DPDK FAQ

Release 2.1.0

August 17, 2015

CONTENTS

This document contains some Frequently Asked Questions that arise when working with DPDK.

Contents

CHAPTER

ONE

WHAT DOES “EAL: MAP_ALL_HUGEPAGES(): OPEN FAILED: PERMISSION DENIED CANNOT INIT MEMORY” MEAN?

This is most likely due to the test application not being run with sudo to promote the user to a superuser. Alternatively, applications can also be run as regular user. For more information, please refer to DPDK Getting Started Guide.

CHAPTER

TWO

IF I WANT TO CHANGE THE NUMBER OF TLB HUGEPAGES ALLOCATED, HOW DO I REMOVE THE ORIGINAL PAGES ALLOCATED?

The number of pages allocated can be seen by executing the following command:

```
grep Huge /proc/meminfo
```

Once all the pages are mmapped by an application, they stay that way. If you start a test application with less than the maximum, then you have free pages. When you stop and restart the test application, it looks to see if the pages are available in the `/dev/huge` directory and mmaps them. If you look in the directory, you will see n number of 2M pages files. If you specified 1024, you will see 1024 page files. These are then placed in memory segments to get contiguous memory.

If you need to change the number of pages, it is easier to first remove the pages. The tools/setup.sh script provides an option to do this. See the “Quick Start Setup Script” section in the DPDK Getting Started Guide for more information.

CHAPTER

THREE

IF I EXECUTE “L2FWD -C F -M 64 -N 3 – -P 3”, I GET THE FOLLOWING OUTPUT, INDICATING THAT THERE ARE NO SOCKET 0 HUGEPAGES TO ALLOCATE THE MBUF AND RING STRUCTURES TO?

I have set up a total of 1024 Hugepages (that is, allocated 512 2M pages to each NUMA node).

The -m command line parameter does not guarantee that huge pages will be reserved on specific sockets. Therefore, allocated huge pages may not be on socket 0. To request memory to be reserved on a specific socket, please use the –socket-mem command-line parameter instead of -m.

CHAPTER

FOUR

I AM RUNNING A 32-BIT DPDK APPLICATION ON A NUMA SYSTEM, AND SOMETIMES THE APPLICATION INITIALIZES FINE BUT CANNOT ALLOCATE MEMORY. WHY IS THAT HAPPENING?

32-bit applications have limitations in terms of how much virtual memory is available, hence the number of hugepages they are able to allocate is also limited (1 GB per page size). If your system has a lot (>1 GB per page size) of hugepage memory, not all of it will be allocated. Due to hugepages typically being allocated on a local NUMA node, the hugepages allocation the application gets during the initialization depends on which NUMA node it is running on (the EAL does not affinitize cores until much later in the initialization process). Sometimes, the Linux OS runs the DPDK application on a core that is located on a different NUMA node from DPDK master core and therefore all the hugepages are allocated on the wrong socket.

To avoid this scenario, either lower the amount of hugepage memory available to 1 GB per page size (or less), or run the application with taskset affinitizing the application to a would-be master core.

For example, if your EAL coremask is 0xff0, the master core will usually be the first core in the coremask (0x10); this is what you have to supply to taskset:

```
taskset 0x10 ./l2fwd -c 0xff0 -n 2
```

In this way, the hugepages have a greater chance of being allocated to the correct socket. Additionally, a `--socket-mem` option could be used to ensure the availability of memory for each socket, so that if hugepages were allocated on the wrong socket, the application simply will not start.

CHAPTER

FIVE

ON APPLICATION STARTUP, THERE IS A LOT OF EAL INFORMATION PRINTED. IS THERE ANY WAY TO REDUCE THIS?

Yes, each EAL has a configuration file that is located in the /config directory. Within each configuration file, you will find CONFIG_RTE_LOG_LEVEL=8. You can change this to a lower value, such as 6 to reduce this printout of debug information. The following is a list of LOG levels that can be found in the rte_log.h file. You must remove, then rebuild, the EAL directory for the change to become effective as the configuration file creates the rte_config.h file in the EAL directory.

```
#define RTE_LOG_EMERG 1U    /* System is unusable. */
#define RTE_LOG_ALERT 2U    /* Action must be taken immediately. */
#define RTE_LOG_CRIT 3U     /* Critical conditions. */
#define RTE_LOG_ERR 4U      /* Error conditions. */
#define RTE_LOG_WARNING 5U  /* Warning conditions. */
#define RTE_LOG_NOTICE 6U   /* Normal but significant condition. */
#define RTE_LOG_INFO 7U     /* Informational. */
#define RTE_LOG_DEBUG 8U    /* Debug-level messages. */
```

CHAPTER

SIX

HOW CAN I TUNE MY NETWORK APPLICATION TO ACHIEVE LOWER LATENCY?

Traditionally, there is a trade-off between throughput and latency. An application can be tuned to achieve a high throughput, but the end-to-end latency of an average packet typically increases as a result. Similarly, the application can be tuned to have, on average, a low end-to-end latency at the cost of lower throughput.

To achieve higher throughput, the DPDK attempts to aggregate the cost of processing each packet individually by processing packets in bursts. Using the testpmd application as an example, the “burst” size can be set on the command line to a value of 16 (also the default value). This allows the application to request 16 packets at a time from the PMD. The testpmd application then immediately attempts to transmit all the packets that were received, in this case, all 16 packets. The packets are not transmitted until the tail pointer is updated on the corresponding TX queue of the network port. This behavior is desirable when tuning for high throughput because the cost of tail pointer updates to both the RX and TX queues can be spread across 16 packets, effectively hiding the relatively slow MMIO cost of writing to the PCIe* device.

However, this is not very desirable when tuning for low latency, because the first packet that was received must also wait for the other 15 packets to be received. It cannot be transmitted until the other 15 packets have also been processed because the NIC will not know to transmit the packets until the TX tail pointer has been updated, which is not done until all 16 packets have been processed for transmission.

To consistently achieve low latency even under heavy system load, the application developer should avoid processing packets in bunches. The testpmd application can be configured from the command line to use a burst value of 1. This allows a single packet to be processed at a time, providing lower latency, but with the added cost of lower throughput.

CHAPTER

SEVEN

WITHOUT NUMA ENABLED, MY NETWORK THROUGHPUT IS LOW, WHY?

I have a dual Intel® Xeon® E5645 processors 2.40 GHz with four Intel® 82599 10 Gigabit Ethernet NICs. Using eight logical cores on each processor with RSS set to distribute network load from two 10 GbE interfaces to the cores on each processor.

Without NUMA enabled, memory is allocated from both sockets, since memory is interleaved. Therefore, each 64B chunk is interleaved across both memory domains.

The first 64B chunk is mapped to node 0, the second 64B chunk is mapped to node 1, the third to node 0, the fourth to node 1. If you allocated 256B, you would get memory that looks like this:

```
256B buffer
Offset 0x00 - Node 0
Offset 0x40 - Node 1
Offset 0x80 - Node 0
Offset 0xc0 - Node 1
```

Therefore, packet buffers and descriptor rings are allocated from both memory domains, thus incurring QPI bandwidth accessing the other memory and much higher latency. For best performance with NUMA disabled, only one socket should be populated.

CHAPTER

EIGHT

I AM GETTING ERRORS ABOUT NOT BEING ABLE TO OPEN FILES. WHY?

As the DPDK operates, it opens a lot of files, which can result in reaching the open files limits, which is set using the ulimit command or in the limits.conf file. This is especially true when using a large number (>512) of 2 MB huge pages. Please increase the open file limit if your application is not able to open files. This can be done either by issuing a ulimit command or editing the limits.conf file. Please consult Linux* manpages for usage information.

CHAPTER

NINE

DOES MY KERNEL REQUIRE PATCHING TO RUN THE DPDK?

Any kernel greater than version 2.6.33 can be used without any patches applied. The following kernels may require patches to provide hugepage support:

- Kernel version 2.6.32 requires the following patches applied:
  - mm: hugetlb: add hugepage support to pagemap
  - mm: hugetlb: fix hugepage memory leak in walk_page_range()
  - hugetlb: add nodemask arg to huge page alloc, free and surplus adjust functions (not mandatory, but recommended on a NUMA system to support per-NUMA node hugepages allocation)
- Kernel version 2.6.31, requires the above patches plus the following:
  - UIO: Add name attributes for mappings and port regions

CHAPTER

TEN

VF DRIVER FOR IXGBE DEVICES CANNOT BE INITIALIZED

Some versions of Linux* IXGBE driver do not assign a random MAC address to VF devices at initialization. In this case, this has to be done manually on the VM host, using the following command:

```
ip link set <interface> vf <VF function> mac <MAC address>
```

where <interface> being the interface providing the virtual functions for example, eth0, <VF function> being the virtual function number, for example 0, and <MAC address> being the desired MAC address.

CHAPTER

ELEVEN

IS IT SAFE TO ADD AN ENTRY TO THE HASH TABLE WHILE RUNNING?

Currently the table implementation is not a thread safe implementation and assumes that locking between threads and processes is handled by the user's application. This is likely to be supported in future releases.

CHAPTER

TWELVE

WHAT IS THE PURPOSE OF SETTING IOMMU=PT?

DPDK uses a 1:1 mapping and does not support IOMMU. IOMMU allows for simpler VM physical address translation. The second role of IOMMU is to allow protection from unwanted memory access by an unsafe device that has DMA privileges. Unfortunately, the protection comes with an extremely high performance cost for high speed NICs.

Setting `iommu=pt` disables IOMMU support for the hypervisor.

CHAPTER

THIRTEEN

WHEN TRYING TO SEND PACKETS FROM AN APPLICATION TO ITSELF, MEANING SMAC==DMAC, USING INTEL(R) 82599 VF PACKETS ARE LOST.

Check on register LLE(PFVMTXSSW[n]), which allows an individual pool to send traffic and have it looped back to itself.

CHAPTER

FOURTEEN

CAN I SPLIT PACKET RX TO USE DPDK AND HAVE AN APPLICATION'S HIGHER ORDER FUNCTIONS CONTINUE USING LINUX* PTHREAD?

The DPDK's lcore threads are Linux* pthreads bound onto specific cores. Configure the DPDK to do work on the same cores and run the application's other work on other cores using the DPDK's "coremask" setting to specify which cores it should launch itself on.

CHAPTER

FIFTEEN

IS IT POSSIBLE TO EXCHANGE DATA BETWEEN DPDK PROCESSES AND REGULAR USERSPACE PROCESSES VIA SOME SHARED MEMORY OR IPC MECHANISM?

Yes - DPDK processes are regular Linux/BSD processes, and can use all OS provided IPC mechanisms.

CHAPTER

SIXTEEN

CAN THE MULTIPLE QUEUES IN INTEL(R) I350 BE USED WITH DPDK?

I350 has RSS support and 8 queue pairs can be used in RSS mode. It should work with multi-queue DPDK applications using RSS.

CHAPTER

SEVENTEEN

HOW CAN HUGEPAGE-BACKED MEMORY BE SHARED AMONG MULTIPLE PROCESSES?

See the Primary and Secondary examples in the multi-process sample application.