**《软件安全》实验报告**

姓名：禹相祐 学号：2312900 班级：计算机科学与技术

**实验名称：**

程序插桩及Hook实验

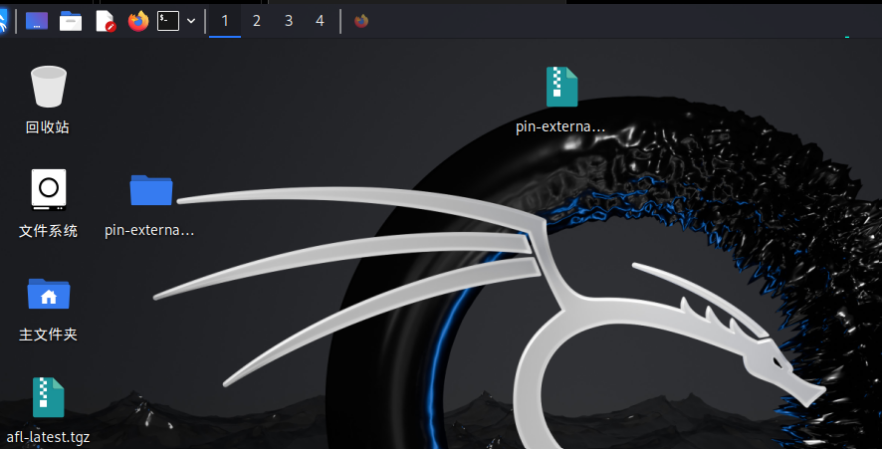
**实验要求：**

复现实验一，基于WindowsMyPinTool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：**

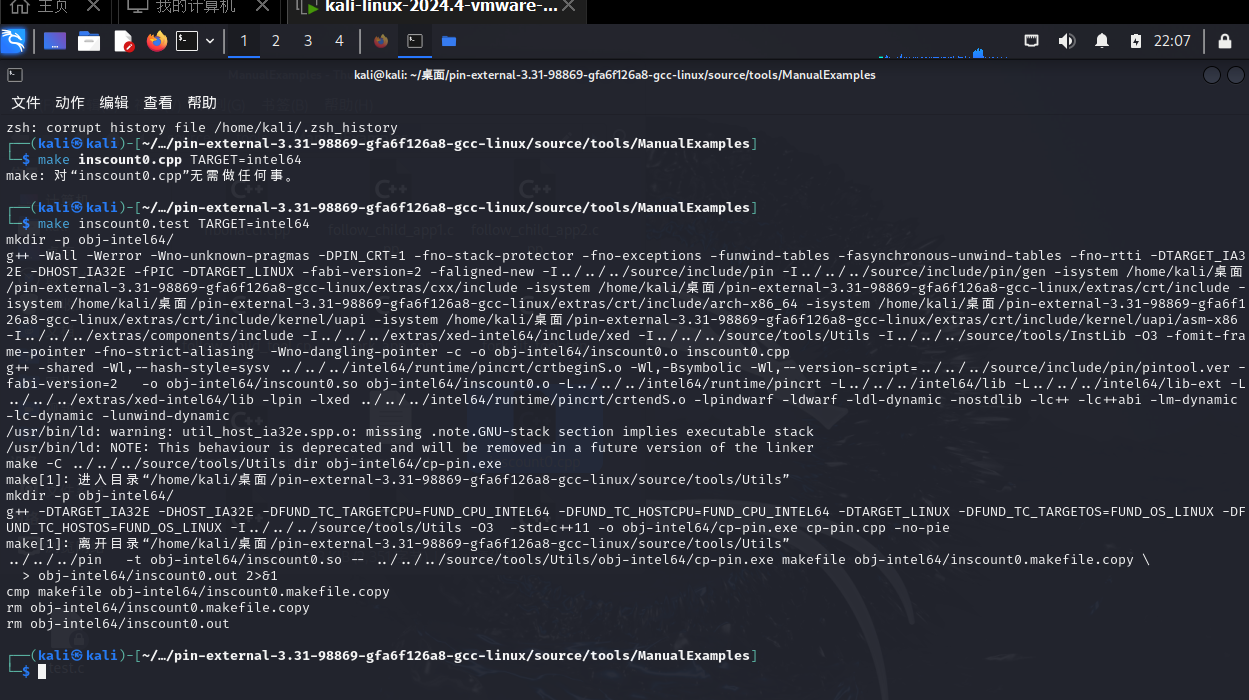
1. **在kali内安装WindowsMyPinTool工具：**

直接在官网上下载最新的linux版本的压缩包并拖进linux虚拟机内解压即可，如图：

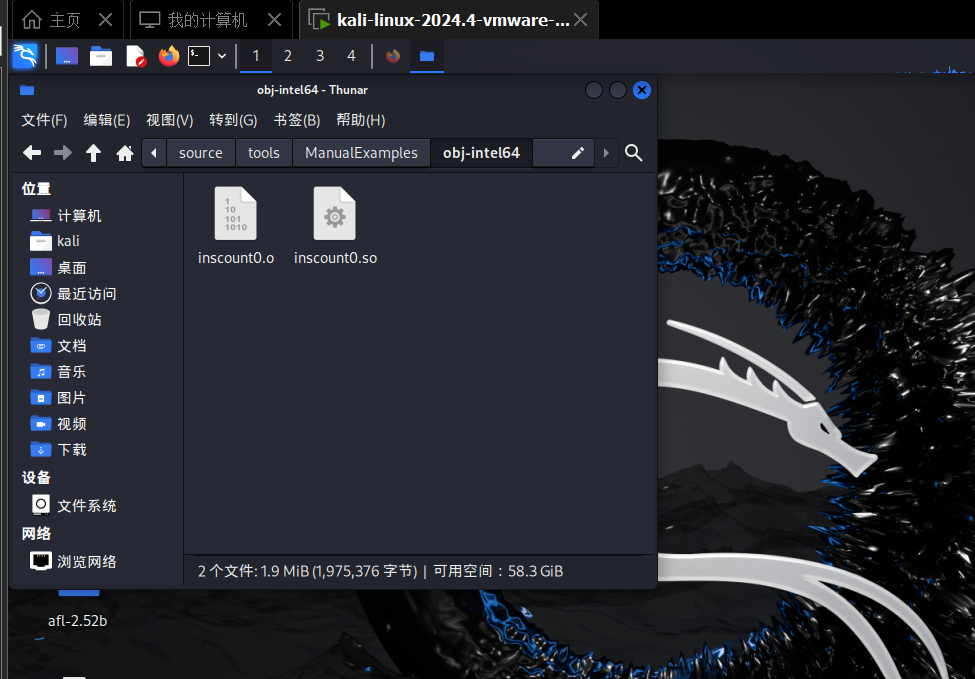


1. **编译inscount0.cpp：**

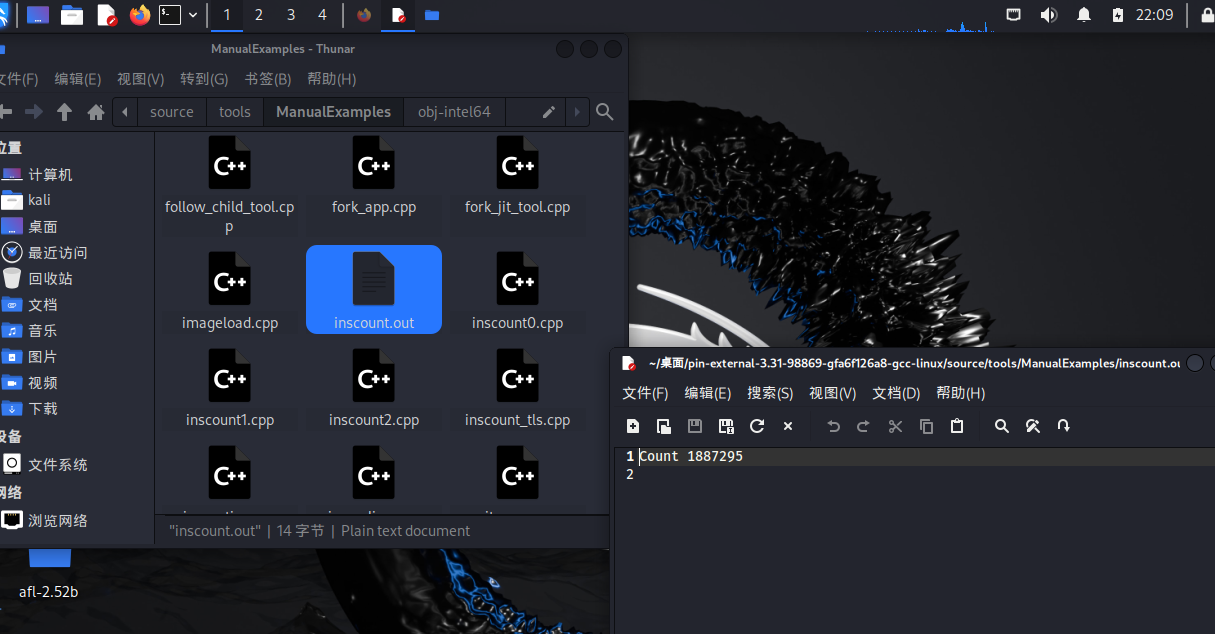
首先我们打开pin文件夹内的source文件夹，再打开tools文件夹中的ManualExamples，选择其中的inscount0.cpp，然后终端打开该文件并输入指令：make inscount0.test TARGET=intel64进行编译，如图：



编译完后我们可以看到：

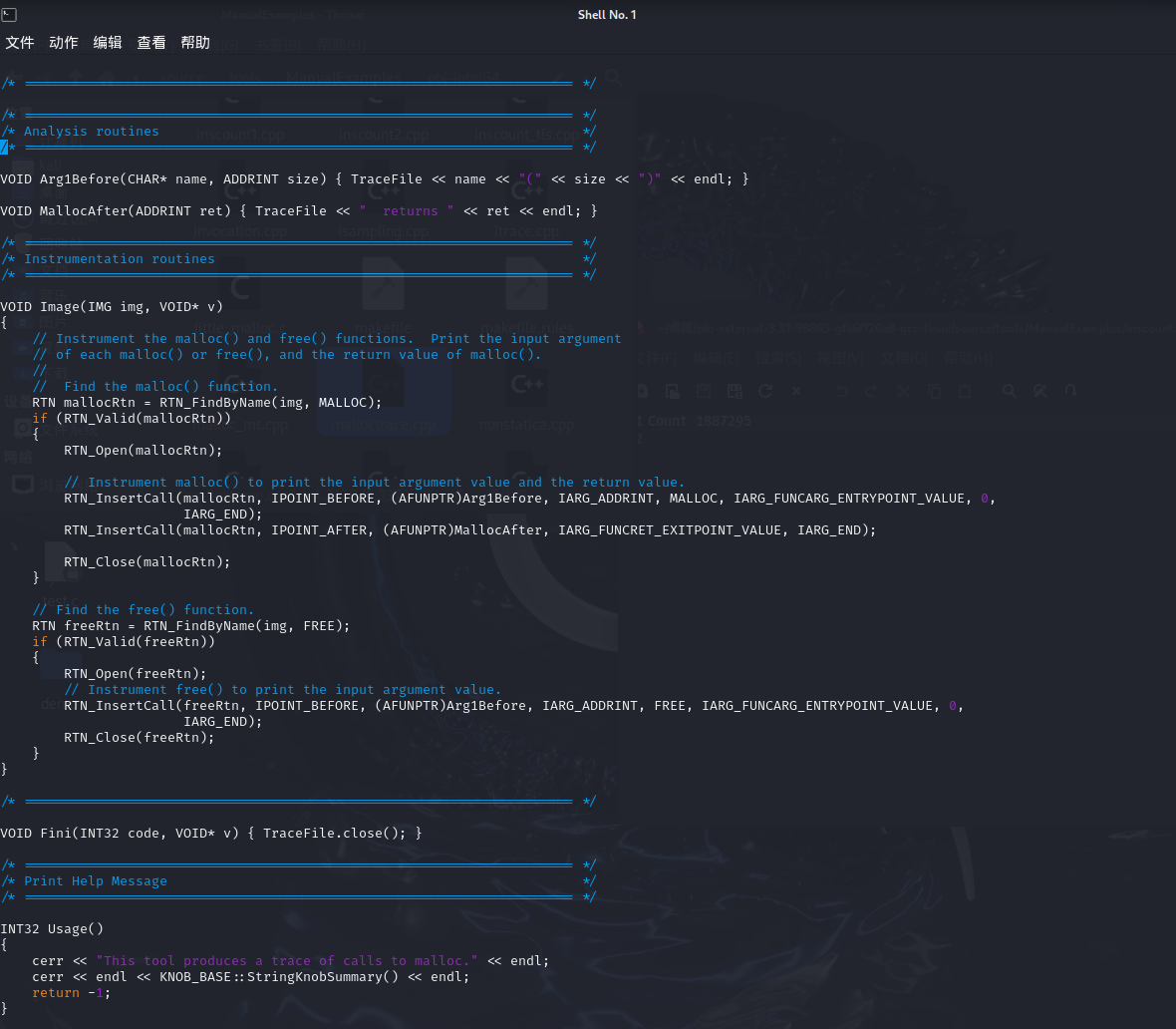


我们还可以打开文件inscount.out查看输出的count数量，此处为1887295，如图：

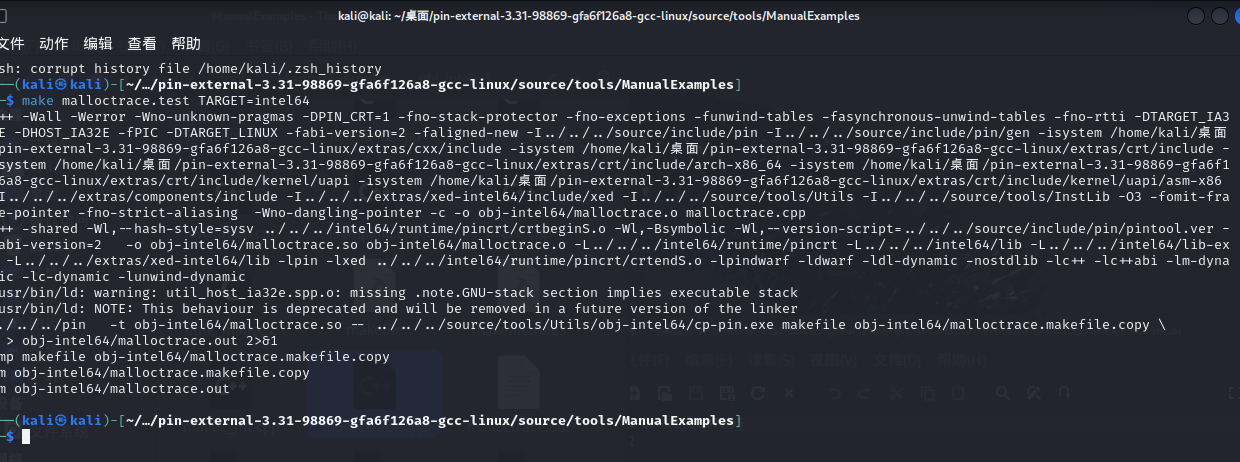


1. **编译malloctrace.cpp：**

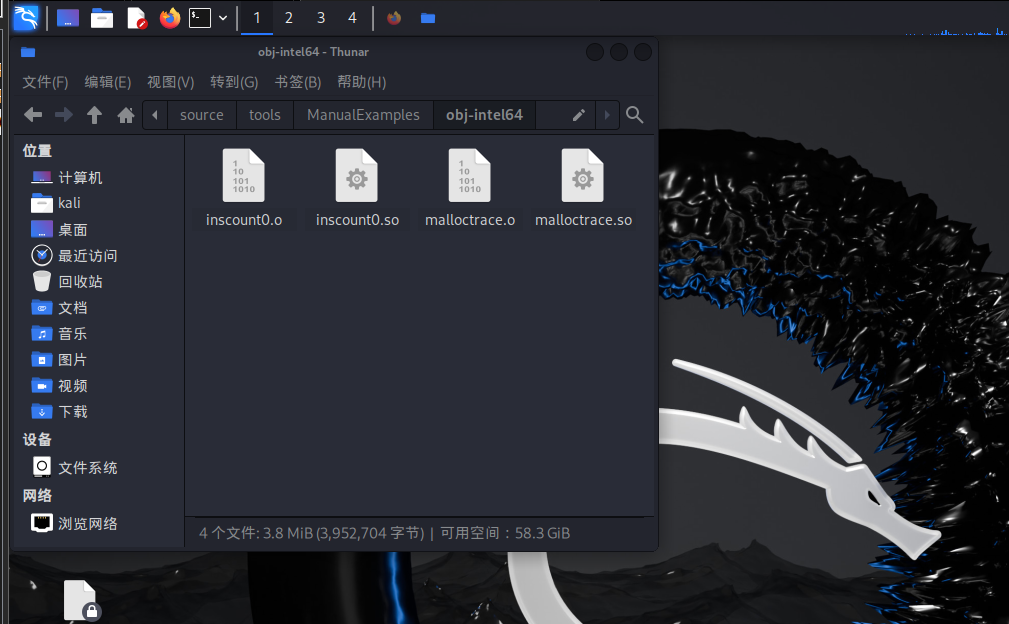
依然打开source/tools/ManualExamples,打开malloctrace.cpp查看源码：



依然新建一个终端输入命令：make malloctrace.test TARGET=intel64 进行编译，如图：

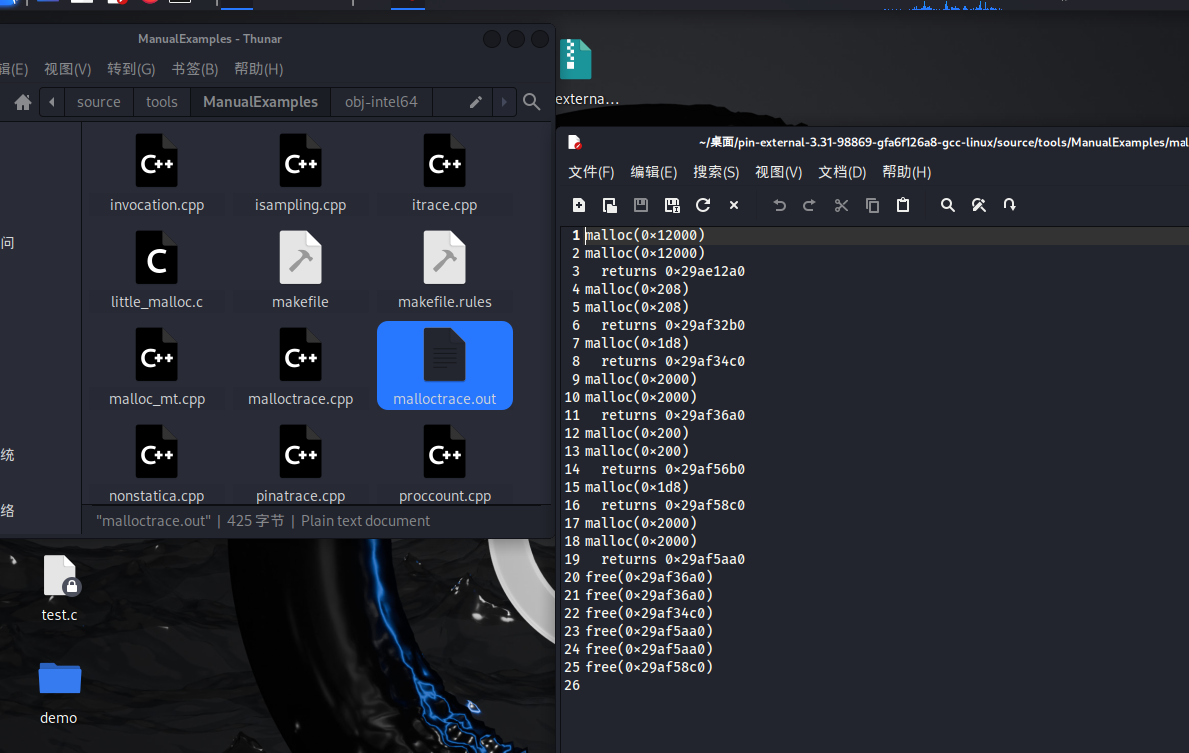


看见下图证明已经完成：



1. **进行插桩实验：**

我们直接打开生成的out文件进行查看，发现其输出了malloc和free函数的具体数值，如图：



1. **对输出结果的分析:**

从上述的输出结果可知：输出由malloc函数、free函数以及returns构成，每次使用malloc函数申请空间时，都会输出申请空间的大小以及起始的地址；每次调用free函数的时候，都会输出释放空间的初始地址。

1. **Pintool的基本分析：**

Malloc代码如下：

|  |
| --- |
| #include "pin.H"   #include <iostream>   #include <fstream>   using std::cerr;   using std::endl;   using std::hex;   using std::ios;   using std::string;   std::ofstream TraceFile;   KNOB< string > KnobOutputFile(KNOB\_MODE\_WRITEONCE, "pintool", "o",  "malloctrace.out", "specify trace file name");   VOID Arg1Before(CHAR**\*** name, ADDRINT size) { TraceFile << name << "(" << size <<  ")" << endl; }   VOID MallocAfter(ADDRINT ret) { TraceFile << "  returns " << ret << endl; }   VOID Image(IMG img, VOID**\*** v)   {      RTN mallocRtn **=** RTN\_FindByName(img, MALLOC);  **if** (RTN\_Valid(mallocRtn))      {          RTN\_Open(mallocRtn);          RTN\_InsertCall(mallocRtn, IPOINT\_BEFORE, (AFUNPTR)Arg1Before,  IARG\_ADDRINT, MALLOC, IARG\_FUNCARG\_ENTRYPOINT\_VALUE, 0,                         IARG\_END);          RTN\_InsertCall(mallocRtn, IPOINT\_AFTER, (AFUNPTR)MallocAfter,  IARG\_FUNCRET\_EXITPOINT\_VALUE, IARG\_END);          RTN\_Close(mallocRtn);      }      RTN freeRtn **=** RTN\_FindByName(img, FREE);  **if** (RTN\_Valid(freeRtn))      {          RTN\_Open(freeRtn);          RTN\_InsertCall(freeRtn, IPOINT\_BEFORE, (AFUNPTR)Arg1Before, IARG\_ADDRINT,  FREE, IARG\_FUNCARG\_ENTRYPOINT\_VALUE, 0,                         IARG\_END);          RTN\_Close(freeRtn);      }   }   VOID Fini(INT32 code, VOID**\*** v) { TraceFile.close(); }   INT32 Usage()   {      cerr << "This tool produces a trace of calls to malloc." << endl;      cerr << endl << KNOB\_BASE::StringKnobSummary() << endl;  **return** **-**1;   }   int main(int argc, char**\*** argv[])   {  **//** Initialize pin & symbol manager      PIN\_InitSymbols();  **if** (PIN\_Init(argc, argv))      {  **return** Usage();      }  **//** Write to a file since cout **and** cerr maybe closed by the application      TraceFile.open(KnobOutputFile.Value().c\_str());      TraceFile << hex;      TraceFile.setf(ios::showbase);  **//** Register Image to be called to instrument functions.      IMG\_AddInstrumentFunction(Image, 0);      PIN\_AddFiniFunction(Fini, 0);  **//** Never returns      PIN\_StartProgram();  **return** 0;   } |

由代码可知：其中包含两个分析函数:Arg1Before和MallocAfter。Arg1Before在调用函数malloc和free前执行，拿来记录所调用的函数名称以及其对应参数；MallocAfter则是在之后执行，拿来记录返回值。而Image函数则是拿来找到并插桩malloc和free函数对应的代码。而RTN\_InsertCall函数则会在找到malloc和free函数后前后插入分析函数Arg1Before和MallocAfter。Fini函数则负责关闭输出文件，Usage函数负责在程序报错的时候输出有误的命令行参数。

**心得体会：**

通过此次试验，我对Pin插桩工具有了更深的理解，简单总结如下：

1. 首先是编写需要进行插桩的程序，然后通过编译生成可执行文件；
2. 选择需要使用的插桩工具，并同样对其进行编译以生成动态链接库；
3. 将pintool插入要执行的程序，就成功实现了插桩，打开out文件也能查看插桩后的对应输出。