RV32 Syscall Intercept Mechanism

Campbell McClendon

July 2024

1 Considerations

There are multiple relevant factors we must consider to design a mechanism to intercept syscalls on RISC-V.

1. Most RISC-V instructions are 32 bits wide, but the C extension provides compressed instructions that are 16 bits wide.
2. Although a RV32 processor could be connected to a bus wider than 32 bits, it would only use the bottom 2^{32} bytes of memory.

2 Mechanism

We are able to apply an approach similar to what is used for x86_64.
There are three available approaches to replacing an `ecall` instruction.

1. We will replace the `ecall` instruction with a `jalr` instruction, utilizing the 12-bit immediate offered by `jalr` to jump up to 2^{12} bytes in the direction of the destination (`pc`-relative).
2. If this is not far enough to jump directly to the destination, we will examine the instruction preceding the `ecall`. If this preceding instruction is not effected by the `pc` (such as `auipc`), we will relocate it to a position mentioned later. We will replace it with `auipc`, which allows us to fill the upper 20 bits of a register (`pc`-relative) and use that register with the aforementioned `jalr` to achieve a full 32-bit jump.
3. If we are unable to relocate the preceding instruction, we will use `jalr` to jump to a trampoline, which will be a series of replaced `nop` or `c.nop` that will use the same sequence of instructions defined above to jump either directly to the destination or to another trampoline.

The destination in this case is the intercept table, which will contain the relocated instruction (if any). The intercept table is a series of copies of a template that will, in order:

1. Save registers to the stack.

2. Jump to the user intercept code and allow it to return.
3. Restore registers from the stack.
4. Set the return value, if desired by the user intercept code.
5. Execute the relocated instruction if present. Otherwise, default to a `nop` instruction.
6. Return to the original `ecall` location.

Each copy will be unique to a syscall, thus allowing it to know which `ecall` it should to return to.

3 Development

There are many RV32 development boards available on the market at the moment, but very few (if any) seem to have enough memory to run Linux. Most development boards that match this requirement are discontinued in favor of RV64 development boards as more memory is often beneficial for development. However, I believe that, as RISC-V continues to grow in market share, RV32 will be used more frequently in production. In particular, it could grow to replace ARM cores typically used in embedded devices. Although RV64 might be used in development, more programs that target RV32 could arise as a result.
I plan to develop using Qemu. I will test with a sufficient development board (for example, the RV32 Sipeed M1s Dock).
Finally, some of my testing work from task 2 is available at the following GitHub repo: `https://github.com/jcrm1/syscall_intercept/tree/risc-v-scratch-work`
This is also where I will upload my code for task 3 (albeit in a different branch).