HyperKernel在RISC-V的移植与CPU结合验证

计54 秦岳
计54 陈宇

HyperKernel

- xv6
- 正确性验证
- 部分“简化”

预期目标

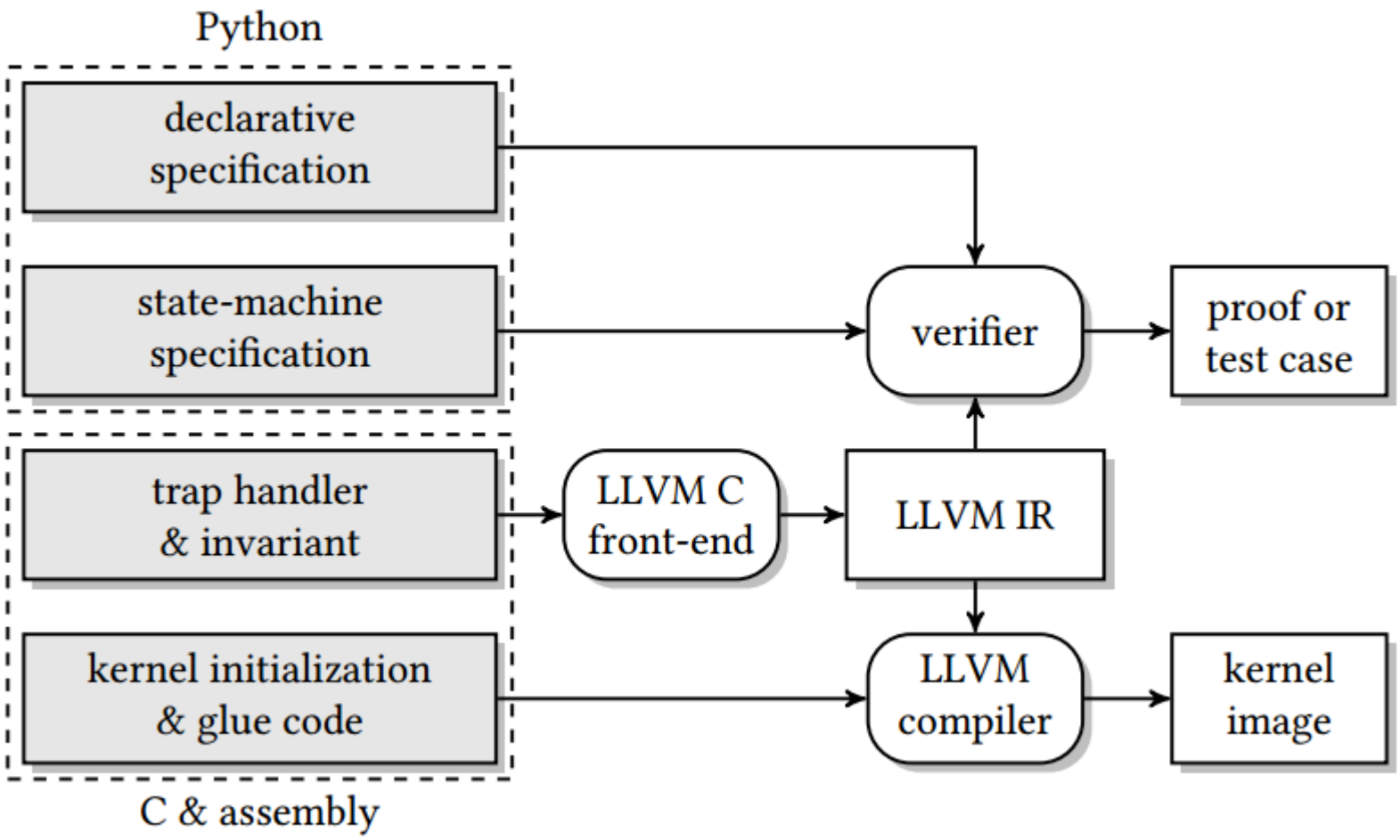
- 将hv6移植到RISC-V
- 修改在RISC-V下hv6的spec并完成验证
- 结合CPU执行流程规范化描述尝试自动发现'熔断'等漏洞

Z3

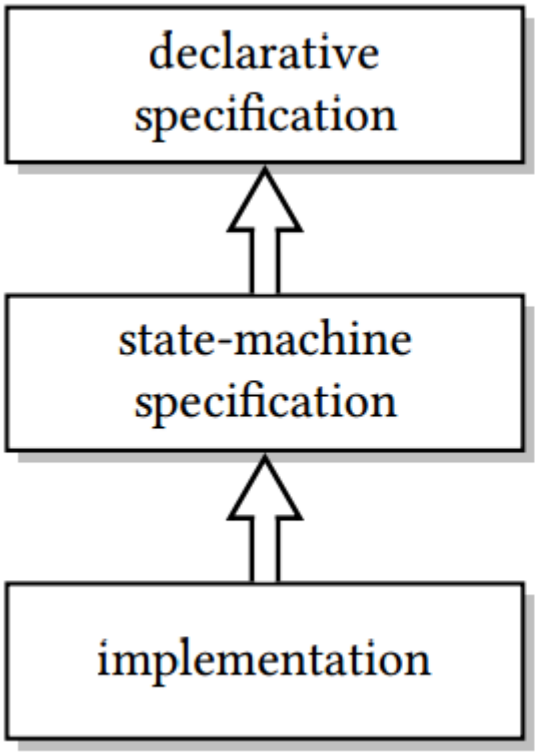
- SMT
- 数值理论，数组理论，布尔代数理论，一阶谓词逻辑理论

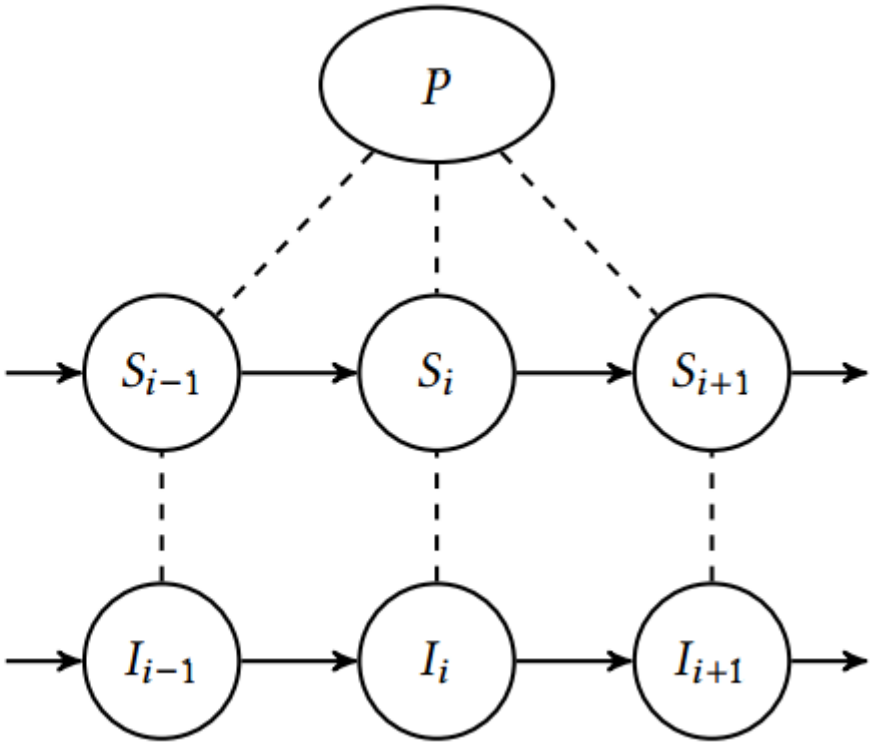
- sat or unsat
- Test Case

HyperKernel



HyperKernel





状态机规约

```
class AbstractKernelState(object):
  current       = PidT()
  proc_fd_table = Map((PidT, FdT), FileT)
  proc_nr_fds   = RefcntMap(PidT, SizeT)
  file_nr_fds   = RefcntMap(FileT, SizeT)
  ...
```

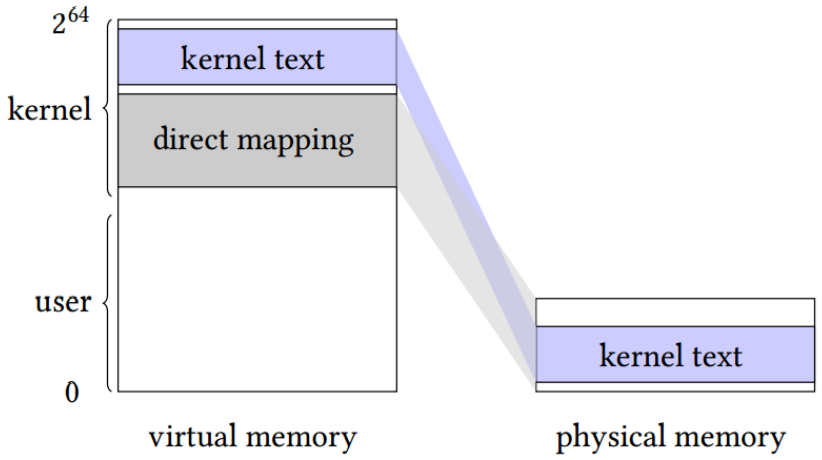
```
def spec_dup(state, oldfd, newfd):
  # state is an instance of AbstractKernelState
  pid = state.current
  # validation condition for system call arguments
  valid = And(
    # oldfd is in [0, NR_FDS)
    oldfd >= 0, oldfd < NR_FDS,
    # oldfd refers to an open file
    state.proc_fd_table(pid, oldfd) < NR_FILES,
    # newfd is in [0, NR_FDS)
    newfd >= 0, newfd < NR_FDS,
    # newfd does not refer to an open file
    state.proc_fd_table(pid, newfd) >= NR_FILES,
  )

  # make the new state based on the current state
  new_state = state.copy()
  f = state.proc_fd_table(pid, oldfd)
  # newfd refers to the same file as oldfd
  new_state.proc_fd_table[pid, newfd] = f
  # bump the FD counter for the current process
  new_state.proc_nr_fds(pid).inc(newfd)
  # bump the counter in the file table
  new_state.file_nr_fds(f).inc(pid, newfd)

  return valid, new_state
```

上层属性约束

```
ForAll([f, pid, fd], Implies(file_nr_fds(f) == 0,
                             proc_fd_table(pid, fd) != f))
```

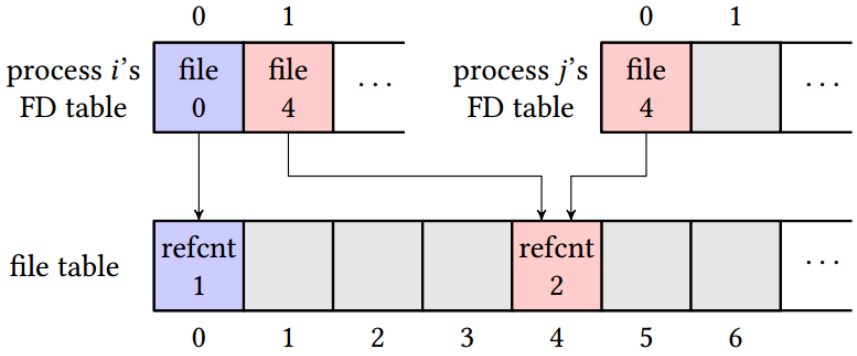
HyperKernel简化

- 内核空间与用户空间完全分离(各占一半)
- 内核空间和物理内存对等映射
- 内核和用户进程分别使用独立的页表在root和guest下运行。



HyperKernel简化

- 有限接口设计
  - 例:dup(oldfd) -> dup(oldfd,newfd)
- 禁止无限循环和递归，不允许嵌套中断，内核处理过程关中断
- 对DMA划分专用内存区域隔离影响
- 内核设计限制语法(LLVM IR)



HyperKernel简化

- 用户进程置于虚拟化环境ringo中(硬件虚拟化)
- 将中断描述符表（IDT）暴露给用户进程
- 用户空间直接handle难验证的异常
- 用户显示的管理资源
- 提供系统调用让用户显示的回收页面
- 内核数据结构使用数组而非链表简化验证
- 复杂语义的系统调用交由用户库(fork,exec)

已完成的工作

- 配置SMT工具链
  - 1.配置Z3等运行环境
  - 2.基本了解符号执行与Z3基本原理
- 重现hyperkernel project
  - 1.可以编译并运行hv6操作系统
  - 2.重现状态机规约与实现一致性验证
  - 3.重现上层属性与状态机规约的一致性验证
  - 4.基本了解HyperKernel基本原理

已完成的工作

- RISC-V 64 工具链搭建
  - 1. 编译运行工具链基本完成(gcc, qemu, spike)
  - 2. 能够运行简单的代码，能够运行下载的linux镜像
  - 3. 能够编译ucore_os_lab仓库riscv64-priv-1.10分支的代码，可以使用spike运行生成的镜像
  - 4.初步了解risc-v规范

移植RISC-V计划

- 1. Boot Loader
- 2.修改中断处理
- 3.修改内存管理
- 4.修改进程管理

正确性证明移植计划

- 1.根据hv6的移植过程修改必要的状态机spec
- 2.针对RISC-V规范增加/删除/修改上层属性
- 3.编译LLVM IR并通过emmiter生成python接口代码，并对中途发生的错误进行必要的修改
- 4.部分重写equiv对等函数

谢谢观看