24	Waiting (3)	1	0x80015724 IopConsoleDispatchThread
3	18	Y	24	Waiting (3)	1	0x80017f4b KiShellThread
4	19	Y	24	Waiting (3)	1	0x80017f4b KiShellThread
5	20	Y	24	Waiting (3)	1	0x80017f4b KiShellThread
6	21	Y	24	Waiting (3)	1	0x80017f4b KiShellThread

进程列表

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
2	24	N	8	2	26	"A:\lab3_2.exe"

线程列表

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	26	N	8	Waiting (3)	24	0x8001f97e PspProcessStartup
2	27	N	8	Running (2)	24	0x401d00 AppThread

PspCurrentThread

数据源: POBJECT_TYPE PspThreadType
源文件: ps\psobject.c

线程基本信息

线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名称 (StartAddress And FuncName)	剩余时间片 (RemainderTicks)
27	N	8	Running (2)	24	0x401d00 AppThread	6

PspCurrentThread

线程执行在内核状态的上下文环境状态(KernelContext)

Eax	0x0	Ebp	0x0	Ebx	0x0	Ecx	0x0
Edi	0x0	Edx	0x0	EFlag	0x200	Eip	0x8001f930
Esi	0x0	Esp	0xca0012ffc	SegCs	0x100008	SegDs	0x77e50010
SegEs	0x77e50010	SegFs	0x77e50010	SegGs	0x77e50010	SegSs	0x77e50010

所在状态队列的链表项(StateListEntry)

Prev	0x0	Next	0x0
------	-----	------	-----

有限等待唤醒的计时器(WaitTimer)

IntervalTicks	0x0	ElapsedTicks	0x0
Parameter	0x0	TimerRoutine	0x0

线程在执行内核代码时绑定的进程地址空间(AttachedPas)

进程地址空间的开始虚页号	0x10	进程地址空间的结束虚页号	0x7ffef
页目录	0x409	PTE计数器数据库的页框号	0x408

阻塞在该线程上的线程链表(WaitListHead)

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	26	N	8	Waiting (3)	24	0x8001f97e PspProcessStartup

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。

起始页start.c

```
58 MmInitializeSystem1(LoaderBlock);
59 ObInitializeSystem1();
60 PsInitializeSystem1();
61 IoInitializeSystem1();
62
63 //
64 // 创建系统启动进程。
65 //
66 PsCreateSystemProcess(KiSystemProcessRoutine);
67
68 //
69 // 执行到这里时，所有函数仍然在使用由 Loader 初始化的堆栈，所有系统线程
70 // 都已处于就绪状态。执行线程调度后，系统线程开始使用各自的线程堆栈运行。
71 //
72 KeThreadSchedule();
73
74 //
75 // 本函数永远不会返回。
76 //
77 ASSERT(FALSE);
78 }
```

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。

数据源: POBJECT_TYPE PspProcessType、POBJECT_TYPE PspThreadType
源文件: ps\psobject.c

进程列表

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
1	1	Y	24	1	2	"N/A"

线程列表

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	2	Y	24	Ready (1)	1	0x80017e40 KiSystemProcessRoutine

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。

数据源: OBJECT_TYPE PspProcessType、OBJECT_TYPE PspThreadType
源文件: ps\psobject.c

进程列表

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
1	1	Y	24	1	2	"N/A"

线程列表

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	2	Y	24	Running (2)	1	0x80017e40 KiSystemProcessRoutine

PspCurrentThread

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。

数据源: OBJECT_TYPE PspProcessType、OBJECT_TYPE PspThreadType
源文件: ps\psobject.c

进程列表

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
1	1	Y	24	2	2	"N/A"

线程列表

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	2	Y	24	Running (2)	1	0x80017e40 KiSystemProcessRoutine
2	3	Y	24	Ready (1)	1	0x80017ed0 KiInitializationThread

PspCurrentThread

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。

```
160 KiInitializeSystemIO;
161 KiInitializeSystemIO;
162 KiInitializeSystemIO;
163 KiInitializeSystemIO;
164
165 // 为 4 个控制台各创建一个线程。
166 //
167 for (ConsoleIndex = 0; ConsoleIndex < 4; ConsoleIndex++) {
168     Status = PsCreateThread(0,
169                             KiShellThread,
170                             (PVOID)ConsoleIndex,
171                             0,
172                             &ThreadHandle,
173                             NULL);
174     if (GOS_SUCCESS(Status)) {
175         ObCloseHandle(ThreadHandle);
176     } else {
177         break;
178     }
179 }
180
181 return 0;
182
183
184
185
186
187 ULONG
188 KiShellThread(
189     IN PVOID Parameter
190 )
191 {
192     STATUS Status;
193     HANDLE StdHandle;
194     HANDLE InputHandle;
195     CHAR Line[64];
```

数据源: OBJECT_TYPE PspProcessType、OBJECT_TYPE PspThreadType
源文件: ps\psobject.c

进程列表

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
1	1	Y	24	7	2	

线程列表

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	2	Y	0	Ready (1)	1	
2	3	Y	24	Running (2)	1	
3	17	Y	24	Ready (1)	1	
4	18	Y	24	Ready (1)	1	
5	19	Y	24	Ready (1)	1	
6	20	Y	24	Ready (1)	1	
7	21	Y	24	Ready (1)	1	

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。

源文件: ps\psobject.c

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
1	1	Y	24	3	2	"N/A"

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	2	Y	0	Ready (1)	1	0x80017e40 KSystemProcessRoutine
2	3	Y	24	Running (2)	1	0x80017ed0 KInitializationThread
3	17	Y	24	Ready (1)	1	0x80015724 IopConsoleDispatchThread

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。

源文件: ps\psobject.c

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
1	1	Y	24	7	2	"N/A"

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	2	Y	0	Ready (1)	1	0x80017e40 KiSystemProcessRoutine
2	3	Y	24	Running (2)	1	0x80017ed0 KiInitializationThread
3	17	Y	24	Ready (1)	1	0x80015724 IopConsoleDispatchThread
4	18	Y	24	Ready (1)	1	0x80017f4b KiShellThread
5	19	Y	24	Ready (1)	1	0x80017f4b KiShellThread
6	20	Y	24	Ready (1)	1	0x80017f4b KiShellThread
7	21	Y	24	Ready (1)	1	0x80017f4b KiShellThread

[illegible]

營業時間

32个调查点组成的经纬网图

Priority: 0	TCB	
Priority: 1	Id	2
Priority: 2	Priority	0
Priority: 3	State	Ready
Priority: 4	RemainderTicks	0
Priority: 5	StartAddr	KiSystemProcessRoutine

Priority: 8
Priority: 7
Priority: 6
Priority: 5
Priority: 4
Priority: 3
Priority: 2
Priority: 1
Priority: 0
Priority: 15
Priority: 14
Priority: 13
Priority: 12
Priority: 11
Priority: 10
Priority: 9
Priority: 8
Priority: 7
Priority: 6
Priority: 5
Priority: 4
Priority: 3
Priority: 2
Priority: 1
Priority: 0
Priority: 15
Priority: 14
Priority: 13
Priority: 12
Priority: 11
Priority: 10
Priority: 9
Priority: 8
Priority: 7
Priority: 6
Priority: 5
Priority: 4
Priority: 3
Priority: 2
Priority: 1
Priority: 0

Diagram illustrating the execution flow of a thread pool:

- Thread 1 (Priority 8) is initially in the **Idle** state.
- Thread 1 transitions to the **Ready** state when it receives a task (Task 1).
- Thread 1 executes Task 1 and then transitions back to the **Idle** state.
- Thread 2 (Priority 7) is initially in the **Idle** state.
- Thread 2 transitions to the **Ready** state when it receives a task (Task 2).
- Thread 2 executes Task 2 and then transitions back to the **Idle** state.
- Thread 3 (Priority 6) is initially in the **Idle** state.
- Thread 3 transitions to the **Ready** state when it receives a task (Task 3).
- Thread 3 executes Task 3 and then transitions back to the **Idle** state.
- Thread 4 (Priority 5) is initially in the **Idle** state.
- Thread 4 transitions to the **Ready** state when it receives a task (Task 4).
- Thread 4 executes Task 4 and then transitions back to the **Idle** state.
- Thread 5 (Priority 4) is initially in the **Idle** state.
- Thread 5 transitions to the **Ready** state when it receives a task (Task 5).
- Thread 5 executes Task 5 and then transitions back to the **Idle** state.

The diagram shows the thread pool's state transitions and task execution flow. The threads are represented by boxes with columns for **Id**, **Priority**, **State**, **RemainderTasks**, and **StartAddr**. The tasks are represented by boxes with columns for **Id**, **Priority**, **State**, **RemainderTasks**, and **StartAddr**.

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。

4. 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。

5. 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.