GDB 远程调试 NEMU 和 NPC

ysyx_23060224 李心杨

为什么需要？

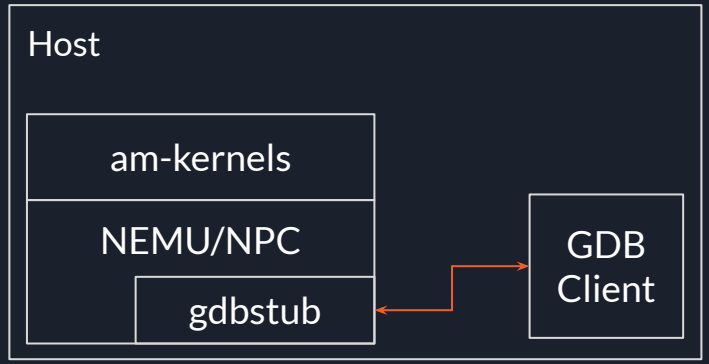
- SDB并不是一个通用的库
  - NEMU上的SDB需要迁移到NPC中才能使用
- SDB在调试过程中不能将指令与源码直接对应
- SDB的简单表达式无法解析ELF文件中的符号
  - 无法使用函数名直接 设置断点，如：`break main`

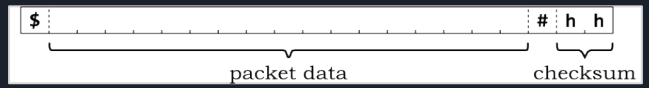
GDB Remote Serial Protocol

- 使用tcp/socket在调试器和程序间交换数据
- RTFM
- 数据包构成
- 通信过程
  - 客户端：发送-等待回复
  - gdbstub：等待消息-处理-回复

以获取内存为例为例：

```
-> m80000000,4
<- +
(wait for processing)
<- 12345678
-> +
```





API设计

- 最初的想法：直接借用difftest的api进行扩展
  - step没有反馈：异常退出？遇到断点？正常退出？
  - 不存在设置断点的功能
  - 耦合严重，难以维护
- 更好的方法：
  - 使用了github.com/RinHizakura/mini-gdbstub

API设计

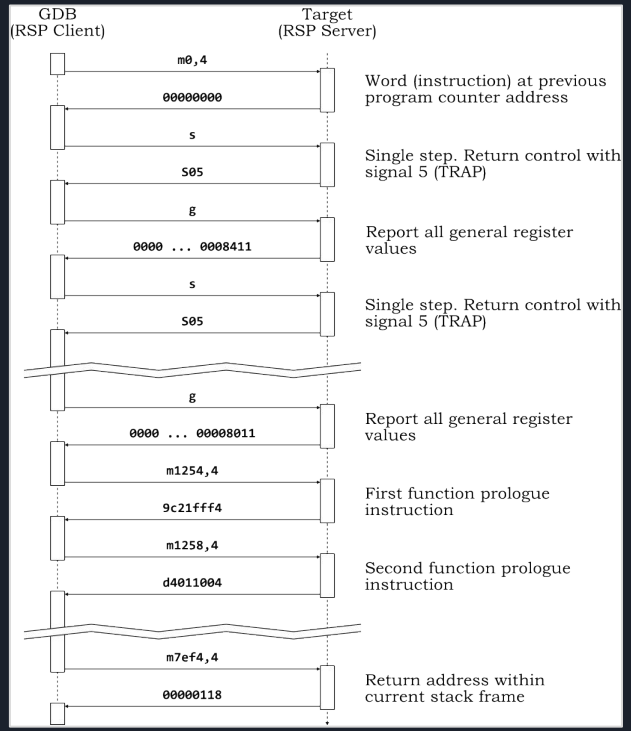
```
struct target_ops {
    void (*cont)(void *args, gdb_action_t *res);
    void (*stepi)(void *args, gdb_action_t *res);
    int (*read_reg)(void *args, int regno, size_t *value);
    int (*write_reg)(void *args, int regno, size_t value);
    int (*read_mem)(void *args, size_t addr, size_t len, void *val);
    int (*write_mem)(void *args, size_t addr, size_t len, void *val);
    bool (*set_bp)(void *args, size_t addr, bp_type_t type);
    bool (*del_bp)(void *args, size_t addr, bp_type_t type);
    void (*on_interrupt)(void *args);
};
```

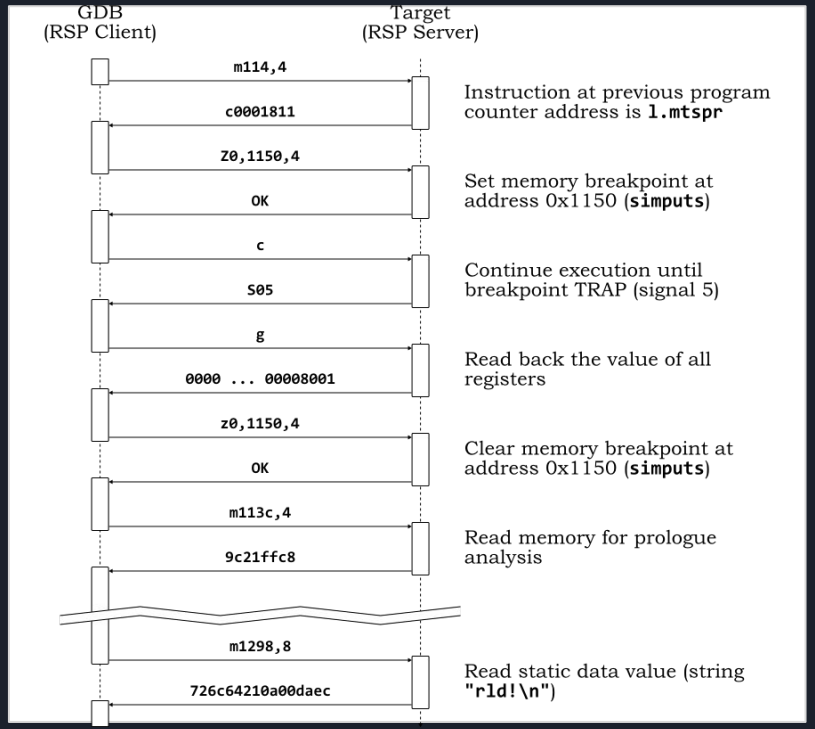
API设计

```
typedef struct {
    enum {
        ACT_NONE,
        ACT_BREAKPOINT,
        ACT_WATCH,
        ACT_RWATCH,
        ACT_WWATCH,
        ACT_SHUTDOWN
    } reason;
    size_t data;
} gdb_action_t;
```

```
typedef enum {
    BP_SOFTWARE = 0,
    BP_WRITE,
    BP_READ,
    BP_ACCESS
} bp_type_t;
```

协议图解
step和continue





优化

- 使用Socket通信
  - Socket的通信开销约为TCP的三分之一
- No-acknowledgment mode
  - 2 RTT -> 1 RTT

为Difftest设计调试器

- stepi和cont不要涉及通信，还是需要使用动态库
- 结果不一致时可以中断执行
- 使用multiprocess扩展来在ref和dut之间切换
- 优势：
  - 动态修正dut或ref的寄存器（或内存），解决可能存在的不同步 问题