Pin插件编写

其实Pin tool的编写还是相对比较容易的，因为有很多的例子程序可以参考，只要明白Pin的工作流程就可以了。

Pin tool由int main(int argc, char \* argv[])函数开始，由NMAKE编译选项编译成特定的动态链接库，如果要编译自己的动态链接库，在Nmakefile文件中把要编译的动态链接库名字加到COMMON\_TOOLS=后面，使用..\nmake.bat TARGET=ia32 xxx.dll命令进行编译。

如果在程序中要使用符号，要调用PIN\_InitSymbols();

初始化PIN\_Init(argc, argv)

PIN tool分指令级插桩（instruction instrumentatio），通过函数INS\_AddInstrumentFunctio实现。

轨迹级插装（trace instrumentation），通过函数TRACE\_AddInstrumentFunction实现。（貌似就是基本块插装）

镜像级插装（image instrumentation），使用IMG\_AddInstrumentFunction函数，由于其依赖于符号信息去确定函数边界，因此必须在调用PIN\_Init之前调用PIN\_InitSymbols。

函数级的插装（routine instrumentation），使用RTN\_AddInstrumentFunction函数。函数级插装比镜像级插装更有效，因为只有镜像中的一小部分函数被执行。

四种插装粒度。其中，IMG\_AddInstrumentFunction和RTN\_AddInstrumentFunction需要先调用PIN\_InitSymbols()，来分析出符号。在无符号的程序中，IMG\_AddInstrumentFunction和RTN\_AddInstrumentFunction无法分析出相应的需要插装的块。

在各种粒度的插装函数调用时，可以添加自己的处理函数在代码中，程序被加载后，在被插装的代码运行时，自己添加的函数会被调用。

INS\_AddInstrumentFunctio、TRACE\_AddInstrumentFunction、IMG\_AddInstrumentFunction、RTN\_AddInstrumentFunction指定的回调函数只有在相应的代码被分析到时才会被调用，即分析到一次只被调用一次，但程序运行过程中一般不再被调用，但INS\_InsertCall之类的程序添加的函数，是在相应的代码位置添加函数，根据程序运行的情况，会被多次调用。

在INS\_AddInstrumentFunctio指令级插装的代码中，只有在INS\_AddInstrumentFunctio指定的函数被调用时INS指令才有效，在INS\_InsertCall函数中，INS无效。

pin的使用

Pintools的使用：

一个简单的例子。使用inscount.so来对一个程序执行的指令数进行计数

Linux下：

进入ManulExample该目录 ../../../pin –t obj-ia32/inscount0.so –o inscount0.log -- /bin/ls 最后生成的log就在该目录里面

../../../pin –t ./obj-ia32/inscount0.so –o inscount0.log -- /bin/ls也可以

执行后查看结果 head xxx.out

**构建示例工具**

要在ia32体系结构的目录中构建所有示例：

$ cd source / tools / ManualExamples

$ make all TARGET=ia32

要在intel64体系结构的目录中构建所有示例：

$ cd source / tools / ManualExamples

$ make all TARGET = intel64

要构建和运行特定示例（例如，inscount0）：

$ cd source / tools / ManualExamples

$ make inscount0.test TARGET = intel64

要在不运行它的情况下构建特定示例（例如，inscount0）：

$ cd source / tools / ManualExamples

$ make obj-intel64 / inscount0.so TARGET = intel64

以上内容适用于英特尔（R）64架构。对于IA-32架构，请改用TARGET = ia32。

$ cd source / tools / ManualExamples

$ make obj-ia32 / inscount0.so TARGET = ia32

生成 直接 make obj-ia32/pinTracer.so

### 生成 PinTool

首先进入根目录下的 source/tools/MyPinTool，我们向其中放入我们的 PinTool 源代码，假定其名字为 myMallocTrace.cpp。

然后在该目录下输入 make obj-intel64/myMallocTrace.so，如果语法没有问题的话，我们就会在 source/tools/MyPinTool/obj-intel64/ 下找到 myMallocTrace.so 文件，这个文件就是我们的自定义 PinTool。

**使用 PinTool**

附加并检测应用程序

**pin –mt 0 –t pintool.so –pid 1234**

启动并检测应用程序

**pin –t pintool.so –- application**

绑定并插桩一个正在运行的程序

../../../pin -t obj-ia32/mypintools.so -pid 1234

(其中mypintools.so 是用户自己编写的插桩工具， 1234是正在运行的程序占用的线程号)

工具说明：

Pintool 编写

main 函数的编写

Pintool 的入口为 main 函数，通常需要完成下面的功能：

初始化 Pin 系统环境：

BOOL LEVEL\_PINCLIENT::PIN\_Init(INT32 argc, CHAR\*\* argv)

初始化符号表（如果需要调用程序符号信息，通常是指令粒度以上）：

VOID LEVEL\_PINCLIENT::PIN\_InitSymbols()

初始化同步变量：

Pin 提供了自己的锁和线程管理 API 给 Pintool 使用。当 Pintool 对多线程程序进行二进制检测，需要用到全局变量时，需要利用 Pin 提供的锁（Lock）机制，使得全局变量的访问互斥。编写时在全局变量中声明锁变量并在 main 函数中对锁进行初始化：VOID LEVEL\_BASE::InitLock(PIN\_LOCK \*lock)。在插桩函数和分析函数中，锁的使用方式如下，应注意在全局变量使用完毕后释放锁，避免死锁的发生：

GetLock(&thread\_lock, threadid);

// 访问全局变量

ReleaseLock(&thread\_lock);

注册不同粒度的回调函数：

TRACE（轨迹）粒度

TRACE 表示一个单入口、多出口的指令序列的数据结构。Pin 将 TRACE 分为若干基本块 BBL（Basic Block），一个 BLL 是一个单入口、单出口的指令序列。TRACE 在指令发生跳转时进行插入，进一步进行基本块分析，常用于记录程序执行序列。注册 TRACE 粒度插桩函数原型为：

TRACE\_AddInstrumentFunction(TRACE\_INSTRUMENT\_CALLBACK fun, VOID \*val)

IMG（镜像）粒度

IMG 表示整个被加载进内存的二进制可执行模块（如可执行文件、动态链接库等）类型的数据结构。每次被插桩进程在执行过程中加载了镜像类型文件时，就会被当做 IMG 类型处理。注册插桩 IMG 粒度加载和卸载的函数原型：

IMG\_AddInstrumentFunction(IMAGECALLBACK fun, VOID \*v)

IMG\_AddUnloadFunction(IMAGECALLBACK fun, VOID \*v)

RTN（例程）粒度

RTN 代表了由面向过程程序语言编译器产生的函数／例成／过程。Pin 使用符号表来查找例程，即需要插入的位置，需要调用内置的初始化表函数 PIN\_InitSymbols()。必须使用 PIN\_InitSymbols 使得符号表信息可用。插桩 RTN 粒度函数原型：

RTN\_AddInstrumentFunction(RTN\_INSTRUMENT\_CALLBACK fun, VOID \*val)

INS（指令）粒度

INS 代表一条指令对应的数据结构，INS 是最小的粒度。INS 的代码插桩是在指令执行前、后插入附加代码，会导致程序执行缓慢。插桩 INS 粒度函数原型：

INS\_AddInstrumentFunction(INS\_INSTRUMENT\_CALLBACK fun, VOID \*val)

注册结束回调函数

插桩程序运行结束时，可以调用结束函数来释放不再使用的资源，输出统计结果等。注册结束回调函数：

VOID PIN\_AddFiniFunction(FINI\_CALLBACK fun, VOID \*val)

启动 Pin 虚拟机进行插桩：

最后调用 VOID PIN\_StartProgram() 启动程序的运行。

插桩、分析函数的编写

在 main 函数中注册插桩回调函数后，Pin 虚拟机将在运行过程中对该种粒度的插桩函数对象选择性的进行插桩。所谓选择性，就是根据被插桩对象的性质和条件，选择性的提取或修改程序执行过程中的信息。

各种粒度的插桩函数：

INS

VOID LEVEL\_PINCLIENT::INS\_InsertCall(INS ins, IPOINT action, AFUNPTR funptr, ...)

RTN

VOID LEVEL\_PINCLIENT::RTN\_InsertCall(RTN rtn, IPOINT action, AFUNPTR funptr, ...)

TRACE

VOID LEVEL\_PINCLIENT::TRACE\_InsertCall(TRACE trace, IPOINT action, AFUNPTR funptr, ...)

BBL

VOID LEVEL\_PINCLIENT::BBL\_InsertCall(BBL bbl, IPOINT action, AFUNPTR funptr, ...)

其中 funptr 为用户自定义的分析函数，函数参数与 ... 参数列表传入的参数个数相同，参数列表以 IARG\_END 标记结束。