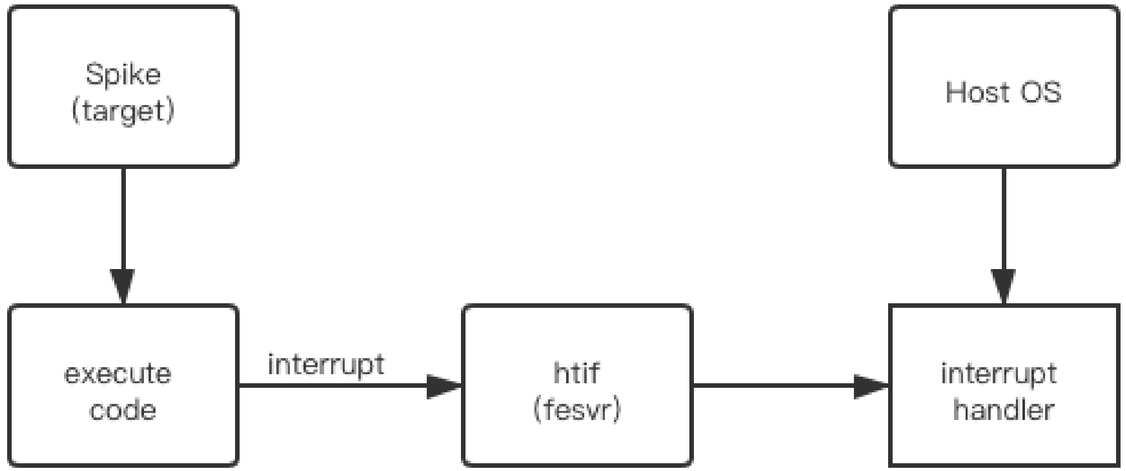
Spike-fesvr及外部设备实现分析

PLCT实验室 实习生 王萌 2020.6.24

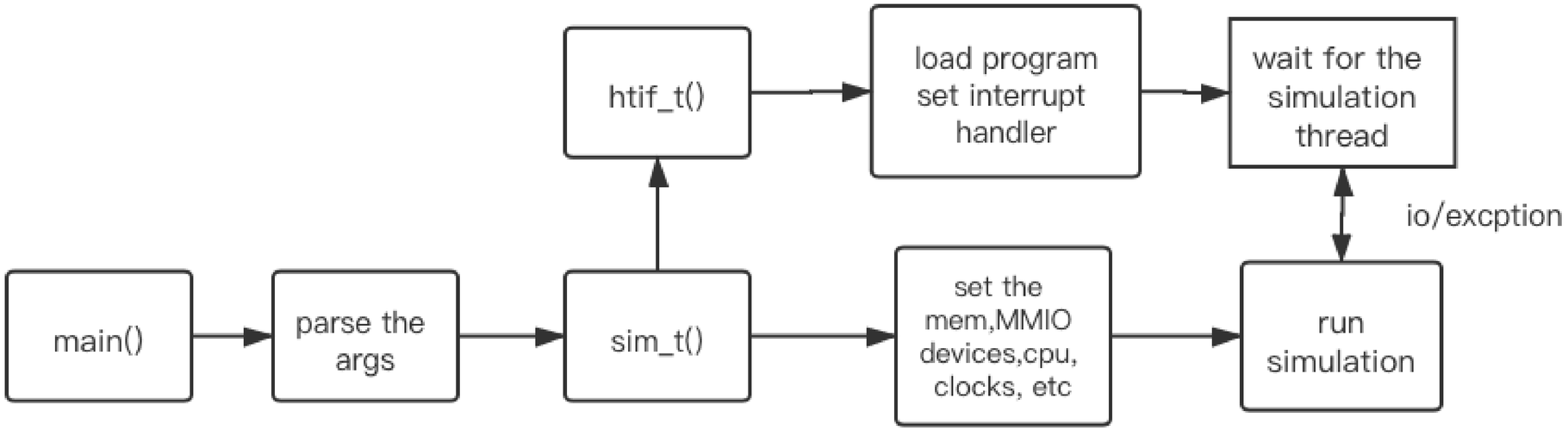
目录

fesvr 模块简介

- fesvr， 全称是 risc-v front-end server
- 主要用来实现simulation target 与host 主机的交互



Spike 启动simulation的过程



Spike 中的设备类

```
class abstract_device_t {
 public:
  virtual bool load(reg_t addr, size_t len, uint8_t* bytes) = 0;
  virtual bool store(reg_t addr, size_t len, const uint8_t* bytes)
  virtual ~abstract_device_t() {}
};
```

设备的类abstract_device_t（riscv/device.h）

- 内建的设备有mem_t内存设备，bus_t总线设备（用于处理所有的内存请求，包括MMIO），clint_t中断控制器设备
- 还实现了用于扩展其他设备的mmio_plugin_device_t

Spike 中的设备类

内存设备mem_t

- 内存设备不使用load,store方法访问
- 内存请求通过总线设备bus_t来访问
- 访问内存的调用链：
- cpu请求内存->mmu进行VA到PA的转换->在总线上查找物理地址对应的内存设备->从主机上对应地址访问主机内存

```
  bool load(reg_t addr, size_t len, uint8_t* bytes) { return false; }
  bool store(reg_t addr, size_t len, const uint8_t* bytes) { return false; }
  char* contents() { return data; }
  size_t size() { return len; }
```

Spike 中的设备类

总线设备bus_t

- 总线设备采用枚举的方式访问：
- 总是访问起始地址小于访问地址且离访问地址最近的设备
- 只有当前地址属于MMIO地址时才会使用load/store方法

```
std::pair<reg_t, abstract_device_t*> bus_t::find_device(reg_t addr)
{
  auto it = devices.upper_bound(addr);
  if (devices.empty() || it == devices.begin()) {
    return std::make_pair((reg_t)0, (abstract_device_t*)NULL);
  }
  it--;
  return std::make_pair(it->first, it->second);
}
```

Spike 中的设备类

MMIO设备mmio_plugin_device_t

- MMIO设备由外部运行库extlib定义
- riscv/mmio_plugin.h中定义了c和c++的接口
- extlib中需要实现alloc，dealloc，load，store方法
- https://github.com/riscv/riscv-isa-sim/pull/315 中提供了多种实现的示例

```
__attribute__((constructor)) static void on_load()
{
  static mmio_plugin_t test_mmio_plugin = {
      test_mmio_plugin_alloc,
      test_mmio_plugin_load,
      test_mmio_plugin_store,
      test_mmio_plugin_dealloc
  };

  register_mmio_plugin("test_mmio_plugin", &test_mmio_plugin);
}
```

需要注意的是，实现设备时需要在构造方法时注册设备，但不需要在注册时实例化

Spike 中的设备类

MMIO设备mmio_plugin_device_t

- MMIO设备由外部运行库extlib定义
- riscv/mmio_plugin.h中定义了c和c++的接口
- extlib中需要实现alloc，dealloc，load，store方法
- https://github.com/riscv/riscv-isa-sim/pull/315 中提供了多种实现的示例

```
__attribute__((constructor)) static void on_load()
{
  static mmio_plugin_t test_mmio_plugin = {
      test_mmio_plugin_alloc,
      test_mmio_plugin_load,
      test_mmio_plugin_store,
      test_mmio_plugin_dealloc
  };

  register_mmio_plugin("test_mmio_plugin", &test_mmio_plugin);
}
```

需要注意的是，实现设备时需要在构造方法中注册设备，但不需要在注册时实例化

Spike 中的设备类

MMIO设备mmio_plugin_device_t

- MMIO设备通过运行时参数—device进行实例化
- —device可以指定的参数包括设备名，base address和一个str类型的参数
- 添加设备后，设备会以mmio_plugin_device_t的形式被加载到总线设备上
- 通过lw，sw指令访问对应地址即可访问对应的设备
- https://github.com/riscv/riscv-isa-sim/pull/315 有完整的设备实现以及驱动程序的示例

Spike 中的设备类

中断控制器设备clint_t

- clint_t设备用于控制设备中断，位于内存中0x2000000处，长度为0xc0000
- 其定义如下
- 其中msip用于控制中断，mtimecmp用于时钟控制的中断，mtime为计时器

```
/* 0000 msip hart 0
 * 0004 msip hart 1
 * 4000 mtimecmp hart 0 lo
 * 4004 mtimecmp hart 0 hi
 * 4008 mtimecmp hart 1 lo
 * 400c mtimecmp hart 1 hi
 * bff8 mtime lo
 * bffc mtime hi
*/
```

借助fesvr实现的虚拟IO

- 如果不借助fesvr，使用MMIO plugin处理中断和系统调用
- 需要手动实现每个设备
- 在fesvr中存在另外一套interface用于实现虚拟I/O
- openSBI和proxy kernel中都采用这套interface实现IO操作

借助fesvr实现的虚拟IO

- 在simulator运行时，htif_t也会运行一个子进程用来与target进行交互和接管IO操作
- 在elf文件中需要预先定义tohost和fromhost两个symbol，指向两个64位地址，用于host和target之间的交互
- 设备可以通过向tohost指向的内存来请求I/O操作
- tohost中的值的定义：[63:56]高8位用于选择设备，[55:48]用于指定command，其余48位为供设备使用的payload

借助fesvr实现的虚拟IO

syscall_proxy设备syscall_t

- 设备0为syscall_proxy设备，用于处理系统调用
- 该设备只实现了command 0，有以下两个子功能：
- 如果最低位为0，则[47:0]将是一个指向特定系统调用的指针
- 否则，payload将被视为一个exit code，为0为正常退出，否则视为异常

借助fesvr实现的虚拟IO

虚拟终端设备bcd_t

- 设备1为虚拟终端设备，用于处理终端的输入输出
- 该设备实现了command0， command1两种操作
- command0表示从host的stdin读入一个字符到tohost的低八位
- command1表示从host的stdout输出一个字符

借助fesvr实现的虚拟IO

riscv-pk中借助fesvr实现的虚拟I/O

```
static void do_tohost_fromhost(uintptr_t dev, uintptr_t cmd, uintptr_t data)
{
  spinlock_lock(&htif_lock);
    __set_tohost(dev, cmd, data);
    while (1) {
      uint64_t fh = fromhost;
      if (fh) {
        if (FROMHOST_DEV(fh) == dev && FROMHOST_CMD(fh) == cmd) {
          fromhost = 0;
          break;
        }
        __check_fromhost();
      }
    }
  spinlock_unlock(&htif_lock);
}
```