

软件安全实验报告

学号: 2310764 姓名: 王亦辉 班次: 计科一班

1 实验名称 :

程序插桩及 Hook实验

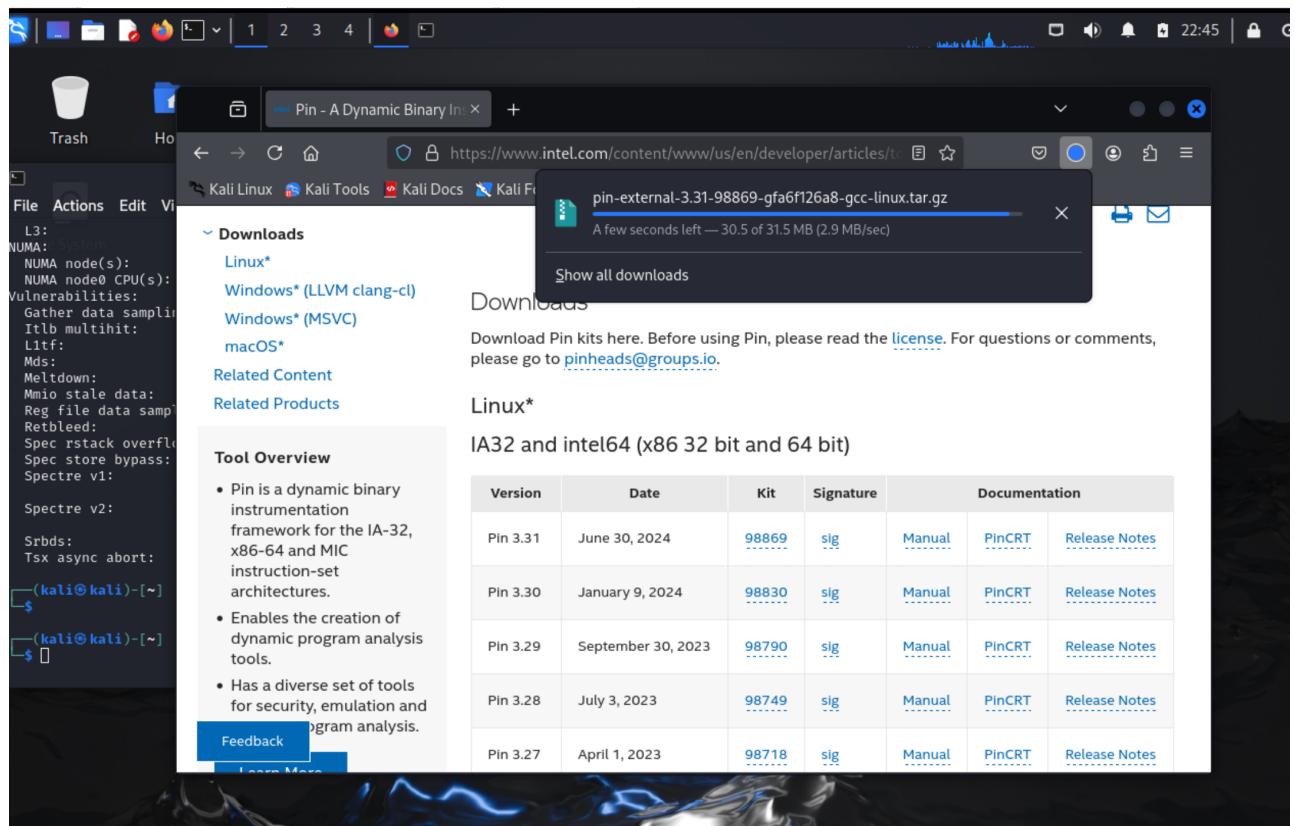
2 实验要求 :

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。

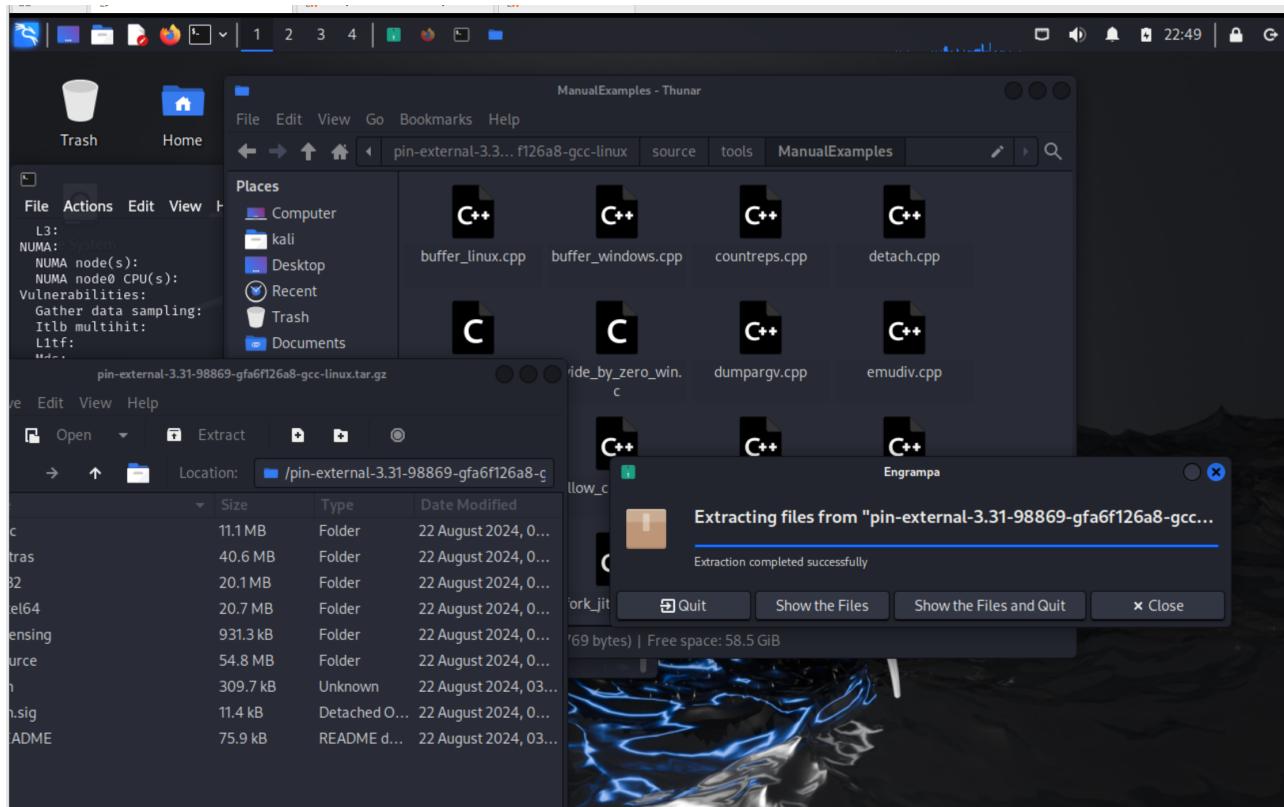
3 实验过程 :

3.1 pin 插桩工具的下载安装

由于上一次实验我们成功让虚拟机通过局域网连接上主机的网络，于是我们可以直接到下载pin tools



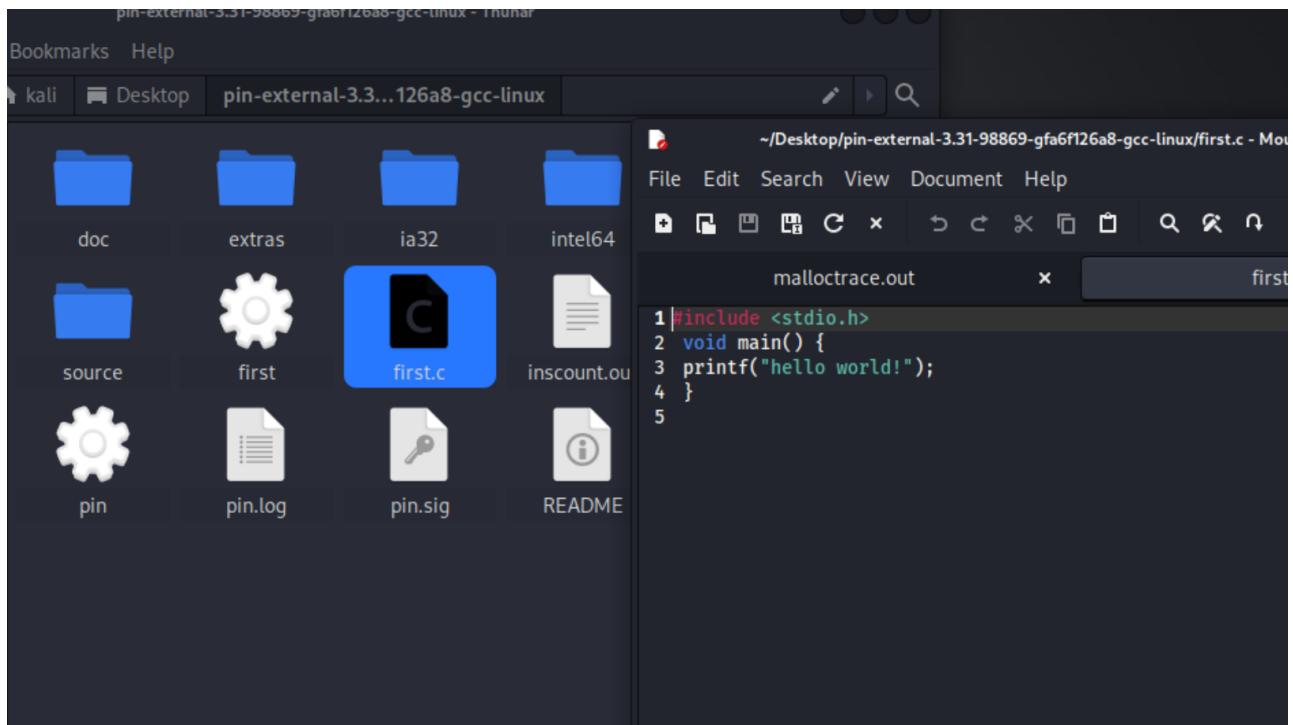
解压，可以看到 `cd source/tools/ManualExamples` 路径里有 pintools 相关 cpp 文件。



接下来的插桩都是使用这个简单的程序

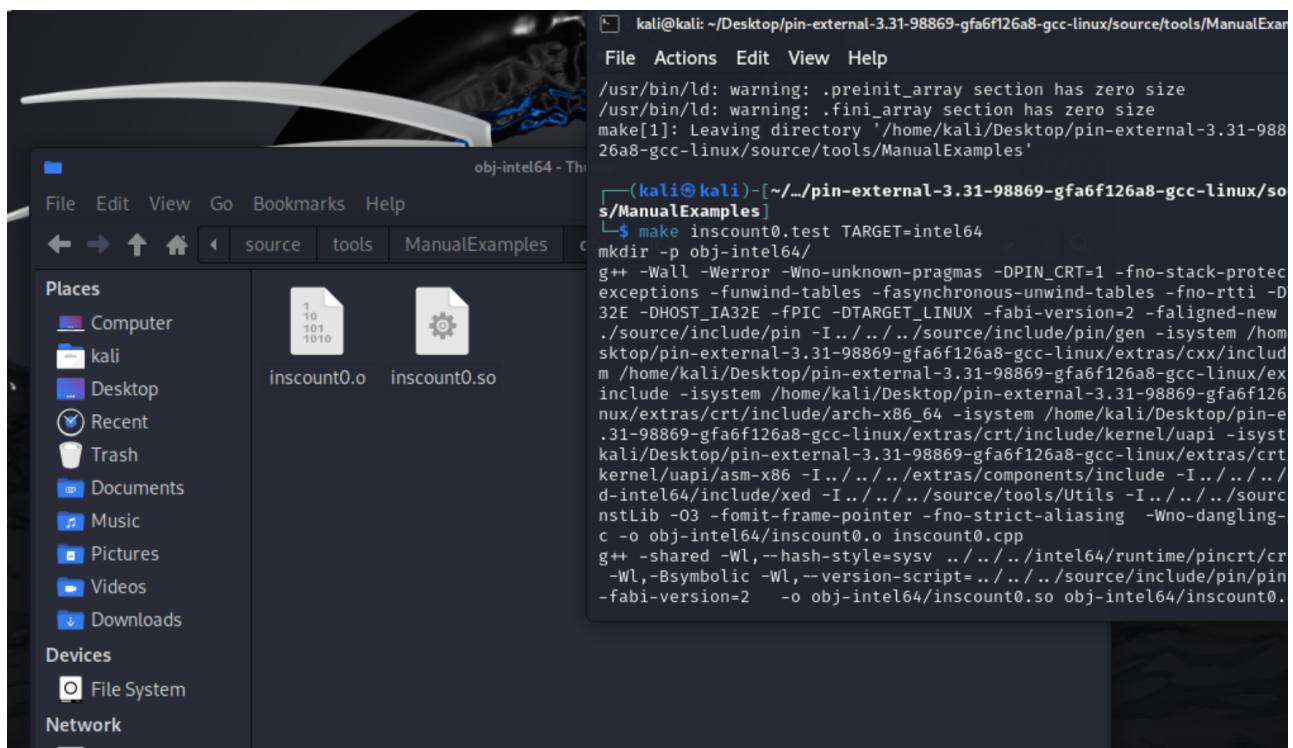
```
1 #include <stdio.h>
2 void main()
3 {
4     printf("hello world!");
5 }
```

将其编译生成可执行文件

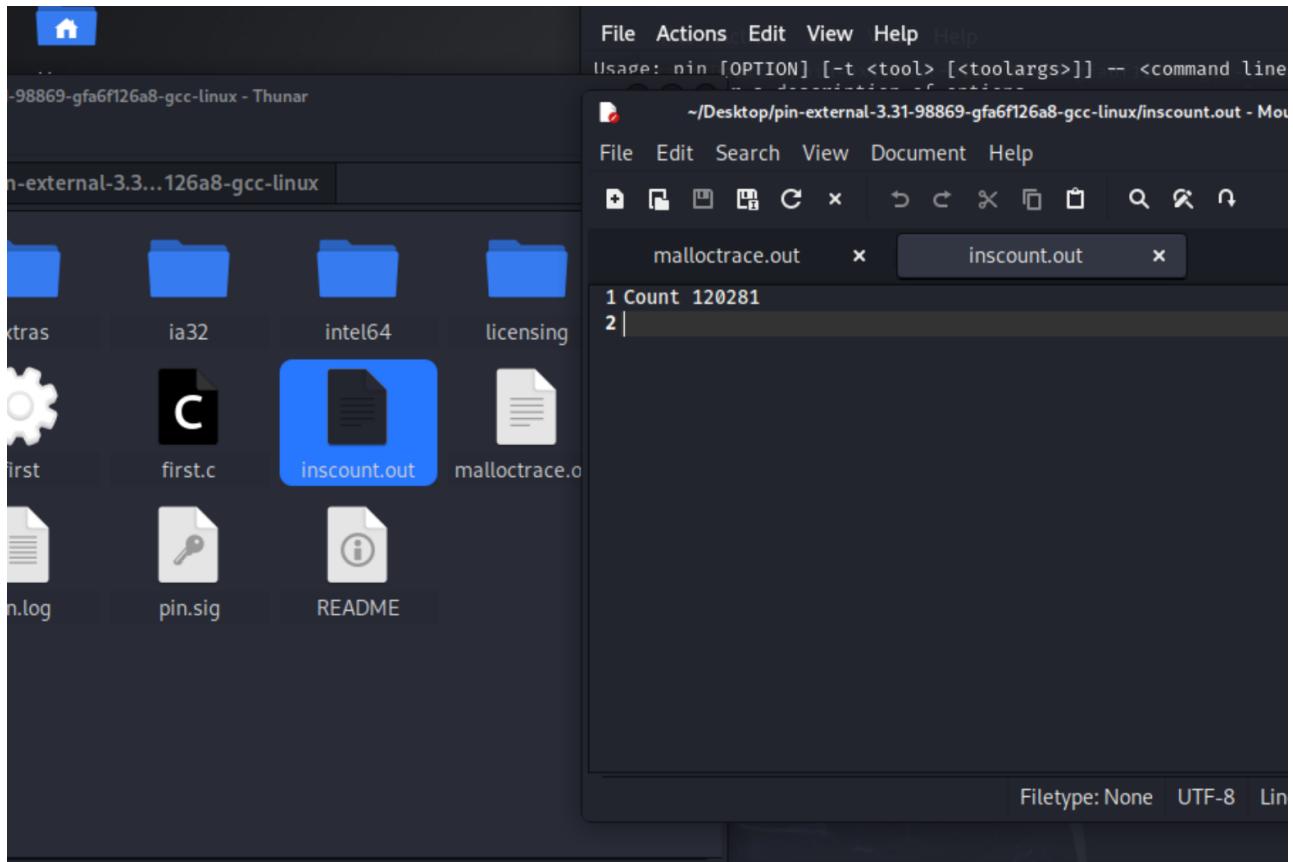


3.2 使用 `inscount0.cpp` 进行插桩测试

编译 `inscount0.test` 进行测试，在 `obj-intel64` 下生成了 `.so` 文件



用编译完的 pintool `inscount0.so` 进行插桩，可以发现路径下多了一个 `inscount0.out` 文件，打开之后显示我们 `first` 程序运行所用的指令数。



3.3 复现 `malloctrace.cpp` pintool 的使用

3.3.1 分析 `malloctrace.cpp` 中关键部分

```
1 VOID Image(IMG img, VOID* v)
2 {
3     // Instrument the malloc() and free() functions. Print the input
4     // argument of each malloc() or free(), and the return value of malloc().
5     //
6     // Find the malloc() function.
7     RTN mallocRtn = RTN_FindByName(img, MALLOC);
8     if (RTN_Valid(mallocRtn))
9     {
10         RTN_Open(mallocRtn);
11
12         // Instrument malloc() to print the input argument value and the
13         // return value.
14         RTN_InsertCall(mallocRtn, IPOINT_BEFORE, (AFUNPTR)Arg1Before,
15                     IARG_ADDRINT, MALLOC, IARG_FUNCARG_ENTRYPOINT_VALUE, 0,
```

```

14             IARG_END);
15         RTN_InsertCall(mallocRtn, IPOINT_AFTER, (AFUNPTR)MallocAfter,
16         IARG_FUNCRET_EXITPOINT_VALUE, IARG_END);
17     RTN_Close(mallocRtn);
18 }
19
20 // Find the free() function.
21 RTN freeRtn = RTN_FindByName(img, FREE);
22 if (RTN_Valid(freeRtn))
23 {
24     RTN_Open(freeRtn);
25     // Instrument free() to print the input argument value.
26     RTN_InsertCall(freeRtn, IPOINT_BEFORE, (AFUNPTR)Arg1Before,
27     IARG_ADDRINT, FREE, IARG_FUNCARG_ENTRYPOINT_VALUE, 0,
28             IARG_END);
29     RTN_Close(freeRtn);
30 }
31
32 int main(int argc, char* argv[])
33 {
34     .....
35     IMG_AddInstrumentFunction(Image, 0);
36     .....
37 }
```

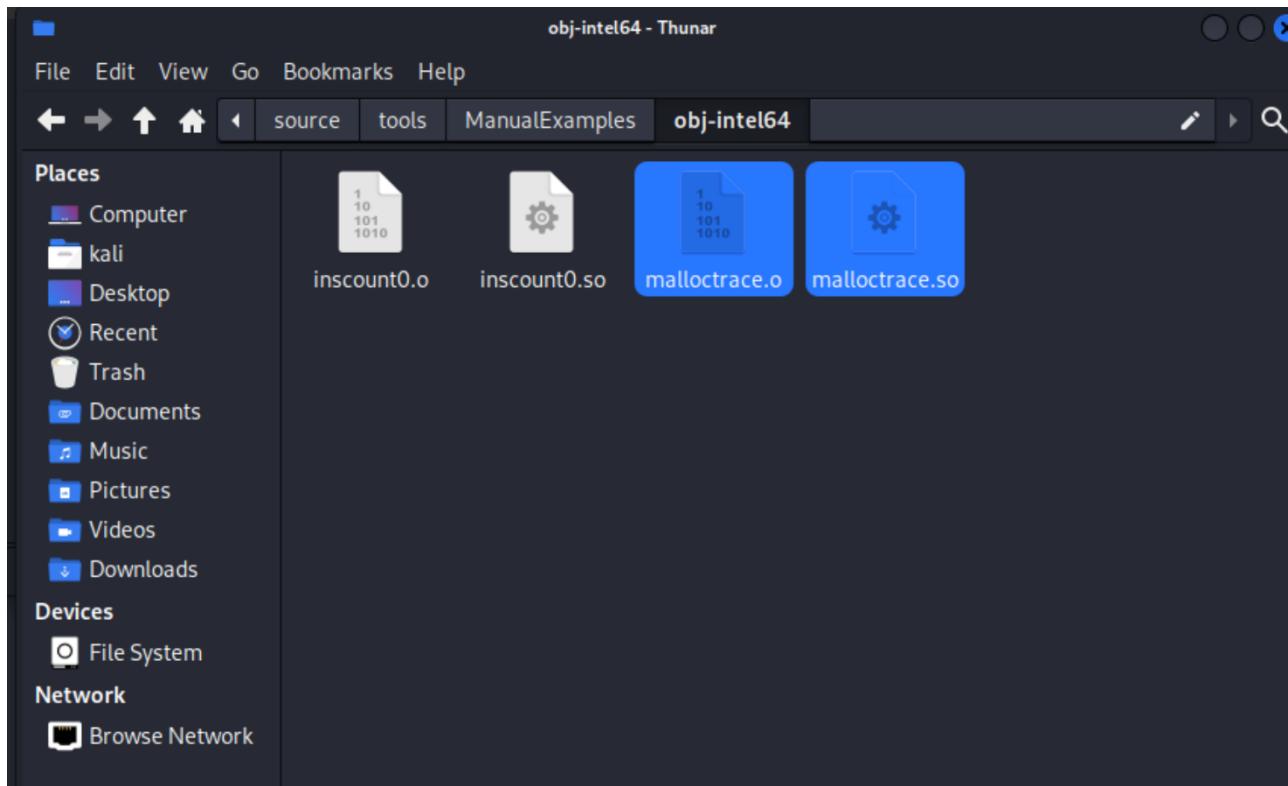
Image 函数实现了镜像插桩的功能，可以看到，`if (RTN_Valid(mallocRtn))` 中的两条 `RTN_InsertCall` 以及 `if (RTN_Valid(freeRtn))` 中的一条 `RTN_InsertCall` 分别对 malloc 的前后、free 之前插入了函数调用，实现了 malloc 和 free 的跟踪，最终在 `.out` 文件中打印出 malloc 和 free 的信息。

而 `IMG_AddInstrumentFunction (Image, 0);` 通过调用 pin 的 API，将前面写的 Image 进行二进制动态插桩。

--

3.3.2 使用 malloctrace 进行插桩

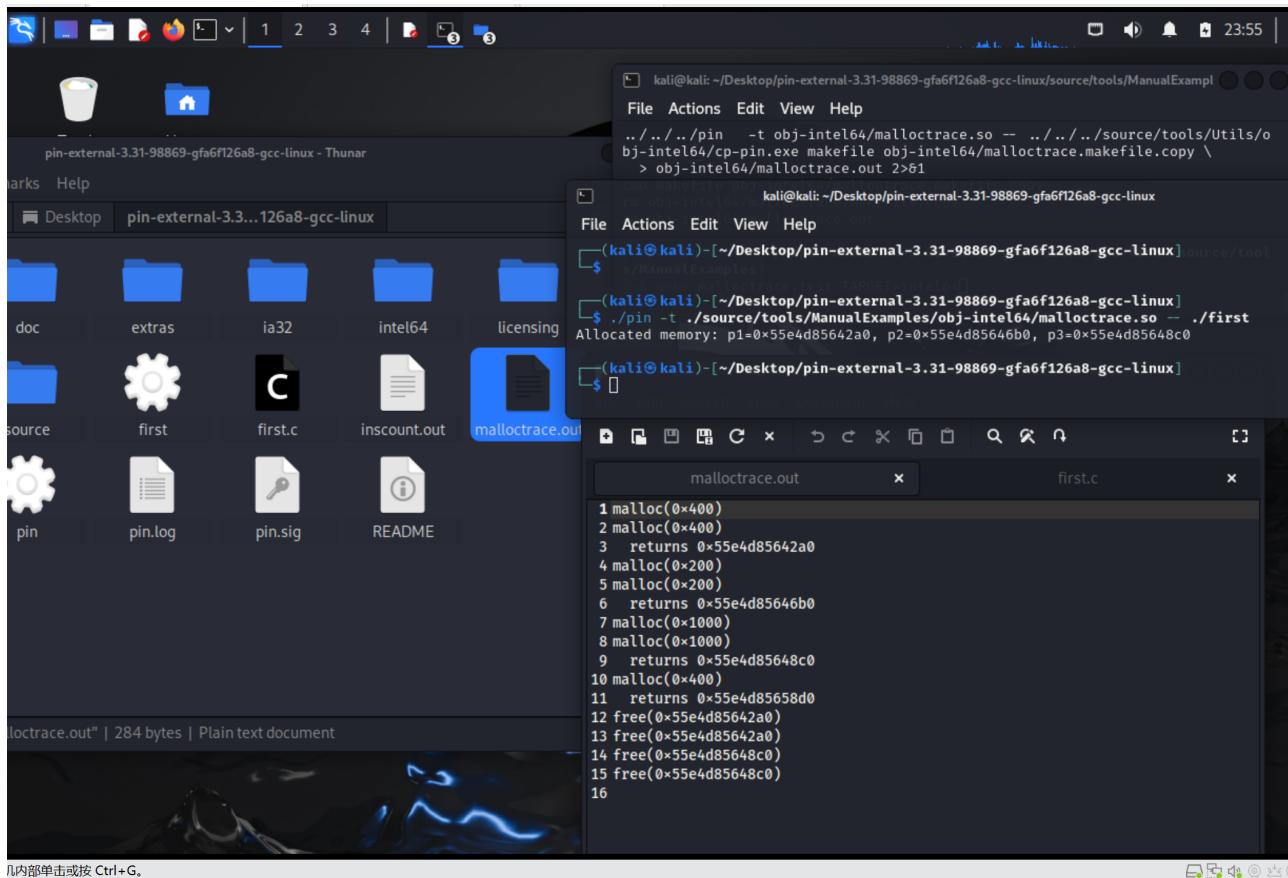
使用 `make malloctrace.test TARGET=intel64` 编译 malloctrace 这个 pintool，可以看到 `obj-intel64` 下多出了相应文件。



使用如下示例程序进行插桩

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     void *p1 = malloc(0x400); // 1024 字节 (1 KB)
6     void *p2 = malloc(0x200); // 512 字节
7     void *p3 = malloc(0x1000); // 4096 字节 (4 KB)
8
9     if (p1) *(char *)p1 = 'A';
10    if (p2) *(char *)p2 = 'B';
11
12    printf("Allocated memory: p1=%p, p2=%p, p3=%p\n", p1, p2, p3);
13
14    free(p1);
15    free(p3);
16
17    return 0;
18 }
```

使用 `./pin -t ./source/tools/ManualExamples/obj-intel64/malloctrace.so -- ./first` 命令进行插桩，结果如下



得到的 `malloctrace.out` 文件内容如下

```
1 malloc(0x400)
2 malloc(0x400)
3     returns 0x55e4d85642a0
4 malloc(0x200)
5 malloc(0x200)
6     returns 0x55e4d85646b0
7 malloc(0x1000)
8 malloc(0x1000)
9     returns 0x55e4d85648c0
10 malloc(0x400)
11     returns 0x55e4d85658d0
12 free(0x55e4d85642a0)
13 free(0x55e4d85642a0)
14 free(0x55e4d85648c0)
15 free(0x55e4d85648c0)
```

3.3.3 分析 malloc 和 free 函数的输入输出信息：

1. 程序调用了三次 malloc: 0x400 (1 KB)、0x200 (512 字节)、0x1000 (4 KB)，返回地址为 0x55e4d85642a0、0x55e4d85646b0、0x55e4d85648c0。额外的 malloc(0x400) (返回 0x55e4d85658d0) 由 printf 触发，可能是 libc 为 stdout 流分配的缓冲区。
2. 程序调用 free(p1) 和 free(p3)，释放了 0x55e4d85642a0 (0x400) 和 0x55e4d85648c0 (0x1000)。p2 (0x55e4d85646b0) 和 libc 的分配 (0x55e4d85658d0) 未释放，符合程序设计和 libc 行为。
3. 每个 malloc 和 free 被记录两次，可能是 malloctrace 的插桩逻辑 (Arg1Before 函数) 在函数调用前后重复记录。隐式 malloc(0x400) 只记录一次，可能因其调用路径不同。

4 心得体会：

1. 通过本次实验，我深入了解了 Intel Pin 工具的动态二进制插桩技术。我在 Kali Linux 环境下安装 Pin 并运行 inscount0 和 malloctrace Pintool，掌握了 Pin 工具的基本使用流程，包括下载、编译、插桩和输出分析。通过对课堂内容的回顾，对插桩这种事情的范畴更清楚了。
2. 本次实验还加深了我的命令行理解，对于工具使用的基本格式 [工具 相应参数](#) 更熟悉了，理解每一行在终端输入的命令是做什么用的。