学习QEMU/RISU的经验和讨论

李威威
PLCT实验室

目 录

RISU是什么?

RISU -- Random Instruction Sequence generator for Userspace testing

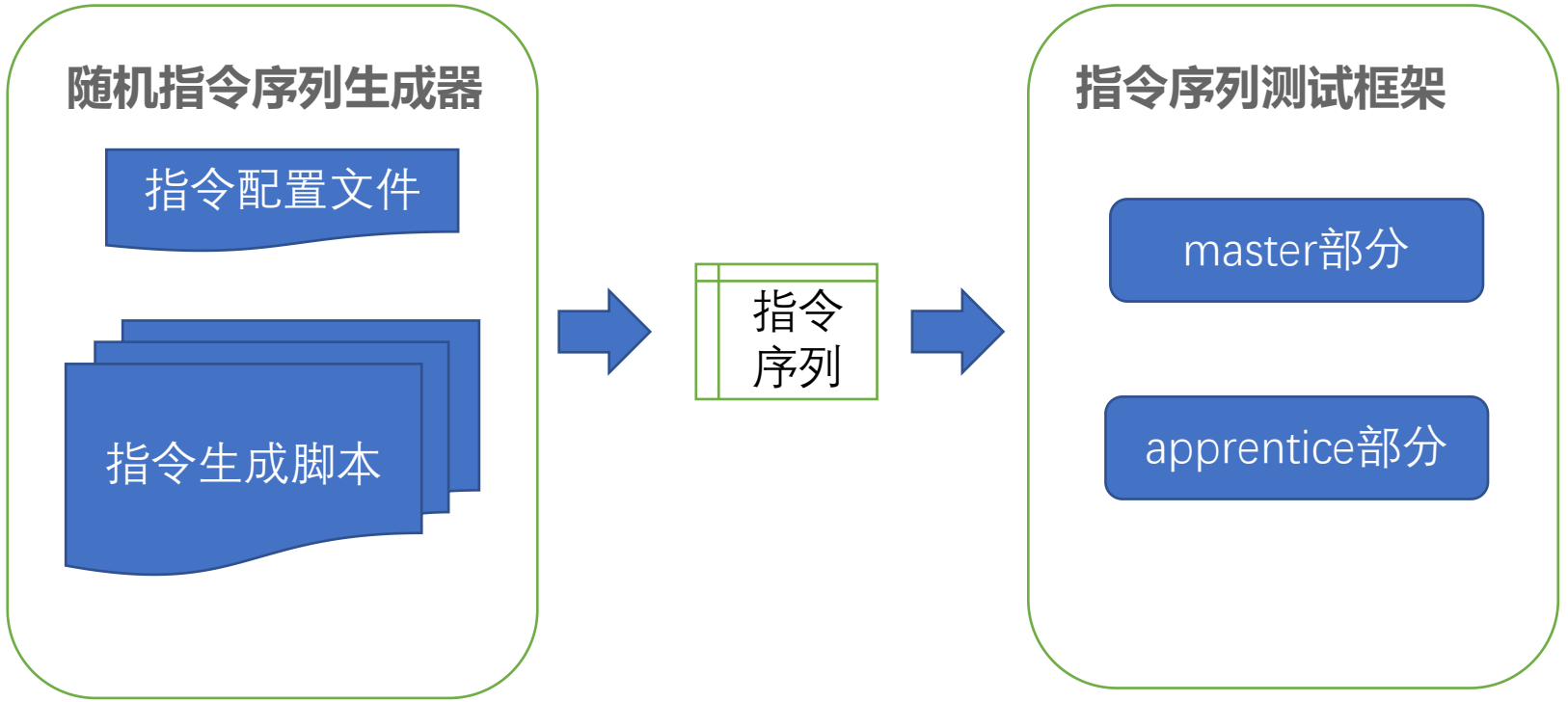
随机指令序列生成器：用于测试**架构模型**，如QEMU

适用于用户态测试：仅针对**linux用户空间可见**的部分进行测试

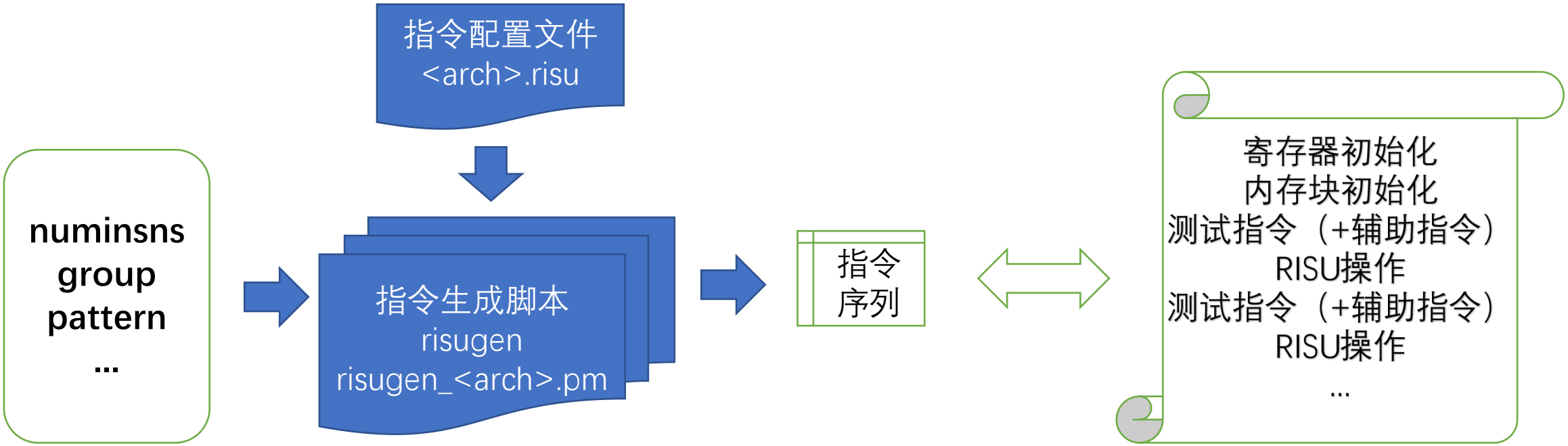
GIT仓库：https://git.linaro.org/people/peter.maydell/risu.git

RISU是如何工作的?

RISU主要包括两个部分：随机指令序列生成器和指令序列测试框架



随机指令序列生成器

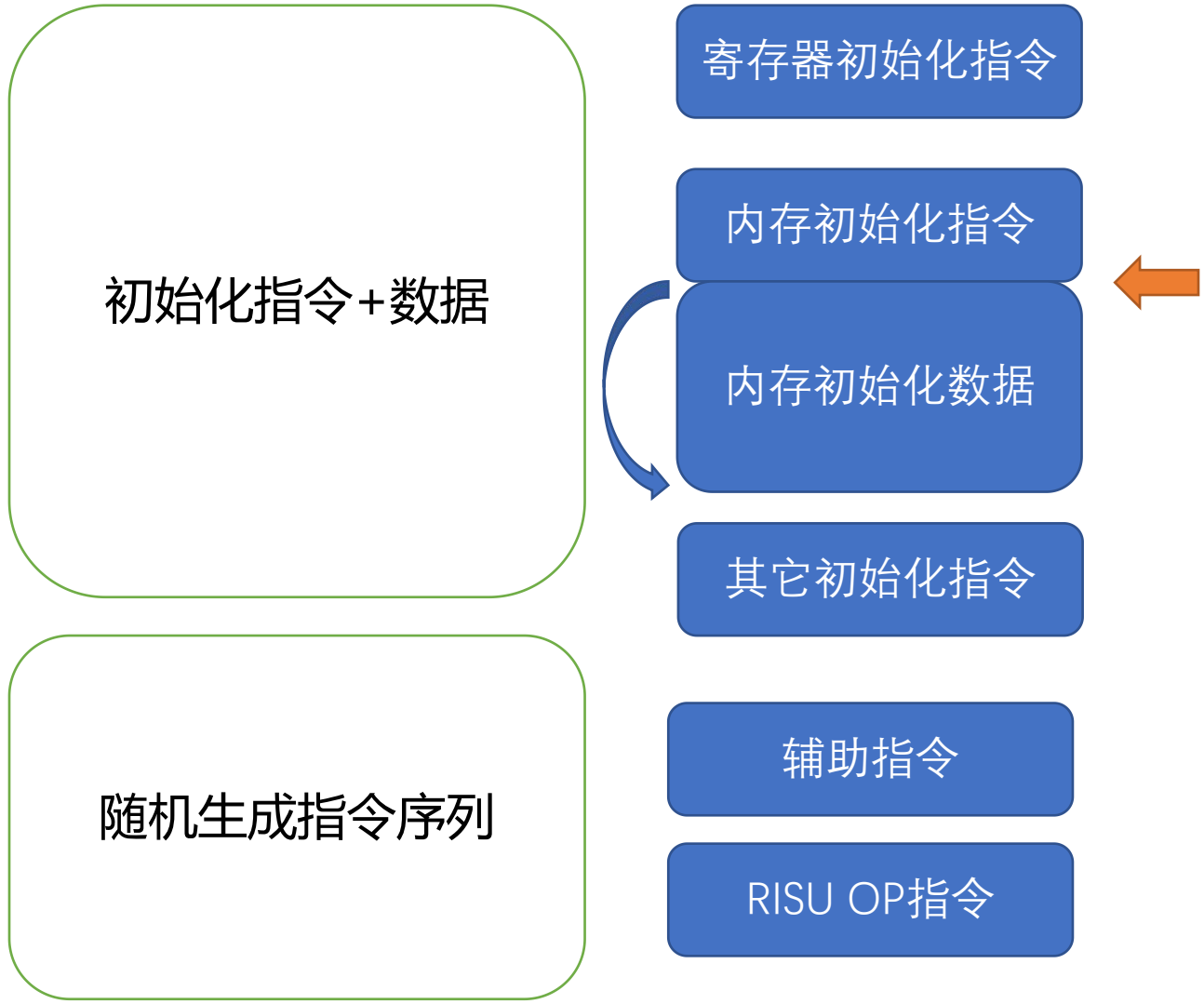

指令配置文件
<arch>.risu
numinsns
group
pattern
...
指令生成脚本
risugen
risugen_<arch>.pm
指令
序列
寄存器初始化
内存块初始化
测试指令（+辅助指令）
RISU操作
测试指令（+辅助指令）
RISU操作
...

指令配置文件<arch>.risu

```
.mode arm.aarch64                                          =>    指示arch/subarch信息
...
@Store                                                     =>    group定义
ST1m_1 A64_V 0 Q:1 001100000 00000 0111 size:2 rn:5 rt:5  \
                                                           =>    指令信息
!constraints { $rn != 31; } \                              =>    指令限制
!memory { align(1 << $size); reg($rn); }                   =>    memory地址及限制
 ...
@                                                          =>    group结束
```

指令生成脚本

- **公用脚本：risugen，risugen_common.pm**
  - 解析命令行输入，指令配置文件，将解析出的numinsns，arch，keys等信息作为参数调用write_test_code接口生成指令序列
  - 提供公用的接口，如insn32()等
- **架构相关脚本：risugen_<arch>.pm**
  - 实现write_test_code接口
  - 实现指令配置文件中constraint，memory等代码块中包含的接口，如 align()，reg()，reg_plus_imm()等



指令序列测试框架

- **测试模型：基于golden model的验证模型（master vs apprentice）**
  - 连接方式：TCP连接
  - 比对触发点：RISU操作触发SIGILL处理
    - RISU操作必须采用一组无效指令
      - OP_COMPARE, OP_CMPAREMEM
      - OP_SETMEMBLOCK, OP_GETMEMBLOCK
      - OP_TESTEND
  - 比对信息：自定义的reginfo结构（resu_reginfo_<arch>.h）和 MEM块内容
    - 从signal 处理函数的sigcontext参数中提取寄存器信息

如何扩展RISU?

如何支持一个新的架构?

- 实现指令配置文件以及架构相关指令生成脚本
  - <arch>.risu：指令配置文件
  - risugen_<arch>.pm： 指令生成脚本
- 实现架构相关接口
  - risu_reginfo_<arch>.h/risu_reginfo_<arch>.c： 架构相关reginfo结构定义，及相关接口实现
  - risu_<arch>.c： 其它架构相关接口
- 实现测试程序（可选）
  - test_<arch>.s：汇编测试文件

测试框架主要需要实现的接口

```c
/* Interface provided by CPU-specific code: */

/* Move the PC past this faulting insn by adjusting ucontext
 */
void advance_pc(void *uc);

/* Set the parameter register in a ucontext_t to the specified value.
 * (32-bit targets can ignore high 32 bits.)
 * vuc is a ucontext_t* cast to void*.
 */
void set_ucontext_paramreg(void *vuc, uint64_t value);

/* Return the value of the parameter register from a reginfo. */
uint64_t get_reginfo_paramreg(struct reginfo *ri);

/* Return the risu operation number we have been asked to do,
 * or -1 if this was a SIGILL for a non-risuop insn.
 */
int get_risuop(struct reginfo *ri);

/* Return the PC from a reginfo */
uintptr_t get_pc(struct reginfo *ri);

/* initialize structure from a ucontext */
void reginfo_init(struct reginfo *ri, ucontext_t *uc);

/* return 1 if structs are equal, 0 otherwise. */
int reginfo_is_eq(struct reginfo *r1, struct reginfo *r2);

/* print reginfo state to a stream, returns 1 on success, 0 on failure */
int reginfo_dump(struct reginfo *ri, FILE * f);

/* reginfo_dump_mismatch: print mismatch details to a stream, ret nonzero=ok */
int reginfo_dump_mismatch(struct reginfo *m, struct reginfo *a, FILE *f);

/* return size of reginfo */
const int reginfo_size(void);
```

- reginfo_init：从ucontext中提取reginfo信息，其中需要将不能比较的寄存器如**栈指针，全局指针，线程指针**等赋值成常数值，如0xdeadbeef
- advance_pc: 修改pc，跳过risu op
- set_ucontext_paramreg: 用于OP_GETMEMBLOCK操作处理过程中，将mem block的地址赋值给**指定的寄存器**
- get_reginfo_paramreg:用于OP_SETMEMBLOCK操作处理过程中，从reginfo的**指定的寄存器**中获取mem block地址保存起来
- get_risuop: 从出错指令中提取risu op编码，**最好采用一组易提取risu op的无效指令**

RISU的局限性

- 不能测试特权指令
- 所有能够比对的寄存器状态都来源于signal handler
- 不宜测试直接针对特殊功能寄存器（如栈指针，全局指针等寄存器）运算的指令
- 不宜测试分支指令
- 其它，如伪随机，当前脚本只支持16/32位指令格式

谢谢各位

欢迎提问、讨论、交流合作