《漏洞利用及渗透测试基础》实验报告

姓名：管昀玫 学号：2013750 班级：计科一班

**实验名称：**

程序插桩及Hook实验

**实验要求：**

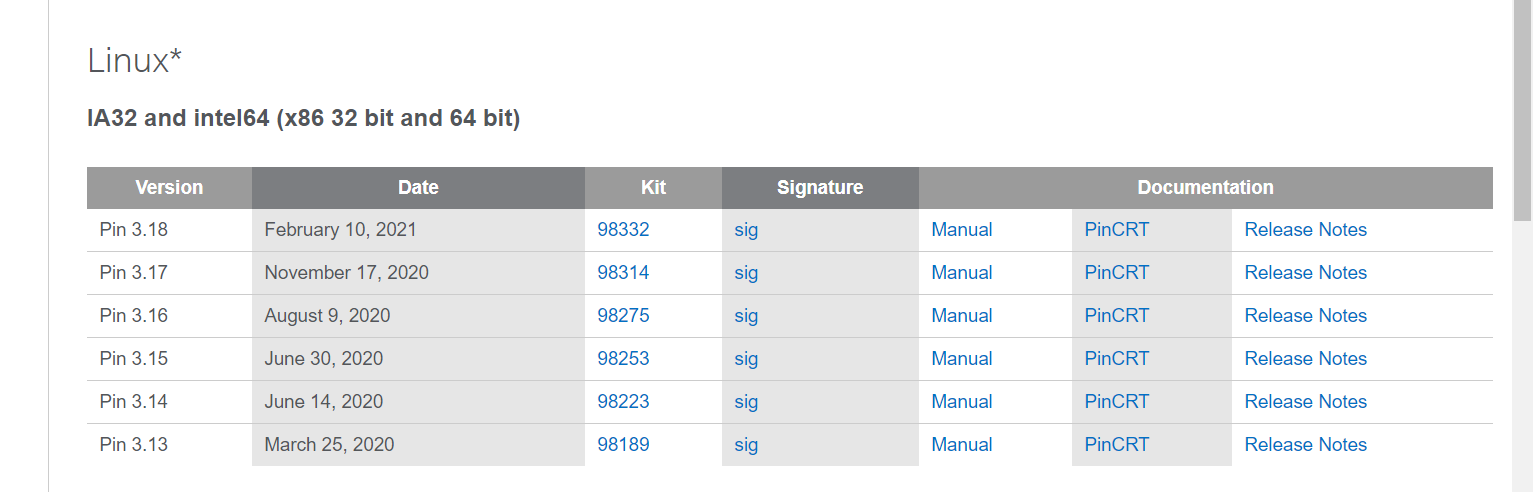
复现实验一，基于Windows MyPinTool或在Kali中复现malloctrace这个PinTool理解Pin插桩工具的核心步骤和相关API，关注malloc和free函数的输入输出信息。

**实验过程：**

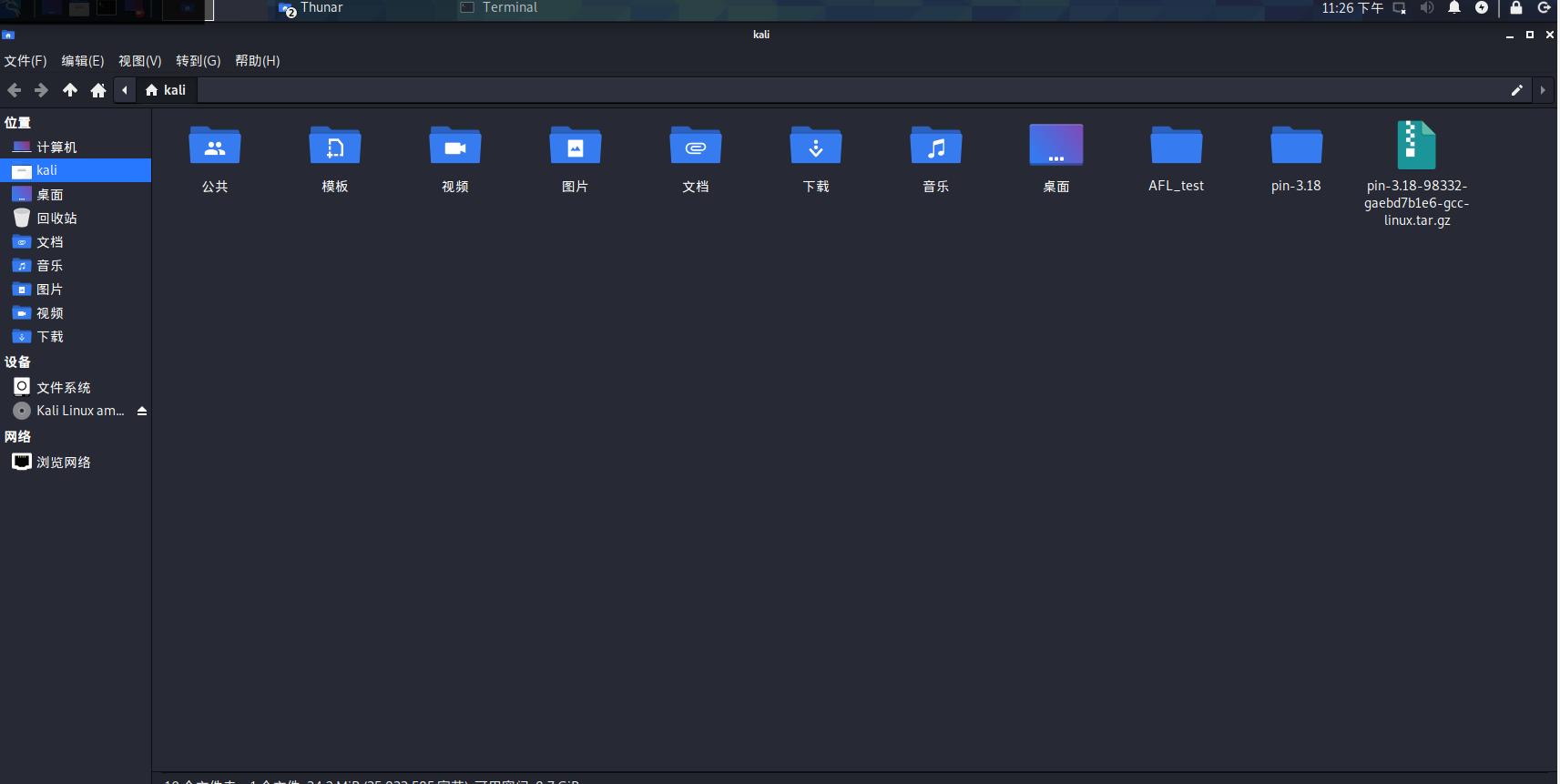
1. 实验准备
   1. 安装pintool工具

进入到pintool的官网

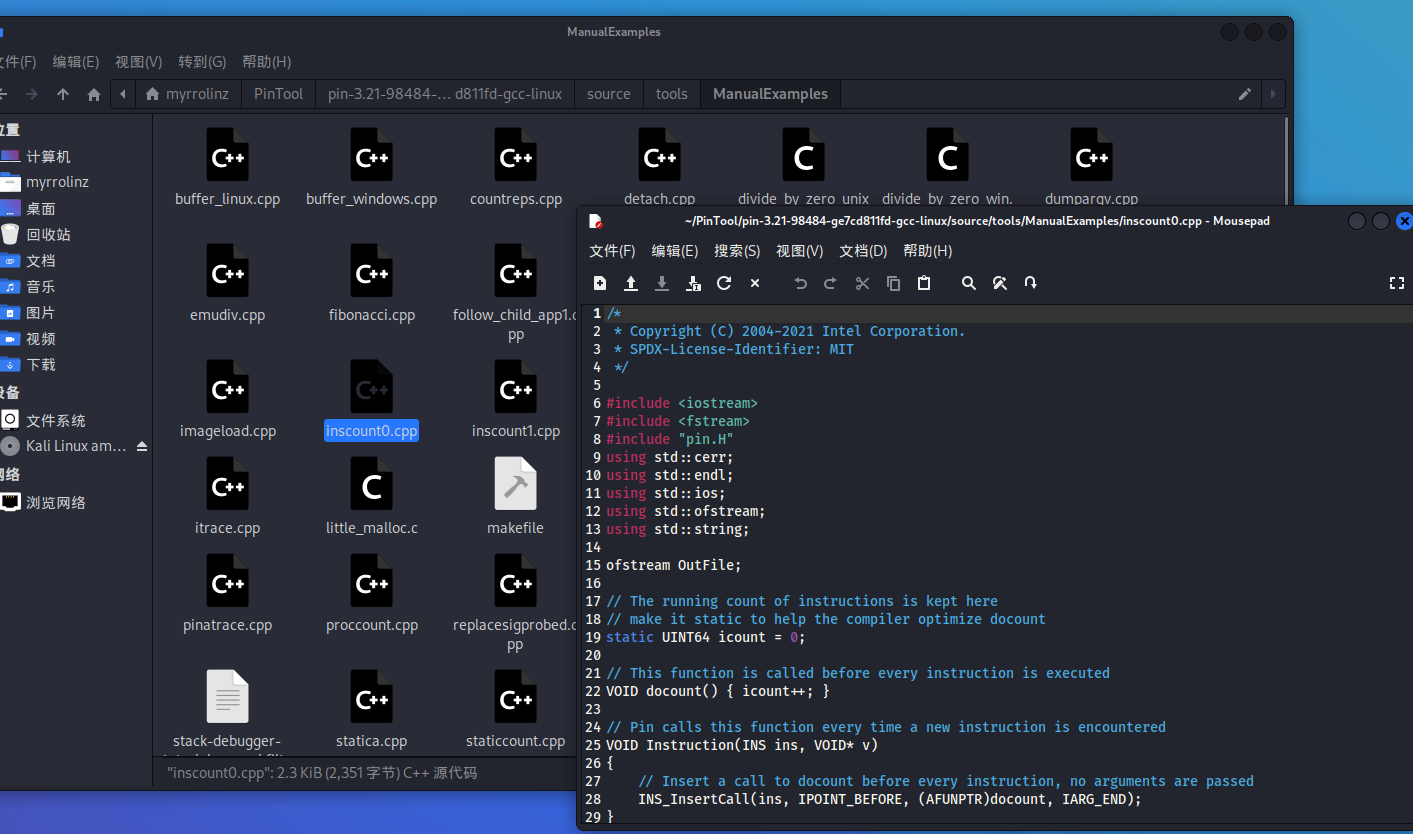
<https://software.intel.com/content/www/us/en/develop/articles/pin-a-binary-instrumentation-tool-downloads.html>



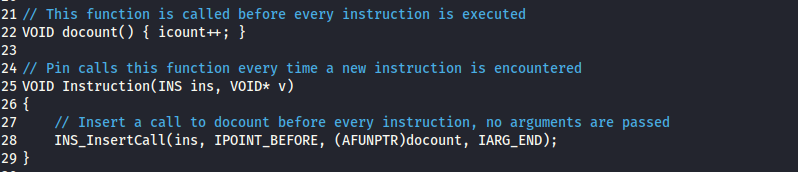
找到对应linux版本下的工具，下载并解压



pintool工具准备完成

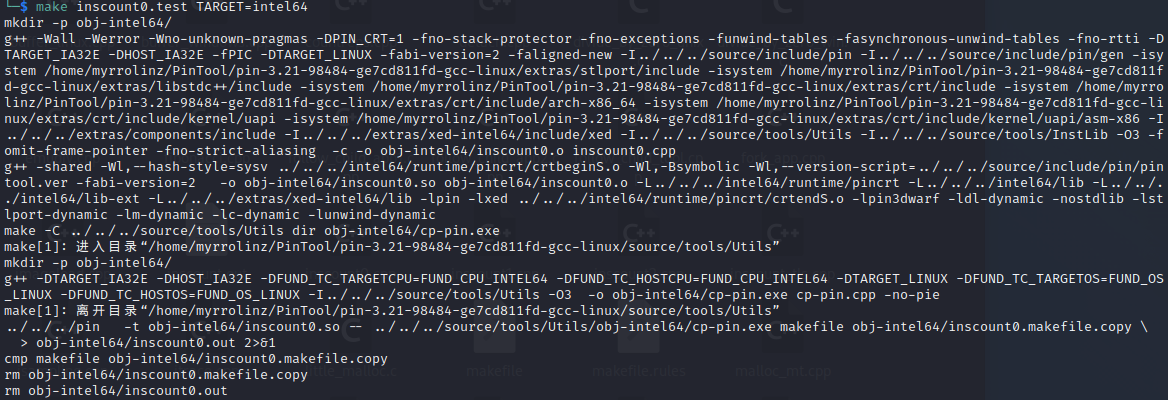


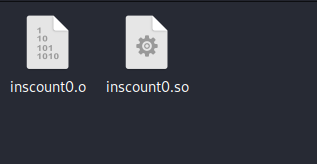
可以看见ManualExamples里有inscount0.cpp文件。



发现这个文件是使用Instruction这个函数来实现插桩，每个指令实现count++。

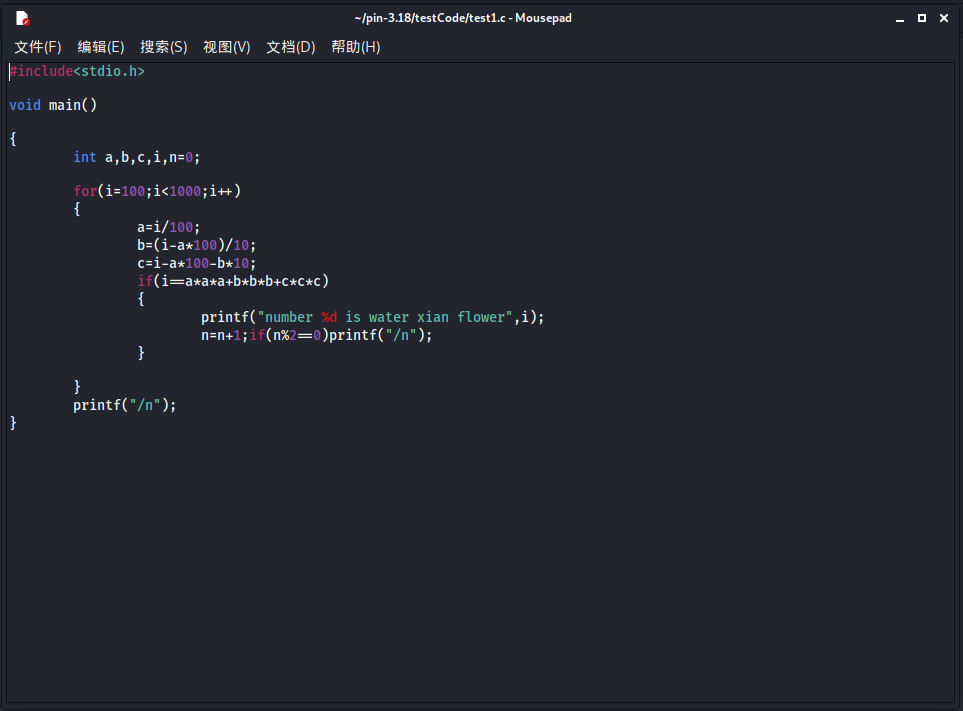
编译运行inscount0.cpp文件，以产生动态链接库。





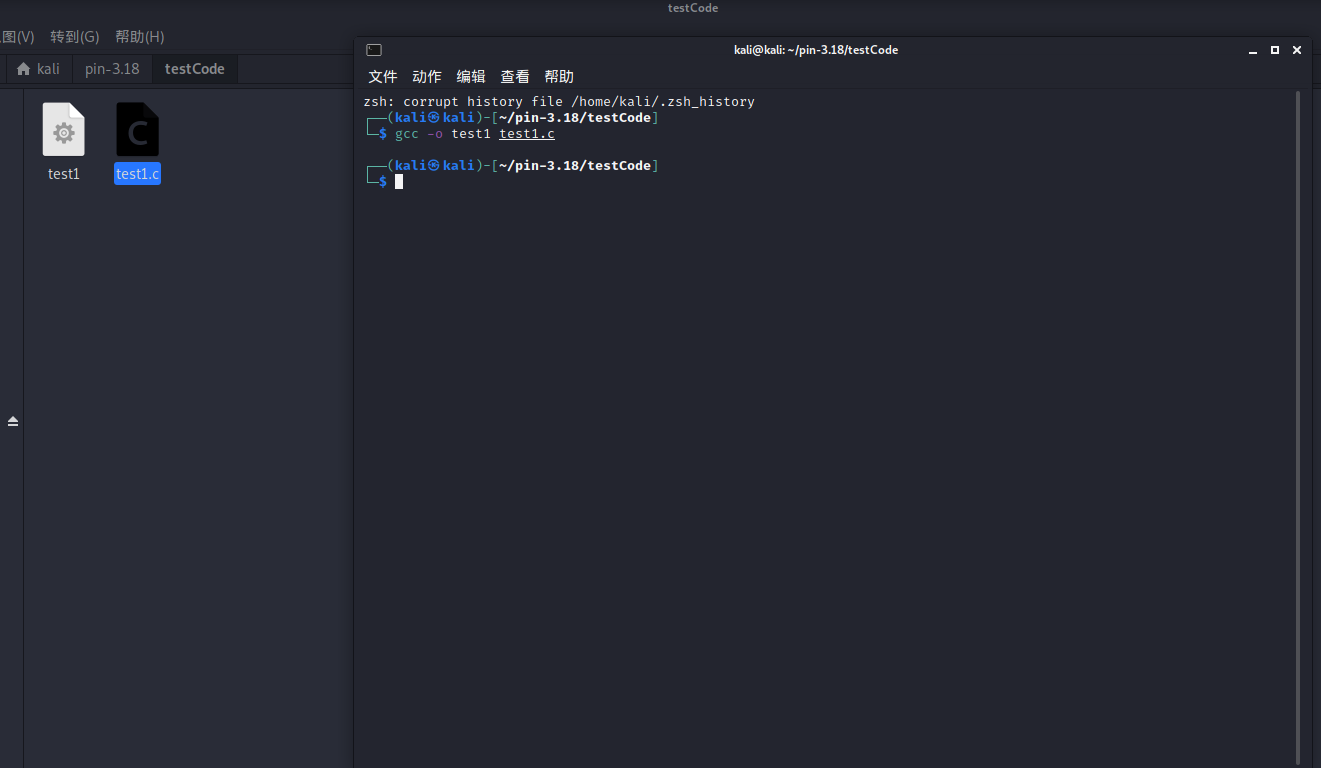
编译成功后，出现了inscount0.so文件。

1. 源程序准备



写了一个输出水仙花数的C语言小程序（本来想用C++的，但是C++的程序貌似不能用gcc编译，还没有尝试过用make编译，不知道这样编译了以后行不行，于是使用了C的程序）

使用gcc -o test1 test1.c进行编译，得到test1



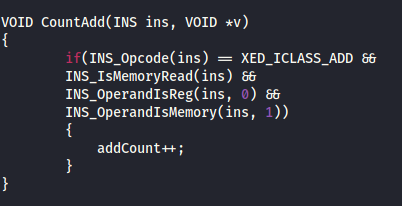
1. 编写MyPinTool

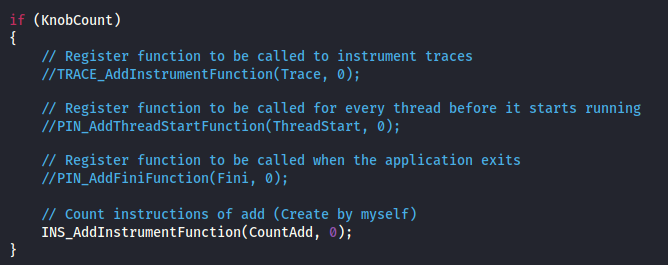
想法：对每一个add操作都进行一次计数

借鉴icount.cpp中的代码编写如下：

新增一个自己的变量，用于记录add指令的数目

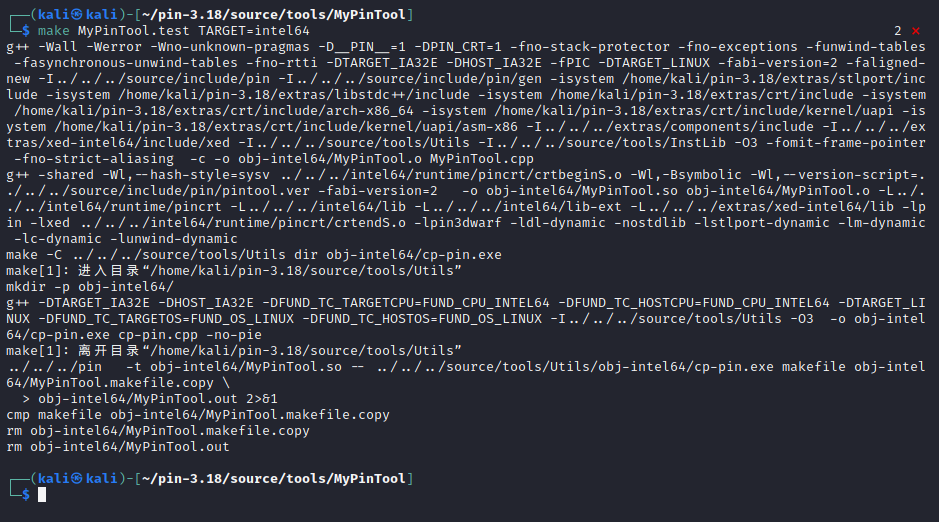




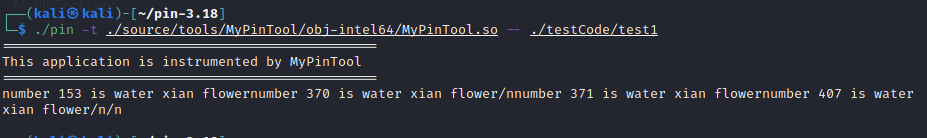


并且添加了incount0.cpp中关于日志输出的代码

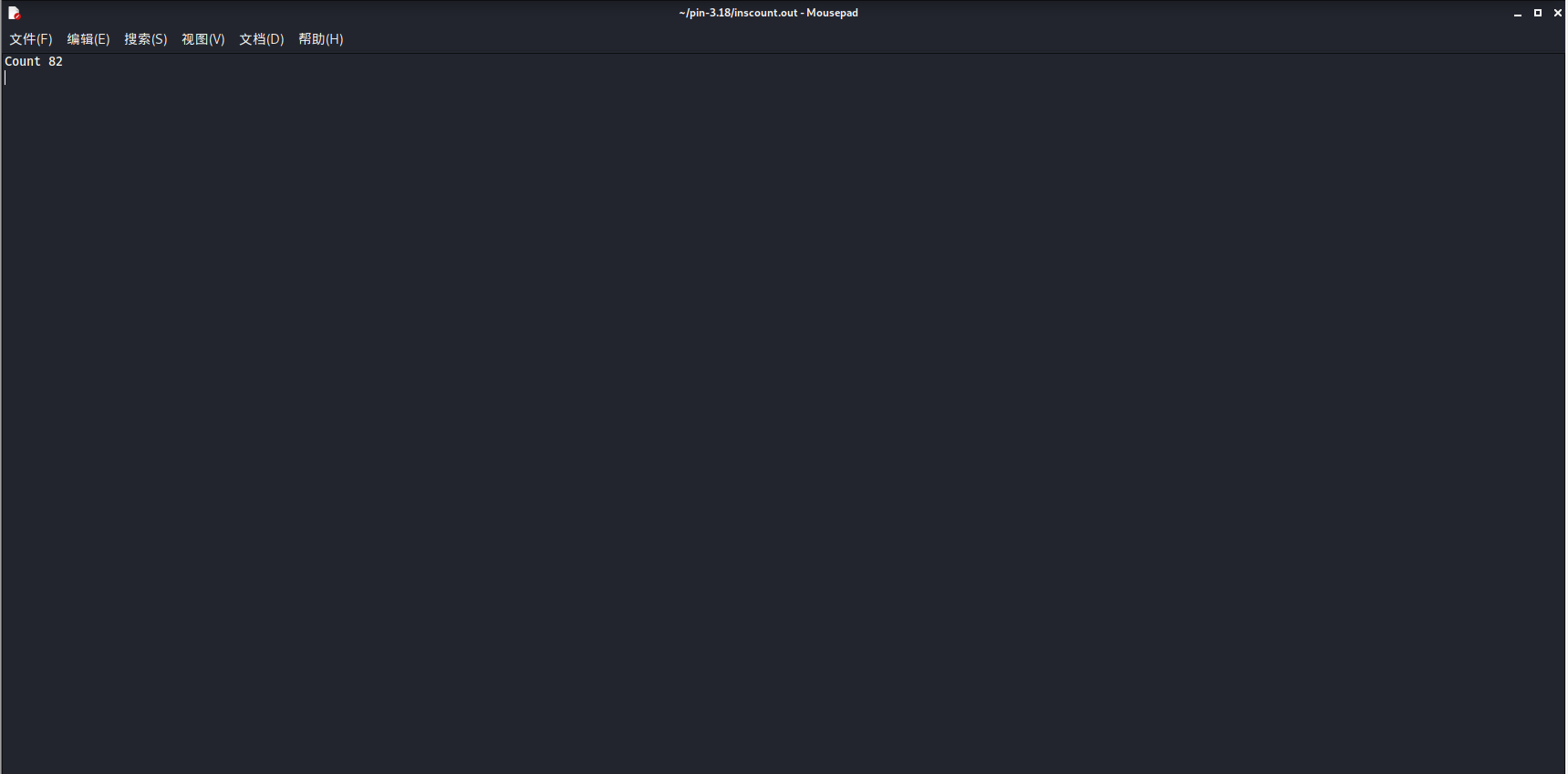
编译MyPinTool



使用MyPinTool进行插桩操作



查看输出文件



可以看到有82条add的指令

实验复现成功

4.指定插桩位置：

将回调函数Instruction修改如下：

VOID Instruction(INS ins, VOID \*v)

{

if (INS\_Opcode(ins) == XED\_ICLASS\_MOV &&

INS\_IsMemoryRead(ins) &&

INS\_OperandIsReg(ins, 0) &&

INS\_OperandIsMemory(ins, 1))

{

icount++;

}

}

且使用视频中所写的First.cpp文件进行插桩，能看见执行结果：



**心得体会：**

Pintool的基本框架：

在main函数中：

* 初始化。通过调用函数PIN\_Init完成初始化。
* 注册插桩函数。通过使用INS\_AddInstrumentFunction注册一个插桩函数，在原始程序的每条指令被执行前，都会进入Instruction这个函数中，其第2个参数为一个额外传递给Instruction的参数，即对应VOID \*v这个参数，这里没有使用。而Instruction接受的第一个参数为INS结构，用来表示一条指令。
* 注册退出回调函数。通过使用PIN\_AddFiniFunction注册一个程序退出时的回调函数Fini，当应用退出的时候会调用函数Fini。
* 启动程序。使用函数PIN\_StartProgram启动程序。

指令集插桩：指令级插桩的对象就是所有指令。很明显，inscount0.cpp这个Pintool是指令级插桩，通过调用INS\_AddInstrumentFunction注册了一个回调函数Instruction。在Instruction函数中，使用函数INS\_InsertCall注册了一个回调函数docount，意为在指令执行之前插入一个对docount函数的调用。而docount的作用即是将一个全局变量加1，以达到统计执行指令条数的目的。故此处插桩的分析代码即是将指令数加1。可以在inscount0的基础上，扩展出更复杂的插桩分析程序。

例如，将回调函数Instruction修改如下：

VOID Instruction(INS ins, VOID \*v)

{

if (INS\_Opcode(ins) == XED\_ICLASS\_MOV &&

INS\_IsMemoryRead(ins) &&

INS\_OperandIsReg(ins, 0) &&

INS\_OperandIsMemory(ins, 1))

{

icount++;

}

}

只有当下述条件满足的时候才会计数：命令是mov指令、是一条内存读指令、指令的第一个操作数是寄存器、指令的第二个操作数是内存。实际上，通过组合这些API就可以非常精确地筛选出想要插桩的指令了。

一些心得：

pin这个工具很好用，就是里面的代码编写的时候完全不知道都有哪些状态可以用来记录，都有哪些函数或者代码是对应的这个功能，具体更多的使用还需要阅读pintool的相关文档才能更加深入的学习。

也更加体会到了程序插桩和hook的重要性和趣味性，对于hook的执行方式或者说是他的功能具体是什么有了更加深刻的了解。