计算机体系结构（实验）

2021年春

HITSZ 实验与创新实践教育中心
Education Center of Experiments and Innovations, HITSZ

实验概况

- **实验项目**

	实验项目名称	学时
实验一	Pin入门及指令依赖距离分析	4
实验二	分支预测器设计	6
实验三	存储层次分析及程序优化	6

- **实验成绩**（共**40**分）
  - 每个实验均按百分制：现场验收30%，代码+报告70%
  - 实验总分 = (3个实验成绩之和 / 3) * 40%
  - **附加题**：在实验总分的基础上加分，**满40分为止**

计算机体系结构
实验一 PIN入门及指令依赖距离分析

实验目的

- 了解评估CPU特性和程序特性的基本方法
- 了解程序插桩分析的基本原理，掌握Pin插桩工具的编写和使用方法
- 了解指令依赖距离的概念，掌握使用指令依赖距离分布图分析程序特性的方法

实验内容

1. 跑Demo，读源码

   目录：<pin-root>/source/tools/ManualExamples

   Demo：inscount0、inscount1、 inscount2，etc.

2. 编写指令依赖距离分析的插桩工具
3. 思考并回答问题

1. 继续完成董老师发布的lab0
2. 按**网页版指导书**的要求，撰写报告
3. 将网页版指导书的实验1跑一遍，**熟悉流程** (不计分)

实验步骤

- 网页版指导书：
  - https://hitsz-cslab.gitee.io/arch/
- 实验包及PPT下载：见指导书首页
- 虚拟机下载：
  - http://10.249.12.124/arch/
  - http://10.xxx.xx.xxx/arch/ (临时)

实验原理

- **1、Why Pin?**
  - 为了设计高性能的CPU/程序，必须充分了解其特性
  - 评估方法
    - 软件模拟　建立CPU模型，在模型上运行测试程序
      - 优点：信息更丰富
      - 缺点：慢
    - 程序插桩　在测试程序中注入代码，在物理CPU运行
      - 优点：快，容易实现
      - 缺点：与模拟相比，获得的信息没那么丰富

实验原理

- **1、Why Pin?**
  - 程序插桩
    - 源码插桩　直接修改测试程序的源码
      - 优点：直观
      - 缺点：需重新编译、效率低
    - 二进制插桩　直接在二进制可执行文件中注入代码
      - 优点：不需重新编译，效率高
  - Pin：**动态二进制**插桩工具包

    动　态——运行时注入，不生成额外的文件

    二进制——直接对可执行文件进行插桩，不需重新编译

实验原理

- 2、插桩原理
  - 基本原理：将额外的代码插入到目标程序，以收集目标程序的信息。
  - 插桩工具 = 插桩代码 + 分析代码

决定在哪里插入哪些代码　　被插入的代码

  - inscount0：

```
// This function is called before every instruction is executed
VOID docount() { icount++; }

// Pin calls this function every time a new instruction is encountered
VOID Instruction(INS ins, VOID *v)
{
    // Insert a call to docount before every instruction, no arguments are passed
    INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)docount, IARG_END);
}
```

实验原理

- **2、插桩原理**
  - 插桩工具也需考虑效率
  - Pin插桩粒度：
    - 指令级插桩 ——以指令为插桩的基本单位，最细粒度
    - 轨迹级插桩 ——以基本块BBL为插桩的基本单位
    - 函数级插桩 ——镜像加载时，对整个函数进行遍历和插桩
    - 镜像级插桩 ——镜像加载时，对整个镜像进行遍历和插桩

实验原理

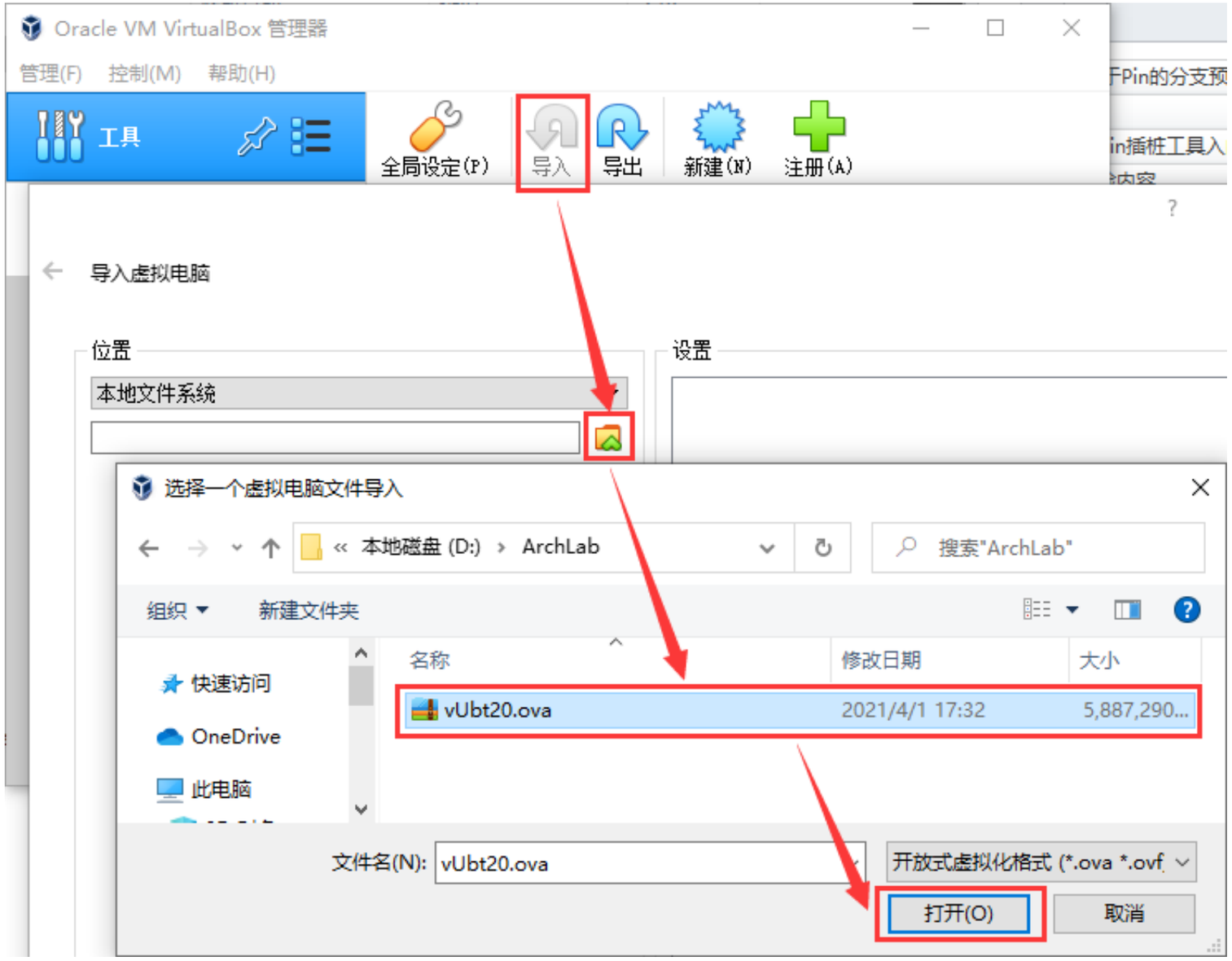
- **2、插桩原理**
  - 插桩工具开销
    - 插桩代码开销
    - 分析代码开销 (**执行次数**×**执行开销**)
  - 插桩代码：只在可执行文件的指令被**首次**执行时执行
  - 分析代码：**每次**执行指令时都会执行
  - 原则：尽可能减少分析代码**执行次数**、减小分析代码**执行开销**
    - 使用大粒度的插桩
    - 简化分析代码
  - 优化方法：分析代码任务转移、控制流消除、编译器优化，etc.

实验原理

- **3、指令依赖距离**
  - **定义**：一个寄存器被某条指令写入，且被后续指令读取时，这两条指令之间的指令数

实验步骤

- 1. 导入虚拟机


Oracle VM VirtualBox 管理器
管理(F)
控制(M)
帮助(H)
工具
全局设定(P)
导入
导出
新建(N)
注册(A)
导入虚拟电脑
位置
本地文件系统
设置
选择一个虚拟电脑文件导入
本地磁盘 (D:)
ArchLab
搜索"ArchLab"
组织
新建文件夹
名称
修改日期
大小
快速访问
OneDrive
此电脑
vUbt20.ova
2021/4/1 17:32
5,887,290...
文件名(N):
vUbt20.ova
开放式虚拟化格式 (*.ova *.ovf
打开(O)
取消

实验步骤

- 2. 运行Pin
  - (1) 进入范例程序的目录：source/tools/ManualExamples

```
arch@arch:~/pin-3.18$ cd source/tools/ManualExamples/ && ls
buffer_linux.cpp         imageload.cpp      obj-intel64
buffer_windows.cpp       inscount0.cpp      pinatrace.cpp
countreps.cpp            inscount1.cpp      proccount.cpp
detach.cpp               inscount2.cpp      replacesigprobed.cpp
divide_by_zero_unix.c    inscount_tls.cpp   safecopy.cpp
```

  - (2) 使用pin命令运行插桩工具

```
pin -t <toolname> [-o <output_file>] -- <target_executable>
```

实验步骤

- 3. 编写insDependDist插桩工具
  - (1) 参考范例程序，补全insDependDist.cpp中的代码
  - (2) 在makefile.rules中添加新的插桩工具
  - (3) 编译并运行

附加题

- **题目：插桩效率优化（+2分）**
  - insDependDist工具基于指令级插桩粒度开发，效率较低
  - 内容：
    - 参考inscount1和inscount2源码，修改insDependDist代码，使其能够基于轨迹级粒度进行插桩，以提高效率
  - 要求：
    - 比较优化前后的插桩效率
    - 将设计思路、关键代码实现、测试方法及分析写入实验报告

实验检查与提交

- 检查指令依赖距离分布图
- 将**源码**、**实验报告**打包提交
  - 命名规则：**学号_姓名_ARCH实验1.zip**
  - 提交方法：https://hitsz-cslab.gitee.io/arch/ojguide
  - Deadline：下周同一上课时间前
- 附加题：将设计思路、关键代码等写入报告，与源码一起打包提交（**+2**分）

开始实验