SBI extension:
Supervisor Software Events

Clément Léger <cleger@rivosinc.com>
SSE: Why?

- Allows injecting events to [H]S-mode from higher privilege mode
  - Similar to existing ARM SDEI (Software Delegated Exception Interface)
- Need to inject high priority (non maskable) events
  - Reliability, Availability and Serviceability errors
  - PMU overflow IRQ for performance events
  - Paravirtualized Asynchronous Page Fault
- SBI Specification submitted by Himanshu Chauhan [1]
  - [https://lists.riscv.org/g/tech-prs/message/515](https://lists.riscv.org/g/tech-prs/message/515)
SSE: How?

- [H]S-mode can register event handler to be called upon specific events
  - Events RAS, PMU, etc
  - Events can be local (per-hart) or global
  - [H]S-mode allocates memory space to store interrupted/entry context

- SBI then « injects » the SSE events upon specific hardware events
  - Divert execution flow by replacing interrupted context with SSE handler entry context
  - Then resume execution at specified entry context “pc” content.

- Upon SSE handle completion, handlers does a SBI SSE complete ecall.
  - Previously interrupted context is restored and resumed
SSE : Supervisor OS PoV

- Almost orthogonal to OS normal operations
  - ie: no impact at all on normal operations
  - Does not use regular interrupt path
- Avoid waiting too long due to irqs off section in kernel
  - SSE can interrupt the supervisor mode at any place
  - Can be seen as an NMI
- Allows handling of RAS fault as fast as possible to avoid fault propagation
  - Critical to avoid taking “serious” action at a later time
- Does not use much OS specific resources
  - Except some memory for SSE event contexts
SSE : Priority

- Events encoding uses 32 bits integer
  - Type (global/local)
  - Priority
  - Platform specific
- IDs encoded the default priority (0 is highest priority)

```
#define SBI_SSE_EVENT_LOCAL_RAS         0x00000000
#define SBI_SSE_EVENT_GLOBAL_RAS        0x00008000
#define SBI_SSE_EVENT_LOCALASYNC_PF     0x00010000
#define SBI_SSE_EVENT_LOCAL_PF          0x00010001
#define SBI_SSE_EVENT_LOCAL_DEBUG       0xffffffff
#define SBI_SSE_EVENT_GLOBAL_DEBUG      0xffffffff
#define SBI_SSE_EVENT_GLOBAL            (1 << 15)
#define SBI_SSE_EVENT_PLATFORM          (1 << 14)
```
```c
struct sse_entry_state {
    /** Entry program counter */
    unsigned long pc;
    /** ra register state */
    unsigned long ra;
    /** sp register state */
    unsigned long sp;
    ...  
    /** t5 register state */
    unsigned long t5;
    /** t6 register state */
    unsigned long t6;
};

struct sse_interrupted_state {
    /** Interrupted program counter */
    unsigned long pc;
    /** ra register state */
    unsigned long ra;
    /** sp register state */
    unsigned long sp;
    ...  
    /** t5 register state */
    unsigned long t5;
    /** t6 register state */
    unsigned long t6;
    /** Exception mode */
    unsigned long exc_mode;
};

struct abi_sse_handler_ctx {
    struct sse_entry_state entry;
    struct sse_entry_state entry;
    struct sse_interrupted_state interrupted;
};
```
<table>
<thead>
<tr>
<th>SSE</th>
<th>Pseudo-NMI [1]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pros:</strong></td>
<td><strong>Pros:</strong></td>
</tr>
<tr>
<td>○ True NMI-like events that can interrupt the kernel at any time</td>
<td>○ Does no require any SBI support</td>
</tr>
<tr>
<td>○ Allows nesting of events</td>
<td>○ Only a few part of interrupts handling modified</td>
</tr>
<tr>
<td>○ Faster delivery than standard IRQ path (TBC)</td>
<td>○ Simpler than SSE</td>
</tr>
<tr>
<td>○ Minimal modification of existing codebase</td>
<td></td>
</tr>
<tr>
<td>○ Easily extensible (pure software)</td>
<td></td>
</tr>
<tr>
<td><strong>Cons:</strong></td>
<td><strong>Cons:</strong></td>
</tr>
<tr>
<td>○ Requires a SSE compatible SBI</td>
<td>○ Critical sections (exception handling) still uninterruptible</td>
</tr>
<tr>
<td>○ Additional work to retrieve current() task struct</td>
<td>○ Does not support nesting nor priority</td>
</tr>
<tr>
<td></td>
<td>○ Performance loss for existing use cases (-1.90%)</td>
</tr>
</tbody>
</table>

[1] https://lore.kernel.org/linux-riscv/20231023082911.23242-1-luxu.kernel@bytedance.com/
With SSE, kernel can be interrupted anywhere, including during exception handling and we need the `current()` task for accounting.

On RISC-V, `CSR_SSCRATCH` is used to store `current()` task

- But also used as the temporary register to make room for temporary stack computation (context saving) → !/\ Content is unreliable !/\

Need a way to know exactly where is the `task_struct` is located based on code addresses

- Using address comparison (ie check kernel one) is unreliable
- Using known labels seems a bit fixed
- One way is to annotate source code with « fixup-like » data
SSE: Where is my task struct? (2)

- Annotations are actually stored in a separate section and used to locate current task based on pc register.

```c
#define __SSE_TASK_LOC(s_loc, u_loc) .pushsection __task_loc,"a";
RISCV_PTR 99f;
.byte TASK_LOC(s_loc, u_loc);
.popsection;
99:

SYM_CODE_START(handle_exception)
__SSE_TASK_LOC(IN_TP, IN_SSCRATCH)
csrrw tp, CSR_SCRATCH, tp
__SSE_TASK_LOC(IN_SSCRATCH, IN_TP)
bnez tp, _save_context

_restore_kernel_tpsp:
csrr tp, CSR_SCRATCH
__SSE_TASK_LOC(IN_TP, IN_TP)
REG_S sp, TASK_TI_KERNEL_SP(tp)
...```
SSE : Specification problem (?)

- Some concerns with the specification were raised
  - Creates a bond between the SBI spec and the Risc-v unprivileged ISA
  - CHERI spec in particular seems to have been reported

- Possibility to save less general purpose registers
  - But at some point we need a safe state to enter « nested » execution in kernel
  - Other registers/architectural state can be saved if needed
Numbers

- Measured within spike
  - PoC with PMU overflow IRQs in M-mode triggering local PMU SSE events

- From interrupt being set to final IRQ/SSE event handler:
  - Normal IRQ handling: ~1590 instructions
  - SSE event handling: ~790 instructions

- With SSE, no jitter due to interrupts being disabled.

- Next step: gather more precise numbers using hardware platforms
  - Measure privilege level switch impact on caches, etc.
Status

- Kernel (~1000 SLOC)
  - https://github.com/rivosinc/linux/tree/dev/cleger/sse
- OpenSBI (~1000 SLOC)
  - https://github.com/rivosinc/opensbi/tree/dev/cleger/sse
- SBI extension specification is still in review, could be ratified ~Q3 2024 for SBI v3.0
  - https://lists.riscv.org/g/tech-prs/message/515