DPDK

DATA PLANE DEVELOPMENT KIT

Mempool Device Driver

Release 20.08.0-rc2

Jul 22, 2020

The following are a list of mempool PMDs, which can be used from an application through the mempool API.

CHAPTER

ONE

OCTEON TX FPAVF MEMPOOL DRIVER

The OCTEON TX FPAVF PMD (**librte_mempool_octeontx**) is a mempool driver for offload mempool device found in **Cavium OCTEON TX** SoC family.

More information can be found at Cavium, Inc Official Website.

1.1 Features

Features of the OCTEON TX FPAVF PMD are:

- 32 SR-IOV Virtual functions
- 32 Pools
- HW mempool manager

1.2 Supported OCTEON TX SoCs

- CN83xx

1.3 Prerequisites

See :doc: *../platform/octeontx.rst* for setup information.

1.4 Pre-Installation Configuration

1.4.1 Config File Options

The following options can be modified in the `config` file. Please note that enabling debugging options may affect system performance.

- `CONFIG_RTE_MBUF_DEFAULT_MEMPOOL_OPS` (set to `octeontx_fpavf`)

  Set default mempool ops to octeontx_fpavf.

- `CONFIG_RTE_LIBRTE_OCTEONTX_MEMPOOL` (default `y`)

  Toggle compilation of the `librte_mempool_octeontx` driver.

1.4.2 Driver Compilation

To compile the OCTEON TX FPAVF MEMPOOL PMD for Linux arm64 gcc target, run the following `make` command:

```
cd <DPDK-source-directory>
make config T=arm64-thunderx-linux-gcc
```

1.5 Initialization

The OCTEON TX fpavf mempool initialization similar to other mempool drivers like ring. However user need to pass –base-virtaddr as command line input to application example test_mempool.c application.

Example:

```
./build/app/test -c 0xf --base-virtaddr=0x100000000000 \
                    --mbuf-pool-ops-name="octeontx_fpavf"
```

CHAPTER

TWO

OCTEON TX2 NPA MEMPOOL DRIVER

The OCTEON TX2 NPA PMD (**librte_mempool_octeontx2**) provides mempool driver support for the integrated mempool device found in **Marvell OCTEON TX2** SoC family.

More information about OCTEON TX2 SoC can be found at Marvell Official Website.

2.1 Features

OCTEON TX2 NPA PMD supports:

- Up to 128 NPA LFs
- 1M Pools per LF
- HW mempool manager
- Ethdev Rx buffer allocation in HW to save CPU cycles in the Rx path.
- Ethdev Tx buffer recycling in HW to save CPU cycles in the Tx path.

2.2 Prerequisites and Compilation procedure

See ../platform/octeontx2 for setup information.

2.3 Pre-Installation Configuration

2.3.1 Compile time Config Options

The following option can be modified in the `config` file.

- `CONFIG_RTE_LIBRTE_OCTEONTX2_MEMPOOL` (default `y`)

  Toggle compilation of the `librte_mempool_octeontx2` driver.

2.3.2 Runtime Config Options

- `Maximum number of mempools per application` (default `128`)

  The maximum number of mempools per application needs to be configured on HW during mempool driver initialization. HW can support up to 1M mempools, Since each mempool costs set

of HW resources, the `max_pools devargs` parameter is being introduced to configure the number of mempools required for the application. For example:

```
-w 0002:02:00.0,max_pools=512
```

With the above configuration, the driver will set up only 512 mempools for the given application to save HW resources.

Note: Since this configuration is per application, the end user needs to provide `max_pools` parameter to the first PCIe device probed by the given application.

- `Lock NPA contexts in NDC`

  Lock NPA aura and pool contexts in NDC cache. The device args take hexadecimal bitmask where each bit represent the corresponding aura/pool id.

  For example:

  ```
  -w 0002:02:00.0,npa_lock_mask=0xf
  ```

2.3.3 Debugging Options

Table 2.1: OCTEON TX2 mempool debug options

#	Component	EAL log command
1	NPA	–log-level='pmd.mempool.octeontx2,8'

2.3.4 Standalone mempool device

The `usertools/dpdk-devbind.py` script shall enumerate all the mempool devices available in the system. In order to avoid, the end user to bind the mempool device prior to use ethdev and/or eventdev device, the respective driver configures an NPA LF and attach to the first probed ethdev or eventdev device. In case, if end user need to run mempool as a standalone device (without ethdev or eventdev), end user needs to bind a mempool device using `usertools/dpdk-devbind.py`

Example command to run `mempool_autotest` test with standalone OCTEONTX2 NPA device:

```
echo "mempool_autotest" | build/app/test -c 0xf0 --mbuf-pool-ops-name="octeontx2_npa"
```

CHAPTER

THREE

RING MEMPOOL DRIVER

rte_mempool_ring is a pure software mempool driver based on the `rte_ring` DPDK library. This is a default mempool driver. The following modes of operation are available for the ring mempool driver and can be selected via mempool ops API:

- `ring_mp_mc`

  The underlying **rte_ring** operates in multi-thread producer, multi-thread consumer sync mode. For more information please refer to: Ring_Library_MPMC_Mode.

- `ring_sp_sc`

  The underlying **rte_ring** operates in single-thread producer, single-thread consumer sync mode. For more information please refer to: Ring_Library_SPSC_Mode.

- `ring_sp_mc`

  The underlying **rte_ring** operates in single-thread producer, multi-thread consumer sync mode.

- `ring_mp_sc`

  The underlying **rte_ring** operates in multi-thread producer, single-thread consumer sync mode.

- `ring_mt_rts`

  For underlying **rte_ring** both producer and consumer operate in multi-thread Relaxed Tail Sync (RTS) mode. For more information please refer to: Ring_Library_MT_RTS_Mode.

- `ring_mt_hts`

  For underlying **rte_ring** both producer and consumer operate in multi-thread Head-Tail Sync (HTS) mode. For more information please refer to: Ring_Library_MT_HTS_Mode.

For 'classic' DPDK deployments (with one thread per core) the `ring_mp_mc` mode is usually the most suitable and the fastest one. For overcommitted scenarios (multiple threads share same set of cores) the `ring_mt_rts` or `ring_mt_hts` modes usually provide a better alternative. For more information about `rte_ring` structure, behaviour and available synchronisation modes please refer to: ../prog_guide/ring_lib.