DPDK

DATA PLANE DEVELOPMENT KIT

Getting Started Guide for FreeBSD

Release 20.08.0

Aug 08, 2020

1 Introduction **1**
1.1 Documentation Roadmap 1

2 Installing DPDK from the Ports Collection **2**
2.1 Installing the DPDK Package for FreeBSD 2
2.2 Installing the DPDK FreeBSD Port 2
2.3 Compiling and Running the Example Applications 3

3 Compiling the DPDK Target from Source **5**
3.1 Prerequisites 5
3.2 Building DPDK 5
3.3 Loading the DPDK contigmem Module 6
3.4 Loading the DPDK nic_uio Module 7

4 Compiling and Running Sample Applications **9**
4.1 Compiling a Sample Application 9
4.2 Running a Sample Application 9
4.3 Running DPDK Applications Without Root Privileges 10

5 EAL parameters **11**
5.1 Common EAL parameters 11
5.2 FreeBSD-specific EAL parameters 14

CHAPTER

ONE

INTRODUCTION

This document contains instructions for installing and configuring the Data Plane Development Kit (DPDK) software. It is designed to get customers up and running quickly and describes how to compile and run a DPDK application in a FreeBSD application (freebsd) environment, without going deeply into detail.

For a comprehensive guide to installing and using FreeBSD, the following handbook is available from the FreeBSD Documentation Project: FreeBSD Handbook.

Note: DPDK is now available as part of the FreeBSD ports collection and as a pre-built package. Installing via the ports collection or FreeBSD *pkg* infrastructure is now the recommended way to install DPDK on FreeBSD, and is documented in the next chapter, *Installing DPDK from the Ports Collection*.

1.1 Documentation Roadmap

The following is a list of DPDK documents in the suggested reading order:

- **Release Notes** : Provides release-specific information, including supported features, limitations, fixed issues, known issues and so on. Also, provides the answers to frequently asked questions in FAQ format.
- **Getting Started Guide** (this document): Describes how to install and configure the DPDK; designed to get users up and running quickly with the software.
- **Programmer's Guide**: Describes:
  - The software architecture and how to use it (through examples), specifically in a Linux* application (linux) environment
  - The content of the DPDK, the build system (including the commands that can be used in the root DPDK Makefile to build the development kit and an application) and guidelines for porting an application
  - Optimizations used in the software and those that should be considered for new development

  A glossary of terms is also provided.
- **API Reference**: Provides detailed information about DPDK functions, data structures and other programming constructs.
- **Sample Applications User Guide**: Describes a set of sample applications. Each chapter describes a sample application that showcases specific functionality and provides instructions on how to compile, run and use the sample application.

CHAPTER

TWO

INSTALLING DPDK FROM THE PORTS COLLECTION

The easiest way to get up and running with the DPDK on FreeBSD is to install it using the FreeBSD *pkg* utility or from the ports collection. Details of installing applications from packages or the ports collection are documented in the FreeBSD Handbook, chapter Installing Applications: Packages and Ports.

Note: Please ensure that the latest patches are applied to third party libraries and software to avoid any known vulnerabilities.

2.1 Installing the DPDK Package for FreeBSD

DPDK can be installed on FreeBSD using the command:

```
pkg install dpdk
```

After the installation of the DPDK package, instructions will be printed on how to install the kernel modules required to use the DPDK. A more complete version of these instructions can be found in the sections *Loading the DPDK contigmem Module* and *Loading the DPDK nic_uio Module*. Normally, lines like those below would be added to the file `/boot/loader.conf`.

```
# Reserve 2 x 1G blocks of contiguous memory using contigmem driver:
hw.contigmem.num_buffers=2
hw.contigmem.buffer_size=1073741824
contigmem_load="YES"

# Identify NIC devices for DPDK apps to use and load nic_uio driver:
hw.nic_uio.bdfs="2:0:0,2:0:1"
nic_uio_load="YES"
```

2.2 Installing the DPDK FreeBSD Port

If so desired, the user can install DPDK using the ports collection rather than from a pre-compiled binary package. On a system with the ports collection installed in `/usr/ports`, the DPDK can be installed using the commands:

```
cd /usr/ports/net/dpdk

make install
```

2.3 Compiling and Running the Example Applications

When the DPDK has been installed from the ports collection it installs its example applications in `/usr/local/share/dpdk/examples`. These examples can be compiled and run as described in *Compiling and Running Sample Applications*.

Note: DPDK example applications must be complied using *gmake* rather than BSD *make*. To detect the installed DPDK libraries, *pkg-config* should also be installed on the system.

Note: To install a copy of the DPDK compiled using gcc, please download the official DPDK package from https://core.dpdk.org/download/ and install manually using the instructions given in the next chapter, *Compiling the DPDK Target from Source*

An example application can therefore be copied to a user's home directory and compiled and run as below, where we have 2 memory blocks of size 1G reserved via the contigmem module, and 4 NIC ports bound to the nic_uio module:

```
cp -r /usr/local/share/dpdk/examples/helloworld .

cd helloworld/

gmake
cc -O3 -I/usr/local/include -include rte_config.h -march=corei7 -D__BSD_VISIBLE  main.c -o buil
ln -sf helloworld-shared build/helloworld

sudo ./build/helloworld -l 0-3
EAL: Sysctl reports 8 cpus
EAL: Detected 8 lcore(s)
EAL: Detected 1 NUMA nodes
EAL: Multi-process socket /var/run/dpdk/rte/mp_socket
EAL: Selected IOVA mode 'PA'
EAL: Contigmem driver has 2 buffers, each of size 1GB
EAL: Mapped memory segment 0 @ 0x1040000000: physaddr:0x180000000, len 1073741824
EAL: Mapped memory segment 1 @ 0x1080000000: physaddr:0x1c0000000, len 1073741824
EAL: PCI device 0000:00:19.0 on NUMA socket 0
EAL:   probe driver: 8086:153b net_e1000_em
EAL:   0000:00:19.0 not managed by UIO driver, skipping
EAL: PCI device 0000:01:00.0 on NUMA socket 0
EAL:   probe driver: 8086:1572 net_i40e
EAL: PCI device 0000:01:00.1 on NUMA socket 0
EAL:   probe driver: 8086:1572 net_i40e
EAL: PCI device 0000:01:00.2 on NUMA socket 0
EAL:   probe driver: 8086:1572 net_i40e
EAL: PCI device 0000:01:00.3 on NUMA socket 0
EAL:   probe driver: 8086:1572 net_i40e
hello from core 1
hello from core 2
hello from core 3
hello from core 0
```

Note: To run a DPDK process as a non-root user, adjust the permissions on the `/dev/contigmem` and `/dev/uio device` nodes as described in section *Running DPDK Applications Without Root Privileges*

Note: For an explanation of the command-line parameters that can be passed to an DPDK application, see section *Running a Sample Application*.

CHAPTER

THREE

COMPILING THE DPDK TARGET FROM SOURCE

3.1 Prerequisites

The following FreeBSD packages are required to build DPDK:

- meson
- ninja
- pkgconf

These can be installed using (as root):

```
pkg install meson pkgconf
```

To compile the required kernel modules for memory management and working with physical NIC devices, the kernel sources for FreeBSD also need to be installed. If not already present on the system, these can be installed via commands like the following, for FreeBSD 12.1 on x86_64:

```
fetch http://ftp.freebsd.org/pub/FreeBSD/releases/amd64/12.1-RELEASE/src.txz
tar -C / -xJvf src.txz
```

To enable the telemetry library in DPDK, the jansson library also needs to be installed, and can be installed via:

```
pkg install jansson
```

Individual drivers may have additional requirements. Consult the relevant driver guide for any driver-specific requirements of interest.

3.2 Building DPDK

The following commands can be used to build and install DPDK on a system. The final, install, step generally needs to be run as root:

```
meson build
cd build
ninja
ninja install
```

This will install the DPDK libraries and drivers to */usr/local/lib* with a pkg-config file *libdpdk.pc* installed to */usr/local/lib/pkgconfig*. The DPDK test applications, such as *dpdk-testpmd* are installed to */usr/local/bin*. To use these applications, it is recommended that the *contigmem* and *nic_uio* kernel modules be loaded first, as described in the next section.

Note: It is recommended that pkg-config be used to query information about the compiler and linker flags needed to build applications against DPDK. In some cases, the path */usr/local/lib/pkgconfig* may not be in the default search paths for *.pc* files, which means that queries for DPDK information may fail. This can be fixed by setting the appropriate path in *PKG_CONFIG_PATH* environment variable.

3.3 Loading the DPDK contigmem Module

To run a DPDK application, physically contiguous memory is required. In the absence of non-transparent superpages, the included sources for the contigmem kernel module provides the ability to present contiguous blocks of memory for the DPDK to use. The contigmem module must be loaded into the running kernel before any DPDK is run. Once DPDK is installed on the system, the module can be found in the */boot/modules* directory.

The amount of physically contiguous memory along with the number of physically contiguous blocks to be reserved by the module can be set at runtime prior to module loading using:

```
kenv hw.contigmem.num_buffers=n
kenv hw.contigmem.buffer_size=m
```

The kernel environment variables can also be specified during boot by placing the following in `/boot/loader.conf`:

```
hw.contigmem.num_buffers=n
hw.contigmem.buffer_size=m
```

The variables can be inspected using the following command:

```
sysctl -a hw.contigmem
```

Where n is the number of blocks and m is the size in bytes of each area of contiguous memory. A default of two buffers of size 1073741824 bytes (1 Gigabyte) each is set during module load if they are not specified in the environment.

The module can then be loaded using kldload:

```
kldload contigmem
```

It is advisable to include the loading of the contigmem module during the boot process to avoid issues with potential memory fragmentation during later system up time. This can be achieved by placing lines similar to the following into `/boot/loader.conf`:

```
hw.contigmem.num_buffers=1
hw.contigmem.buffer_size=1073741824
contigmem_load="YES"
```

Note: The contigmem_load directive should be placed after any definitions of `hw.contigmem.num_buffers` and `hw.contigmem.buffer_size` if the default values are not to be used.

An error such as:

```
kldload: can't load ./x86_64-native-freebsd-gcc/kmod/contigmem.ko:
         Exec format error
```

is generally attributed to not having enough contiguous memory available and can be verified via dmesg or `/var/log/messages`:

```
kernel: contigmalloc failed for buffer <n>
```

To avoid this error, reduce the number of buffers or the buffer size.

3.4 Loading the DPDK nic_uio Module

After loading the contigmem module, the `nic_uio` module must also be loaded into the running kernel prior to running any DPDK application, e.g. using:

```
kldload nic_uio
```

Note: If the ports to be used are currently bound to a existing kernel driver then the `hw.nic_uio.bdfs sysctl` value will need to be set before loading the module. Setting this value is described in the next section below.

Currently loaded modules can be seen by using the `kldstat` command and a module can be removed from the running kernel by using `kldunload <module_name>`.

To load the module during boot place the following into `/boot/loader.conf`:

```
nic_uio_load="YES"
```

Note: `nic_uio_load="YES"` must appear after the contigmem_load directive, if it exists.

By default, the `nic_uio` module will take ownership of network ports if they are recognized DPDK devices and are not owned by another module. However, since the FreeBSD kernel includes support, either built-in, or via a separate driver module, for most network card devices, it is likely that the ports to be used are already bound to a driver other than `nic_uio`. The following sub-section describe how to query and modify the device ownership of the ports to be used by DPDK applications.

3.4.1 Binding Network Ports to the nic_uio Module

Device ownership can be viewed using the pciconf -l command. The example below shows four Intel® 82599 network ports under `if_ixgbe` module ownership.

```
pciconf -l
ix0@pci0:1:0:0: class=0x020000 card=0x00038086 chip=0x10fb8086 rev=0x01 hdr=0x00
ix1@pci0:1:0:1: class=0x020000 card=0x00038086 chip=0x10fb8086 rev=0x01 hdr=0x00
ix2@pci0:2:0:0: class=0x020000 card=0x00038086 chip=0x10fb8086 rev=0x01 hdr=0x00
ix3@pci0:2:0:1: class=0x020000 card=0x00038086 chip=0x10fb8086 rev=0x01 hdr=0x00
```

The first column constitutes three components:

1. Device name: `ixN`
2. Unit name: `pci0`
3. Selector (Bus:Device:Function): `1:0:0`

Where no driver is associated with a device, the device name will be `none`.

By default, the FreeBSD kernel will include built-in drivers for the most common devices; a kernel rebuild would normally be required to either remove the drivers or configure them as loadable modules.

To avoid building a custom kernel, the nic_uio module can detach a network port from its current device driver. This is achieved by setting the hw.nic_uio.bdfs kernel environment variable prior to loading nic_uio, as follows:

```
kenv hw.nic_uio.bdfs="b:d:f,b:d:f,..."
```

Where a comma separated list of selectors is set, the list must not contain any whitespace.

For example to re-bind ix2@pci0:2:0:0 and ix3@pci0:2:0:1 to the nic_uio module upon loading, use the following command:

```
kenv hw.nic_uio.bdfs="2:0:0,2:0:1"
```

The variable can also be specified during boot by placing the following into /boot/loader.conf, before the previously-described nic_uio_load line - as shown:

```
hw.nic_uio.bdfs="2:0:0,2:0:1"
nic_uio_load="YES"
```

3.4.2 Binding Network Ports Back to their Original Kernel Driver

If the original driver for a network port has been compiled into the kernel, it is necessary to reboot FreeBSD to restore the original device binding. Before doing so, update or remove the hw.nic_uio.bdfs in /boot/loader.conf.

If rebinding to a driver that is a loadable module, the network port binding can be reset without rebooting. To do so, unload both the target kernel module and the nic_uio module, modify or clear the hw.nic_uio.bdfs kernel environment (kenv) value, and reload the two drivers - first the original kernel driver, and then the nic_uio driver. Note: the latter does not need to be reloaded unless there are ports that are still to be bound to it.

Example commands to perform these steps are shown below:

```
kldunload nic_uio
kldunload <original_driver>

# To clear the value completely:
kenv -u hw.nic_uio.bdfs

# To update the list of ports to bind:
kenv hw.nic_uio.bdfs="b:d:f,b:d:f,..."

kldload <original_driver>

kldload nic_uio  # optional
```

CHAPTER

FOUR

COMPILING AND RUNNING SAMPLE APPLICATIONS

The chapter describes how to compile and run applications in a DPDK environment. It also provides a pointer to where sample applications are stored.

4.1 Compiling a Sample Application

The DPDK example applications make use of the pkg-config file installed on the system when DPDK is installed, and so can be built using GNU make.

Note: BSD make cannot be used to compile the DPDK example applications. GNU make can be installed using *pkg install gmake* if not already installed on the FreeBSD system.

The following shows how to compile the helloworld example app, following the installation of DPDK using *ninja install* as described previously:

```
$ export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig

$ cd examples/helloworld/

$ gmake
cc -O3 -I/usr/local/include -include rte_config.h -march=native
-D__BSD_VISIBLE  main.c -o build/helloworld-shared
-L/usr/local/lib -lrte_telemetry -lrte_bpf -lrte_flow_classify
-lrte_pipeline -lrte_table -lrte_port -lrte_fib -lrte_ipsec
-lrte_stack -lrte_security -lrte_sched -lrte_reorder -lrte_rib
-lrte_rcu -lrte_rawdev -lrte_pdump -lrte_member -lrte_lpm
-lrte_latencystats -lrte_jobstats -lrte_ip_frag -lrte_gso -lrte_gro
-lrte_eventdev -lrte_efd -lrte_distributor -lrte_cryptodev
-lrte_compressdev -lrte_cfgfile -lrte_bitratestats -lrte_bbdev
-lrte_acl -lrte_timer -lrte_hash -lrte_metrics -lrte_cmdline
-lrte_pci -lrte_ethdev -lrte_meter -lrte_net -lrte_mbuf
-lrte_mempool -lrte_ring -lrte_eal -lrte_kvargs
ln -sf helloworld-shared build/helloworld
```

4.2 Running a Sample Application

1. The `contigmem` and `nic_uio` modules must be set up prior to running an application.
2. Any ports to be used by the application must be already bound to the `nic_uio` module, as described in section *Binding Network Ports to the nic_uio Module*, prior to running the application.

The application is linked with the DPDK target environment's Environment Abstraction Layer (EAL) library, which provides some options that are generic to every DPDK application.

A large number of options can be given to the EAL when running an application. A full list of options can be got by passing *–help* to a DPDK application. Some of the EAL options for FreeBSD are as follows:

- `-c COREMASK` or `-l CORELIST`: A hexadecimal bit mask of the cores to run on. Note that core numbering can change between platforms and should be determined beforehand. The corelist is a list of cores to use instead of a core mask.
- `-b <domain:bus:devid.func>`: Blacklisting of ports; prevent EAL from using specified PCI device (multiple `-b` options are allowed).
- `--use-device`: Use the specified Ethernet device(s) only. Use comma-separate `[domain:]bus:devid.func` values. Cannot be used with `-b` option.
- `-v`: Display version information on startup.
- `-m MB`: Memory to allocate from hugepages, regardless of processor socket.

Other options, specific to Linux and are not supported under FreeBSD are as follows:

- `socket-mem`: Memory to allocate from hugepages on specific sockets.
- `--huge-dir`: The directory where hugetlbfs is mounted.
- `mbuf-pool-ops-name`: Pool ops name for mbuf to use.
- `--file-prefix`: The prefix text used for hugepage filenames.

The `-c` or `-l` option is mandatory; the others are optional.

4.3 Running DPDK Applications Without Root Privileges

Although applications using the DPDK use network ports and other hardware resources directly, with a number of small permission adjustments, it is possible to run these applications as a user other than "root". To do so, the ownership, or permissions, on the following file system objects should be adjusted to ensure that the user account being used to run the DPDK application has access to them:

- The userspace-io device files in `/dev`, for example, `/dev/uio0`, `/dev/uio1`, and so on
- The userspace contiguous memory device: `/dev/contigmem`

Note: Please refer to the DPDK Release Notes for supported applications.

CHAPTER

FIVE

EAL PARAMETERS

This document contains a list of all EAL parameters. These parameters can be used by any DPDK application running on FreeBSD.

5.1 Common EAL parameters

The following EAL parameters are common to all platforms supported by DPDK.

5.1.1 Lcore-related options

- `-c <core mask>`

  Set the hexadecimal bitmask of the cores to run on.

- `-l <core list>`

  List of cores to run on

  The argument format is `<c1>[-c2][,c3[-c4],...]` where `c1`, `c2`, etc are core indexes between 0 and 128.

- `--lcores <core map>`

  Map lcore set to physical cpu set

  The argument format is:

  ```
  <lcores[@cpus]>[<,lcores[@cpus]>...]
  ```

  Lcore and CPU lists are grouped by `(` and `)` Within the group. The `-` character is used as a range separator and `,` is used as a single number separator. The grouping `()` can be omitted for single element group. The `@` can be omitted if cpus and lcores have the same value.

Note: At a given instance only one core option `--lcores`, `-l` or `-c` can be used.

- `--master-lcore <core ID>`

  Core ID that is used as master.

- `-s <service core mask>`

  Hexadecimal bitmask of cores to be used as service cores.

5.1.2 Device-related options

- `-b,--pci-blacklist <[domain:]bus:devid.func>`

  Blacklist a PCI device to prevent EAL from using it. Multiple -b options are allowed.

Note: PCI blacklist cannot be used with `-w` option.

- `-w,--pci-whitelist <[domain:]bus:devid.func>`

  Add a PCI device in white list.

Note: PCI whitelist cannot be used with `-b` option.

- `--vdev <device arguments>`

  Add a virtual device using the format:

  ```
  <driver><id>[,key=val, ...]
  ```

  For example:

  ```
  --vdev 'net_pcap0,rx_pcap=input.pcap,tx_pcap=output.pcap'
  ```

- `-d <path to shared object or directory>`

  Load external drivers. An argument can be a single shared object file, or a directory containing multiple driver shared objects. Multiple -d options are allowed.

- `--no-pci`

  Disable PCI bus.

5.1.3 Multiprocessing-related options

- `--proc-type <primary|secondary|auto>`

  Set the type of the current process.

- `--base-virtaddr <address>`

  Attempt to use a different starting address for all memory maps of the primary DPDK process. This can be helpful if secondary processes cannot start due to conflicts in address map.

5.1.4 Memory-related options

- `-n <number of channels>`

  Set the number of memory channels to use.

- `-r <number of ranks>`

  Set the number of memory ranks (auto-detected by default).

- `-m <megabytes>`

  Amount of memory to preallocate at startup.

- `--in-memory`

  Do not create any shared data structures and run entirely in memory. Implies `--no-shconf` and (if applicable) `--huge-unlink`.

- `--iova-mode <pa|va>`

  Force IOVA mode to a specific value.

5.1.5 Debugging options

- `--no-shconf`

  No shared files created (implies no secondary process support).

- `--no-huge`

  Use anonymous memory instead of hugepages (implies no secondary process support).

- `--log-level <type:val>`

  Specify log level for a specific component. For example:

  ```
  --log-level lib.eal:debug
  ```

  Can be specified multiple times.

- `--trace=<regex-match>`

  Enable trace based on regular expression trace name. By default, the trace is disabled. User must specify this option to enable trace. For example:

  Global trace configuration for EAL only:

  ```
  --trace=eal
  ```

  Global trace configuration for ALL the components:

  ```
  --trace=.*
  ```

  Can be specified multiple times up to 32 times.

- `--trace-dir=<directory path>`

  Specify trace directory for trace output. For example:

  Configuring `/tmp/` as a trace output directory:

  ```
  --trace-dir=/tmp
  ```

  By default, trace output will created at `home` directory and parameter must be specified once only.

- `--trace-bufsz=<val>`

  Specify maximum size of allocated memory for trace output for each thread. Valid unit can be either `B` or `K` or `M` for `Bytes`, `KBytes` and `MBytes` respectively. For example:

  Configuring `2MB` as a maximum size for trace output file:

  ```
  --trace-bufsz=2M
  ```

  By default, size of trace output file is `1MB` and parameter must be specified once only.

- `--trace-mode=<o[verwrite] | d[iscard] >`

  Specify the mode of update of trace output file. Either update on a file can be wrapped or discarded when file size reaches its maximum limit. For example:

  To `discard` update on trace output file:

  ```
  --trace-mode=d or --trace-mode=discard
  ```

  Default mode is `overwrite` and parameter must be specified once only.

5.1.6 Other options

- `-h, --help`

  Display help message listing all EAL parameters.

- `-v`

  Display the version information on startup.

- `mbuf-pool-ops-name`:

  Pool ops name for mbuf to use.

- `--telemetry`:

  Enable telemetry (enabled by default).

- `--no-telemetry`:

  Disable telemetry.

5.2 FreeBSD-specific EAL parameters

There are currently no FreeBSD-specific EAL command-line parameters available.