sd_event_wait

- Description
- State Machine
- Return Value
  - Errors
- History
- See Also

systemd

sd_event_wait

3

sd_event_wait

sd_event_prepare

sd_event_dispatch

sd_event_get_state

sd_event_get_iteration

SD_EVENT_INITIAL

SD_EVENT_PREPARING

SD_EVENT_ARMED

SD_EVENT_PENDING

SD_EVENT_RUNNING

SD_EVENT_EXITING

SD_EVENT_FINISHED

Low-level event loop operations

#include <systemd/sd-event.h>

enum

{

SD_EVENT_INITIAL

,

SD_EVENT_PREPARING

,

SD_EVENT_ARMED

,

SD_EVENT_PENDING

,

SD_EVENT_RUNNING

,

SD_EVENT_EXITING

,

SD_EVENT_FINISHED

, };

int

sd_event_prepare

sd_event *

event

int

sd_event_wait

sd_event *

event

uint64_t

usec

int

sd_event_dispatch

sd_event *

event

int

sd_event_get_state

sd_event *

event

int

sd_event_get_iteration

sd_event *

event

uint64_t *

ret

Description

The low-level `sd_event_prepare()`, `sd_event_wait()` and `sd_event_dispatch()` functions may be used to execute specific phases of an event loop. See `sd_event_run(3)` and `sd_event_loop(3)` for higher-level functions that execute individual but complete iterations of an event loop or run it continuously.

`sd_event_prepare()` checks for pending events and arms necessary timers. If any events are ready to be processed ("pending"), it returns a positive, non-zero value, and the caller should process

these events with `sd_event_dispatch()`.

`sd_event_dispatch()` dispatches the highest priority event source that has a pending event. On success, `sd_event_dispatch()` returns either zero, which indicates that no further event sources may be dispatched and exiting of the event loop was requested via `sd_event_exit(3)`; or a positive non-zero value, which means that an event source was dispatched and the loop returned to its initial state, and the caller should initiate the next event loop iteration by invoking `sd_event_prepare()` again.

In case `sd_event_prepare()` returned zero, `sd_event_wait()` should be called to wait for further events or a timeout. If any events are ready to be processed, it returns a positive, non-zero value, and the events should be dispatched with `sd_event_dispatch()`. Otherwise, the event loop returned to its initial state and the next event loop iteration should be initiated by invoking `sd_event_prepare()` again.

`sd_event_get_state()` may be used to determine the state the event loop is currently in. It returns one of the states described below.

`sd_event_get_iteration()` may be used to determine the current iteration of the event loop. It returns an unsigned 64-bit integer containing a counter that increases monotonically with each iteration of the event loop, starting with 0. The counter is increased at the time of the `sd_event_prepare()` invocation.

All five functions take, as the first argument, the event loop object `event` that has been created with `sd_event_new()`. The timeout for `sd_event_wait()` is specified in `usec` in microseconds. `(uint64_t) -1` may be used to specify an infinite timeout.

State Machine

The event loop knows the following states, that may be queried with `sd_event_get_state()`.

`SD_EVENT_INITIAL`
The initial state the event loop is in, before each event loop iteration. Use `sd_event_prepare()` to transition the event loop into the `SD_EVENT_ARMED` or `SD_EVENT_PENDING` states.

`SD_EVENT_PREPARING`
An event source is currently being prepared, i.e. the preparation handler is currently being executed, as set with `sd_event_source_set_prepare(3)`. This state is only seen in the event source preparation handler that is invoked from the `sd_event_prepare()` call and is immediately followed by `SD_EVENT_ARMED` or `SD_EVENT_PENDING`.

`SD_EVENT_ARMED`
`sd_event_prepare()` has been called and no event sources were ready to be dispatched. Use `sd_event_wait()` to wait for new events, and transition into `SD_EVENT_PENDING` or back into

`SD_EVENT_INITIAL`.

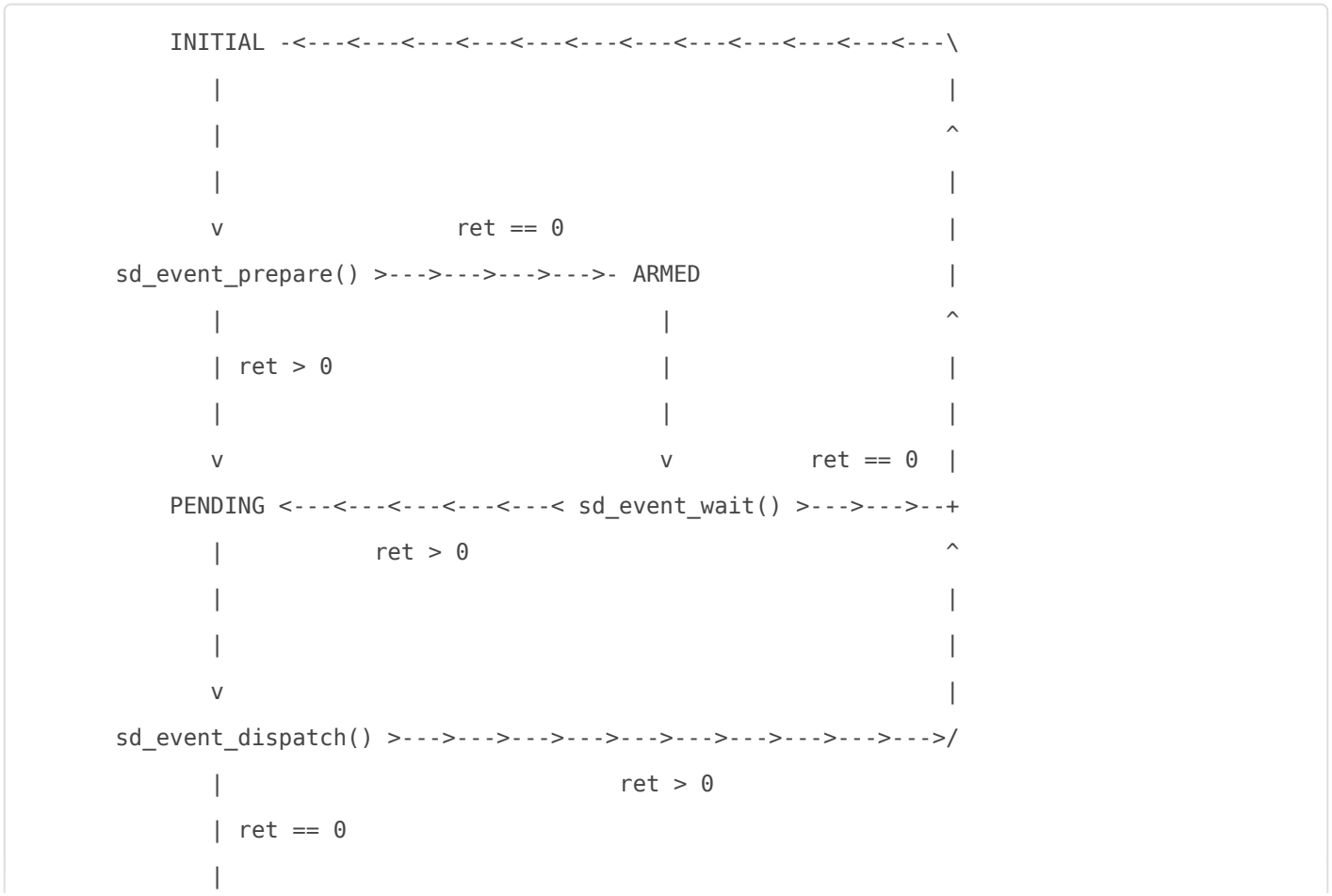
`SD_EVENT_PENDING`
`sd_event_prepare()` or `sd_event_wait()` have been called and there were event sources with events pending. Use `sd_event_dispatch()` to dispatch the highest priority event source and transition back to `SD_EVENT_INITIAL`, or `SD_EVENT_FINISHED`.

`SD_EVENT_RUNNING`
A regular event source is currently being dispatched. This state is only seen in the event source handler that is invoked from the `sd_event_dispatch()` call, and is immediately followed by `SD_EVENT_INITIAL` or `SD_EVENT_FINISHED` as soon the event source handler returns. Note that during dispatching of exit event sources the `SD_EVENT_EXITING` state is seen instead.

`SD_EVENT_EXITING`
Similar to `SD_EVENT_RUNNING` but is the state in effect while dispatching exit event sources. It is followed by `SD_EVENT_INITIAL` or `SD_EVENT_FINISHED` as soon as the event handler returns.

`SD_EVENT_FINISHED`
The event loop has exited. All exit event sources have run. If the event loop is in this state it serves no purpose anymore, and should be freed.

A simplified flow chart of the states and the calls to transition between them is shown below. Note that `SD_EVENT_PREPARING`, `SD_EVENT_RUNNING` and `SD_EVENT_EXITING` are not shown here.



```
            v
        FINISHED
```

Return Value

On success, these functions return 0 or a positive integer. On failure, they return a negative errno-style error code. In case of `sd_event_prepare()` and `sd_event_wait()`, a positive, non-zero return code indicates that events are ready to be processed and zero indicates that no events are ready. In case of `sd_event_dispatch()`, a positive, non-zero return code indicates that the event loop returned to its initial state and zero indicates the event loop has exited. `sd_event_get_state()` returns a positive or zero state on success.

Errors

Returned errors may indicate the following problems:

`-EINVAL`
The `event` parameter is invalid or `NULL`.

`-EBUSY`
The event loop object is not in the right state.

`-ESTALE`
The event loop is already terminated.

`-ECHILD`
The event loop has been created in a different process, library or module instance.

Other errors are possible, too.

History

`sd_event_prepare()`, `sd_event_wait()`, `sd_event_dispatch()`, and `sd_event_get_state()` were added in version 221.

`sd_event_get_iteration()` was added in version 231.

See Also

`systemd(1)`, `sd_event_new(3)`, `sd_event_add_io(3)`, `sd_event_add_time(3)`, `sd_event_add_signal(3)`, `sd_event_add_child(3)`, `sd_event_add_inotify(3)`, `sd_event_add_defer(3)`, `sd_event_run(3)`, `sd_event_get_fd(3)`, `sd_event_source_set_prepare(3)`

□□□ #2
□ □□□ □□□ 2026-10-31 14:56:41 UTC
□ □□□ □□□ 2026-10-31 14:56:42 UTC