# Data Path Development Kit on Layerscape platforms

DPDK provides a simple, complete framework for fast packet processing in data plane applications. Using the APIs provided as part of the framework, applications can leverage the capabilities of underlying network infrastructure.

This document describes DPDK basic introduction, DPDK core components, DPDK Linux networking, DPDK Crypto Subsystem, DPDK memory manager and DPDK implementation on Layerscape platforms.

## 1. DPDK Basic Introduction

DPDK is a set of libraries and drivers for fast packet processing across CPU architecture

a) High performance with dedicated user space cores.
b) Core optimized run-time libraries
c) Implements a run to completion model or pipeline model
d) Scales from low to high end processors

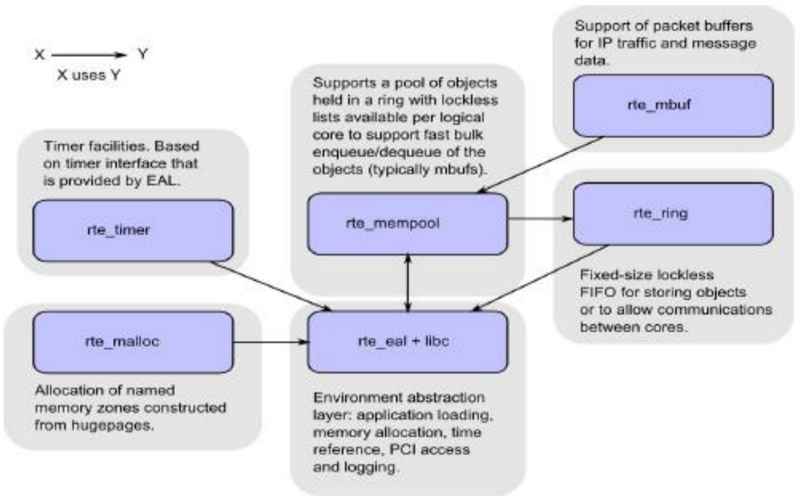
Why Use DPDK?

a) Bypass the operating system and kernel over-head getting data directly to your application
b) Core optimized run-time C-libraries with APIs

DPDK is originally designed by Intel is now a Linux Foundation de-facto standard for userspace networking – adopted by most vendors.

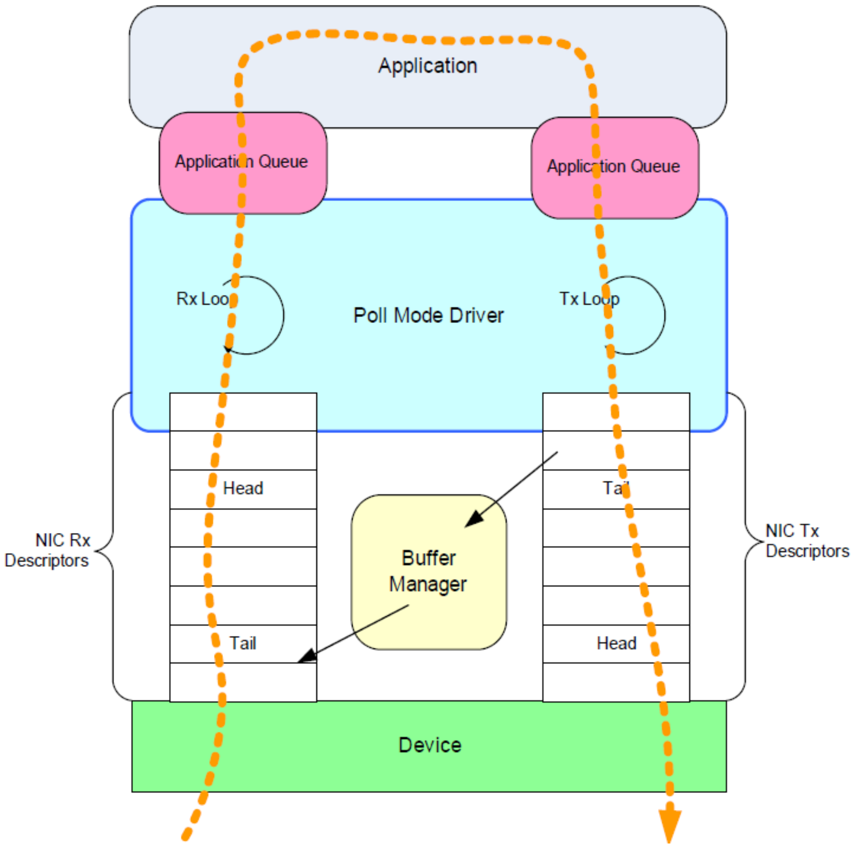
## 2. DPDK core components

DPDK architecture overview is


X → Y
X uses Y
Timer facilities. Based on timer interface that is provided by EAL.
rte_timer
Supports a pool of objects held in a ring with lockless lists available per logical core to support fast bulk enqueue/dequeue of the objects (typically mbufs).
rte_mempool
Support of packet buffers for IP traffic and message data.
rte_mbuf
rte_ring
Fixed-size lockless FIFO for storing objects or to allow communications between cores.
rte_malloc
Allocation of named memory zones constructed from hugepages.
rte_eal + libc
Environment abstraction layer: application loading, memory allocation, time reference, PCI access and logging.

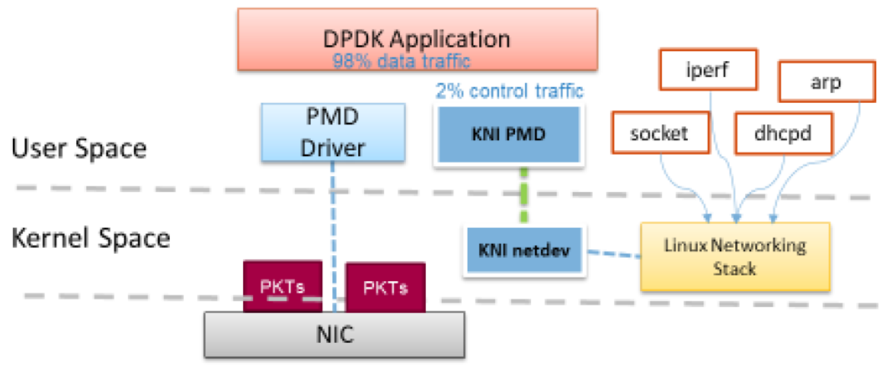
RTE or Run-Time Environment  is the core library – interfacing between application and drivers (eth, crypto etc)
Packet buffer represented by rte_mbuf are enqueued/dequeued from rings represented by rte_ring
All rte_mbuf are part of a buffer pool, represented by rte_mempool
Pools are in-turn allocated from an internal memory allocator represented by rte_malloc, which can get memory from hugepages.
All the above is glued together using the EAL (Environment Abstraction Layer)

The work flow of typical DPDK poll mode driver(PMD) is as the following.



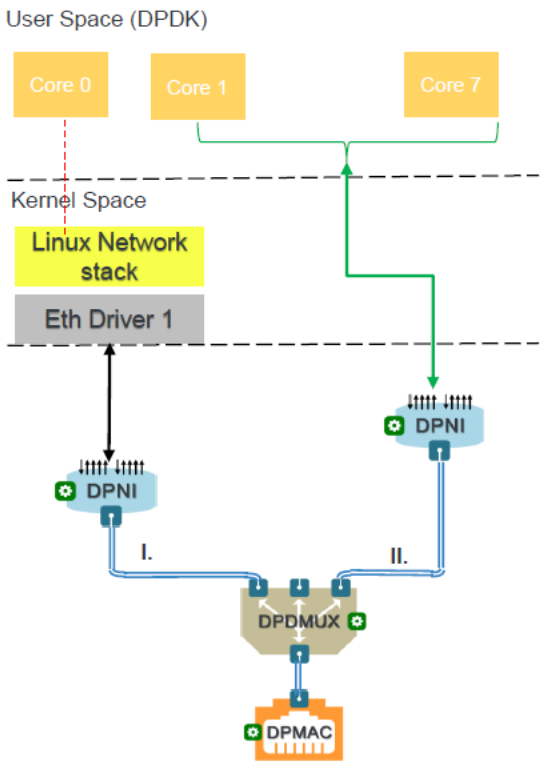
## 3. DPDK Linux Networking

Typically, DPDK PMD receives all the traffic.
DPDK Application may registers a virtual/logical ethernet interface in kernel netdev for each physical interface it own.
Based on the application logic, it sends the linux host bound traffic to Linux networking stack using this interface



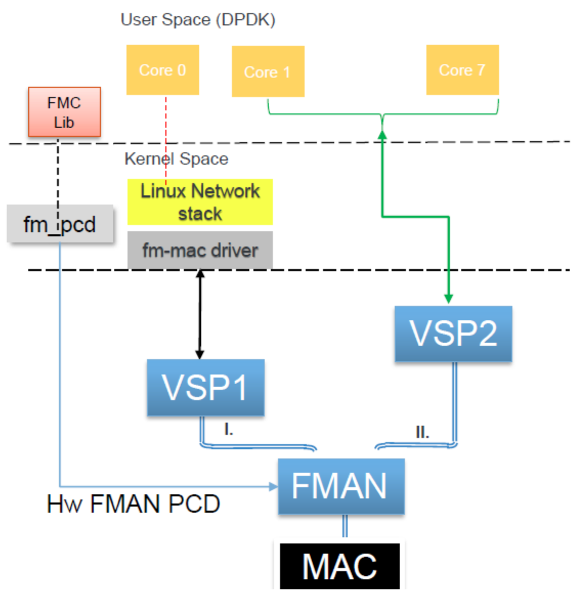
DPAA2 support HW (WRIOP) based traffic splitting with different logical ethernet interfaces.

- Default method is on VLAN or MAC
- Custom method is via DPDK DPAA2 PMD programming API interface on Protocol (e.g. GTP), ethernet header (e.g. eCPRI) etc.



DPAA support HW (FMAN) – Virtual   based traffic splitting with different interfaces.

- Virtual storage profile can be configured
- Custom method to split the traffic can be programmed via FMAN PCD interface configurations.



## 4. DPDK Crypto Subsystem

Session-less Mode

For each job, software defines:

The data to be operated upon (input buffers, lengths, offsets).

The output buffers to hold results.

The cryptographic operations to be performed.

Keys & context for the cryptographic operations.

Session Oriented Mode

For each job, software defines:

The data to be operated upon (input buffers, lengths, offsets)

The output buffers to hold results

Cryptographic operations, keys & context are defined at session establishment time, and referenced for each job

Supports virtual and physical crypto devices

- Virtual Device (Software Implementation)
  Intel AES-NI/vector operations
  ARM NEON instructions
- Physical Device (Hardware Accelerated)
  QAT
  CAAM-Job ring
  DPAA-CAAM
  DPAA2-CAAM

Test Applications

L2fwd with crypto

ipsec forward application

## 4.1 DPDK Crypto Subsystem APIs

Device creation and configuration

rte_cryptodev_configure, rte_cryptodev_queue_pair_setup

Device capabilities.

rte_cryptodev_info_get

Pool creations

rte_crypto_op_pool_create, rte_crypto_op_alloc

Session Management

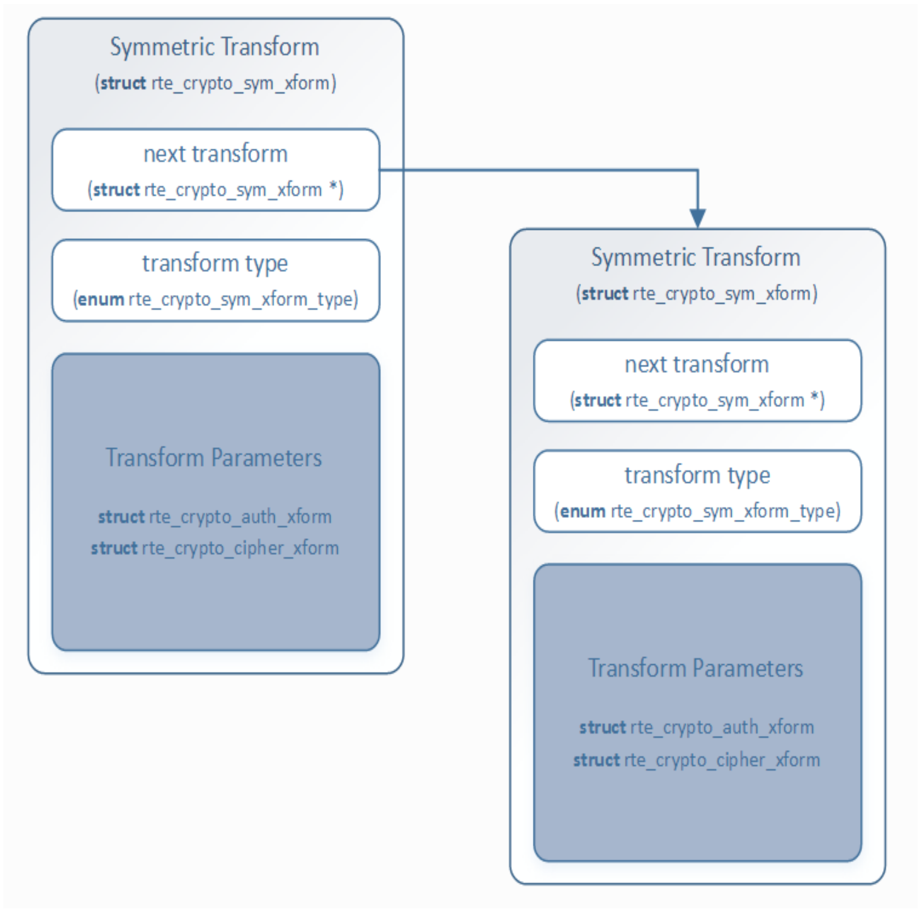
rte_cryptodev_sym_session_create

rte_cryptodev_sym_session_free

Packet operations

rte_cryptodev_enqueue_burst

rte_cryptodev_dequeue_burst

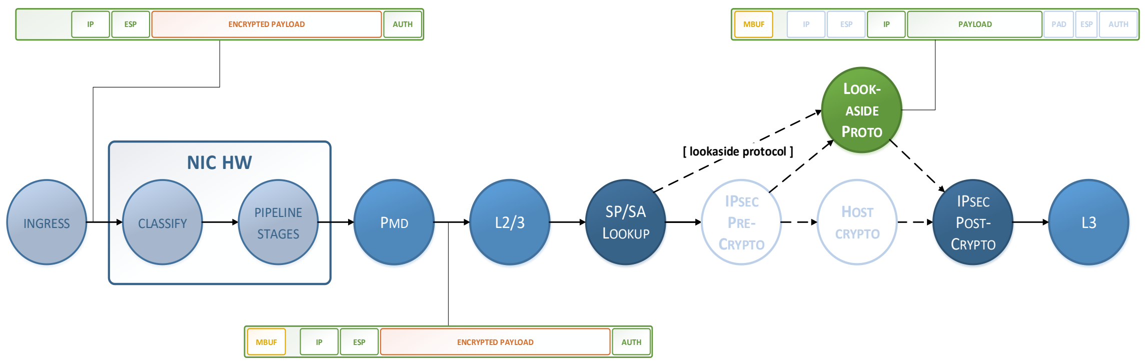

Symmetric Transform
(struct rte_crypto_sym_xform)
next transform
(struct rte_crypto_sym_xform *)
transform type
(enum rte_crypto_sym_xform_type)
Transform Parameters
struct rte_crypto_auth_xform
struct rte_crypto_cipher_xform
Symmetric Transform
(struct rte_crypto_sym_xform)
next transform
(struct rte_crypto_sym_xform *)
transform type
(enum rte_crypto_sym_xform_type)
Transform Parameters
struct rte_crypto_auth_xform
struct rte_crypto_cipher_xform

## 4.2 DPDK Security Offload – rte_security

Complete protocol offload to NIC or to crypto device.

Current focus is on IPSEC protocol offload only. Other protocol can be added

Already part of upstream in DPDK 17.11



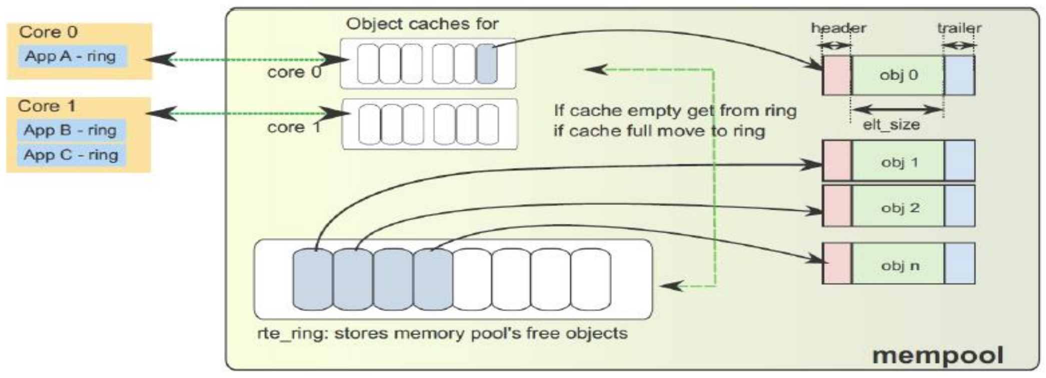
## 5. DPDK memory manager

### 5.1 Multi-layered memory architecture

Pools memory represented by rte_mempool which is used as raw buffers or packet buffers (rte_mbuf)
Each pool is backed by physical memory represented by Memory zones of rte_memzone
A Memzone has backing of multiple physical pages, including hugepages, represented by rte_memsegs
This multi-layered hierarchy allows for a flexible memory layout – for applications as well as drivers.
DPDK 1807 onwards, memory hotpluging is supported where physical memory is allocated on-demand

mempool structure:

- Pool of fixed-sized objects
- Uses a ring to store free objects
- Per core cache (optional)



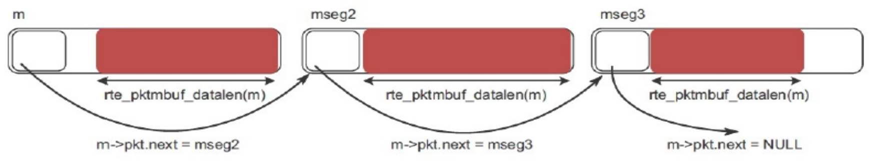
## 5.2 Buffer Manager

mbuf structure used to store network packets

- Created before runtime
- ”Allocation”: take a free mbuf from a mempool
- ”Deallocation”: put the mbuf back to the mempool
- Small size to fit in one cache-line (→ mbuf-chaining)

mbuf contains:

1. Metadata: control information, e.g. packet length
2. Pointer to next mbuf
3. Packet data: header and payload



## 5.3 Packet Buffer（mbuf）

A representation equivalent to Linux/BSD skb or Socket buffer. Referred to as *mbuf* or rte_mbuf

- Used to manage packet + metadata associated with the packet
- Multiple *mbufs* can be chained together to provide a larger virtual buffer for Tx/Rx of jumbo packets
- Metadata and packet data are sequentially stored in memory, allowing faster access and optimal cache utilization
- Alignments of rte_mbuf is strictly controlled and can impact performance to a large degree

Memory channels and layout impact the performance

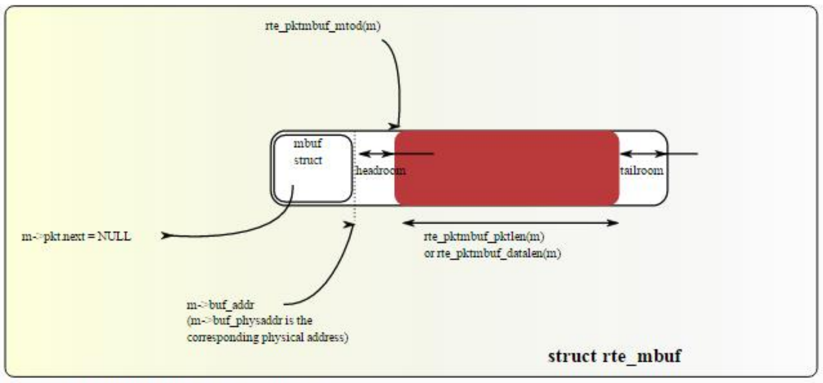


*Fig. 6.1 An mbuf with One Segment*

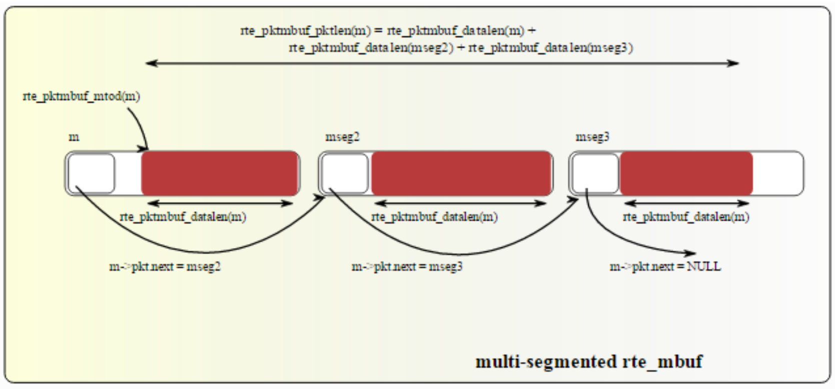
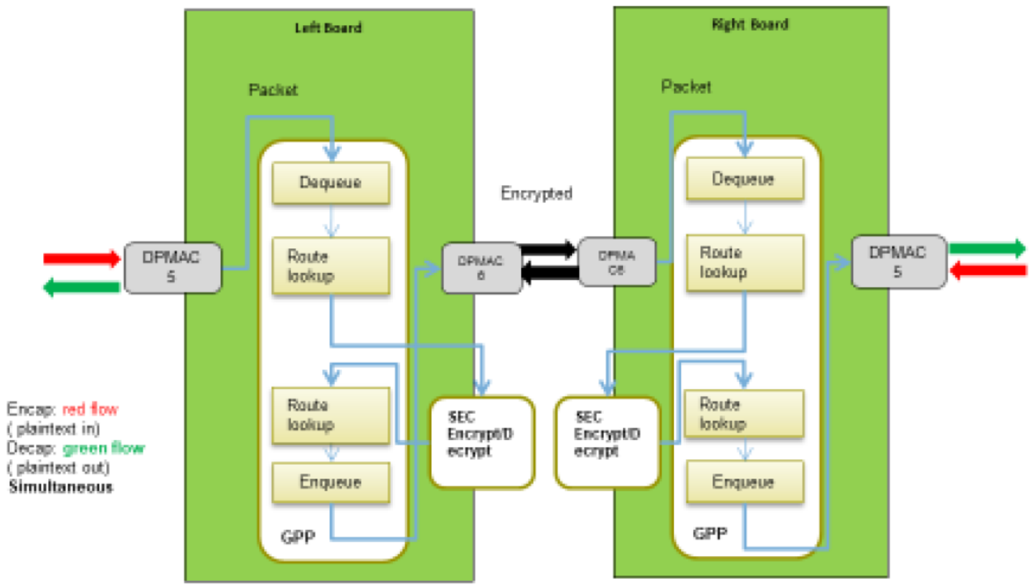


*Fig. 6.2 An mbuf with Three Segments*

## 6. DPDK implementation on Layerscape platforms

### DPDK IPSecgw Test Setup



Bidirectional traffic flow, 64 streams (flows) per direction
Plaintext packets sent to the DPMAC5(25G) of of DUT1 and DUT2 are encrypted by DPDK-IPSecgw application, then forwarded to the other board for decryption.

```
root@localhost:~# source /root/dpdkfiles/dynamic_dpl.sh dpmac.1 dpmac.2 dpmac.3 dpmac.4
parent - dprc.1
Creating Non nested DPRC
NEW DPRCs
dprc.1
   dprc.2
Using board type as 2088
Using High Performance Buffers
```

```
##################### Container [0;32m dprc.2 [0m is created #####################

Container dprc.2 have following resources :=>

  * 1 DPMCP
  * 16 DPBP
  * 8 DPCON
  * 8 DPSECI
  * 4 DPNI
  * 16 DPIO
  * 2 DPCI
  * 8 DPDMAI
  * 0 DPRTC


######################### Configured Interfaces #########################

Interface Name          Endpoint              Mac Address
==============          ========              ==================
dpni.1                  dpmac.1               -Dynamic-
dpni.2                  dpmac.2               -Dynamic-
dpni.3                  dpmac.3               -Dynamic-
dpni.4                  dpmac.4               -Dynamic-

root@localhost:~# export DPRC=dprc.2
root@localhost:~# echo $?
0
root@localhost:~# dmesg -D
root@localhost:~# echo $?
0
root@localhost:~# source /usr/local/dpdk/enable_performance_mode.sh
Setting performance mode on 8 cores only.
Check that all I/O cores required has 'performance' mode enabled
CPU governor mode 0:performance
CPU governor mode 1:performance
CPU governor mode 2:performance
CPU governor mode 3:performance
CPU governor mode 4:performance
CPU governor mode 5:performance
CPU governor mode 6:performance
CPU governor mode 7:performance
-bash: echo: write error: Device or resource busy
WARN: Hereafter, don't execute RT tasks on Core 0
Increasing priority of ksoftirqd on all cores
```

```
root@localhost:~# dmesg -D
root@localhost:~#  /root/dpdkfiles/ipsec-secgw  -v  -c  0xff  -n  1  --  -p  0xf  -P  -u  0xa  --
config="(0,0,0),(0,1,1),(1,0,2),(1,1,3),(2,0,4),(2,1,5),(3,0,6),(3,1,7)"                    -f
/root/dpdkfiles/ep0_64X64_proto.cfg
EAL: Detected 8 lcore(s)
EAL: Detected 1 NUMA nodes
EAL: RTE Version: 'DPDK 19.11.4'
EAL: Multi-process socket /var/run/dpdk/rte/mp_socket
fslmc: Skipping invalid device (power)
EAL: Selected IOVA mode 'VA'
EAL: No available hugepages reported in hugepages-2048kB
EAL: No available hugepages reported in hugepages-32768kB
EAL: No available hugepages reported in hugepages-64kB
EAL: Probing VFIO support...
EAL: VFIO support initialized
EAL: PCI device 0000:01:00.0 on NUMA socket -1
EAL:     Invalid NUMA socket, default to 0
EAL:     probe driver: 8086:107d net_e1000_em
PMD: dpni.1: netdev created
PMD: dpni.2: netdev created
PMD: dpni.3: netdev created
PMD: dpni.4: netdev created
Promiscuous mode selected
librte_ipsec usage: disabled
replay window size: 0
ESN: disabled
SA flags: 0
Frag TTL: 10000000000 ns
Allocated mbuf pool on socket 0
CRYPTODEV: elt_size 80 is expanded to 176
```