Integration of the LTTng user-space tracer with the RSEQ system call
What are Restartable Sequences?

How is RSEQ useful for LTTng-UST?

Restartable Sequences upstreaming status

Missing Pieces

Solutions

Ongoing effort
What are Restartable Sequences (RSEQ)?

- Linux kernel system call registering a Thread-Local Storage area allowing user-space to perform updates on per-cpu data efficiently,

- Achieve critical section atomicity with respect to scheduler by aborting critical sections on preemption and signal delivery rather than disabling preemption.
RSEQ Structure Members

Thread-Local Storage __rseq_abi:

```c
struct rseq {
    int32_t cpu_id;
    struct rseq_cs *rseq_cs;
    [...]  
};
```

```c
struct rseq_cs {
    void *start_ip;
    void *post_commit_ip;
    void *abort_ip;
    [...]  
};
```
How is RSEQ useful for LTTng-UST?

- LTTng-UST implements per-CPU ring buffers:
  - Eliminate false-sharing,
  - Reserve and commit counters scheme,
- RSEQ accelerates reading the current cpu number,
- RSEQ replaces atomic operations for reserve and commit on per-CPU data by faster non-atomic loads and stores.
Other uses of RSEQ

- Per-CPU pool memory allocation,
- Per-CPU ring buffer,
- Per-CPU statistics accounting,
- Per-CPU RCU grace period tracking,
- User-space PMU counters read from user-space on big/LITTLE ARM64.
RSEQ Benchmarks: Get Current CPU Number

Reading the current CPU number (arm32)

Reading the current CPU number (x86-64)
RSEQ Benchmarks: Statistics Counter

Per-CPU statistics counter increment (arm32)

- getcpu+atomic: 344.0
- rseq: 31.4

Per-CPU counter increment (x86-64)

- getcpu+atomic: 15.3
- rseq: 2.0
RSEQ Benchmarks: LTTng-UST Ring Buffer

LTTng-UST write event into trace per-cpu buffer (arm32)

- getcpu+atomic: 2502.0 ns/op
- rseq: 2250.0 ns/op

LTTng-UST write event into trace per-cpu buffer (x86-64)

- getcpu+atomic: 117.4 ns/op
- rseq: 98.0 ns/op
Restartable Sequences Linux Integration

- Linux 4.18:
  - RSEQ system call merged,
  - RSEQ wired up for x86 32/64, powerpc 32/64, arm 32, mips 32/64,

- Linux 4.19:
  - RSEQ wired up for arm 64, s390 32/64,
Restartable Sequences glibc Integration

- Submitted for glibc 2.31,
- Includes:
  - RSEQ TLS registration,
  - Use of RSEQ to accelerate sched_getcpu(3).
Perform update of per-CPU data from **other cpus**: 
- The case of lttng-consumerd live and switch timers.
- Cannot be done reliably with CPU affinity due to CPU hotplug.
Missing Pieces

- **Early/late use** in libc initialization and thread lifetime, where the RSEQ TLS is not yet registered:
  - Within libc and dynamic linker initialization,
  - Preloaded libraries constructors,
  - Audit libraries,
  - IFUNC resolvers,
  - Signal handlers,
- Guarantee **progress** under debugger single-stepping for current debuggers.
Solutions

- New system call: `do_on_cpu()` (previously submitted as `cpu_opv()`)
- eBPF bytecode interpreter within the kernel,
  - Running either in IPI handler or thread context with preemption disabled on the target CPU,
  - Specialized to load and store exclusively from/to user-space memory.
- Use this system call as fallback when RSEQ is not registered for the current thread or aborts due to preemption.
1) Upstreaming RSEQ TLS registration within glibc,
   - Submitted, being reviewed by maintainers,
2) Justify further RSEQ-related kernel code by showing RSEQ adoption by the community,
3) Consider upstreaming do_on_cpu() eBPF bytecode interpreter system call into Linux,
4) Complete integration of RSEQ+do_on_cpu() within LTTng-UST.