
# (Event) Dispatcher Library

# Agenda

DPDK
DATA PLANE DEVELOPMENT KIT

- History
- Context
- Purpose
- Example
- Components
- (API Walk-through)
- (Performance)
- Future Development

# History

- Proprietary variants (ODP and DPDK)
- First RFC early 2021 (as Eventdev library extension)
- Work restarted mid-2023
- PATCH v3 early September 2023 (now as a separate library)

# Context: Eventdev

- Each EAL thread (and core) usually owns exactly one Eventdev port
- Events destined for different software modules (protocol layers) usually comes multiplexed over the same Eventdev port
- Some piece of logic must demultiplex this stream of events

# Dispatcher Purpose

- Decouple application modules in Eventdev-based applications
  - Replace the conditional logic deciding which-event-goes-where
  - Optional
- Particularly useful when
  - Modules are developed by different organizations
  - Modules are reused across different applications
  - Applications consist of many modules
- *Conceptually* similar an event loop library (e.g., libevent)
  - select() versus rte_event_dequeue_burst()
- Provide a convenient for DPDK application to deploy as DPDK services

# Pre-Dispatcher Example

```
for (;;) {
        struct rte_event events[MAX_BURST];
        unsigned int n;

        n = rte_event_dequeue_burst(dev_id, port_id, events, MAX_BURST, 0);

        for (i = 0; i < n; i++) {
                const struct rte_event *event = &events[i];

                switch (event->queue_id) {
                case MODULE_A_QUEUE_ID:
                        module_a_process(event);
                        break;
                case MODULE_B_STAGE_0_QUEUE_ID:
                        module_b_process_stage_0(event);
                        break;
                case MODULE_B_STAGE_1_QUEUE_ID:
                        module_b_process_stage_1(event);
                        break;
                }
        }
}
```

# Post-Dispatcher Example

DPDK
DATA PLANE DEVELOPMENT KIT

```
static bool
module_a_match(const struct rte_event *event, void *cb_data)
{
        return event->queue_id == MODULE_A_QUEUE_ID;
}

static void
module_a_process_events(uint8_t event_dev_id, uint8_t event_port_id,
                        const struct rte_event *events,
                        uint16_t num, void *cb_data)
{
        uint16_t i;

        for (i = 0; i < num; i++)
                module_a_process_event(&events[i]);
}

/* In the module's initialization code */
rte_dispatcher_register(DISPATCHER_ID, module_a_match, NULL,
                        module_a_process_events, module_a_data);
```

# Components

- Library (2 files, 1200 LOC)
- Programming guide
- API documentation
- Test suite

# API

DPDK
DATA PLANE DEVELOPMENT KIT

## Data Structures

struct **rte_dispatcher_stats**

## Typedefs

typedef bool(* **rte_dispatcher_match_t**) (const struct **rte_event** *event, void *cb_data)

typedef void(* **rte_dispatcher_process_t**) (uint8_t event_dev_id, uint8_t event_port_id, struct **rte_event** *events, uint16_t num, void *cb_data)

typedef void(* **rte_dispatcher_finalize_t**) (uint8_t event_dev_id, uint8_t event_port_id, void *cb_data)

## Functions

__rte_experimental int **rte_dispatcher_create** (uint8_t id, uint8_t event_dev_id)

__rte_experimental int **rte_dispatcher_free** (uint8_t id)

__rte_experimental int **rte_dispatcher_service_id_get** (uint8_t id, uint32_t *service_id)

__rte_experimental int **rte_dispatcher_bind_port_to_lcore** (uint8_t id, uint8_t event_port_id, uint16_t batch_size, uint64_t timeout, unsigned int lcore_id)

__rte_experimental int **rte_dispatcher_unbind_port_from_lcore** (uint8_t id, uint8_t event_port_id, unsigned int lcore_id)

__rte_experimental int **rte_dispatcher_register** (uint8_t id, **rte_dispatcher_match_t** match_fun, void *match_cb_data, **rte_dispatcher_process_t** process_fun, void *process_cb_data)

__rte_experimental int **rte_dispatcher_unregister** (uint8_t id, int handler_id)

__rte_experimental int **rte_dispatcher_finalize_register** (uint8_t id, **rte_dispatcher_finalize_t** finalize_fun, void *finalize_data)

__rte_experimental int **rte_dispatcher_finalize_unregister** (uint8_t id, int reg_id)

__rte_experimental int **rte_dispatcher_start** (uint8_t id)

__rte_experimental int **rte_dispatcher_stop** (uint8_t id)

__rte_experimental int **rte_dispatcher_stats_get** (uint8_t id, struct **rte_dispatcher_stats** *stats)

__rte_experimental int **rte_dispatcher_stats_reset** (uint8_t id)

# Performance

- Adds ~10 cc/event
- Search for event handler is O(N)
- Handler callbacks are reshuffled (in run-time, per lcore)
  - Often-matching handlers “bubble” up, matching earlier
- In a particular dequeue burst, all events destined for a particular handler are delivered as one callback call
  - Event ordering between events destined different handlers are *not* maintained
  - Improves cache locality (compared to the naïve example)

# Future Development

- Support for matching-function-less handlers
  - I.e., Eventdev queue id matching, or some template event+bitmask
- Add flags parameter to create (?)
  - E.g., to ask for strict cross-handler ordering to be maintained
- Support for run-time reconfiguration (?)
- Support for non-Eventdev event sources (?)
- eBPF for matching (??)
- Tracing support
- Telemetry suport


# Questions?

Mattias Rönnblom
<mattias.ronnblom@ericsson.com>