

实验三进程的创建

一、实验目的

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。

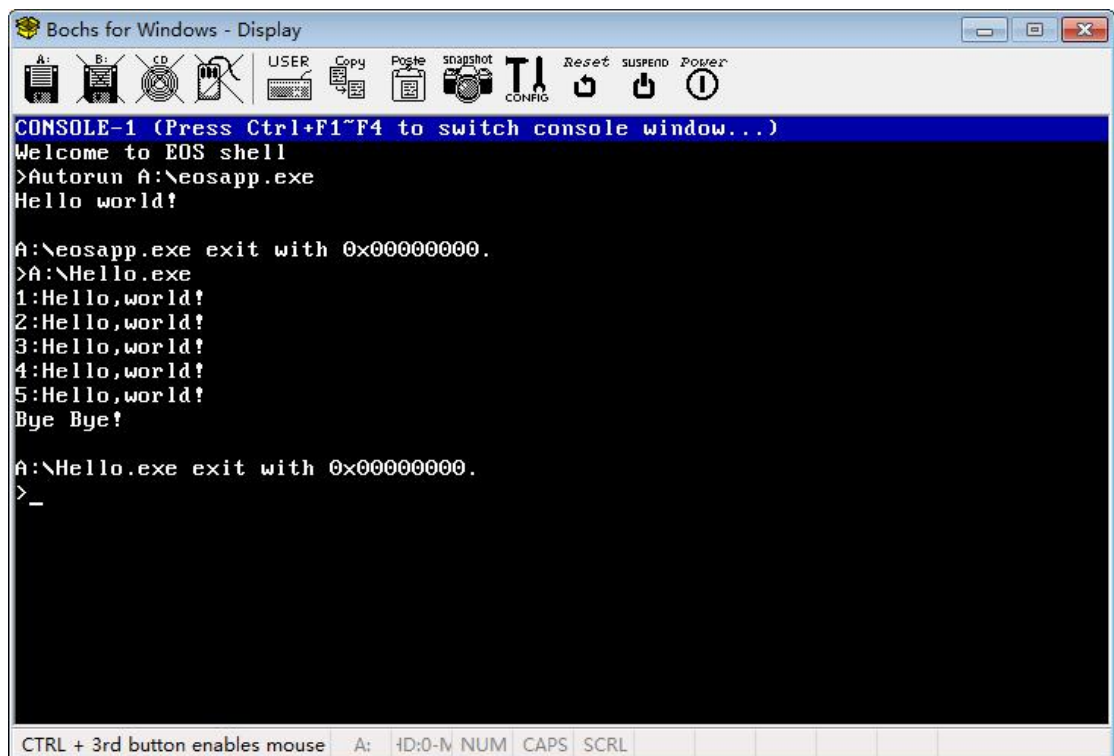
二、实验内容

任务（一）

1. 用 OS Lab 提供的“从 Git 远程库新建项目”功能将个人项目克隆到本地磁盘中，然后生成完整的（debug 和 release）的 kernel 工程，复制 SDK 文件作为自己的应用程序运行环境，保证在调试应用程序时能够调试进入内核并显示代码。

2. 练习使用控制台命令创建一个 EOS 应用程序的进程

按着实验指导书的步骤进行操作，观察到 `hello.exe` 程序执行的结构如果：



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

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

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

如图说明此进程被激活并运行成功

3. 练习通过编程的方式让应用程序创建另一个应用程序的进程

按着步骤完成实验，第一步要获取正确的启动信息后父进程才会调用子进程，同时自己进入阻塞状态，然后在子进程合法结束并返回正确的退出码后，父进程才会继续执行自身的程序。

```
000 Hello,world! 1
Wai Hello,world! 2
Con Hello,world! 3
    Hello,world! 4
    Hello,world! 5
    Bye-bye!

The process exit with 0.

A:\eosapp.exe exit with 0x00000000.
>_
```

4.从应用程序的角度理解进程的创建过程

按着步骤完成实验，从应用程序的角度理解进程创建过程

The screenshot displays the EOSApp development environment. The left pane shows the source code for `EOSApp.c`, which includes a `main` function that prints "Hello world!\n" and returns 0. The right pane shows the "进程过程" (Process Process) window, which contains three tables: "进程列表" (Process List), "线程列表" (Thread List), and "进程列表" (Process List) again. The "进程列表" table shows the current process (ID 1, Name eosapp.exe, Priority 24, ThreadCount 6, ParentProcessID 0). The "线程列表" table shows the threads of the current process (ID 1, Name eosapp.exe, Priority 24, ThreadCount 6, ParentProcessID 0). The "进程列表" table shows the list of processes (ID 1, Name eosapp.exe, Priority 24, ThreadCount 6, ParentProcessID 0).

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数 (ThreadCount)	主线程ID (PrimaryThreadID)
1	1	Y	24	6	2

序号	线程 ID	系统进程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)
1	2	Y	0	Ready (1)	1
2	17	Y	24	Waiting (3)	1
3	18	Y	24	Waiting (3)	1
4	19	Y	24	Waiting (3)	1
5	20	Y	24	Waiting (3)	1
6	21	Y	24	Waiting (3)	1

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数 (ThreadCount)	主线程ID (PrimaryThreadID)
2	24	N	8	1	26

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	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 lopConsoleDispatchThread
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:/eosapp.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

数据源: OBJECT_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:/eosapp.exe"

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	0x409	PTE计数器数据库的页框号	0x408

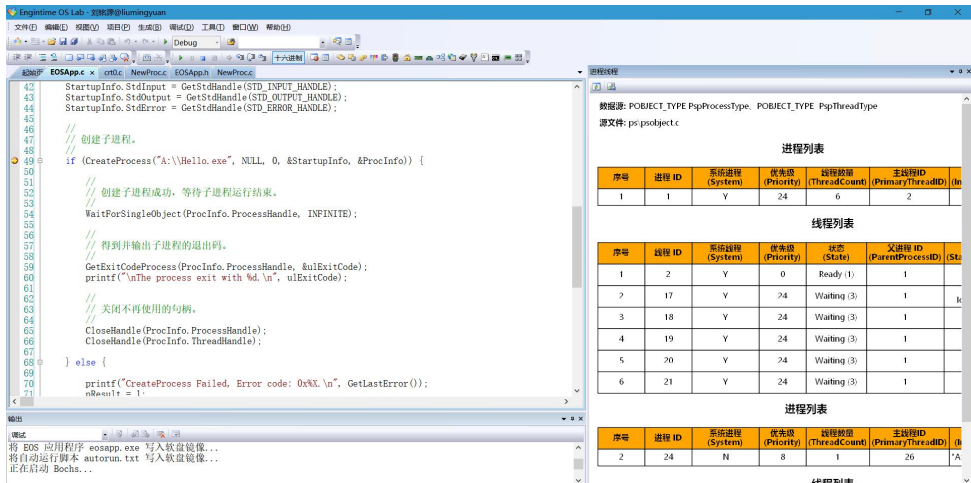
进程的内核对象句柄表(ObjectTable)

HandleTable	0xa0003000	FreeEntryListHead	0x6	HandleCount	0x5
-------------	------------	-------------------	-----	-------------	-----

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

5.从内核的角度理解进程的创建过程
(1) 调试 CreateProcess 函数



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 lopConsoleDispatchThread
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:\eosapp.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

PID=1:

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\psph 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

反汇编:

Engine OS Lab - 刘松源@lumingyuan

文件(F) 编辑(E) 视图(V) 项目(B) 生成(S) 调试(D) 工具(T) 窗口(W) 帮助(H)

Debug

编译 运行 断点 十六进制 反汇编 内存 寄存器 堆栈 调用堆栈 性能 网络 设备 系统 安全 其他

EOSApp.c x crt0.c NewProc.c EOSApp.h NewProc.h

```
36 printf("Create a process and wait for the process exit...\n\n");
37
38
39 // 使子进程和父进程使用相同的标准句柄。
40
41 StartupInfo.StdInput = GetStdHandle(STD_INPUT_HANDLE);
42 StartupInfo.StdOutput = GetStdHandle(STD_OUTPUT_HANDLE);
43 StartupInfo.StdError = GetStdHandle(STD_ERROR_HANDLE);
44
45 // 创建子进程。
46
47
48
```

反汇编

```
0x8001d82a <PsCreateProcess+0>: push ebp
0x8001d82b <PsCreateProcess+1>: mov ebp,esp
0x8001d82d <PsCreateProcess+3>: push ebx
0x8001d82e <PsCreateProcess+4>: sub esp,0x44
0x8001d831 <PsCreateProcess+7>: mov DWORD PTR [ebp-0xc],0x0
0x8001d838 <PsCreateProcess+14>: mov DWORD PTR [ebp-0x10],0x0
0x8001d83f <PsCreateProcess+21>: mov DWORD PTR [ebp-0x14],0x0
0x8001d846 <PsCreateProcess+28>: mov DWORD PTR [ebp-0x18],0x0
0x8001d84d <PsCreateProcess+35>: mov DWORD PTR [ebp-0x1c],0x0
0x8001d854 <PsCreateProcess+42>: mov DWORD PTR [ebp-0x20],0x0
0x8001d85b <PsCreateProcess+49>: mov DWORD PTR [ebp-0x24],0x0
0x8001d862 <PsCreateProcess+56>: cmp DWORD PTR [ebp-0x51],0x0
0x8001d866 <PsCreateProcess+60>: je 0x8001d86e <PsCreateProcess+68>
0x8001d868 <PsCreateProcess+62>: cmp DWORD PTR [ebp-0x14],0x0
0x8001d86c <PsCreateProcess+66>: jne 0x8001d87a <PsCreateProcess+80>
0x8001d86e <PsCreateProcess+68>: mov DWORD PTR [ebp-0x28],0xc0000004
0x8001d875 <PsCreateProcess+75>: jmp 0x8001dcb1 <PsCreateProcess+1159>
0x8001d87a <PsCreateProcess+80>: lea eax,[ebp-0x20]
0x8001d87d <PsCreateProcess+83>: mov DWORD PTR [esp+0x4],eax
0x8001d881 <PsCreateProcess+87>: mov eax,ds:0x80025460
0x8001d886 <PsCreateProcess+92>: mov eax,DWORD PTR [eax]
0x8001d888 <PsCreateProcess+94>: mov eax,DWORD PTR [eax+0x8]
```

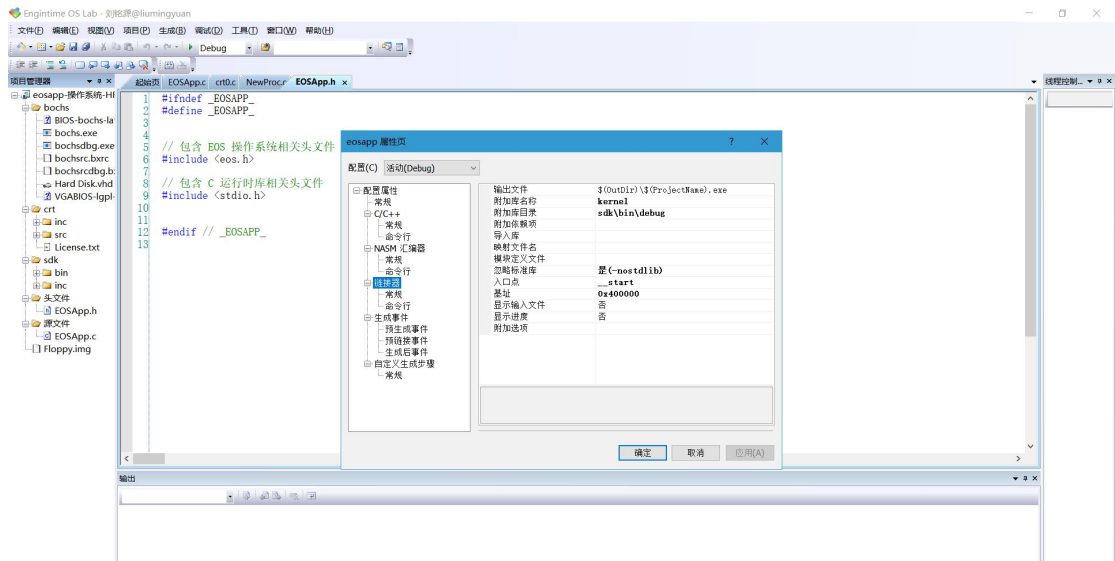
进程控制块 PID = 1

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\psph mm\mi.h

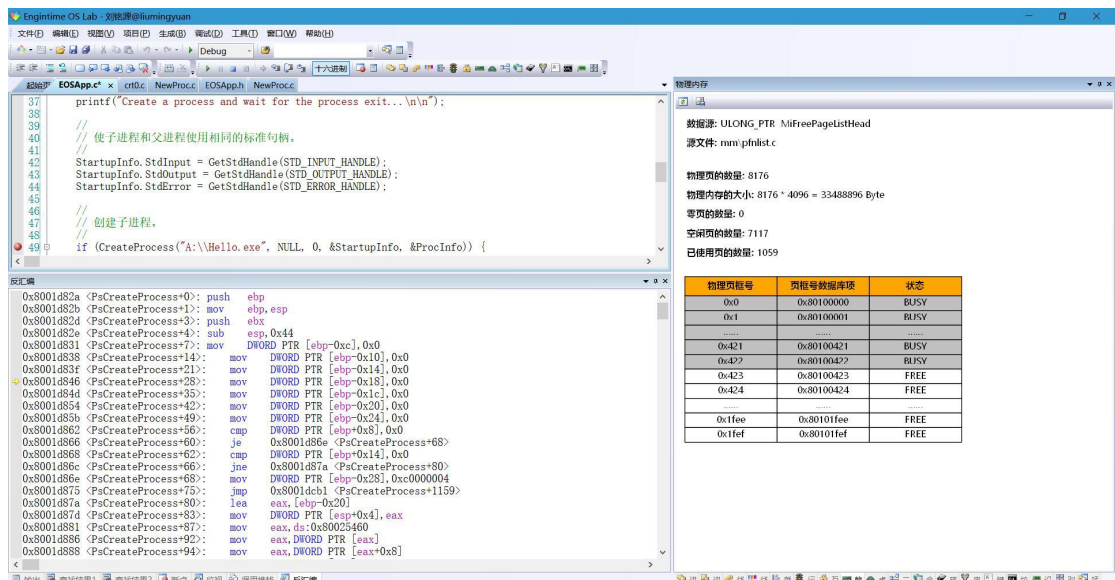
Total Vpn Count: 2048
Allocated Vpn Count: 16
Free Vpn Count: 2032
Total Vpn From 655360 to 657407 (0xa0000000 - 0xa07ff000)

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

属性页:



物理内存:



(2) 调试 PsCreateProcess 函数

定义了一个进程控制块的指针变量 NewProcess。在此函数中查找到创建进程控制块的代码行（create.c 文件的第 418 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)) {
425         break;
426     }
427
428     //
429     // 如果是系统进程则直接使用系统进程地址空间，否则新建一个进程地址空间。
430     //
431     if (NULL == ImageName) {
432
```

进程控制块 PID=27

进程控制块

进程控制块 PID = 27

数据源: OBJECT_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)

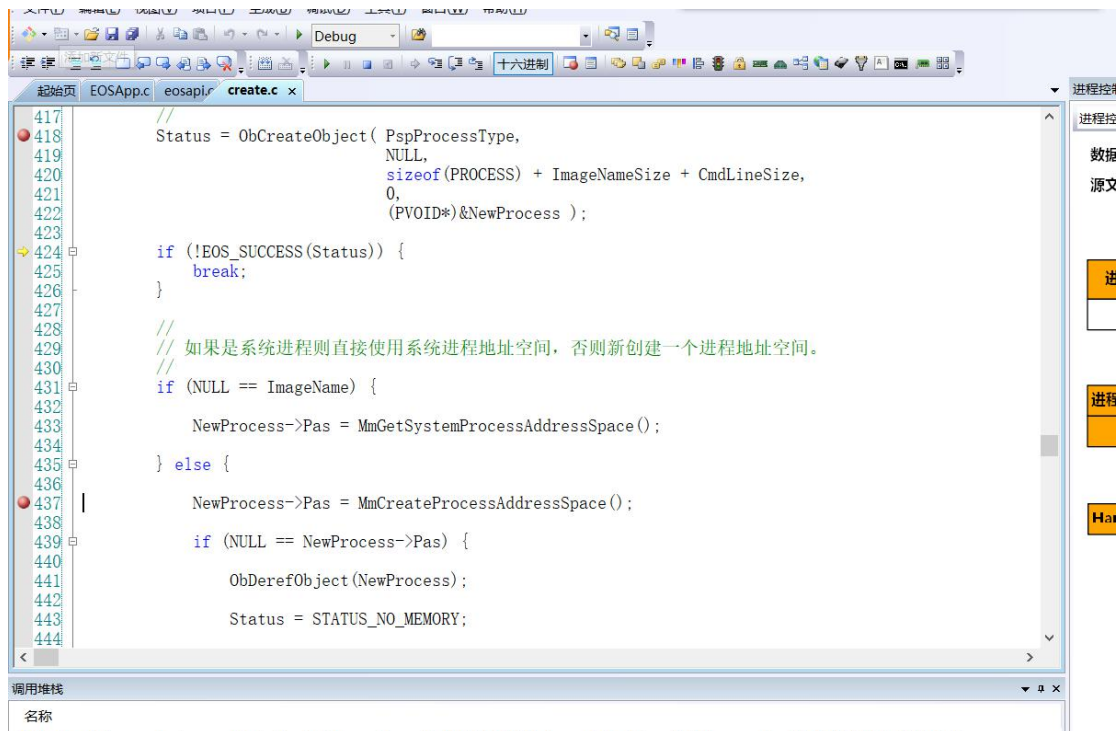
进程地址空间的开始虚页号	0x0	进程地址空间的结束虚页号	0x0
页目录	0x0	PTE计数器数据库的页框号	0x0

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)

创建进程空间



更新之后进程控制模块 PID=27

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"

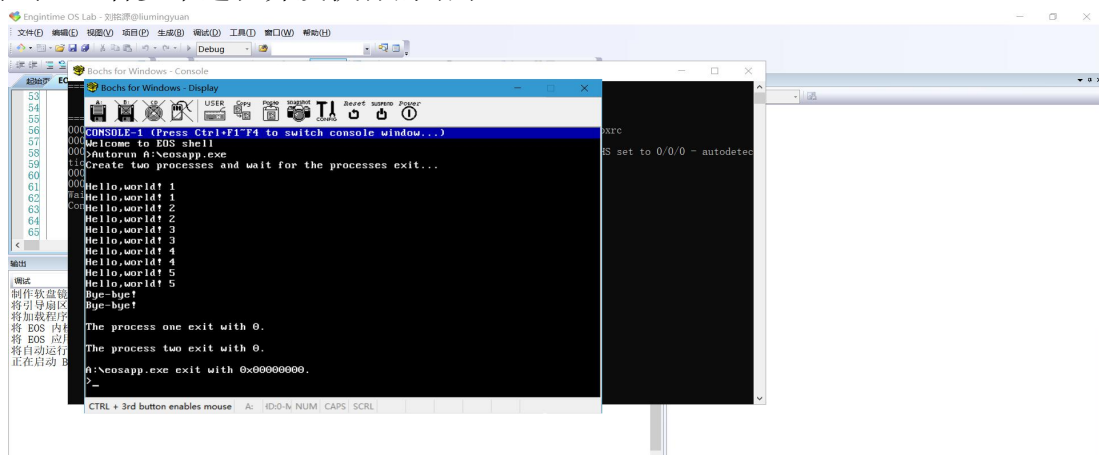
任务（二）

先登录 CodeCode.net 平台领取本次实验对应的任务，从而在 CodeCode.net 平台上创建一个

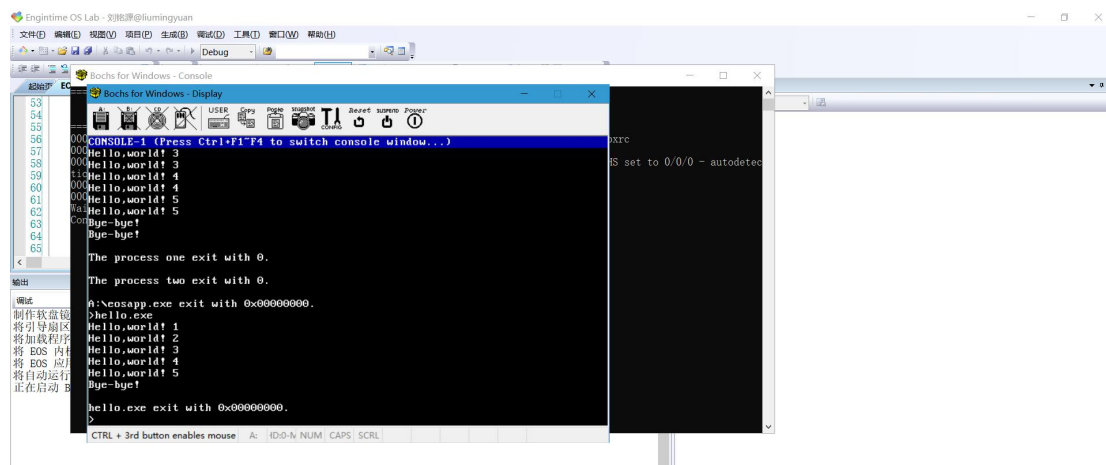
人项目，然后使用 OS Lab 提供的“从 Git 远程库新建项目”功能将个人项目克隆到本地磁盘中。

1. 创建一个应用程序的多个进程

（1）替换 EOS 应用程序项目中 EOSApp.c 文件 内的源代码，生成后启动调试，查看多个进程并发执行的结果

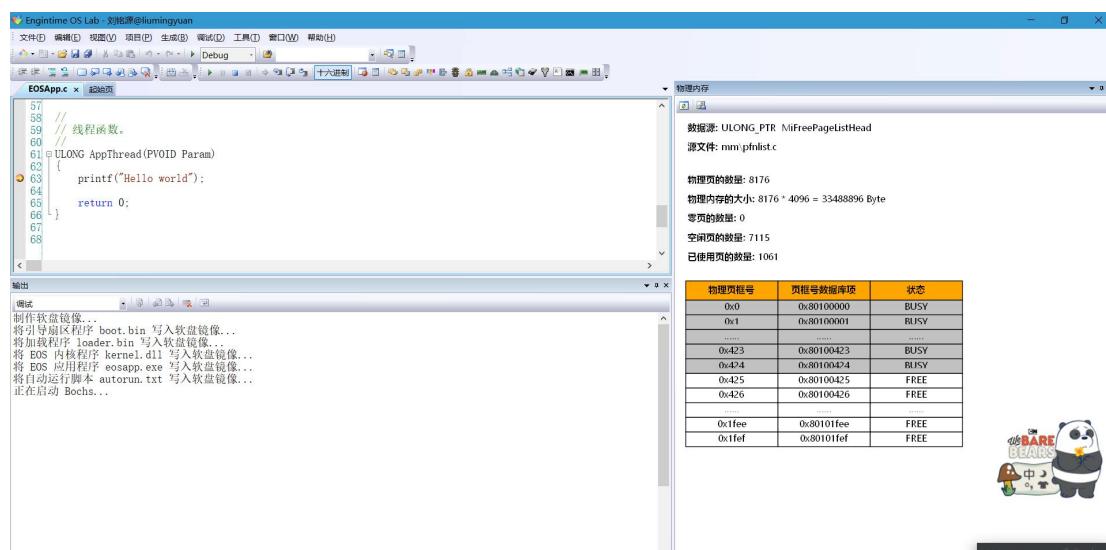


(2) 多个进程并发时, EOS 操作系统中运行的用户进程从而验证一个程序(hello.exe)可以同时创建多个进程。



任务（三）

1.先登录 CodeCode.net 平台领取本次实验对应的任务，从而在 CodeCode.net 平台上创建个人项目，然后使用 OS Lab 提供的“从 Git 远程库新建项目”功能将个人项目克隆到本地磁盘中。



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: 7115
已使用页的数量: 1061

物理页框号	页框号数据库项	状态
0x0	0x80100000	BUSY
0x1	0x80100001	BUSY
.....
0x423	0x80100423	BUSY
0x424	0x80100424	BUSY
0x425	0x80100425	FREE
0x426	0x80100426	FREE
.....
0x1fee	0x80101fee	FREE
0x1fef	0x80101fef	FREE

数据源: OBJECT_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	0x200	Ebp	0xa0012fa8	Ebx	0x0	Ecx	0x803fadf0
Edi	0x0	Edx	0x10	EFlag	0x206	Eip	0x80020649
Esi	0x0	Esp	0xa0012fa4	SegCs	0x8	SegDs	0x10
SegEs	0x10	SegFs	0x10	SegGs	0x10	SegSs	0x10

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(AttachedPps)

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

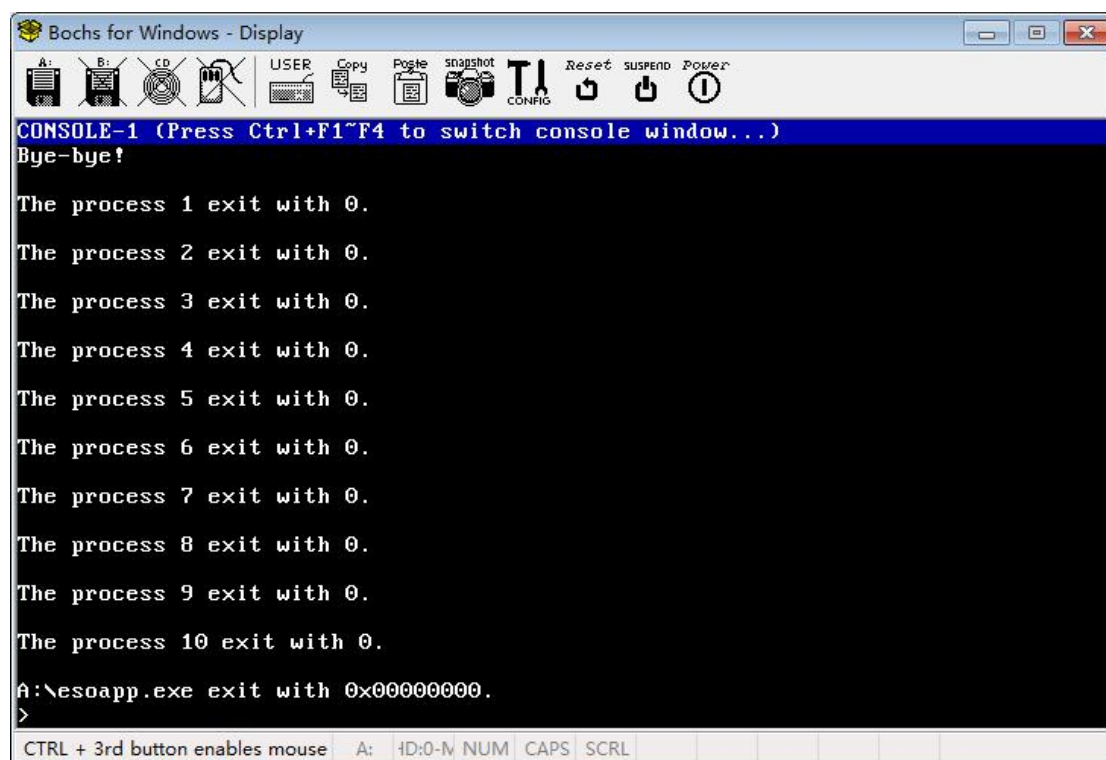
三、思考题

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：核心代码：

```
PROCESS_INFORMATION ProcInfo[10];
printf("Create ten processes and wait for the processes exit...\n\n");
int i = 0;
for(; i < 10 && CreateProcess("A:\\Hello.exe", NULL, 0, &StartupInfo,
&ProcInfo[i]; i++){
}
if (i == 10) {
//
// 创建子进程成功，等待子进程运行结束。
//
for(i = 0; i < 10; i++){
WaitForSingleObject(ProcInfo[i].ProcessHandle, INFINITE);
}
//
// 得到并输出子进程的退出码。
//
for(i = 0; i < 10; i++){
GetExitCodeProcess(ProcInfo[i].ProcessHandle, &ulExitCode);
printf("\nThe process one exit with %d.\n", ulExitCode);
}
//
// 关闭不再使用的句柄。
//
for(i = 0; i < 10; i++){
CloseHandle(ProcInfo[i].ProcessHandle);
CloseHandle(ProcInfo[i].ThreadHandle);}
} else {
printf("CreateProcess Failed, Error code: 0x%X.\n", GetLastError());
nResult = 1;
}
```



2. 在 `PspCreateProcess` 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` 的主要功能是创建进程控制块并且为程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。

刘锐源
2020.5.28