NAME

rseq − Restartable sequences and cpu number cache

SYNOPSIS

#include <linux/rseq.h>

int rseq(struct rseq * *rseq***, uint32_t** *rseq_len***, int** *flags***, uint32_t** *sig***);**

DESCRIPTION

A restartable sequence is a sequence of instructions guaranteed to be executed atomically with respect to other threads and signal handlers on the current CPU. If its execution does not complete atomically, the kernel changes the execution flow by jumping to an abort handler defined by user-space for that restartable sequence.

Using restartable sequences requires to register a **__rseq_abi** thread-local storage data structure (struct rseq) through the **rseq**() system call. Only one **__rseq_abi** can be registered per thread, so user-space libraries and applications must follow a user-space ABI defining how to share this resource. The ABI defining how to share this resource between applications and libraries is defined by the C library.

The **__rseq_abi** contains a *rseq_cs* field which points to the currently executing critical section. For each thread, a single rseq critical section can run at any given point. Each critical section need to be implemented in assembly.

The **rseq**() ABI accelerates user-space operations on per-cpu data by defining a shared data structure ABI between each user-space thread and the kernel.

It allows user-space to perform update operations on per-cpu data without requiring heavy-weight atomic operations.

The term CPU used in this documentation refers to a hardware execution context. For instance, each CPU number returned by **sched_getcpu**() is a CPU. The current CPU means to the CPU on which the registered thread is running.

Restartable sequences are atomic with respect to preemption (making it atomic with respect to other threads running on the same CPU), as well as signal delivery (user-space execution contexts nested over the same thread). They either complete atomically with respect to preemption on the current CPU and signal delivery, or they are aborted.

Restartable sequences are suited for update operations on per-cpu data.

Restartable sequences can be used on data structures shared between threads within a process, and on data structures shared between threads across different processes.

Some examples of operations that can be accelerated or improved by this ABI:

- Memory allocator per-cpu free-lists,
- Querying the current CPU number,
- Incrementing per-CPU counters,
- Modifying data protected by per-CPU spinlocks,
- Inserting/removing elements in per-CPU linked-lists,
- Writing/reading per-CPU ring buffers content.

- Accurately reading performance monitoring unit counters with respect to thread migration.

Restartable sequences must not perform system calls. Doing so may result in termination of the process by a segmentation fault.

The *rseq* argument is a pointer to the thread-local rseq structure to be shared between kernel and user-space.

The layout of **struct rseq** is as follows:

Structure alignment

This structure is aligned on 32-byte boundary.

Structure size

This structure is fixed-size (32 bytes). Its size is passed as parameter to the rseq system call.

```
struct rseq {
    __u32 cpu_id_start;
    __u32 cpu_id;
    union {
        /* Edited out for conciseness. [...] */
    } rseq_cs;
    __u32 flags;
} __attribute__((aligned(32)));
```

Fields

cpu_id_start

Optimistic cache of the CPU number on which the registered thread is running. Its value is guaranteed to always be a possible CPU number, even when rseq is not registered. Its value should always be confirmed by reading the cpu_id field before user-space performs any side-effect (e.g. storing to memory).

This field is an optimistic cache in the sense that it is always guaranteed to hold a valid CPU number in the range [0 .. nr_possible_cpus - 1]. It can therefore be loaded by user-space and used as an offset in per-cpu data structures without having to check whether its value is within the valid bounds compared to the number of possible CPUs in the system.

Initialized by user-space to a possible CPU number (e.g., 0), updated by the kernel for threads registered with rseq.

For user-space applications executed on a kernel without rseq support, the cpu_id_start field stays initialized at 0, which is indeed a valid CPU number. It is therefore valid to use it as an offset in per-cpu data structures, and only validate whether it's actually the current CPU number by comparing it with the cpu_id field within the rseq critical section. If the kernel does not provide rseq support, that cpu_id field stays initialized at -1, so the comparison always fails, as intended.

It is up to user-space to implement a fall-back mechanism for scenarios where rseq is not available.

cpu_id

Cache of the CPU number on which the registered thread is running. Initialized by user-space to -1, updated by the kernel for threads registered with rseq.

rseq_cs
The rseq_cs field is a pointer to a struct rseq_cs. Is is NULL when no rseq assembly block critical section is active for the registered thread. Setting it to point to a critical section descriptor (struct rseq_cs) marks the beginning of the critical section.

Initialized by user-space to NULL.

Updated by user-space, which sets the address of the currently active rseq_cs at the beginning of assembly instruction sequence block, and set to NULL by the kernel when it restarts an assembly instruction sequence block, as well as when the kernel detects that it is preempting or delivering a signal outside of the range targeted by the rseq_cs. Also needs to be set to NULL by user-space before reclaiming memory that contains the targeted struct rseq_cs.

Read and set by the kernel.

flags
Flags indicating the restart behavior for the registered thread. This is mainly used for debugging purposes. Can be a combination of:

- RSEQ_CS_FLAG_NO_RESTART_ON_PREEMPT: Inhibit instruction sequence block restart on preemption for this thread.
- RSEQ_CS_FLAG_NO_RESTART_ON_SIGNAL: Inhibit instruction sequence block restart on signal delivery for this thread.
- RSEQ_CS_FLAG_NO_RESTART_ON_MIGRATE: Inhibit instruction sequence block restart on migration for this thread.

Initialized by user-space, used by the kernel.

The layout of **struct rseq_cs** version 0 is as follows:

Structure alignment
This structure is aligned on 32-byte boundary.

Structure size
This structure has a fixed size of 32 bytes.

```
struct rseq_cs {
    __u32   version;
    __u32   flags;
    __u64   start_ip;
    __u64   post_commit_offset;
    __u64   abort_ip;
} __attribute__((aligned(32)));
```

Fields

version
Version of this structure. Should be initialized to 0.

flags
Flags indicating the restart behavior of this structure. Can be a combination of:

- RSEQ_CS_FLAG_NO_RESTART_ON_PREEMPT: Inhibit instruction sequence block restart on preemption for this critical section.
- RSEQ_CS_FLAG_NO_RESTART_ON_SIGNAL: Inhibit instruction sequence block restart on signal delivery for this critical section.

- RSEQ_CS_FLAG_NO_RESTART_ON_MIGRATE: Inhibit instruction sequence block restart on migration for this critical section.

start_ip
Instruction pointer address of the first instruction of the sequence of consecutive assembly instructions.

post_commit_offset
Offset (from start_ip address) of the address after the last instruction of the sequence of consecutive assembly instructions.

abort_ip
Instruction pointer address where to move the execution flow in case of abort of the sequence of consecutive assembly instructions.

The *rseq_len* argument is the size of the *struct rseq* to register.

The *flags* argument is 0 for registration, and *RSEQ_FLAG_UNREGISTER* for unregistration.

The *sig* argument is the 32-bit signature to be expected before the abort handler code.

A single library per process should keep the rseq structure in a thread-local storage variable. The *cpu_id* field should be initialized to -1, and the *cpu_id_start* field should be initialized to a possible CPU value (typically 0).

Each thread is responsible for registering and unregistering its rseq structure. No more than one rseq structure address can be registered per thread at a given time.

Reclaim of rseq object's memory must only be done after either an explicit rseq unregistration is performed or after the thread exits.

In a typical usage scenario, the thread registering the rseq structure will be performing loads and stores from/to that structure. It is however also allowed to read that structure from other threads. The rseq field updates performed by the kernel provide relaxed atomicity semantics (atomic store, without memory ordering), which guarantee that other threads performing relaxed atomic reads (atomic load, without memory ordering) of the cpu number cache will always observe a consistent value.

RETURN VALUE

A return value of 0 indicates success. On error, −1 is returned, and *errno* is set appropriately.

ERRORS

EINVAL
Either *flags* contains an invalid value, or *rseq* contains an address which is not appropriately aligned, or *rseq_len* contains an incorrect size.

ENOSYS
The **rseq**() system call is not implemented by this kernel.

EFAULT
rseq is an invalid address.

EBUSY
Restartable sequence is already registered for this thread.

EPERM
The *sig* argument on unregistration does not match the signature received on registration.

VERSIONS

The **rseq**() system call was added in Linux 4.18.

CONFORMING TO

rseq() is Linux-specific.

SEE ALSO

sched_getcpu(3), **membarrier**(2)