DPDK

DATA PLANE DEVELOPMENT KIT

DPDK Tools User Guides

Release 16.11.0

November 13, 2016

CONTENTS

CHAPTER

ONE

DPDK-PROCINFO APPLICATION

The dpdk-procinfo application is a Data Plane Development Kit (DPDK) application that runs as a DPDK secondary process and is capable of retrieving port statistics, resetting port statistics and printing DPDK memory information. This application extends the original functionality that was supported by dump_cfg.

1.1 Running the Application

The application has a number of command line options:

```
./$(RTE_TARGET)/app/dpdk-procinfo -- -m | [-p PORTMASK] [--stats | --xstats |
--stats-reset | --xstats-reset]
```

1.1.1 Parameters

-p PORTMASK: Hexadecimal bitmask of ports to configure.

–stats The stats parameter controls the printing of generic port statistics. If no port mask is specified stats are printed for all DPDK ports.

–xstats The stats parameter controls the printing of extended port statistics. If no port mask is specified xstats are printed for all DPDK ports.

–stats-reset The stats-reset parameter controls the resetting of generic port statistics. If no port mask is specified, the generic stats are reset for all DPDK ports.

–xstats-reset The xstats-reset parameter controls the resetting of extended port statistics. If no port mask is specified xstats are reset for all DPDK ports.

-m: Print DPDK memory information.

CHAPTER

TWO

DPDK-PDUMP APPLICATION

The `dpdk-pdump` tool is a Data Plane Development Kit (DPDK) tool that runs as a DPDK secondary process and is capable of enabling packet capture on dpdk ports.

Note:

- The `dpdk-pdump` tool can only be used in conjunction with a primary application which has the packet capture framework initialized already.
- The `dpdk-pdump` tool depends on libpcap based PMD which is disabled by default in the build configuration files, owing to an external dependency on the libpcap development files which must be installed on the board. Once the libpcap development files are installed, the libpcap based PMD can be enabled by setting CONFIG_RTE_LIBRTE_PMD_PCAP=y and recompiling the DPDK.

2.1 Running the Application

The tool has a number of command line options:

```
./build/app/dpdk-pdump --
                       --pdump '(port=<port id> | device_id=<pci id or vdev name>),
                                (queue=<queue_id>),
                                (rx-dev=<iface or pcap file> |
                                 tx-dev=<iface or pcap file>),
                                [ring-size=<ring size>],
                                [mbuf-size=<mbuf data size>],
                                [total-num-mbufs=<number of mbufs>]'
                       [--server-socket-path=<server socket dir>]
                       [--client-socket-path=<client socket dir>]
```

The `--pdump` command line option is mandatory and it takes various sub arguments which are described in below section.

Note:

- Parameters inside the parentheses represents mandatory parameters.
- Parameters inside the square brackets represents optional parameters.
- Multiple instances of `--pdump` can be passed to capture packets on different port and queue combinations.

The `--server-socket-path` command line option is optional. This represents the server socket directory. If no value is passed default values are used i.e. `/var/run/.dpdk/` for root users and `~/.dpdk/` for non root users.

The `--client-socket-path` command line option is optional. This represents the client socket directory. If no value is passed default values are used i.e. `/var/run/.dpdk/` for root users and `~/.dpdk/` for non root users.

2.1.1 The `--pdump` parameters

`port`: Port id of the eth device on which packets should be captured.

`device_id`: PCI address (or) name of the eth device on which packets should be captured.

Note:

- As of now the `dpdk-pdump` tool cannot capture the packets of virtual devices in the primary process due to a bug in the ethdev library. Due to this bug, in a multi process context, when the primary and secondary have different ports set, then the secondary process (here the `dpdk-pdump` tool) overwrites the `rte_eth_devices[]` entries of the primary process.

`queue`: Queue id of the eth device on which packets should be captured. The user can pass a queue value of `*` to enable packet capture on all queues of the eth device.

`rx-dev`: Can be either a pcap file name or any Linux iface.

`tx-dev`: Can be either a pcap file name or any Linux iface.

Note:

- To receive ingress packets only, `rx-dev` should be passed.
- To receive egress packets only, `tx-dev` should be passed.
- To receive ingress and egress packets separately `rx-dev` and `tx-dev` should both be passed with the different file names or the Linux iface names.
- To receive ingress and egress packets together, `rx-dev` and `tx-dev` should both be passed with the same file name or the same Linux iface name.

`ring-size`: Size of the ring. This value is used internally for ring creation. The ring will be used to enqueue the packets from the primary application to the secondary. This is an optional parameter with default size 16384.

`mbuf-size`: Size of the mbuf data. This is used internally for mempool creation. Ideally this value must be same as the primary application's mempool's mbuf data size which is used for packet RX. This is an optional parameter with default size 2176.

`total-num-mbufs`: Total number mbufs in mempool. This is used internally for mempool creation. This is an optional parameter with default value 65535.

2.2 Example

```
$ sudo ./build/app/dpdk-pdump -- --pdump 'port=0,queue=*,rx-dev=/tmp/rx.pcap'
```

CHAPTER

THREE

DPDK-PMDINFO APPLICATION

The `dpdk-pmdinfo` tool is a Data Plane Development Kit (DPDK) utility that can dump a PMDs hardware support info.

3.1 Running the Application

The tool has a number of command line options:

```
dpdk-pmdinfo [-hrtp] [-d <pci id file] <elf-file>

-h, --help            Show a short help message and exit
-r, --raw             Dump as raw json strings
-d FILE, --pcidb=FILE Specify a pci database to get vendor names from
-t, --table           Output information on hw support as a hex table
-p, --plugindir       Scan dpdk for autoload plugins
```

Note:

- Parameters inside the square brackets represents optional parameters.

CHAPTER

FOUR

DPDK-DEVBIND APPLICATION

The `dpdk-devbind` tool is a Data Plane Development Kit (DPDK) utility that helps binding and unbinding devices from specific drivers. As well as checking their status in that regard.

4.1 Running the Application

The tool has a number of command line options:

```
dpdk-devbind [options] DEVICE1 DEVICE2 ....
```

4.2 OPTIONS

- `--help, --usage`

  Display usage information and quit

- `-s, --status`

  Print the current status of all known network interfaces. For each device, it displays the PCI domain, bus, slot and function, along with a text description of the device. Depending upon whether the device is being used by a kernel driver, the `igb_uio` driver, or no driver, other relevant information will be displayed: - the Linux interface name e.g. `if=eth0` - the driver being used e.g. `drv=igb_uio` - any suitable drivers not currently using that device e.g. `unused=igb_uio` NOTE: if this flag is passed along with a bind/unbind option, the status display will always occur after the other operations have taken place.

- `-b driver, --bind=driver`

  Select the driver to use or “none” to unbind the device

- `-u, --unbind`

  Unbind a device (Equivalent to `-b none`)

- `--force`

  By default, devices which are used by Linux - as indicated by having routes in the routing table - cannot be modified. Using the `--force` flag overrides this behavior, allowing active links to be forcibly unbound. WARNING: This can lead to loss of network connection and should be used with caution.

Warning: Due to the way VFIO works, there are certain limitations to which devices can be used with VFIO. Mainly it comes down to how IOMMU groups work. Any Virtual Function device can be used with VFIO on its own, but physical devices will require either all ports bound to VFIO, or some of them bound to VFIO while others not being bound to anything at all.
If your device is behind a PCI-to-PCI bridge, the bridge will then be part of the IOMMU group in which your device is in. Therefore, the bridge driver should also be unbound from the bridge PCI device for VFIO to work with devices behind the bridge.

Warning: While any user can run the `dpdk-devbind.py` script to view the status of the network ports, binding or unbinding network ports requires root privileges.

4.3 Examples

To display current device status:

```
dpdk-devbind --status
```

To bind eth1 from the current driver and move to use igb_uio:

```
dpdk-devbind --bind=igb_uio eth1
```

To unbind 0000:01:00.0 from using any driver:

```
dpdk-devbind -u 0000:01:00.0
```

To bind 0000:02:00.0 and 0000:02:00.1 to the ixgbe kernel driver:

```
dpdk-devbind -b ixgbe 02:00.0 02:00.1
```

To check status of all network ports, assign one to the igb_uio driver and check status again:

```
# Check the status of the available devices.
dpdk-devbind --status
Network devices using DPDK-compatible driver
============================================
<none>

Network devices using kernel driver
===================================
0000:0a:00.0 '82599ES 10-Gigabit' if=eth2 drv=ixgbe unused=


# Bind the device to igb_uio.
sudo dpdk-devbind -b igb_uio 0000:0a:00.0


# Recheck the status of the devices.
dpdk-devbind --status
Network devices using DPDK-compatible driver
============================================
0000:0a:00.0 '82599ES 10-Gigabit' drv=igb_uio unused=
```