Lab2: send and receive packets with DPDK

519021910913 黄喆敏

1. 问题解答

- Q1: What's the purpose of using hugepage?
  - 减少TLB缓存项的使用，从而大大降低TLB Miss的概率。
  - 减少页表的级数，从而提升查询页表的效率。

- Q2: Take examples/helloworld as an example, describe the execution flow of DPDK programs?

  答：helloworld代码如下所示。

```
int main(int argc, char **argv)
{
   int ret;
   unsigned lcore_id;

   ret = rte_eal_init(argc, argv);
   if (ret < 0)
      rte_panic("Cannot init EAL\n");

   /* call lcore_hello() on every worker lcore */
   RTE_LCORE_FOREACH_WORKER(lcore_id) {
      rte_eal_remote_launch(lcore_hello, NULL, lcore_id);
   }

   /* call it on main lcore too */
   lcore_hello(NULL);
   rte_eal_mp_wait_lcore();

   /* clean up the EAL */
   rte_eal_cleanup();
   return 0;
}
```

  控制流如下：

  1. 调用`rte_eal_init`，初始化基础运行环境，若初始化失败则报错。
  2. 对多核运行初始化。即遍历EAL指定可以使用的lcore，然后通过`rte_eal_remote_launch`在每个lcore上，启动被指定的线程。
  3. 运行当前线程的函数locore_hello。
  4. 主线程等待从线程结束执行。
  5. 执行`rte_eal_cleanup`，释放资源，防止hugepage内存泄漏。

- Q3: Read the codes of examples/skeleton, describe DPDK APIs related to sending and receiving packets.

  答：以下分别为收包、发包所对应的API。通过指定的端口与队列，收/发缓存区中的数据。

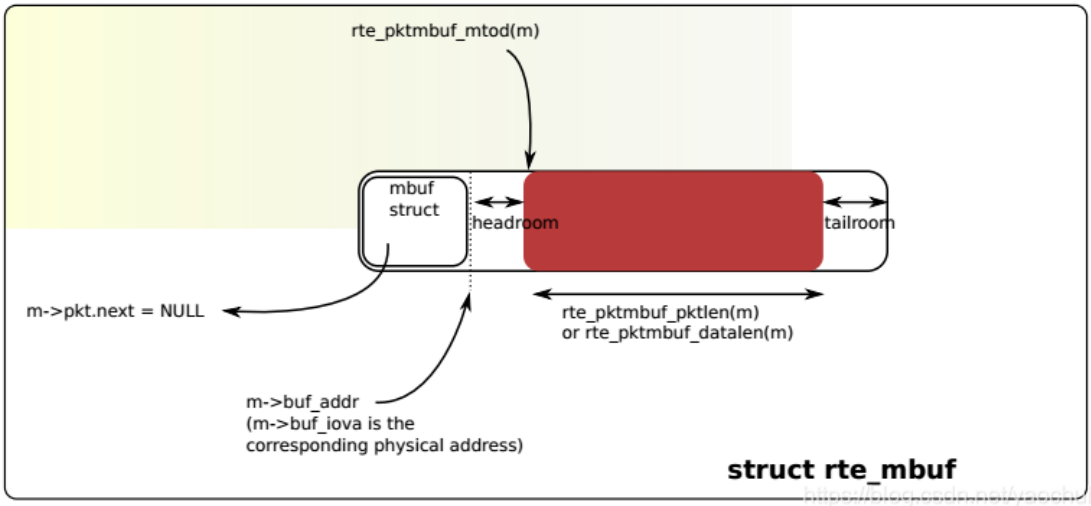
其中，最后一个参数`nb_pkts`为指定一次函数调用来处理的包的个数。当设置为1时，每次收/发一个包。

dpdk在样例程序，例如skeleton中，使用了burst模式，即收/发包数量为32个。这样可以减少内存访问，提高性能。

```
static inline uint16_t rte_eth_rx_burst(uint8_t port_id, uint16_t queue_id,
struct rte_mbuf **rx_pkts, const uint16_t nb_pkts)
```

```
static inline uint16_t rte_eth_tx_burst(uint8_t port_id, uint16_t queue_id,
struct rte_mbuf **tx_pkts, uint16_t nb_pkts)
```

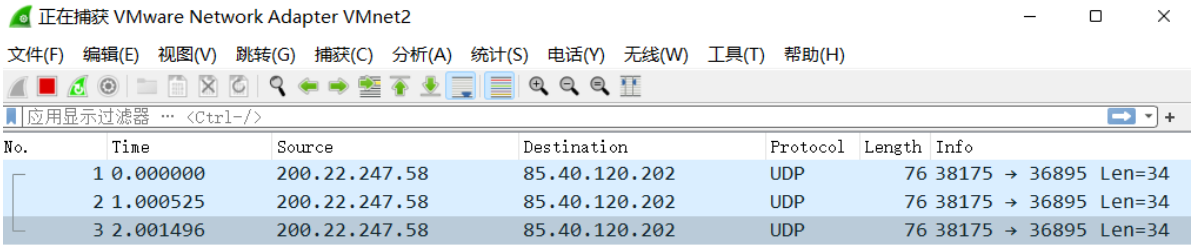
- Q4: Describe the data structure of 'rte_mbuf'.
- 答：`rte_mbuf`的结构如下所示。

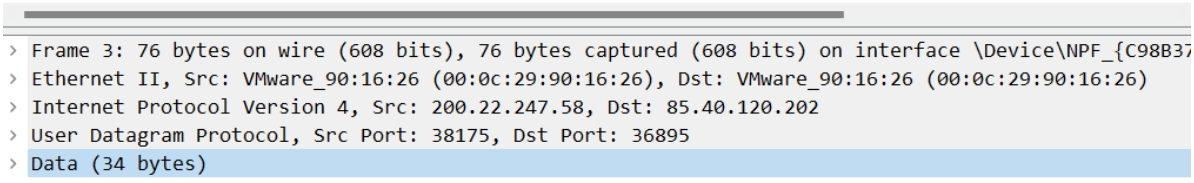


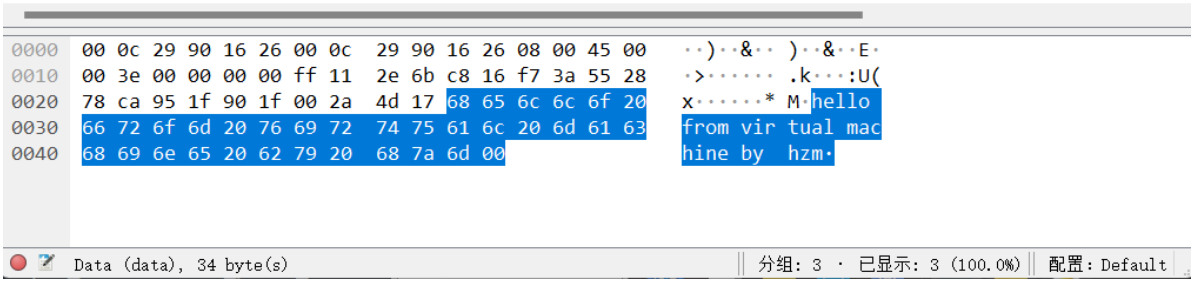
  - `headroom`为`mbuf`头部与实际包数据的一段空间，存储控制信息、帧内容、事件等。`headroom`的起始地址保存在`buff_addr`指针中。
  - 在`headroom`后为实际数据所占空间。数据帧的长度可通过调用`pkt_len(m)`或`data_len(m)`获得。
  - 实际数据后剩余的空间为`tailroom`。通过`headroom`与`tailroom`，可方便应用解封报文。
  - `pkt`的`next`字段指向下一个segment的地址；`buf_addr`指向`headroom`的起始地址；`rte_pktmbuf_mtod(m)`指向实际data的起始地址。
  - 此外，还记录了所属的mempool，时间戳，端口，私有数据大小等信息。

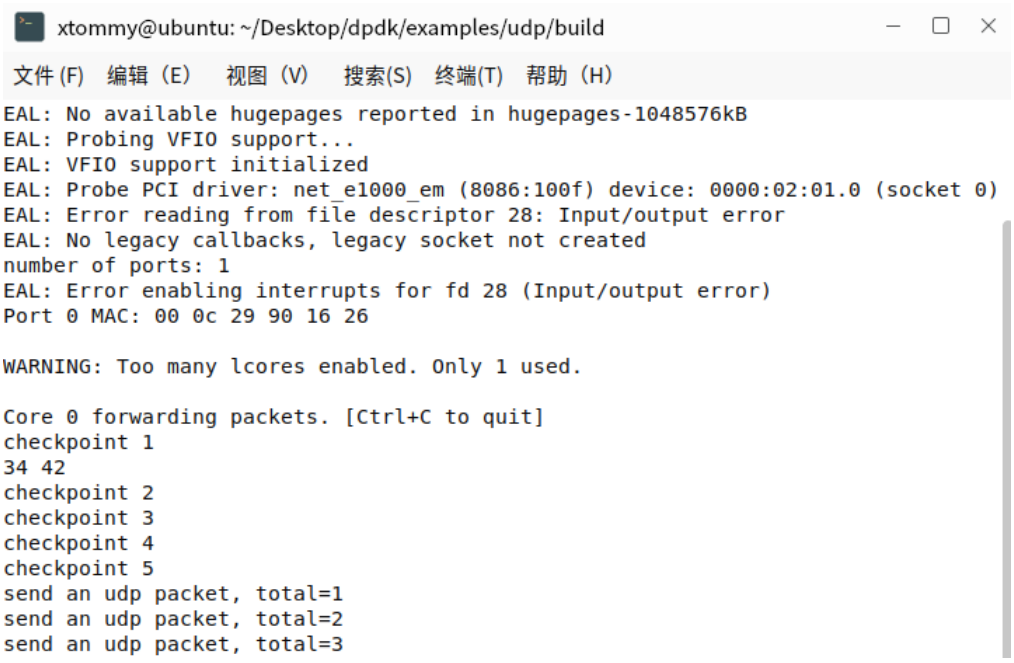
2. 检验正确性

通过wireshark，我们可以监听来自虚拟机的UDP包，且UDP包可以正常解析，内容正确。









部分报文解析如下所示:

```
Frame 3: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface
\Device\NPF_{C98B37FA-1912-418B-B26C-EC3757C968CB}, id 0
    Interface id: 0 (\Device\NPF_{C98B37FA-1912-418B-B26C-EC3757C968CB})
    Encapsulation type: Ethernet (1)
    Arrival Time: Mar 12, 2022 15:46:15.490546000 中国标准时间
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1647071175.490546000 seconds
    [Time delta from previous captured frame: 1.000971000 seconds]
    [Time delta from previous displayed frame: 1.000971000 seconds]
    [Time since reference or first frame: 2.001496000 seconds]
    Frame Number: 3
    Frame Length: 76 bytes (608 bits)
    Capture Length: 76 bytes (608 bits)
```

```
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:udp:data]
    [Coloring Rule Name: UDP]
    [Coloring Rule String: udp]
Ethernet II, Src: VMware_90:16:26 (00:0c:29:90:16:26), Dst: VMware_90:16:26
(00:0c:29:90:16:26)
    Destination: VMware_90:16:26 (00:0c:29:90:16:26)
    Source: VMware_90:16:26 (00:0c:29:90:16:26)
    Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 200.22.247.58, Dst: 85.40.120.202
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 62
    Identification: 0x0000 (0)
    Flags: 0x00
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 255
    Protocol: UDP (17)
    Header Checksum: 0x2e6b [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 200.22.247.58
    Destination Address: 85.40.120.202
User Datagram Protocol, Src Port: 38175, Dst Port: 36895
    Source Port: 38175
    Destination Port: 36895
    Length: 42
    Checksum: 0x4d17 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
    [Timestamps]
    UDP payload (34 bytes)
Data (34 bytes)
```

可以看到，ethernet的src与dst地址正确；ipv4的src, dst, 包长度, TTL, Protocol等均正确；UDP的src, dst, payload长度均正确。因此可验证程序的正确性。

References

[1] 深入浅出dpdk chapter1.7 实例讲解

[2] 深入浅出dpdk chapter6.6 Mbuf与Mempool

[3] https://blog.csdn.net/XuVowkin/article/details/117064512

[4] DPDK总结之常用API https://blog.csdn.net/gerald_jones/article/details/106600175

[5] DPDK基础模块之rte_mbuf详解 https://www.cnblogs.com/ziding/p/4214499.html

[6] https://en.wikipedia.org/wiki/IPv4