sv6 for RV64 SMP - 课程设计期末报告

计53 谭闻德 尹宇峰

选题概述

近年来，人工智能及大数据的众多应用为计算能力提出了新的需求，而处理器核心频率的提高已经遇到困难，现如今，多核心处理器、并行化已成为提高计算系统性能的主要手段。事实上，在2013年，Austin T. Clements等人[1, 2]就提出过一种新的手段，用于优化并行系统软件的性能。此项工作提出了一个名为commuter的工具，借助符号化执行技术[10, 11]以及SMT求解器[12]，用于挖掘系统规范中潜在的并行化优化点；同时，他们用此工具辅助设计实现了一个并行度很好的精简的操作系统，名为sv6。sv6在x86-64指令集架构上实现，考虑到目前新兴的RISC-V指令集架构[3, 4]更开放、灵活性更好、硬件设计更简洁，为此，本课程设计基于上述工作，将上述工作提出的sv6移植到RISC-V 64位多核平台，以期能够推动RISC-V并行架构的发展。此外，我们深入理解了commuter的设计与实现并对其进行总结。最后，本课程设计还基于commuter对unix socket API进行了建模，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。

由此，本课程设计的主体工作分成以下三个部分：

1. **移植** 将sv6操作系统从x86-64移植到RISC-V 64位（以下简称RV64）对称多处理器（以下简称SMP）环境。
2. **分析** 详细阅读[2]的文献和代码，分析和理解commuter工具的设计与实现并对其进行总结，方便后人阅读理解。
3. **建模** 基于commuter工具对unix socket API（从传输层到应用层）进行建模。

相关工作

这一小节简要介绍本课程设计会用到的前人的一些已有工作。

RISC-V

RISC-V [3, 4]是一个新的指令集架构，它更开放、灵活性更好、硬件设计更简洁。sv6操作系统[1]则是一款基于xv6，并行度很好的精简的类POSIX的研究性质的操作系统。

符号化执行与z3求解器

通过符号化执行的方法[10, 11]，可以将一个函数的输入参数指定为符号值，函数的每一个路径可以产生一个先决条件逻辑表达式，通过Z3等SMT求解器[12]进行求解，就可以知道这个路径是否可达，是否会产生特定的情况（大多数情况是断言失败）。

commuter

Austin T. Clements等人在2013年提出的Commuter [2]能够利用上面提到的符号化执行技术以及Z3求解器，完成对系统调用接口规范模型的分析，为设计良好并行化的接口给出建设性意见，此外commuter还将生成测试代码来检测实际操作系统的系统调用并行性，详细的论述分析请见下一小节。

我们的工作

设计方案

本课程设计分为以下9个步骤分阶段进行：

1. 准备开发环境 — 于第6周完成

把RV64 SMP的工具链，包括编译器GCC、模拟器QEMU等环境准备好。

这是基础性的工作，为进行RV64的开发，必不可少。

2. 学习RV64 — 于第6周完成

学习RV64用户态指令集架构和特权指令集架构。

这也是基础性的工作，为进行RV64的开发，必不可少。当然，若之后遇到问题，可以进一步查阅文档。

3. 动手实践RV64 SMP — 于第6周完成

基于bbl编写几个RV64 SMP的小例子，注重SMP的启动以及管理。这个环节旨在熟悉RV64 SMP的一些关键性问题，例如谁负责启动AP（除了启动处理器之外的处理器核心称为AP），启动AP后的状态如何设置等。

4. 开始移植 — 于第6周开始,第11周完成

开始将sv6移植到RV64单核环境。

这个阶段的重点以及难点都在于将原有的x86-64相关的部分在RV64架构下重新实现，包括虚拟内存管理、中断和异常（包括系统调用）、原子操作指令、寄存器上下文、内核栈设置、外设中断配置、设备驱动程序等。

5. 多核移植 — 于第11周完成

进一步将sv6移植到RV64多核系统。

注：根据实际情况，单核和多核的移植可以分开或者同时进行。

6. 真实硬件测试 — 于第12周完成

从辉总那里借来了RV开发板一套，含12V 2A DC5.5电源一个、装板子的瓦楞纸质包装盒一个、保护用粉色塑料海绵纸一张、RV板一个。在RV开发板上成功运行sv6移植。

注：感谢辉总大力支持！

7. 学习并实践符号化执行方法、Z3求解器 — 于第10周完成

学习符号化执行方法以及Z3求解器的使用。

commuter工具使用到了符号化执行方法以及SMT求解器，为了更好地完成sv6的优化，更好地写出接口规范，需要学习这两项内容。

8. 深入分析和理解commuter的设计与实现 — 于第10周开始，第13周完成

详细阅读[2]的文献和代码，分析和理解commuter工具的运行原理和代码实现并对其进行总结，方便后人阅读理解。

9. 基于commuter对socket建模 — 于第13周完成

利用commuter工具对unix socket API（从传输层到应用层）进行建模。

主要工作是接口规范的编写。

小组分工

1. sv6移植RV64 SMP

谭闻德主导，尹宇峰辅助。

2. 深入分析和理解commuter的设计与实现

尹宇峰主导，谭闻德辅助。

3. 基于commuter对socket建模

尹宇峰主导，谭闻德辅助。

具体细节

sv6移植

commuter分析和理解

注：本小节将基于[2]中的文献和代码进行论述，考虑到时间限制和难度较大等原因，我们略去了其中关于sim条件证明的部分，但是这不会影响我们对于commuter工具的整体理解和分析。

commuter简述

本小节根据[2]中的文献Abstract、1. Introduction以及5. Analyzing interfaces using COMMUTER章节整理得到。

在论文中，作者指出当今的操作系统判断并行性的方式较为简单：在实际操作系统不同的核上运行测试代码来检测并行性的瓶颈。这样的方法有着明显的弊端：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。为此作者提出了一个问题：我们是否可以仅仅通过系统调用的行为描述（specification）来判断系统调用与系统调用之间的并行性?

接着，作者提出了一条规则：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。这一规则的提出使得上述问题得到了解决（作者使用了名为Analyzer的工具来判断并行性）。

commuter的提出可以帮助我们分析和设计系统调用的行为描述，并设计相应的实现到达并行效果。例如说，在同一个文件目录下，多个进程调用open系统调用，在类POSIX文件系统中这是无法并行的，这是因为open系统调用需要返回最小的文件描述符，但是当我们把行为描述修改成任意一个合法的文件描述符，他们就可以实现并行了。

更进一步地，针对在行为描述层面证明可以被并行的系统调用序列，作者利用TestGen工具生成了相应的测试代码，并在实际操作系统（sv6和Linux）中利用MTrace工具运行测试代码，以此来检测系统调是否因为不必要的存储共享降低了os的并行性，同时帮助定位了OS并行性的瓶颈。最后，针对测试代码的结果，作者给出了几个可以帮助提升os并行性的例子。

总的来说，commuter的核心设计分成了三个部分Analyzer、TestGen、MTrace，其中Analyzer负责分析系统调用序列是否满足SIM条件，针对系统调用序列中满足条件的path condition，传递给TestGen；TestGen针对这些path condition实例化测试代码`TestGen.c`；MTrace是qemu的修改版本，额外记录了操作系统访问物理内存的信息，MTrace通过分析是否有缓冲行的访问冲突来判断实际os的系统调用之间的并行性。原文中也提到充分理解commuter是一件极其困难的事情，因为它涉及了许多的知识，例如符号化执行，z3求解。此外，更加令人头疼的是，限于篇幅，许多概念和细节，作者并没有真正阐释清楚，例如5.2 TESTGEN章节关于assignments同构这一概念和对于conflict coverage代码的实现。以下，我们将从commuter的这三个部分分别展开详细论述他们的设计与实现，方便后人继续深入研究commuter工具。

Analyzer

本小节根据[2]中的文献5. Analyzing interfaces using COMMUTER和5.1 ANALYZER章节整理得到。

总体来说，Analyzer的功能是输入对符号化的接口进行建模（代码实现在spec.py中），然后针对该模型计算出准确的运行条件，使得系统调用之间可以并行。

注：为了简化问题，作者在考虑系统调用序列并行化问题时候，将序列中调用的个数规定为2个。

接下来，我们逐步讲解：

首先是对符号化的接口进行建模，这是Analyzer的输入。我们需要定义每个系统调用的python model（即行为描述或specification）。例如论文中的rename model，有两个参数，一个是原先的名字，一个是希望重新命名的名字。通过输入参数的不同，rename会有相应不同的操作，当两个参数相同时，不需要任何操作，若不然，则需要进行若干操作。

接着，我们会通过符号化执行的方法来分析系统调用序列是否满足SIM条件，即序列以任意先后顺序调用，最后的系统状态和返回结果都是一致的，而且满足单调特性（系统调用的子序列同样满足SIM条件）。判断是否满足条件的方式也比较暴力，即枚举全部调用序列，然后观察系统状态和返回结果是否一致。

符号化执行的方法，会让我们可以遍历系统调用内部的所有if分支路径（我们称之为path），其中有一些路径可以做到并行，而有一些路径则不行，于是通过Analyzer的分析后，我们会得到一组可以并行化的路径，而在此基础之上，便可以得到运行路径需要path condition，我们将这组满足SIM条件的path condition称之为Commutativity conditions。

再次以rename系统调用为例，当我们输入给Analyzer rename的行为描述后，Analyzer会说renane(a, b)和rename(c, d)可以并行化，如果参数满足以下任意一个条件：

1. 文件a和c存在，而且a，b，c，d两两不相同。
2. 文件a存在，b≠c，文件c不存在；或者文件c存在，d≠a，文件a不存在。
3. 文件a和c都不存在。
4. a=b，c=d。
5. 文件a存在，a=b，a≠c；或者文件c存在，c=d，a≠c。
6. a，c是两个相同的索引节点，而且a≠c，b=d。

接下来，我们详细讨论一下Analyzer的代码实现：

simtest.py的test函数，这个函数就是用来检测sim条件的。它采用了对系统调用序列深度优先搜索方法来检测，对于第一次抵达树根时，记录下系统状态和返回值，此后每次抵达树根就会进行比较，如果不一样就会直接退出函数。值得注意的是，在simtest.py中调用test的方式是符号化执行的，所以说不会出现说特定参数恰好返回值和系统状态一致的情况，符号化变量的结果一致，我们就可以安心地认为他们满足系统调用。

simtest.py的test_callset函数，这个函数利用符号化执行的方式调用了test函数，对于每个path都验证了sim条件，并会将所有满足sim条件的path condition取或打印出来，所有不满足sim条件的path condition也会取或打印出来。

TestGen

我认为这是commuter最难懂的部分。今天我在这里花费了大量时间阅读相关部分的文献，但是仍然有一处句子不理解："TESTGEN partitions most values in isomorphism groups and considers two assignments equivalent if each group has the same pattern of equal and distinct values in both assignments." 我的问题是这里group划分的规则是什么？pattern的含义又是什么？阅读文献的感觉是晦涩难懂，下一步的计划是阅读TestGen的代码。代码有两部分，一个是testgen.py，这个是所有生成测试代码的基类，还有一个是model-specific test code generator，即fs_testgen.py，这是testgen的子类，功能是对于特定系统调用将符号化变量转为特定值的代码。值得注意的是，这里testgen不仅仅是将之前满足Analyzer给出的commutativity conditions的符号化变量转为特定

值，然后生成c测试代码。文献中指出，有的时候，相同的系统调用序列，相同的path路径，也有可能出现一个没有访问冲突，一个出现访问冲突，例如：两个pwrite访问同一个数组，两个角标一致和不一致就会出现不一样的情况。原因在于，他们的访问模式不一样。所以TestGen在遍历所有path路径的情况下，还需要遍历所有的访问模式，例如：不同指令在共享的数据结构中访问同一个元素或是不同元素。对于遍历path路径的方法，文献中指出，TestGen的做法是反复对部分的path condition取反，以达到遍历效果。（具体实现细节我还是吃不准，需要继续阅读实现代码）。

MTrace

这一部分的工作较为简单，功能是运行TestGen的测试代码，测试代码分为两部分：一个是setup函数，负责初始化系统初始状态，系统调用参数，一个是若干个系统调用函数，将在不同的核上运行，MTrace负责检测是否出现访问冲突，如果没有就说明这些系统调用可以并行，不然，则会记录下访问冲突的变量和对应调用他们的代码。

总的来说，Analyzer会先第一步检测定义好行为的系统调用之间的并行性，而Mtrace运行的测试代码会检测实际操作系统中实现的系统调用是否潜在地访问了不必要的共享变量，而导致了并行性变差。

基于commuter对socket建模

结束语

实验成果

实验收获

在本次课程设计中，我们：

- 熟悉了RV64指令集
- 熟悉了一个操作系统的移植过程
- 学习了符号化执行方法
- 学习了Z3 SMT求解器的使用方法
- 了解了sv6
- 深入了解了commuter
- 了解了socket

未来的工作

我们希望可以将基于commuter对unix socket API所建得的模型，结合commuter中的testgen模块生成相应的测试代码，以便可以对实际操作系统的网络子模块进行并行性测试与优化。

致谢

感谢陈渝老师，向勇老师对于我们课程设计的耐心指导；

感谢辉总给予我们硬件的大力支持；

感谢路橙，于志竟成组与我们小组合作与讨论。

参考文献

1. sv6 https://github.com/aclements/sv6
2. Commuter https://pdos.csail.mit.edu/archive/commuter/ https://github.com/aclements/commuter

3. The RISC-V Instruction Set Manual Volume I: User-Level ISA

   https://riscv.org/specifications/

4. The RISC-V Instruction Set Manual Volume II: Privileged Architecture

   https://riscv.org/specifications/privileged-isa/

5. bbl-ucore https://ring00.github.io/bbl-ucore/
6. ucore labs RISC-V 64移植 https://gitee.com/shzhxh/ucore_os_lab/tree/riscv64-priv-1.10
7. Booting a RISC-V Linux Kernel https://www.sifive.com/blog/2017/10/09/all-aboard-part-6-booting-a-risc-v-linux-kernel/
8. Entering and Exiting the Linux Kernel on RISC-V https://www.sifive.com/blog/2017/10/23/all-aboard-part-7-entering-and-exiting-the-linux-kernel-on-risc-v/
9. Paging and the MMU in the RISC-V Linux Kernel https://www.sifive.com/blog/2017/12/11/all-aboard-part-9-paging-and-mmu-in-risc-v-linux-kernel/
10. 符号执行入门 https://zhuanlan.zhihu.com/p/26927127
11. mini-mc https://github.com/xiw/mini-mc
12. Z3 https://github.com/Z3Prover/z3
13. HiFive Unleashed https://www.sifive.com/products/hifive-unleashed/
14. Freedom U540-C000 Manual https://www.sifive.com/documentation/chips/freedom-u540-c000-manual/