

合肥工业大学

操作系统实验报告

实验题目	实验 1 实验环境的使用
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

- 熟悉操作系统集成实验环境 OS Lab 的基本使用方法。
- 练习编译、调试 EOS 操作系统内核以及 EOS 应用程序。

2. 实验原理

实验 1 涉及 EOS 操作系统内核和 EOS 应用程序的源代码生成可执行文件的过程，以及 OS Lab 是如何将这些可执行文件写入软盘镜像文件并开始执行的。这是对 EOS 操作系统使用的学习。

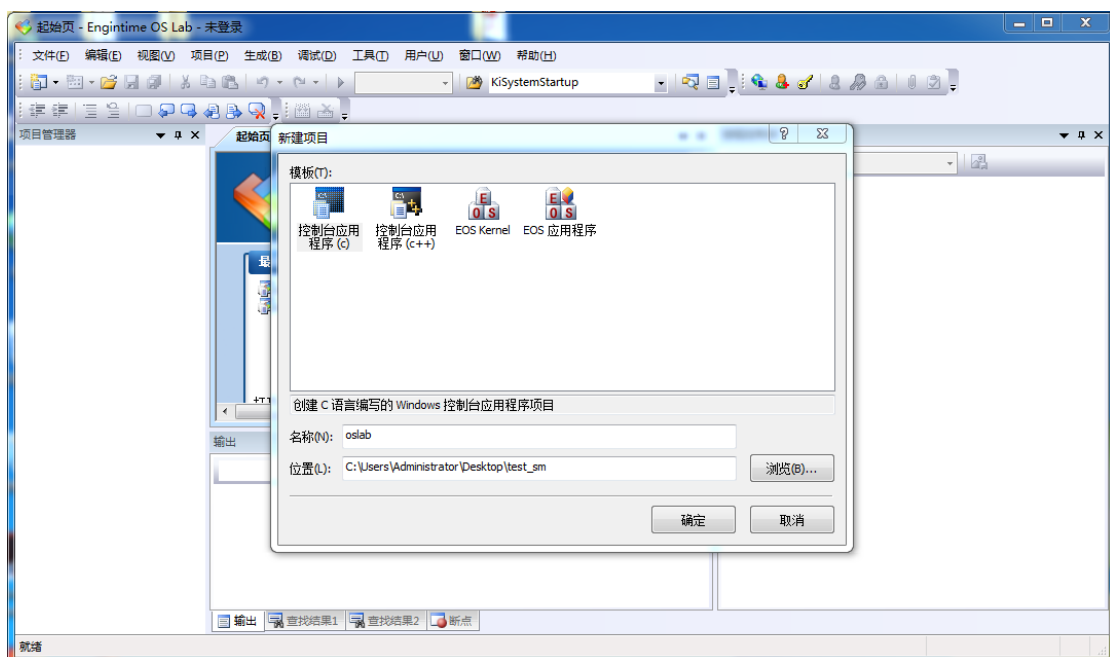
EOS 有配套的 IDE (Integrated Development Environment) 实验环境。该 IDE 环境提供的强大功能可以编辑、编译和调试 EOS 源代码。

编辑功能可以用来阅读和修改 EOS 源代码；编译功能可以将 EOS 源代码编译为二进制文件（包括引导程序和内核）；调试功能可以将编译好的二进制文件写入一个软盘镜像（或软盘），然后让虚拟机（或裸机）运行此软盘中的 EOS，并对其进行远程调试。

EOS 操作系统与 IDE 环境组成了“操作系统集成实验环境 OS Lab”。

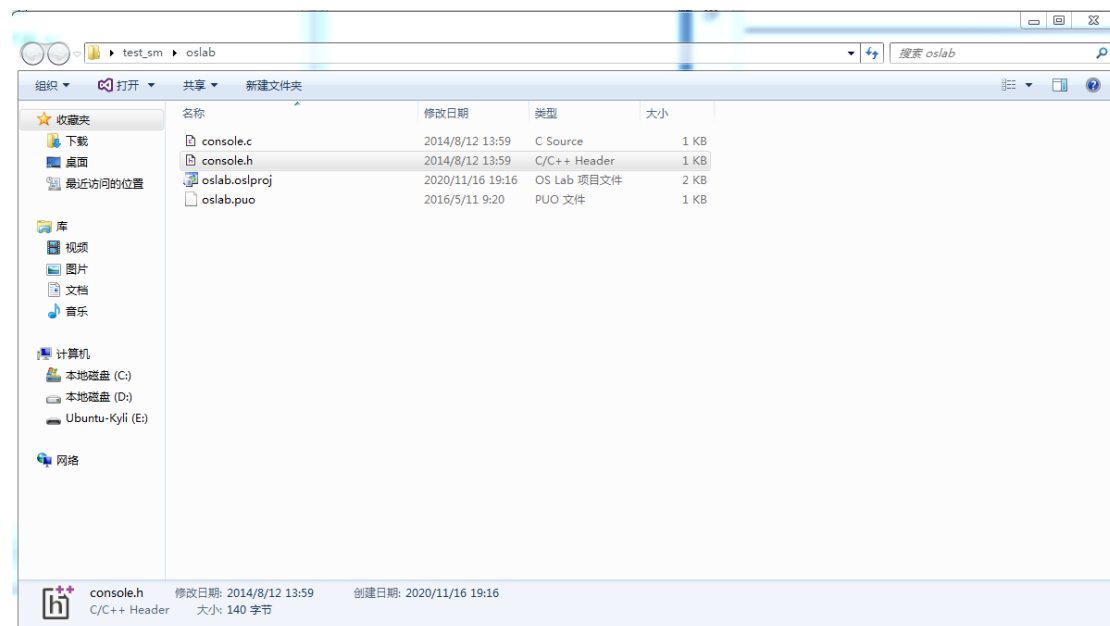
3. 实验内容

第一步新建控制台应用程序的项目

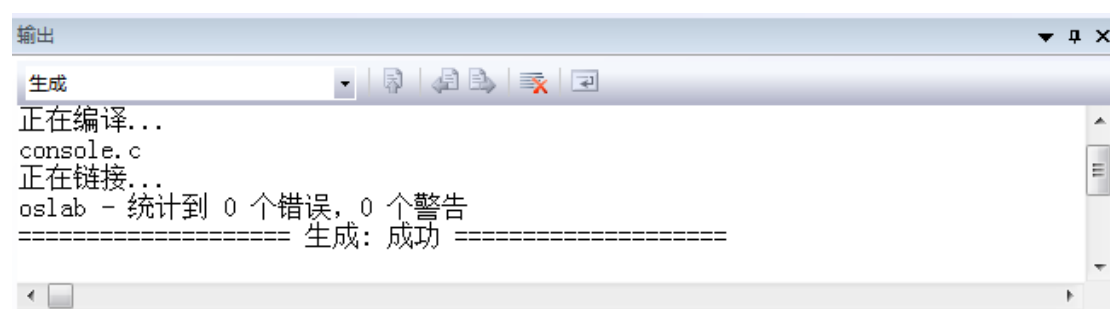


新建完毕后，OS Lab 会自动打开这个新建的项目。在“项目管理器”窗口中（如图 9-1 所示），树的根节点是项目节点，项目的名称是“console”，各个子节点是项目包含的文件夹或者文件。此项目的源代码主要包含一个头文件“console.h”和一个 C 语言源文件“console.c”。使用 Windows 资源管理器打开磁盘上的“C:\test\oslab”文件夹查看项目中包含的文件（提示，在“项目管理器”窗口的项目节点上点击右键，然后在弹出的快捷菜单中选择“打开所在的文件夹”即可）。

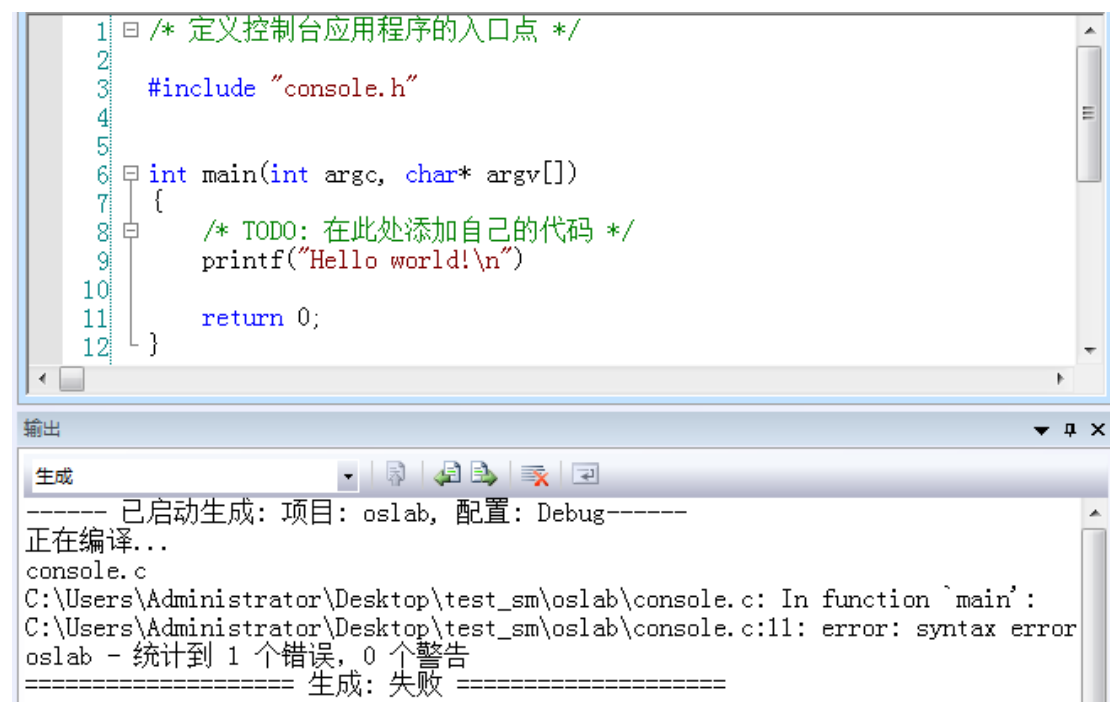
我们在 Windows 资源管理器中查看项目包含的文件：



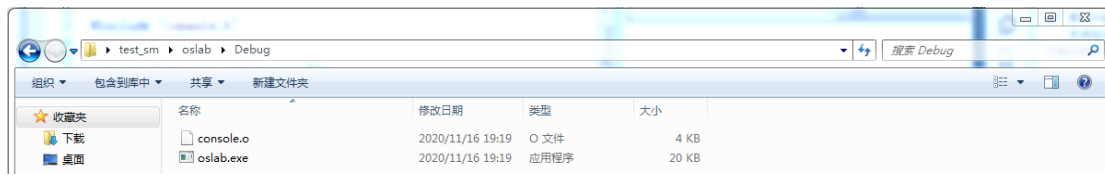
然后生成项目，查看输出窗口



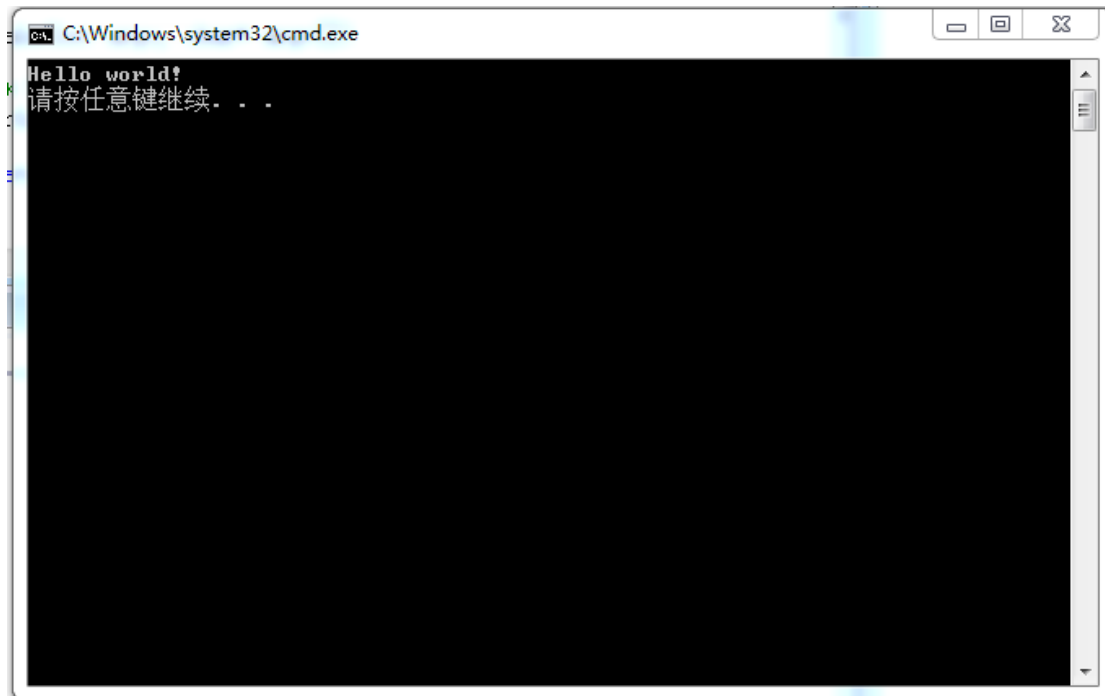
为了使用定位错误代码的功能，我们故意制造错误，并且再次生成项目，由此得到了下面的结果



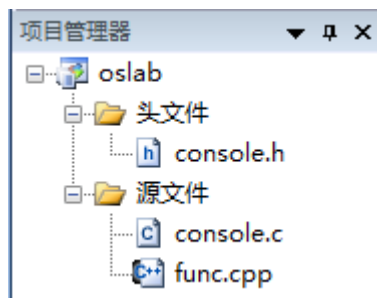
生成结果的检查；



启动执行，我们得到了如下 Windows 控制台窗口

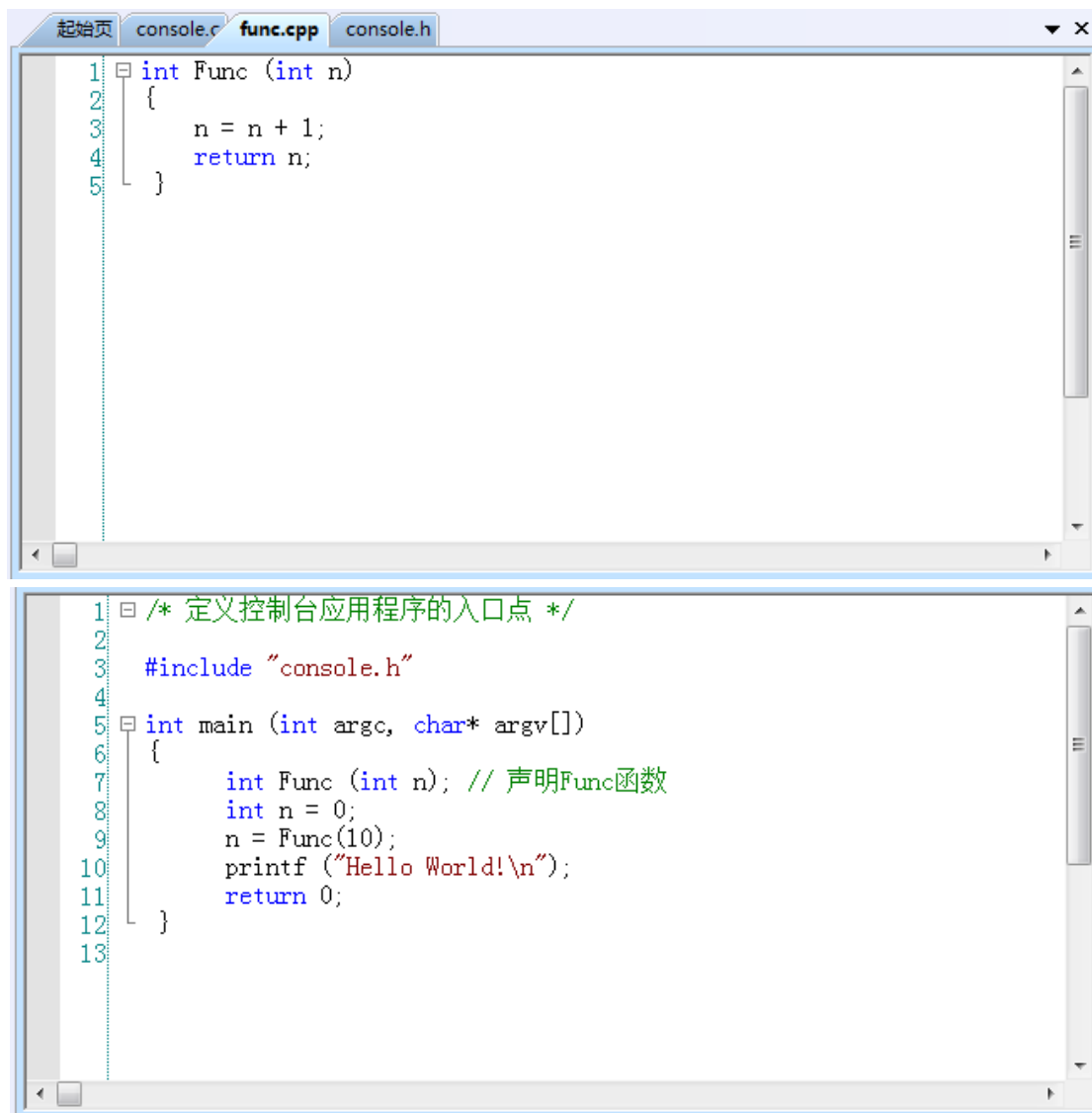


添加了新文件 func



使用此调试器可以观察程序的运行时行为并确定逻辑 错误的位置，可以中断（或挂起）程序的执行以检查代码，计算和编辑程序中的变量，查看寄存器，以及 查看从源代码创建的指令。为了顺利进行后续的各项实验，应该学会灵活使用这些调试功能。

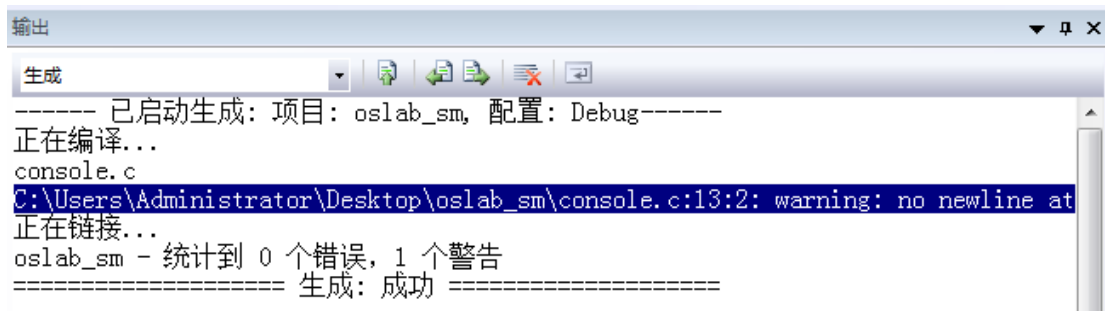
在开始练习各种调试功能之前，首先需要对刚刚创建的例子程序进行必要的修改，在代码编辑器中添加和修改函数：



```
1 int Func (int n)
2 {
3     n = n + 1;
4     return n;
5 }

1 /* 定义控制台应用程序的入口点 */
2
3 #include "console.h"
4
5 int main (int argc, char* argv[])
6 {
7     int Func (int n); // 声明Func函数
8     int n = 0;
9     n = Func(10);
10    printf ("Hello World!\n");
11    return 0;
12 }
13
```

代码修改完毕后按 F7（“生成项目”功能的快捷键）。注意查看“输出”窗口中的内容，如果代码中存 在语法错误，就根据错误信息进行修改，直到成功生成项目。运行成功：



```
输出
生成
----- 已启动生成: 项目: oslab_sm, 配置: Debug-----
正在编译...
console.c
C:\Users\Administrator\Desktop\oslab_sm\console.c:13:2: warning: no newline at
正在链接...
oslab_sm - 统计到 0 个错误, 1 个警告
===== 生成: 成功 =====
```

在 main 函数中定义变量 n 的代码行 `int n = 0;` 上点击鼠标右键，在弹出的快捷菜单中选择“插入/删除断点”，会在此行左侧的空白处显示一 个红色圆点，表示已经成功在此行代码添加了一个断点；
插入断点：

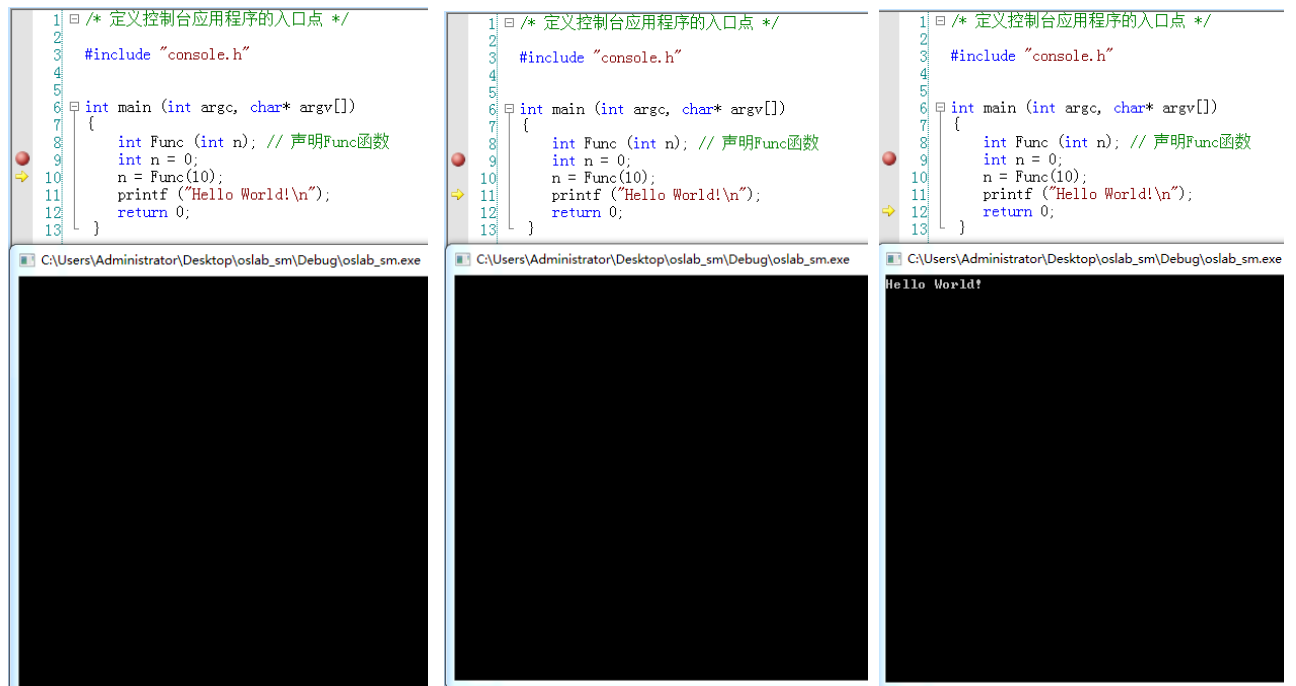
```
1  /* 定义控制台应用程序的入口点 */
2
3  #include "console.h"
4
5
6  int main (int argc, char* argv[])
7  {
8      int Func (int n); // 声明Func函数
9      int n = 0;
10     n = Func(10);
11     printf ("Hello World!\n");
12     return 0;
13 }
```

激活控制台窗口，可以看到窗口中没有输出任何内容，因为 printf 函数还没有被执行。

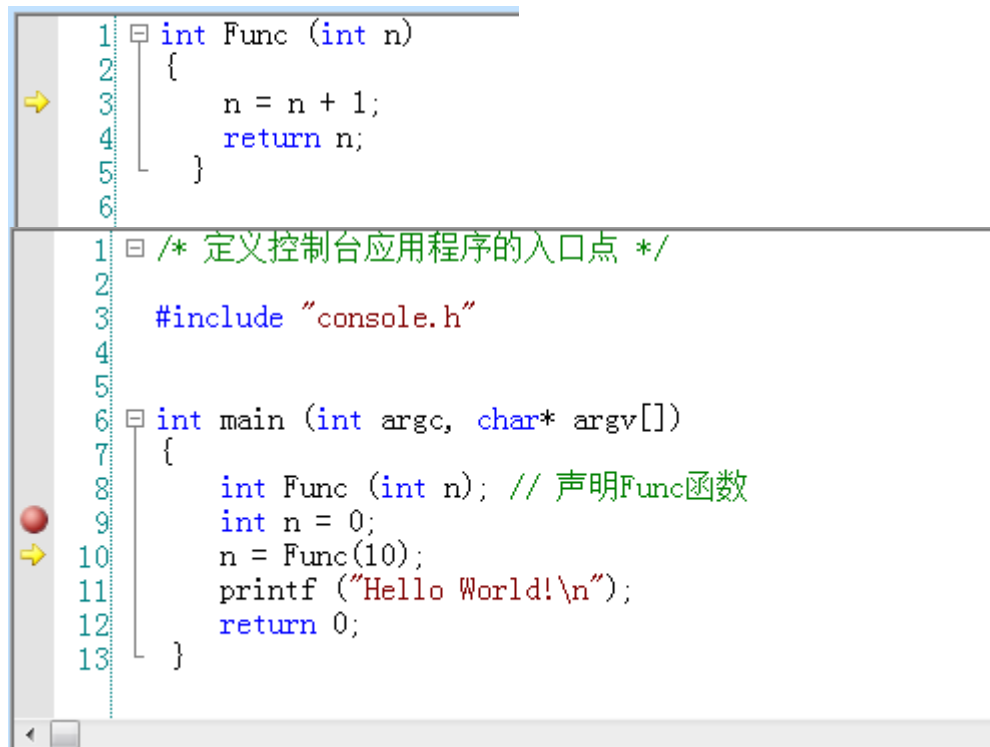
```
1  /* 定义控制台应用程序的入口点 */
2
3  #include "console.h"
4
5
6  int main (int argc, char* argv[])
7  {
8      int Func (int n); // 声明Func函数
9      int n = 0;
10     n = Func(10);
11     printf ("Hello World!\n");
12     return 0;
13 }
```

C:\Users\Administrator\Desktop\oslab_sm\Debug\oslab_sm.exe

下面试试逐过程输出 Hello World!



然后试试逐语句和跳出

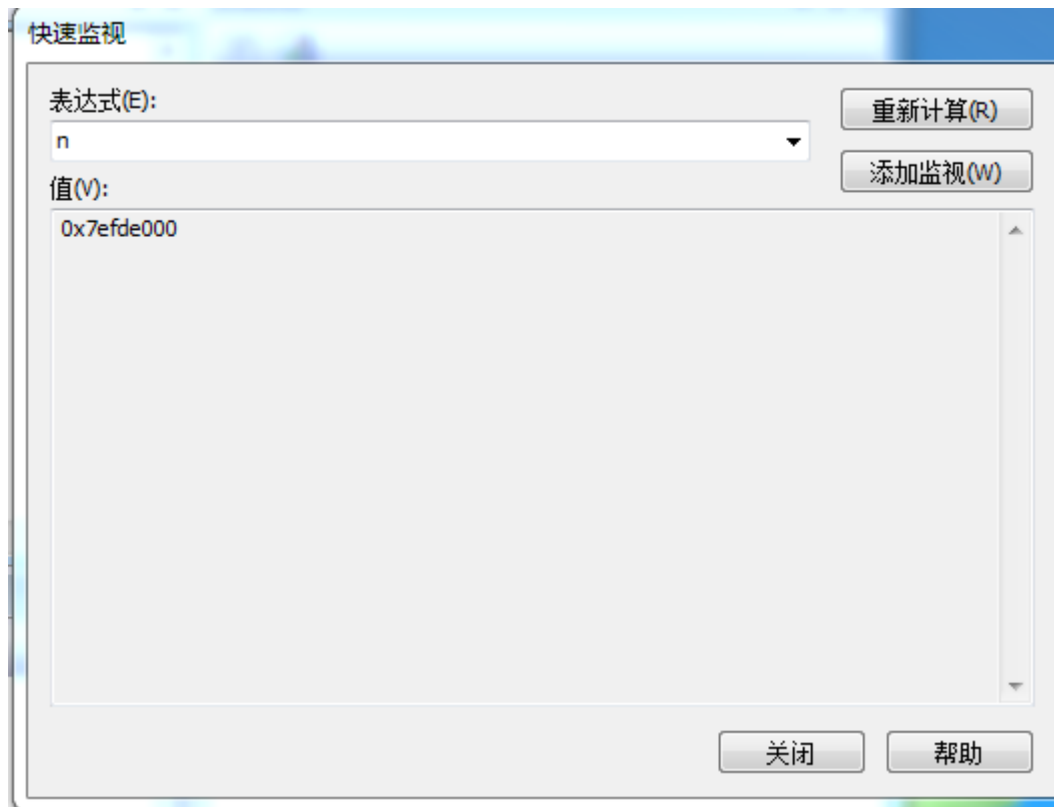


体会“逐过程”和“逐语句”的不同。

在调试的过程中，OS Lab 提供了三种查看变量值的方法，按照下面的步骤练习这些方法：

```
1  /* 定义控制台应用程序的入口点 */
2
3  #include "console.h"
4
5
6  int main (int argc, char* argv[])
7  {
8      int Func (int n); // 声明Func函数
9      int n = 0;
10     n = Func(n = 0x7efde000);
11     printf ("Hello World!\n");
12     return 0;
13 }
```

首先是快速监视



监视		
名称	值	类型
n	0x7efde000	int

如果需要使用十进制查看变量的值，可以点击工具栏上的“十六进制”按钮，从而在十六进制和十进制间切换。可以使用不同的进制和不同的方法来查看变量的值，然后结束调试。


```

1  /* 定义控制台应用程序的入口点 */
2
3  #include "console.h"
4
5
6  int main (int argc, char* argv[])
7  {
8      int Func (int n); // 声明Func函数
9      int n = 0;
10     n = Func(10);
11     printf ("Hello World!\n");
12     return 0;
13 }

```

监视

名称	值	类型
n	0x0	int

监视

名称	值	类型
n	0	int

接下来试试调用堆栈

```

1  /* 定义控制台应用程序的入口点 */
2
3  #include "console.h"
4
5
6  int main (int argc, char* argv[])
7  {
8      int Func (int n); // 声明Func函数
9      int n = 0;
10     n = Func(10);
11     printf ("Hello World!\n");
12     return 0;
13 }

```

监视

名称	值	类型
n	2130567168	int

```

1  /* 定义控制台应用程序的入口点 */
2
3  #include "console.h"
4
5
6  int main (int argc, char* argv[])
7  {
8      int Func (int n); // 声明Func函数
9      int n = 0;
10     n = Func(10);
11     printf ("Hello World!\n");
12     return 0;
13 }

```

调用堆栈

名称
main(argc=1, argv=0x3c2d20) 地址:0x0040131a

```
1 int Func (int n)
2 {
3     n = n + 1;
4     return n;
5 }
6
```

调用堆栈

名称
→ Func(n=10) 地址:0x00401347
main(argc=1, argv=0x3c2d20) 地址:0x0040132d

```
1 /* 定义控制台应用程序的入口点 */
2
3 #include "console.h"
4
5
6 int main (int argc, char* argv[])
7 {
8     int Func (int n); // 声明Func函数
9     int n = 0;
10    n = Func(10);
11    printf ("Hello World!\n");
12    return 0;
13 }
```

调用堆栈

名称
→ Func(n=10) 地址:0x00401347
→ main(argc=1, argv=0x3c2d20) 地址:0x0040132d

```
1 /* 定义控制台应用程序的入口点 */
2
3 #include "console.h"
4
5
6 int main (int argc, char* argv[])
7 {
8     int Func (int n); // 声明Func函数
9     int n = 0;
10    n = Func(10);
11    printf ("Hello World!\n");
12    return 0;
13 }
```

调用堆栈

名称
→ main(argc=1, argv=0x3c2d20) 地址:0x0040131a

反复双击“调用堆栈”窗口中 Func 函数和 main 函数所在的行，查看“监视”窗口中变量 n 的值， 可以看到在不同的堆栈帧被激活时，OS Lab 调试器会自动更新“监视”窗口中的数据，显示出对 应于当前活动堆栈帧的信息。

```
1 int Func (int n)
2 {
3     n = n + 1;
4     return n;
5 }
6
```

调用堆栈	
名称	
→ Func(n=10) 地址:0x00401347	
main(argc=1, argv=0x3c2d20) 地址:0x0040132d	

```
1 /* 定义控制台应用程序的入口点 */
2
3 #include "console.h"
4
5
6 int main (int argc, char* argv[])
7 {
8     int Func (int n); // 声明Func函数
9     int n = 0;
10    n = Func(10);
11    printf ("Hello World!\n");
12    return 0;
13 }
```

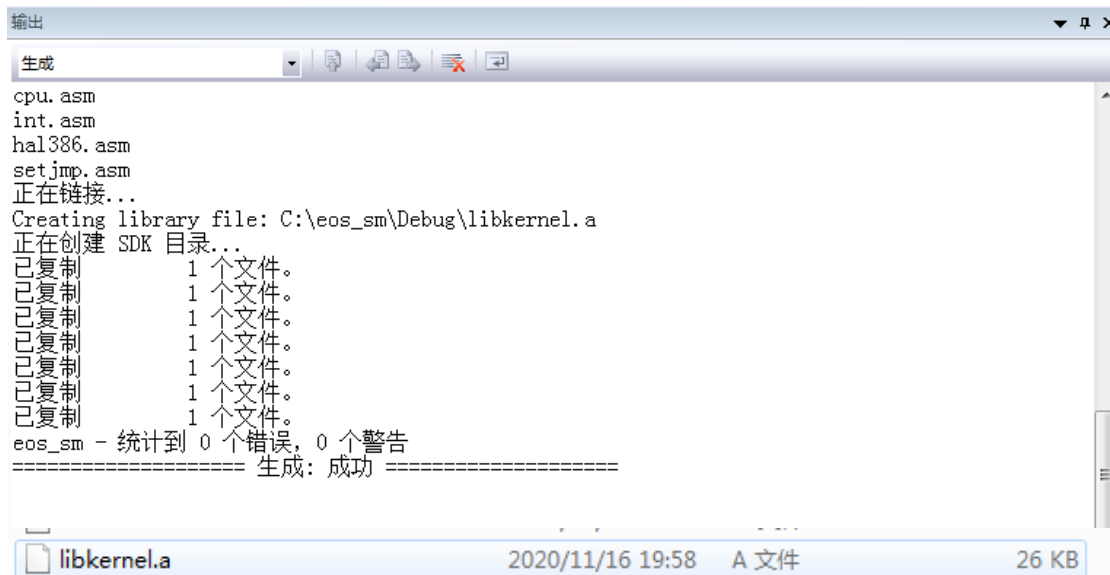
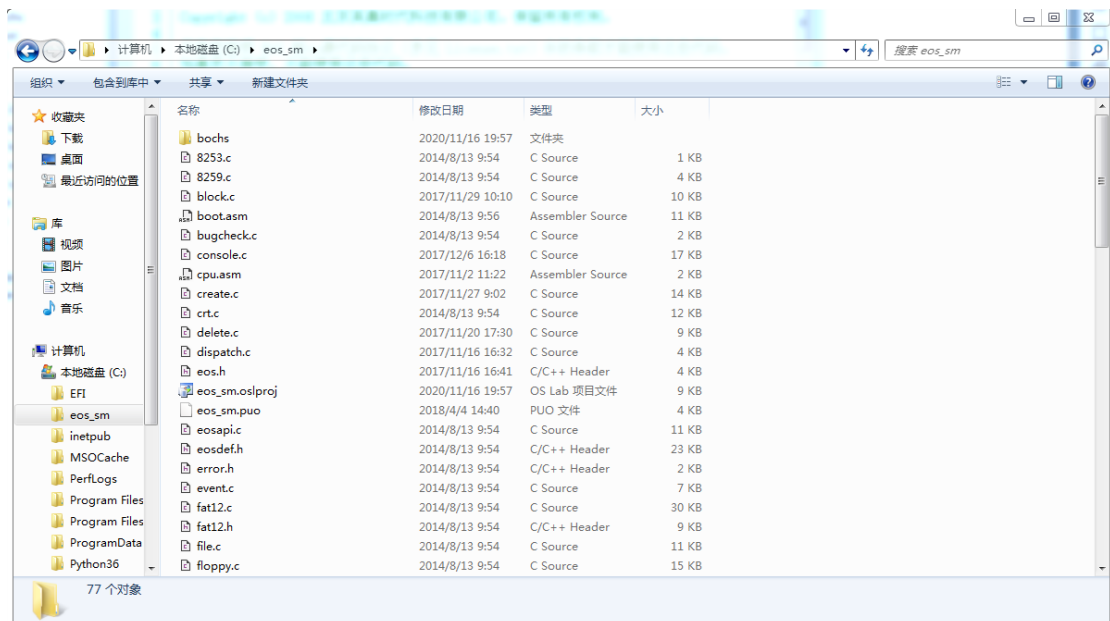
调用堆栈	
名称	
→ Func(n=10) 地址:0x00401347	
→ main(argc=1, argv=0x3c2d20) 地址:0x0040132d	

```
1 int Func (int n)
2 {
3     n = n + 1;
4     return n;
5 }
```

监视			
名称	值	类型	
n	10	int	

```
8 int Func (int n); // 声明Func函数
9 int n = 0;
10 n = Func(10);
11 printf ("Hello World!\n");
12 return 0;
```

监视			
名称	值	类型	
n	0	int	

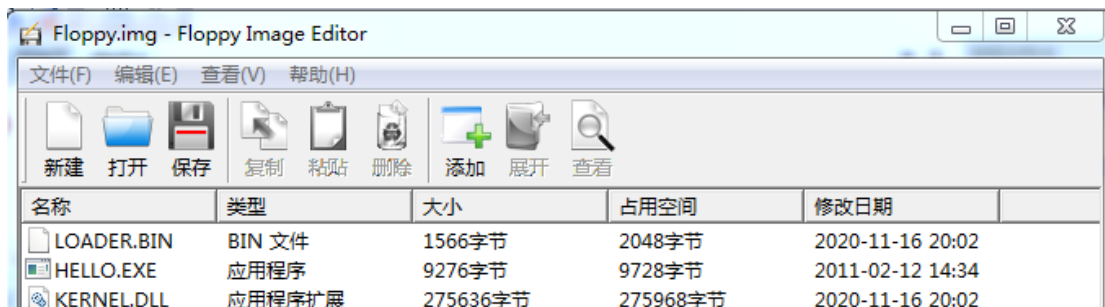
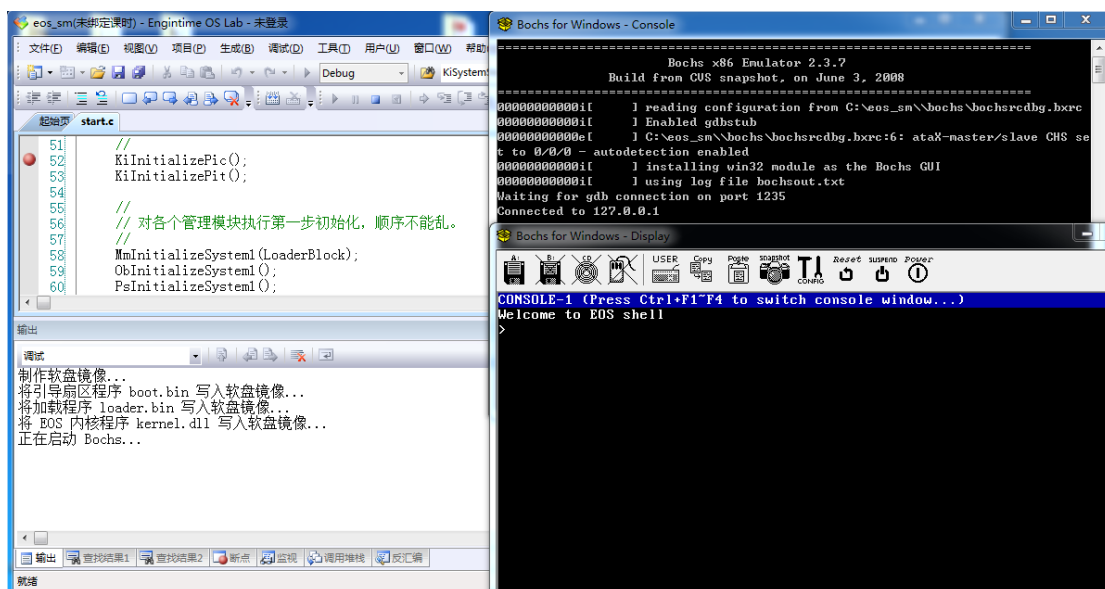
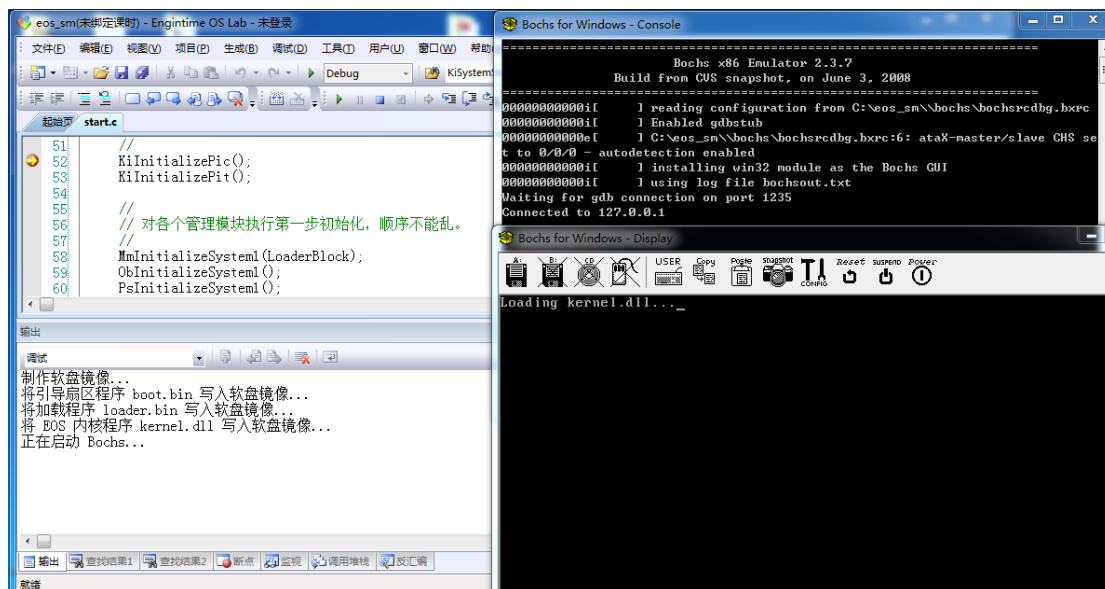


接下来新建一个 EOS 内核项目，并进行相应的操作

对 EOS 内核项目的各种操作（包括新建、生成和各种调试功能等）与对 Windows 控制台项目的操作是完全一致的。所以，接下来实验内容的重点不再 是各种操作的具体步骤，而应将注意力放在对 EOS 内核项目的理解上。

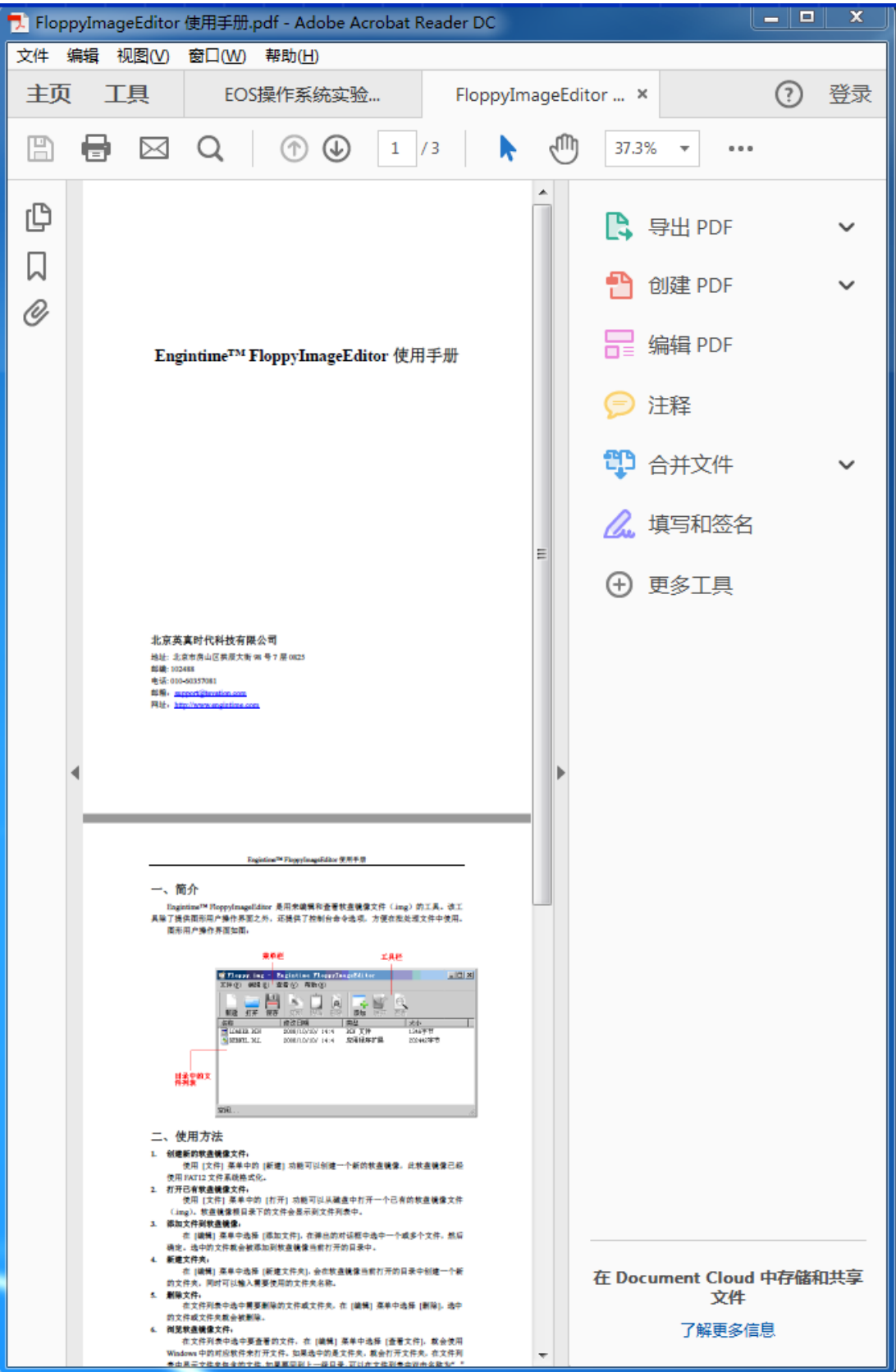
在“项目管理器”窗口中查看 EOS 内核项目包含的文件夹和源代码文件，可以看到不同的文件夹包含了 EOS 操作系统不同模块的源代码文件，例如“mm”文件夹中包含了内存管理模块的源代码文件，“boot”文件夹中包含了软盘引导扇区程序和加载程序的源代码文件。也可以使用 Windows 资源管理器打开项目所在 的文件夹 C:\eos，查看所有源代码文件。

按 F5 启动调试，虚拟机开始运行软盘镜像中的 EOS。在虚拟机窗口中可以看到 EOS 启动的过程。随后 EOS 会在刚刚添加的断点处中断执行。激活虚拟机窗口可以看到 EOS 也不再继续运行了。各 种调试功能（包括单步调试、查看变量的值和各个调试工具窗口）的使用方法与调试 Windows 控制台程序完全相同

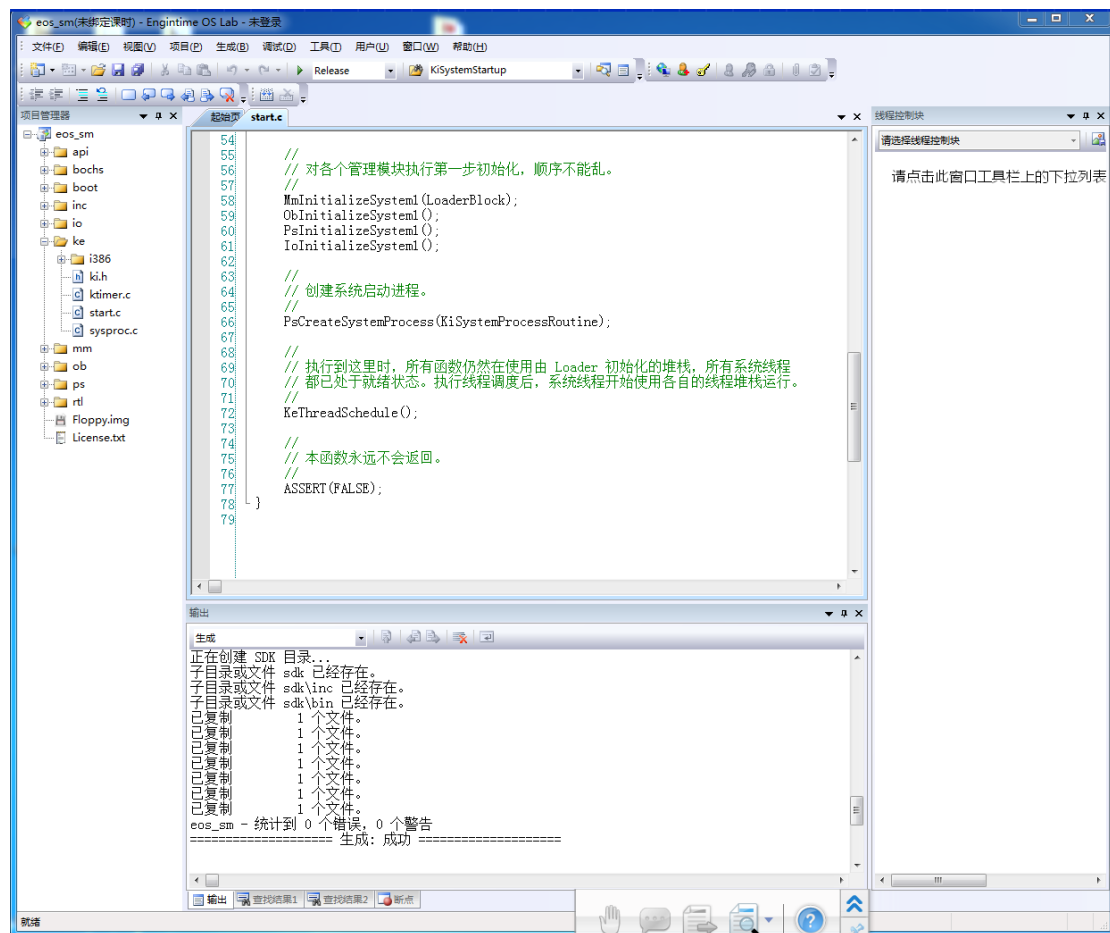


在“项目 管理器”窗口中双击软盘镜像文件 Floppy.img，就会使用 FloppyImageEditor 工具打开此文件(在 FloppyImageEditor 工具中按 F1 可以查看此工具的帮助文件)。在 FloppyImageEditor 工具的文件 列表中可以找到 loader.bin 文件和 kernel.dll 文件，这两个文件都是在启动调试时被写入软盘镜像文件 的(可以查看这两个文件的修改日期) boot.bin 文件在启动调试时被写入了软盘镜像的引导扇区中，不受软盘文件系统的管理，所以在文件列表中找不到此文件。关闭 FloppyImageEditor 工具。

(在 FloppyImageEditor 工具中按 F1 可以查看此工具的帮助文件)



查看 EOS SDK (Software Development Kit) 文件夹在文件夹中多出了一个 SDK 文件夹, 此文件夹就是在生成 EOS Kernel 项目的同时自动生成的。4. SDK 文件夹中提供了开发 EOS 应用程序需要的所有文件。



sdk 2020/11/16 19:58 文件夹

计算机 > 本地磁盘 (C:) > eos_sm > sdk > bin > debug

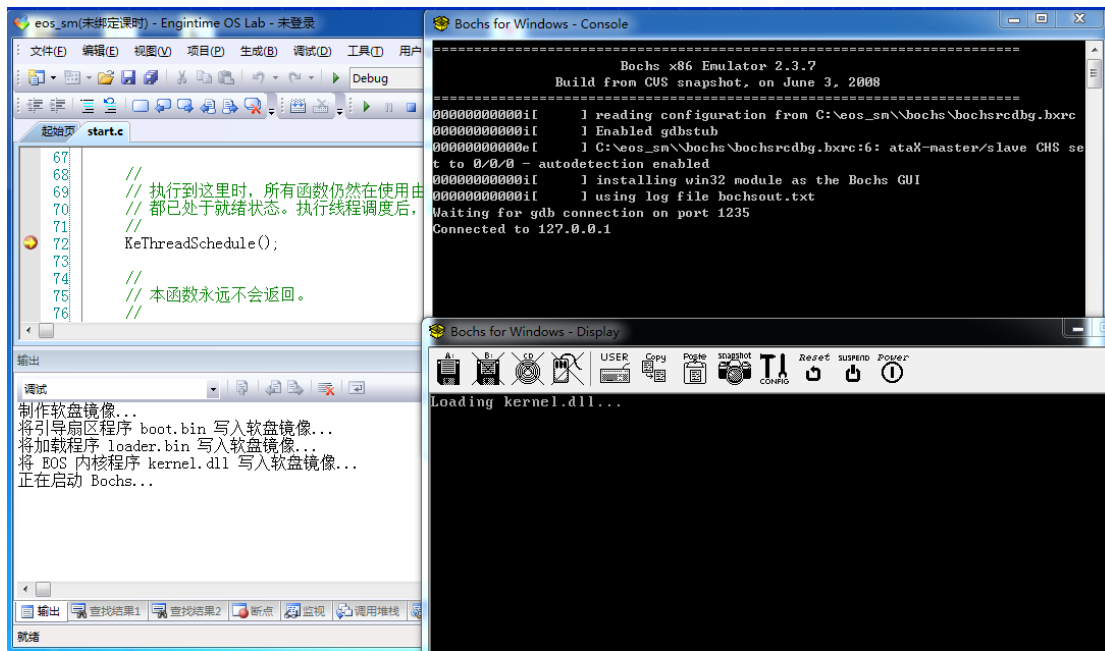
名称	修改日期	类型	大小
boot.bin	2020/11/16 19:58	BIN 文件	1 KB
kernel.dll	2020/11/16 19:58	应用程序扩展	270 KB
libkernel.a	2020/11/16 19:58	A 文件	26 KB
loader.bin	2020/11/16 19:58	BIN 文件	2 KB

计算机 > 本地磁盘 (C:) > eos_sm > sdk > bin > release

名称	修改日期	类型	大小
boot.bin	2020/11/16 20:13	BIN 文件	1 KB
kernel.dll	2020/11/16 20:13	应用程序扩展	95 KB
libkernel.a	2020/11/16 20:13	A 文件	26 KB
loader.bin	2020/11/16 20:13	BIN 文件	2 KB

计算机 > 本地磁盘 (C:) > eos_sm > sdk > inc

名称	修改日期	类型	大小
eos.h	2017/11/16 16:41	C/C++ Header	4 KB
eosdef.h	2014/8/13 9:54	C/C++ Header	23 KB
error.h	2014/8/13 9:54	C/C++ Header	2 KB



初识可视化窗口，点击“对象类型”窗口工具栏上的“刷新”按钮，会显示如图9-10所示的内容。读者可以查看到当前系统中注册的所有对象类型，以及各个对象类型的名称、使用此对象类型创建的对象数量、注册的操作函数等。这些对象将在后面的实验中涉及到，读者在这里有一个初步的了解即可。

对象类型

数据源: SINGLE_LIST_ENTRY ObpTypeListHead
源文件: ob\obtype.c

类型链表

TypeListEntry	
Name	"FILE"
ObjectCount	0
Create	NULL
Delete	0x80015db1 IopCloseFileObject
Wait	NULL
Read	0x80015e3c IopReadFileObject
Write	0x80015f14 IopWriteFileObject

↓

TypeListEntry	
Name	"CONSOLE"
ObjectCount	0
Create	NULL

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

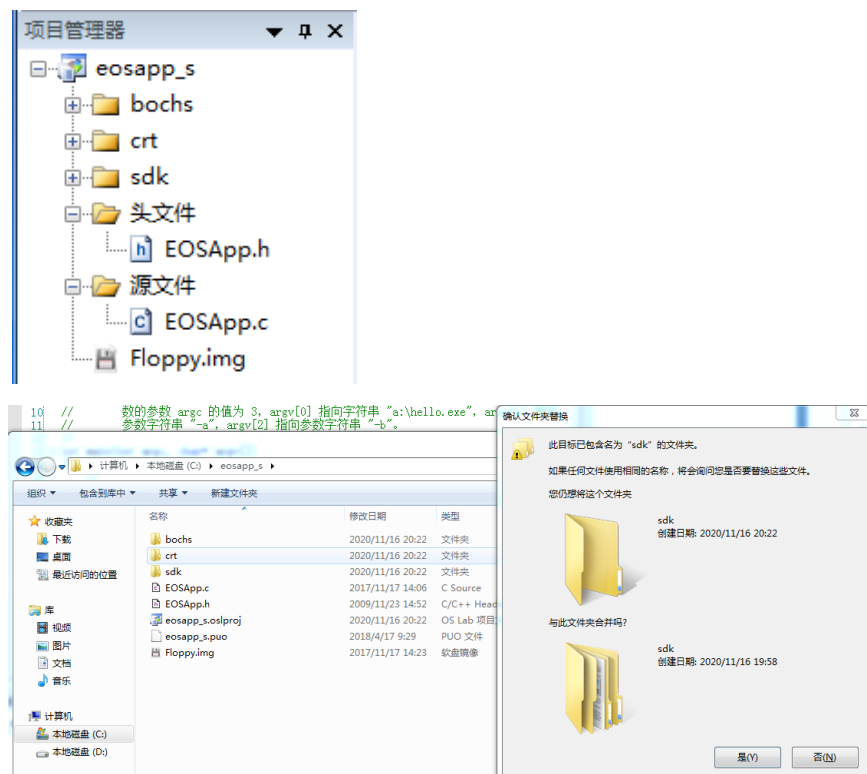
进程列表

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"

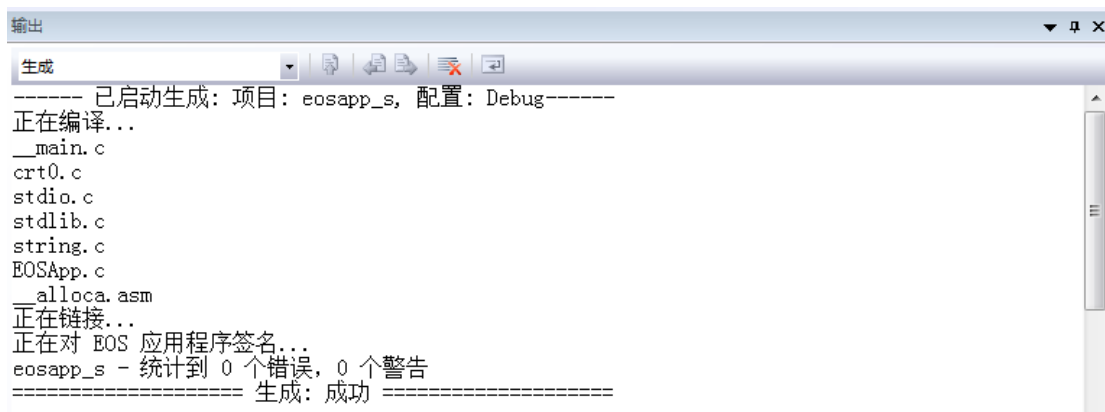
线程列表

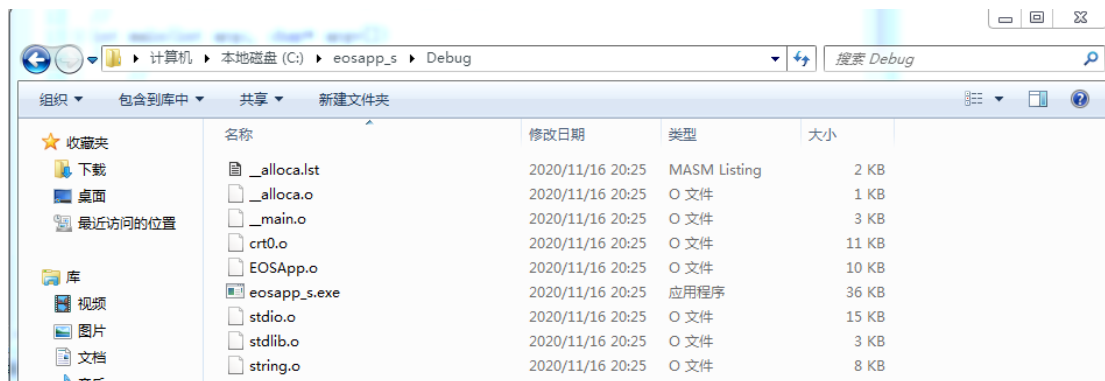
序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	2	Y	24	Ready (1)	1	0x80017e40 KiSystemProcessRoutine

新建 EOS 应用程序项目，接下来，读者使用 Windows 资源管理器将之前由 EOS 内核项目生成的“C:\eos\sdk”文件夹拷贝覆盖 到“C:\eosapp\sdk”位置。这样 EOS 应用程序就可以使用最新版本的 EOS SDK 文件夹了。

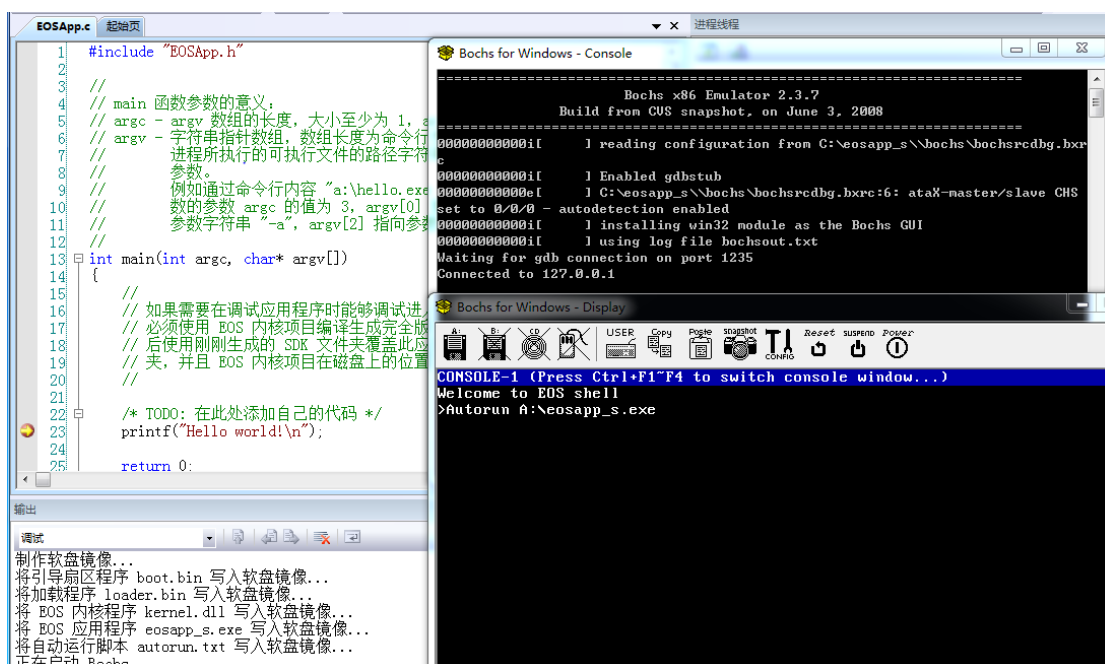


生成项目 OS Lab 每次启动执行 EOS 应用程序时，都会将 EOS 应用程序的可执行文件写入软盘镜像，并且会将 SDK 文件夹中对应配置（Debug 或 Release）的 EOS 内核二进制文件 写入软盘镜像，然后让虚拟机运行软盘镜像中的 EOS，待 EOS 启动后再自动执行 EOS 应用程序





调试项目:调试 EOS 应用程序项目与之前练习的方法类似,在 eosapp.c 的 printf("Hello world!\n"); 代码行添加一个断点。2. 按 F5 启动调试。会在刚刚添加的断点处中断。



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

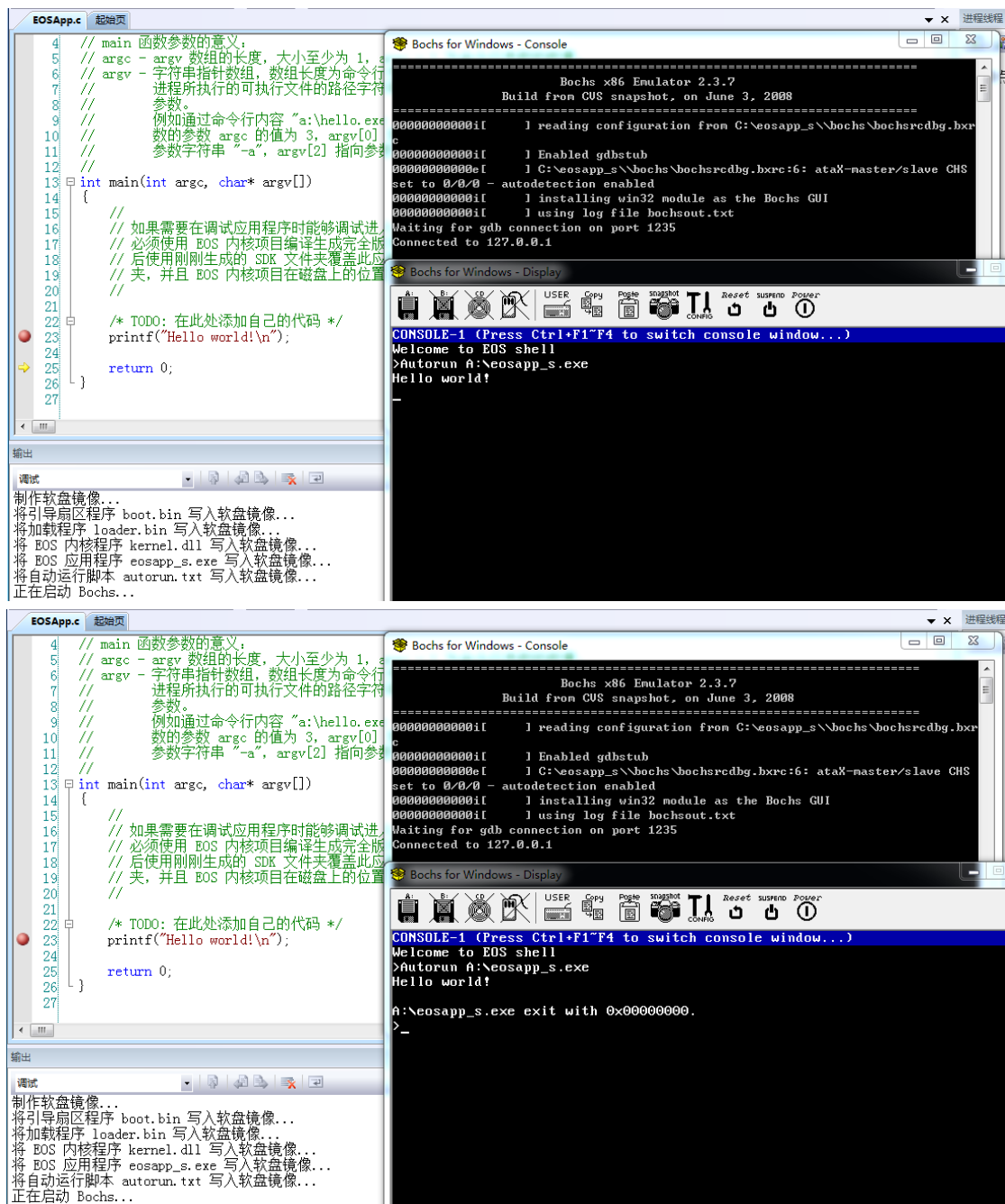
进程列表

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"

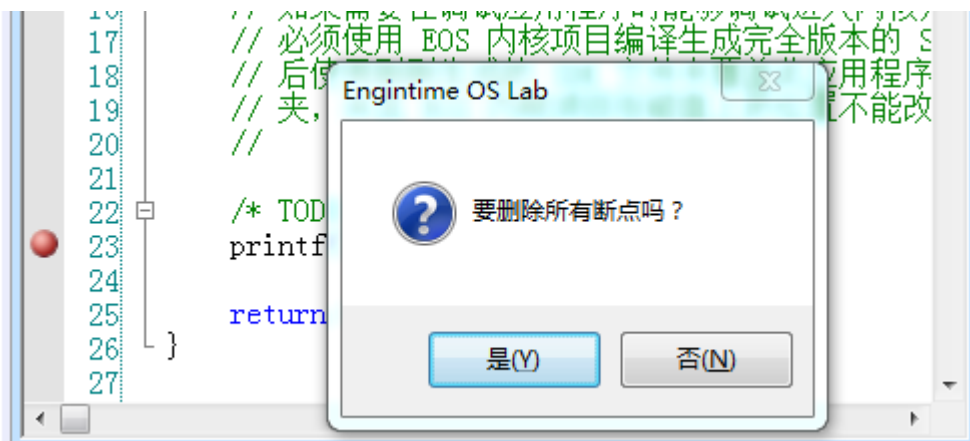
线程列表

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

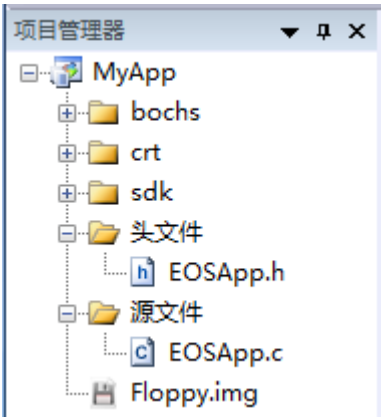


选择“调试”菜单中的“删除所有断点”，可以将程序中的所有断点都删除。



使用 FloppyImageEditor 工具打开该项目中的 Floppy.img 文件，查看软盘镜像中的文件。loader.bin 和 kernel.dll 是从 C:\eosapp\sdk\bin\debug 文件夹写入的，C:\eosapp\sdk\bin\debug\boot.bin 被写入了软盘镜像文件的引导扇区中。eosapp.exe 就是本项目生成的 EOS 应用程序。EOS 操作系统启动后会根据 autorun.txt 文本文件中的内容启动执行 eosapp.exe 程序，双击 autorun.txt 文件查看其内容。

修改 EOS 应用程序名称成功



4. 实验的思考与问题分析

(1) 练习使用单步调试功能（逐过程、逐语句），体会在哪些情况下应该使用“逐过程”调试，在哪些情况下应该使用“逐语句”调试。练习使用各种调试工具（包括“监视”窗口、“调用堆栈”窗口等）。

答；

逐语句，就是每次执行一行语句，如果碰到函数调用，它就会进入到函数里面。

逐过程，碰到函数时，不进入函数，把函数调用当成一条语句执行。因此，在需要进入函数体时用逐语句调试，而不需要进入函数体时用逐过程调试。

(2) 思考生成 EOS SDK 文件夹的目的和作用。查看 EOS SDK 文件夹中的内容，明白文件夹的组织结构和各个文件的来源和作用。查看 EOS 应用程序包含了 SDK 文件夹中的哪些头文件，是如何包含的？

答：

EOS SDK 是为应用程序调用系统 API 提供服务，可作为用户编程中可使用的工具包集合。EOS SDK 文件夹主要包括 INC 头文件、LIB 文件夹导入库文件和 BIN 文件夹动态链接库，可执行程序，二进制文件。EOS SDK 包含的头文件有：eos.h 负责导出 API 函数声明；eosdef.h 负责导出函数类型的定义；error.h 负责导出错误码。

4. 总结和感想体会

熟悉了 OSLab 实验环境，学会了 EOS 操作系统内核和 EOS 应用程序的基本调试方法，对 EOS 相关项目的编程方法增加了了解。通过这次实验，我锻炼了动手实践的能力，提升了对于操作系统相关知识的兴趣，加深对操作系统理论知识的理解，并能将其应用到实际操作当中。

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.