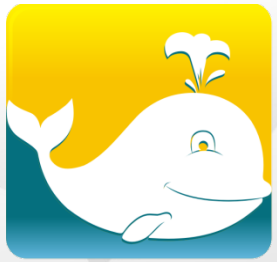


操作系统
Operating System

sv6 for RV64 SMP
课程设计最终报告

计53 谭闻德 尹宇峰

报告流程

- 选题概述
- sv6移植
- Commuter分析与理解
- 基于Commuter对POSIX socket API建模
- 实验收获

选题概述

- 目前，计算机的系统大多是多核的系统，而多核优化比较重要，所以我们的课程设计想研究多核优化问题。

sv6移植

编译

- 修复x86-64下的sv6
- 将汇编代码替换为死循环
- 删除分段相关代码
- 删除驱动程序
- 将其他无法编译的代码变为注释

链接生成可启动代码

- 运行时库问题
- Code Model问题
- 原子操作问题

在单核环境运行

- 页表填写
- trapframe以及context结构体
- 中断和系统调用的处理
- 特权指令
- 其他

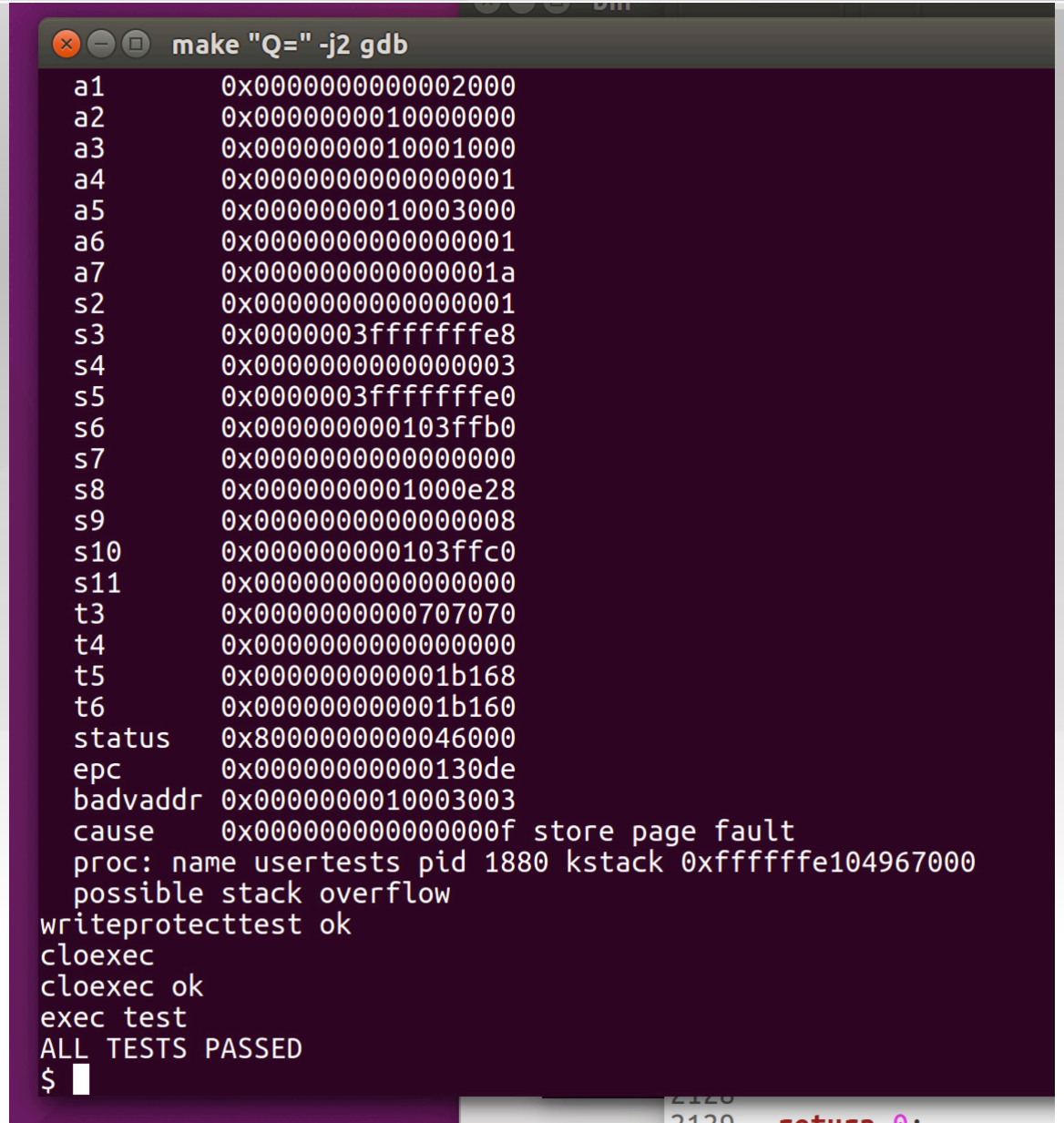
在多核环境运行

- 调用bbl提供的IPI和TLB shootdown
- 对于每一个处理器核心，需要保存自己的数据结构，考虑到sv6本身是多核操作系统，并且我们在单核移植的时候已经考虑到这一点，开启多核就不需要特殊处理了。

在真实硬件运行

- 第一，真实硬件启动时内存的内容是随机的，而模拟器一般会将内存空间全部清零。
  - 为此，需要在内核启动时清零bss段等内存区域。
- 第二，真实硬件有缓存以及TLB，模拟器一般不会特意模拟这些特性。
  - 为此，内核修改共享内存或者页表后，需要使用相应指令刷新缓存或TLB。
- 此外，有和具体开发板相关的问题。

实验结果


make "Q=" -j2 gdb
a1 0x0000000000002000
a2 0x0000000010000000
a3 0x0000000010001000
a4 0x0000000000000001
a5 0x0000000010003000
a6 0x0000000000000001
a7 0x000000000000001a
s2 0x0000000000000001
s3 0x0000003fffffffe8
s4 0x0000000000000003
s5 0x0000003fffffffe0
s6 0x000000000103ffb0
s7 0x0000000000000000
s8 0x0000000001000e28
s9 0x0000000000000008
s10 0x000000000103ffc0
s11 0x0000000000000000
t3 0x0000000000707070
t4 0x0000000000000000
t5 0x000000000001b168
t6 0x000000000001b160
status 0x8000000000046000
epc 0x00000000000130de
badvaddr 0x0000000010003003
cause 0x000000000000000f store page fault
proc: name usertests pid 1880 kstack 0xffffffe104967000
possible stack overflow
writeprotecttest ok
cloexec
cloexec ok
exec test
ALL TESTS PASSED
$

实验结果

NumericalAnalysis — minicom -b 115200 -D /dev/tty.usbserial-142B — minicom — minicom -b 115200 -D /dev...

```
  a3       0x0000000010001000
  a4       0x0000000000000001
  a5       0x0000000010003000
  a6       0x0000000000000001
  a7       0x000000000000001a
  s2       0x0000000000000001
  s3       0x0000003fffffffe8
  s4       0x0000000000000003
  s5       0x0000003fffffffe0
  s6       0x000000000103ffb0
  s7       0x0000000000000000
  s8       0x0000000001000e28
  s9       0x0000000000000008
  s10      0x000000000103ffc0
  s11      0x0000000000000000
  t3       0x0000000000707070
  t4       0x0000000000000000
  t5       0x000000000001b168
  t6       0x000000000001b160
  status   0x8000000200046000
  epc      0x00000000000130de
  badvaddr 0x0000000010003003
  cause    0x000000000000000f store page fault
  proc: name usertests pid 1880 kstack 0xffffffe104967000
  possible stack overflow
writeprotecttest ok
cloexec
cloexec ok
exec test
ALL TESTS PASSED
$ $ $
Meta-Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | tty.usbserial-142B
```

Commuter分析与理解

Commuter简述

- 传统的操作系统判断并行性的方式有着明显的弊端：
- 1. 我们不清楚最重要的并行性的瓶颈在哪里。
- 2. 在操作系统开发阶段的后期，设计层面的修改（例如修改系统调用的接口）显得不切实际。
- 我们是否可以仅仅通过系统调用的行为描述来判断系统调用与系统调用之间的并行性？

Commuter简述

- 如果对于一组系统调用，他们之间的任意调用顺序不会影响最后的系统状态和返回结果（即SIM条件），那么这些系统调用便可以被并行执行。

Commuter简述

- Analyzer
- TestGen
- MTrace

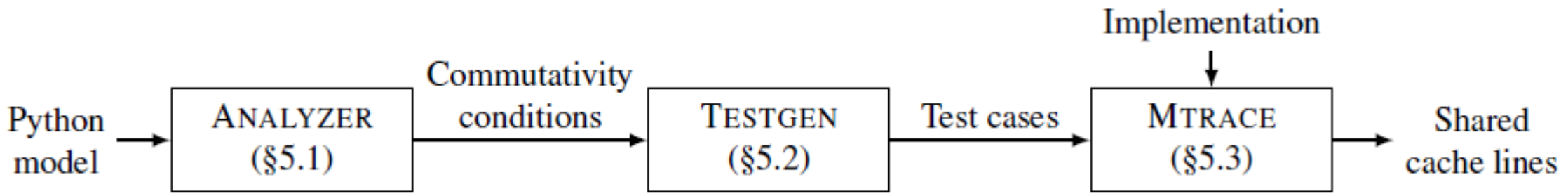


Figure 3: The components of COMMUTER.

Analyzer

- Analyzer的输入是符号化的接口建立的模型
- 通过符号化执行来分析系统调用序列是否满足SIM条件
- 判断方式是通过枚举全部调用序列，观察系统状态和返回结果是否一致
- 返回的是满足SIM条件的path condition，即Commutativity conditions

TestGen分析和理解的难点

- 限于篇幅，许多概念和细节，作者并没有真正阐释清楚
- 杂乱的代码注释，复杂的函数调用关系，以及python本身欠缺类型声明
- 我们是唯一一组对Commuter工具进行分析和理解的小组，没有其他小组可以给予帮助

TestGen

- TestGen的任务是记录下来Analyzer输出的Commutativity conditions，并将其中的符号化变量实例化，然后生成实际的测试代码。
- TestGen生成测试代码的目的就是寻找的潜在的系统调用实现的并行化问题。
- 因此，我们需要构造一种合适的覆盖模式，使得我们的测试代码可以尽可能地找出可能的并行化问题。

conflict coverage

- 两个系统调用访问同一个数组，下标相同或不同对应的路径一致，但是前者会引发访存冲突。
- 一个path condition对应一组参数赋值是不够的。

Assignment 同构

表达式	x=1, y=1	x=2, y=2	x=1, y=3
x - y	0	0	-2
x + y	2	4	4
2x - y	1	2	-1
2y - x	1	2	5

conflict coverage

- 针对一组参数赋值和给定的表达式集合，我们可以求解与这组参数赋值同构的条件，然后将该条件取非再与path condition取交，更新为新的path condition，然后求解得到一组新的参数赋值，那么这组的赋值就一定与原先的参数赋值不同构。

一个简单的例子

```
Assignments 1:
defaultdict(<type 'list'>,
  {
  <testgen.Interpreter object at 0x7f71fb8b4810>:
  [(Fs.proc0.fd_map._map.inum[a.close.fd], SInum!val!0),
  (Fs.proc0.fd_map._map.inum[b.close.fd], SInum!val!0)],
  }
)

Assignments 2:
defaultdict(<type 'list'>,
  {
  <testgen.Interpreter object at 0x7f4e0c8fc8d0>:
  [(Fs.proc0.fd_map._map.inum[a.close.fd], SInum!val!1),
  (Fs.proc0.fd_map._map.inum[b.close.fd], SInum!val!0)]
  }
)
```

MTrace

- MTrace的功能是运行TestGen的测试代码，MTrace会查看每个测试代码是否让内核有访问冲突。通过MTrace，开发者可以利用这些测试代码来寻找并修正代码中的并行性问题。

基于Commuter对POSIX socket API建模

socket总体设计

- 考虑到TCP的协议状态机太过于复杂，我们选择了对无连接不可靠的UDP进行建模，主要建模的系统调用有9个，分别是：socket，read，write，connect，bind，listen，accept，shutdown以及close。

socket总体设计

- 目前的模型描述了从传输层到应用层的socket的行为，未来还可以用此模型对实际操作系统的网络子模块进行并行性测试与优化。

实验收获

- 熟悉了RV64指令集及其工具链
- 进一步熟悉了x86-64指令集
- 了解了sv6的设计与实现
- 了解了C++运行时环境的设计
- 了解了RV64的代码模型
- 了解了Berkeley Boot Loader的设计与实现
- 熟悉并经历了一个多核操作系统的移植过程

实验收获

- 学习了符号化执行方法
- 学习了Z3 SMT求解器的使用方法
- 深入理解了Commuter各个子模块的设计与实现
- 较为深入地了解了POSIX socket API
- 简单了解了QEMU模拟器的设计与实现

操作系统
Operating System

Thanks!

Q & A