The Little Extras in DPDK

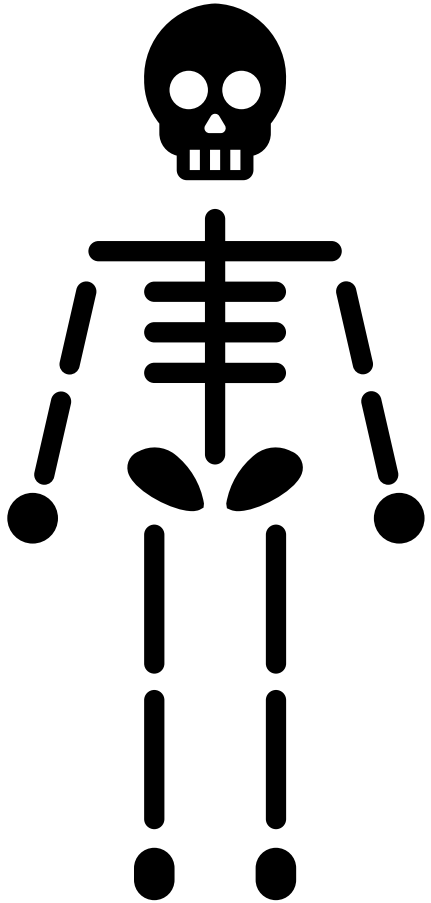
The "other bits" of DPDK I use when writing apps

Bruce Richardson

Disclaimer ☺

- I am not a DPDK end-application developer
  - Nor do I play one on TV! ☺
- I work mostly on the internals of DPDK and DPDK drivers
- However, I have occasionally been known to try to write an app using DPDK
- This talk covers a few tips from my experience of doing so!

Start Point



- Skeleton Example
- Minimal app for DPDK
- Does EAL init and basic forwarding
- Useful port init function

Files

main

Go to file T

- packet_ordering
- pipeline
- ptpclient
- qos_meter
- qos_sched
- rxtx_callbacks
- server_node_efd
- service_cores
- skeleton
  - Makefile
  - basicfwd.c
  - meson.build
- timer
- vdpa
- vhost
- vhost_blk
- vhost_crypto
- vm_power_manager
- vmdq
- vmdq_dcb
- meson.build
- kernel
- lib
- license

Code | Blame | 221 lines (181 loc) · 5.82 KB | Raw

```
#define BURST_SIZE 32

/* basicfwd.c: Basic DPDK skeleton forwarding example. */

/*
 * Initializes a given port using global settings and with the RX buffers
 * coming from the mbuf_pool passed as a parameter.
 */

/* Main functional part of port initialization. 8< */
static inline int
port_init(uint16_t port, struct rte_mempool *mbuf_pool)
{
        struct rte_eth_conf port_conf;
        const uint16_t rx_rings = 1, tx_rings = 1;
        uint16_t nb_rxd = RX_RING_SIZE;
        uint16_t nb_txd = TX_RING_SIZE;
        int retval;
        uint16_t q;
        struct rte_eth_dev_info dev_info;
        struct rte_eth_txconf txconf;

        if (!rte_eth_dev_is_valid_port(port))
                return -1;

        memset(&port_conf, 0, sizeof(struct rte_eth_conf));

        retval = rte_eth_dev_info_get(port, &dev_info);
        if (retval != 0) {
                printf("Error during getting device (port %u) info: %s\n",
                                port, strerror(-retval));
                return retval;
        }

        if (dev_info.tx_offload_capa & RTE_ETH_TX_OFFLOAD_MBUF_FAST_FREE)
                port_conf.txmode.offloads |=
                        RTE_ETH_TX_OFFLOAD_MBUF_FAST_FREE;

        /* Configure the Ethernet device. */
        retval = rte_eth_dev_configure(port, rx_rings, tx_rings, &port_conf);
        if (retval != 0)
                return retval;
```

```
bruce@host:~$ find ~/.local/ -name libdpdk.pc
/home/bruce/.local/lib/x86_64-linux-gnu/pkgconfig/libdpdk.pc

bruce@host:~$ export PKG_CONFIG_PATH=/home/bruce/.local/lib/x86_64-linux-gnu/pkgconfig/

bruce@host:~$ export LD_LIBRARY_PATH=/home/bruce/.local/lib/x86_64-linux-gnu/

bruce@host:~$ mkdir dpdk_summit_2026

bruce@host:~$ cd dpdk_summit_2026

bruce@host:~/dpdk_summit_2026$ cp $HOME/.local/share/dpdk/examples/skeleton/* .

bruce@host:~/dpdk_summit_2026$ make
cc -O3 -I/home/bruce/.local/include -include rte_config.h -march=native -mrtm -I/usr/include/x86_64-linux-gnu -I/usr/include/libnl3 -I/usr/include/dbus-1.0 -
I/usr/lib/x86_64-linux-gnu/dbus-1.0/include  -DALLOW_EXPERIMENTAL_API basicfwd.c -o build/basicfwd-shared  -Wl,--as-needed -L/home/bruce/.local/lib/x86_64-li
nux-gnu -lrte_node -lrte_graph -lrte_pipeline -lrte_table -lrte_pdump -lrte_port -lrte_fib -lrte_pdcp -lrte_ipsec -lrte_vhost -lrte_stack -lrte_security -lrt
e_sched -lrte_reorder -lrte_rib -lrte_mldev -lrte_regexdev -lrte_rawdev -lrte_power -lrte_pcapng -lrte_member -lrte_lpm -lrte_latencystats -lrte_jobstats -lr
te_ip_frag -lrte_gso -lrte_gro -lrte_gpudev -lrte_dispatcher -lrte_eventdev -lrte_efd -lrte_dmadev -lrte_distributor -lrte_cryptodev -lrte_compressdev -lrte_
cfgfile -lrte_bpf -lrte_bitratestats -lrte_bbdev -lrte_acl -lrte_timer -lrte_hash -lrte_metrics -lrte_cmdline -lrte_pci -lrte_ethdev -lrte_meter -lrte_net -l
rte_mbuf -lrte_mempool -lrte_rcu -lrte_ring -lrte_eal -lrte_pmu -lrte_telemetry -lrte_argparse -lrte_kvargs -lrte_log -lbsd
ln -sf basicfwd-shared build/basicfwd

bruce@host:~/dpdk_summit_2026$ ./build/basicfwd -l 1
EAL: Detected CPU lcores: 96
EAL: Detected NUMA nodes: 2
EAL: Detected shared linkage of DPDK
EAL: Multi-process socket /run/user/11304126/dpdk/rte/mp_socket
EAL: Selected IOVA mode 'VA'
EAL: VFIO support initialized
EAL: Using IOMMU type 1 (Type 1)
ICE_INIT: ice_load_pkg_type(): Active package is: 1.3.53.0, ICE OS Default Package (single VLAN mode)
ICE_INIT: ice_load_pkg_type(): Active package is: 1.3.53.0, ICE OS Default Package (single VLAN mode)
ICE_DRIVER: ice_set_tx_function(): Using Vector AVX2 (port 0).
ICE_DRIVER: ice_set_rx_function(): Using Vector AVX2 (port 0).
Port 0 MAC: b4 96 91 cd 7e b8
ICE_DRIVER: ice_set_tx_function(): Using Vector AVX2 (port 1).
ICE_DRIVER: ice_set_rx_function(): Using Vector AVX2 (port 1).
Port 1 MAC: b4 96 91 cd 7e b9

Core 1 forwarding packets. [Ctrl+C to quit]
^C

bruce@host:~/dpdk_summit_2026$
```

DPDK installed as local user

Copy and build example

Running as non-root –
Used chmod/chown on /dev/vfio/*
and hugetlbfs dirs.

Using Ctrl-C to kill the App is rather ugly!

DPDK Command Line Library

- The cmdline library is included in DPDK since very first versions
- Latest versions of DPDK include the "`dpdk-cmdline-gen.py`" script to improve ease of development

```
bruce@host:~/dpdk_summit_2026$ dpdk-cmdline-gen.py --help
usage: dpdk-cmdline-gen.py [-h] [--stubs] [--output-file OUTPUT_FILE] [--context-name CONTEXT_NAME] infile

Script to automatically generate boilerplate for using DPDK cmdline library.

positional arguments:
 infile          File with list of commands

options:
 -h, --help        show this help message and exit
 --stubs          Produce C file with empty function stubs for each command
 --output-file, -o OUTPUT_FILE
                  Output header filename [default to stdout]
 --context-name CONTEXT_NAME
                  Name given to the cmdline context variable in the output header [default=ctx]
```

Step 1 - Create our List of Commands

```
bruce@host:~/dpdk_summit_2026$ cat >> commands.list
# commands for our example app.
# The comment after each command is used as the help text

stats    # display per-port stats
quit     # close the application
```

Step 2 - Create Header on Build

```
+build/commands.h: commands.list Makefile
+	dpdk-cmdline-gen.py -o $@ $<
+
+CFLAGS += -Ibuild
 CFLAGS += -DALLOW_EXPERIMENTAL_API

-build/$(APP)-shared: $(SRCS-y) Makefile $(PC_FILE) | build
+build/$(APP)-shared: $(SRCS-y) Makefile $(PC_FILE) | build build/commands.h
	$(CC) $(CFLAGS) $(SRCS-y) -o $@ $(LDFLAGS) $(LDFLAGS_SHARED)

-build/$(APP)-static: $(SRCS-y) Makefile $(PC_FILE) | build
+build/$(APP)-static: $(SRCS-y) Makefile $(PC_FILE) | build build/commands.h
	$(CC) $(CFLAGS) $(SRCS-y) -o $@ $(LDFLAGS) $(LDFLAGS_STATIC)
```

Step 3 – Create Cmdline Instance (run in thread)

```
+       rte_thread_t cli_thread;
+       struct cmdline *cl = cmdline_stdin_new(ctx, "basicfwd> ");
+       rte_thread_create_control(&cli_thread, "cli_thread", (rte_thread_func)cmdline_interact, cl);
+
        /* Call lcore_main on the main core only. Called on single lcore. 8< */
        lcore_main();
        /* >8 End of called on single lcore. */

+       rte_thread_join(cli_thread, NULL);
+       cmdline_stdin_exit(cl);
```

Step 4 – Implement Callback Functions

```
+void
+cmd_quit_parsed(void *parsed_result, struct cmdline *cl, void *data)
+{
+        quit = true;
+        cmdline_quit(cl);
+}
```

Development with cmdline lib

- If starting with a lot of commands, can run `dpdk-cmdline-gen.py` manually to generate function stubs `[--stubs` flag]

- Thereafter just add commands to list file as you go, e.g.

```
+show port status <UINT16>N # show status of port N
```

- Rebuilding will give you an error message with function name to add (or get prototype from generated header).
  - Function name is build from fixed parts of the command
    - `cmd_show_port_status_parsed(void *, struct cmdline *, void *)`
  - Struct type for the first (parsed_result) parameter uses same base name
    - `struct cmd_show_port_status_result { … }`

Initial Configuration

- Command-line library takes care of runtime interactivity

- Need some method for initial config:
  - Application configuration
  - DPDK configuration

- DPDK Apps and examples all do that through command-line parameters; e.g.
  - /path/to/dpdk-testpmd <eal-args> -- <testpmd-args>

Programmatically Create EAL argc/argv

- The “dpdk default” method is not only way to initialize
- Many/most apps based on DPDK create their own argc/argv values to pass to “`rte_eal_init()`”
- For example:
  - Grout – dpdk_init function: https://github.com/DPDK/grout/blob/main/main/dpdk.c#L92
  - OVS – dpdk_init__ function: https://github.com/openvswitch/ovs/blob/main/lib/dpdk.c#L376

Config File Library

- Need a source of data for building up argc/argv
- DPDK includes the “cfgfile” library for reading/writing .ini configuration files.

```
[global]
corelist=1,2

[ports]
upstream=0000:03:00.0
downstream=vdev,net_af_xdp0,
iface=ens6f0
control=0000:15:00.0
```

```
char *myargv[10] = { [0]=argv[0] };
int myargc = 1;
char *lcores_param;
struct rte_cfgfile *cfg = rte_cfgfile_load("basicfwd.ini", 0);

if (asprintf(&lcores_param, "--lcores=%s", rte_cfgfile_get_entry(cfg, "global", "corelist")) < 0)
        rte_exit(EXIT_FAILURE, "Cannot create lcores parameter\n");
myargv[myargc++] = lcores_param;
myargv[myargc++] = "--allow=00:00.0";

int ret = rte_eal_init(myargc, myargv);
if (ret < 0)
        rte_exit(EXIT_FAILURE, "Error with EAL initialization\n");

/* hotplug in ports post-EAL init*/
rte_dev_probe(rte_cfgfile_get_entry(cfg, "ports", "upstream"));
rte_dev_probe(rte_cfgfile_get_entry(cfg, "ports", "downstream"));

/* Check that there is an even number of ports to send/receive on. */
nb_ports = rte_eth_dev_count_avail();
```

Simple argv array for building up init parameters

"hack" to prevent device discovery at init. Replaced in future with "-A" flag.

Explicitly hotplug in ports specified in config file.

Argument Parsing Library

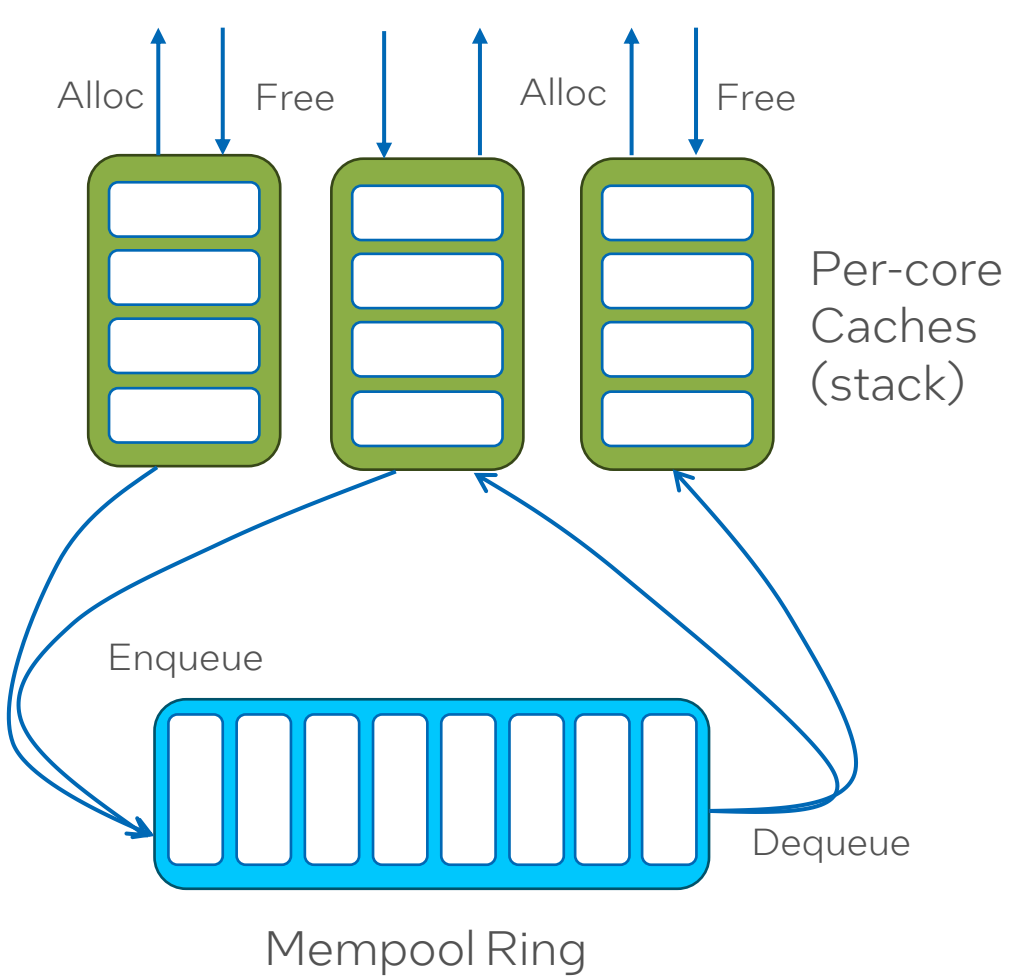
- Introduced January 2024
- More declarative method of handling argc/argv in app compared to getopt()
- Discussed at last years DPDK Summit: https://dpdksummit2025.sched.com/event/1xoZ4/introducing-argparse-library-feng-chengwen-huawei-technologies-co-ltd
- Now used in EAL itself for argument parsing!

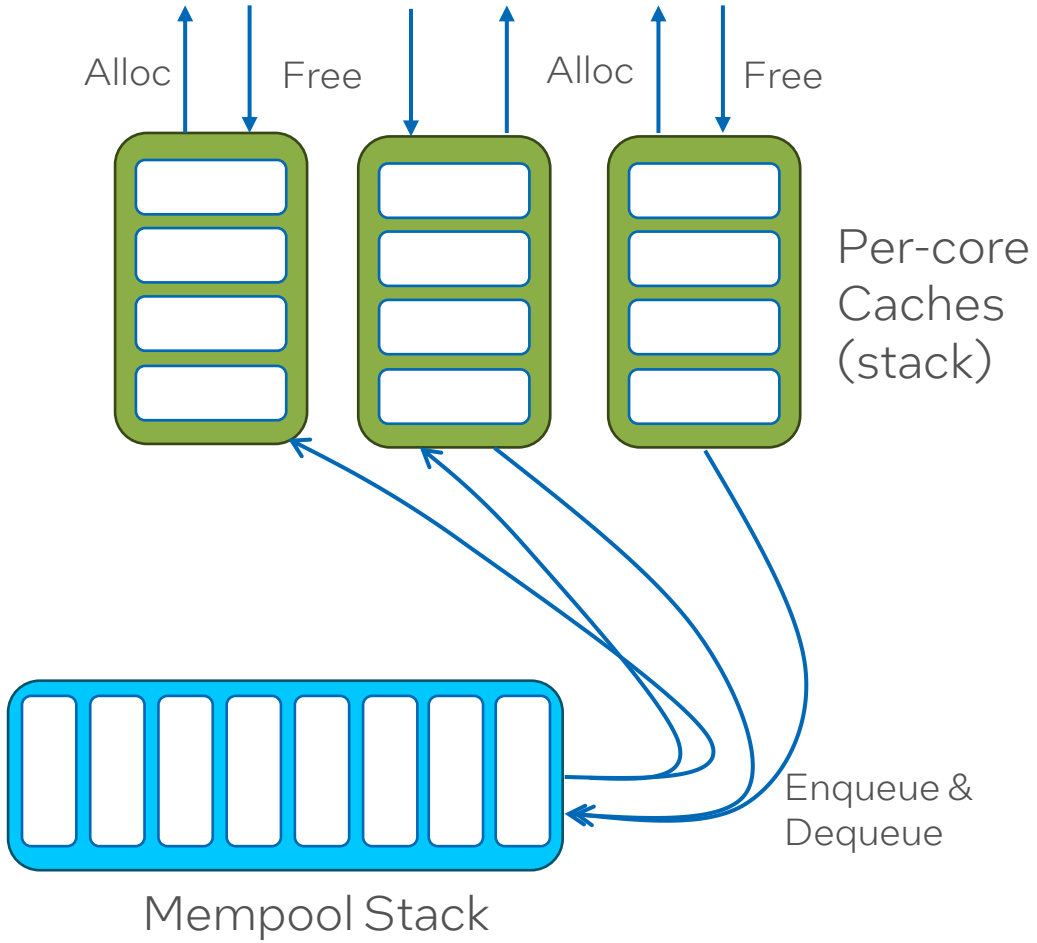
Suggested non-default EAL Args

1. “--in-memory”
   - EAL to no longer keep allocated hugepage files visible on filesystem
   - EAL no longer has keeps its memory config in a file on filesystem
   - Allows multiple instances without having to provide “--file-prefix”
   - No cleanup needed, if process crashes
   - Disables use of secondary processes

2. “--mbuf-pool-ops-name=stack”
   - Switches default mempool allocator to the stack allocator

Stack vs Ring Mempool allocator





With Ring Allocator, you iterate through *whole mempool* before getting a buffer back again!
With Stack Allocator, you only use the subset of the buffers you actually need!

Mempool Allocation

- For “run-to-completion” apps, i.e. no core-to-core transfer:
  - Aim to run entirely out of per-lcore cache
  - If you do need to touch the main mempool store, stack gives you hot/warm buffer back faster
    - Better chance of having buffer still in CPU cache
- For pipeline apps, i.e. apps passing from core to core
  - If alloc and free are on separate cores, ring mempool will always cycle through the full mempool reservation
  - Stack allocator reallocates hot buffers – better LLC utilization
- Stack uses locks, but ring still has atomics (and is not wait-free)

Telemetry Callbacks

- Included in DPDK since 2019
- Exports telemetry using a unix domain socket
- For interactive use script: `./usertools/dpdk-telemetry.py`
  - Can also be used to directly query a value:

```
$ echo /ethdev/stats,1 | ./usertools/dpdk-telemetry.py
{"/ethdev/stats": {"ipackets": 3627661245, "opackets": 1, "ibytes":
218706755992, "obytes": 60, "imissed": 1026227265, "ierrors": 0,
"oerrors": 0, "rx_nombuf": 0}}
```

Telemetry Callbacks

Pipe output through jq for readability:

```
$ echo /ethdev/stats,1 | ./usertools/dpdk-telemetry.py | jq
{
  "/ethdev/stats": {
    "ipackets": 4179770991,
    "opackets": 1,
    "ibytes": 251989543944,
    "obytes": 60,
    "imissed": 1182349486,
    "ierrors": 0,
    "oerrors": 0,
    "rx_nombuf": 0
  }
}
```

Proposed patchset for real-time monitoring:
https://patches.dpdk.org/project/dpdk/list/?series=37285&state=*

```
$ ./usertools/dpdk-telemetry-watcher.py -dT1 eth.rx
Connected to application: "dpdk-testpmd"
Time       /ethdev/stats,0.ipackets /ethdev/stats,1.ipackets                Total
16:30:46                58,393,901               58,393,070          116,786,971
```

Simple telemetry example

1. Create callback with desired stats/output

```
static int
telemetry_stage_stats_cb(const char *cmd __rte_unused,
                const char *params __rte_unused, struct rte_tel_data *d)
{
        rte_tel_data_start_dict(d);
        rte_tel_data_add_dict_uint(d, "foo", stage_stats.foo.processed);
        rte_tel_data_add_dict_uint(d, "foo_drops", stage_stats.foo.dropped);
        rte_tel_data_add_dict_uint(d, "bar", stage_stats.bar.processed);
        rte_tel_data_add_dict_uint(d, "bar_drops", stage_stats.bar.dropped);
        …
        return 0;
}
```

2. Register callback

```
return rte_telemetry_register_cmd("/basicfwd/stage_stats",  telemetry_stage_stats_cb,
                "Returns foo/bar/baz stage packet counters (processed and dropped)");
```

3. Check output using dpdk-telemetry.py script

```
$ dpdk-telemetry.py
...
Connected to application: "basicfwd"
--> /basicfwd/stage_stats
{
  "/basicfwd/stage_stats": {
    "foo_processed": 5288580800,
    "foo_dropped": 0,
    "bar_processed": 5288580992,
    "bar_dropped": 0,
    "baz_processed": 5288581023,
    "baz_dropped": 1
  }
}
```

Summary

- Starting point for a basic DPDK app
- Cmdline library
- EAL init args build-up
- Configfile library
- Argparse library
- Stack mempool vs ring
- Telemetry library