www.bzl.es

BZL
BARCELONA
ZETTASCALE LAB

Enabling Syscall Interception on RISC-V

Petar Andrić, Aaron Call, Ramon Nou

Barcelona Supercomputing Center

The European Union's technological sovereignty strategy centers around the RISC-V Instruction Set Architecture, with the European Processor Initiative leading efforts to build production-ready processors. Focusing on realizing a functional RISC-V ecosystem, the BZL initiative from Spain is making an effort to create a software stack along with the hardware. In this poster, we detail the efforts made in porting a widely used syscall interception library, mainly used on AdHocFS (i.e., DAOS, GekkoFS), to RISC-V and how we overcame some of the limitations encountered.

***syscall_intercept* library:**
https://github.com/GekkoFS/syscall_intercept/tree/riscv

***GekkoFS*:**
https://storage.bsc.es/gitlab/hpc/gekkofs

Implementation

Obstacles

1. The relative jump instruction (`jal`) has a **±1 MB reach** which is not enough to jump out of *libc*, unlike x86's `jmp` (±2 GB) or PowerPC (±32 MB).
2. Instructions are naturally better aligned, so **nops are rarely present** in *libc*, eliminating the possibility of using *nop-trampolines*.
3. The Linux kernel on RISC-V **saves the full context** during interrupts, including caller-saved registers. This is relevant for indirect jumps because no register can be overwritten.

These constraints required the use of an indirect jump sequence (`auipc + jalr`), and the calling convention **must be preserved**. This sequence requires 8 bytes, plus prologue/epilogue, **totalling 16 B** when RVC is supported. **x86 requires only 5 or 2 B** (with nop-trampoline).

Solution

Three patching methods:

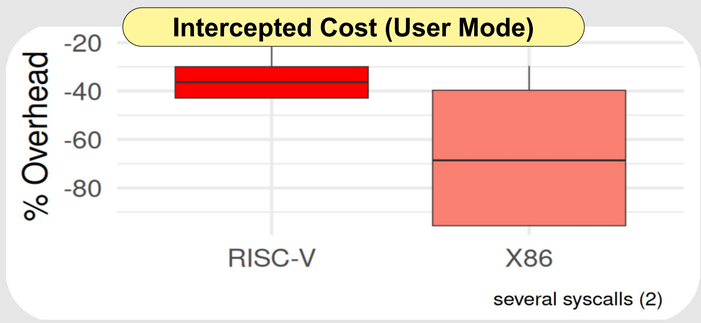
1. **Gateway Patch**: Applied when an *ecall* is surrounded by many relocatable instructions. It creates a ±2 GB jump, serving as the foundation for the library. Smaller patches jump to these gateways to reach the *syscall_intercept* library.
2. **Middle Patch**: Applied when a sufficient number of relocatable instructions are available to preserve the calling convention. It jumps with `jal` to the gateway, where it gets "forwarded" to the *syscall_intercept* library.
3. **Small Patch**: Applied when there is not enough space to preserve the calling convention. Static analysis stores the syscall number (a7) in the patch's structure, allowing `jal` to overwrite it, as its value is restored later inside the library.

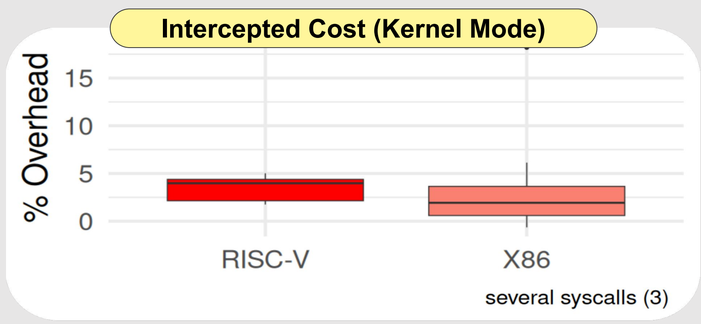
Approximately, *libc* is patched with 40% Gateway, 45% Small, and 15% Middle patch types.

Trade-offs

Less memory is allocated per patch because the library uses an indirect patching approach. Instead of each patch jumping to its own location, all patches redirect to a shared entry point, where they are dynamically identified. This design slightly **increases runtime overhead** (see Overhead), but significantly **reduces overall memory usage**.

Overhead





Platforms: TH1520 RISC-V CPU (4 harts) from Lichee PI 4A
i7-8650U x86 CPU (4 harts) from Dell Latitude

We analyze three distinct cost scenarios:

- **Normal**: The cost of executing the standard *libc* function (e.g., `getpid()`) without interception. These measurements serve as the baseline reference for the overhead charts on the left.
- **Intercepted Cost (User Mode)**: The cost of executing the intercepted *libc* function when the corresponding syscall is bypassed. Since the Linux kernel isn't called to execute the syscall, the overhead is negative in the chart on the left.
- **Intercepted Cost (Kernel Mode)**: The cost of executing the intercepted *libc* function with a call to the corresponding syscall. In this case, the chart shows a positive overhead due to the complexity of patching *libc* and the added indirection to reach the Linux kernel.

Basic examples of patch types

```
f7de0:   addi   sp, sp, -48
f7de2:   sd     ra, 0(sp)
f7de4:   auipc  ra, <offset>
f7de8:   jalr   ra, <offset>(ra)
f7dec:   ld     ra, 0(sp)
f7dee:   addi   sp, sp, 48
```

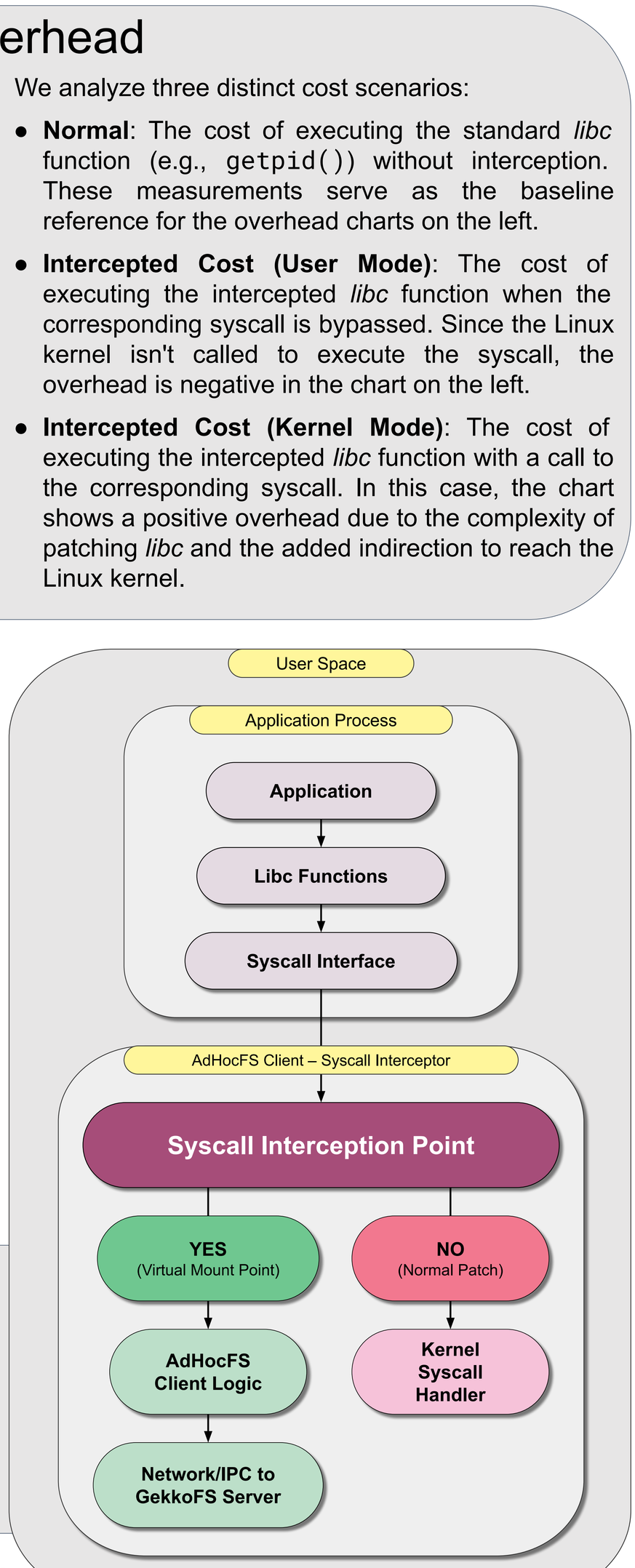
```
e5fa4:   addi   sp, sp, -48
e5fa6:   sd     ra, 8(sp)
e5fa8:   jal    ra, <GW addr offset>
e5fac:   ld     ra, 8(sp)
e5fae:   addi   sp, sp, 48
```

- Case when a register gets set after the ecall:

```
a13ea:   jal    a7, <GW addr offset>
```

- Case when no register gets set after the ecall:

```
a13ea:   jal    a7, <GW addr offset>
a13ec:   li     a7, <syscall number>
```



Use case: GekkoFS

In AdHocFS, the library **reduces** the number of required hooks, improving **compatibility** with *libc* interception. Separate hooks for different `stat()` or `open()` variants are unnecessary. It also enables interception of applications that bypass libc, which would otherwise be difficult to handle and debug.

This project is promoted by the Ministry for Digital Transformation and the Civil Service, within the framework of the Recovery, Transformation, and Resilience Plan - Funded by the European Union - NextGenerationEU (reference REGAGE22e00058408992). This work has been partially financed by the European Commission (EU-HORIZON VITAMIN-V GA 101093062). The work done in this paper was achieved with the support of RISC-V International in their mentorship program, with the participation of Ramon Nou (BSC-CNS) as mentor and Petar Andrić as Mentee.

Financiado por la Unión Europea NextGenerationEU
GOBIERNO DE ESPAÑA
MINISTERIO PARA LA TRANSFORMACIÓN DIGITAL Y LA FUNCIÓN PÚBLICA
SECRETARÍA DE ESTADO DE TELECOMUNICACIONES E INFRAESTRUCTURAS DIGITALES
Plan de Recuperación, Transformación y Resiliencia

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Este proyecto se impulsa por el Ministerio para la Transformación Digital y de la Función Pública, en el marco del Plan de Recuperación, Transformación y Resiliencia – Financiado por la Unión Europea – NextGenerationEU