

Part 1: Get familiar with DPDK Take DPDK

Q1: What's the purpose of using hugepage?

A1: 可以减少TLB miss的情况，从而提升性能。

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

A2: 启动和初始化基础运行环境 EAL，初始化失败则报错 -> 多核运行初始化，通过 `rte_eal_remote_launch` 遍历所有 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 -> 每个线程运行函数 `lcore_hello` -> 等待结束

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

收发包的两个API为：

```
static inline uint16_t rte_eth_rx_burst(uint16_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(uint16_t port_id, uint16_t queue_id,
                                         struct rte_mbuf **tx_pkts, uint16_t nb_pkts)
```

4个参数意义分别为：端口，队列，报文缓冲区以及收发包数。函数返回值为实际收发的包的数量。

这两个API会在指定的端口和队列，以指定的报文缓冲区收发指定数目的包。

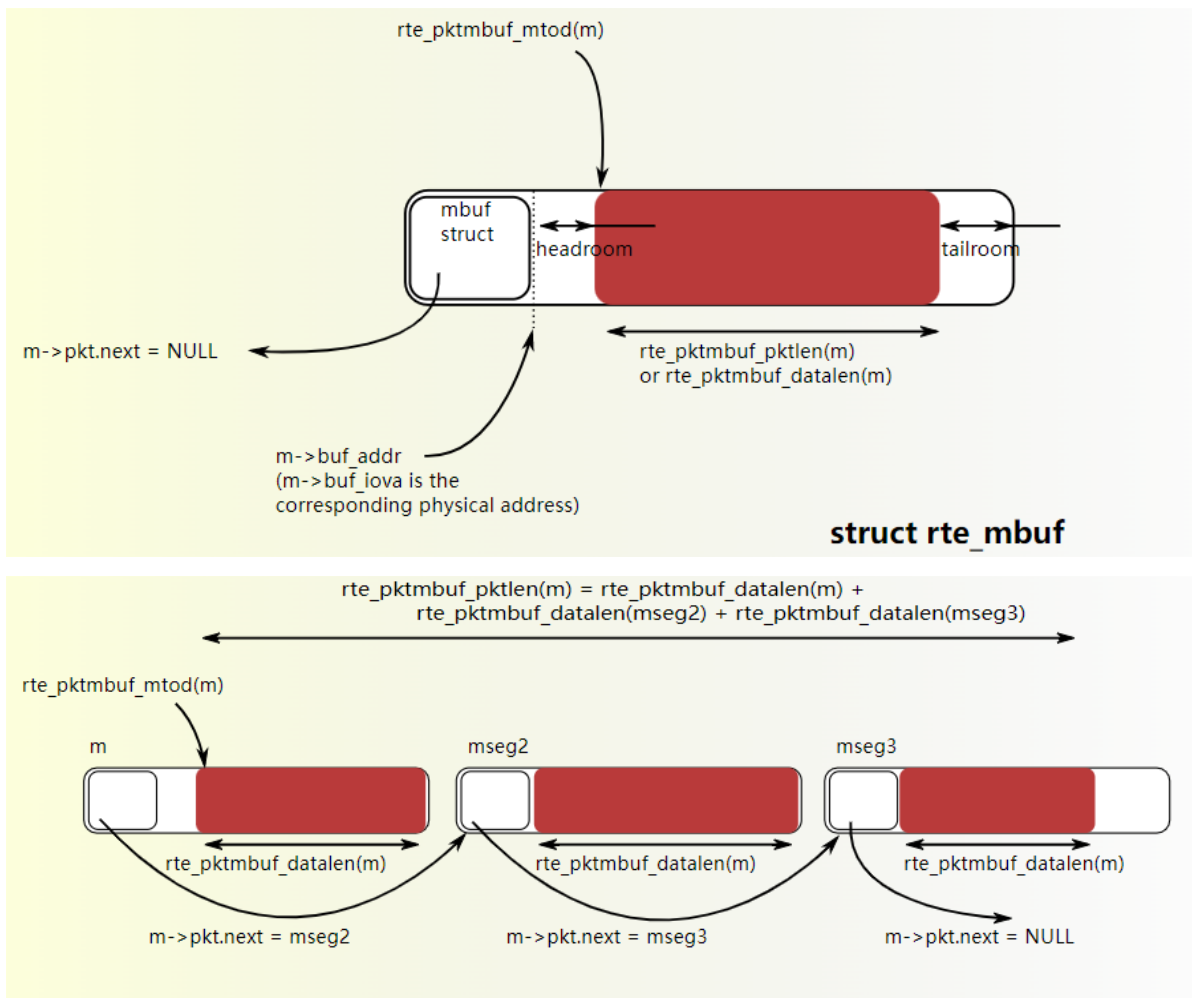
释放用于缓存包的 mbuf 空间的API为：

```
static inline void rte_pktmbuf_free(struct rte_mbuf *m)
```

这个API会将 mbuf 释放回到原先的 mempool 中，可以在发送完包之后调用用于回收空间。

Q4: Describe the data structure of 'rte_mbuf'.

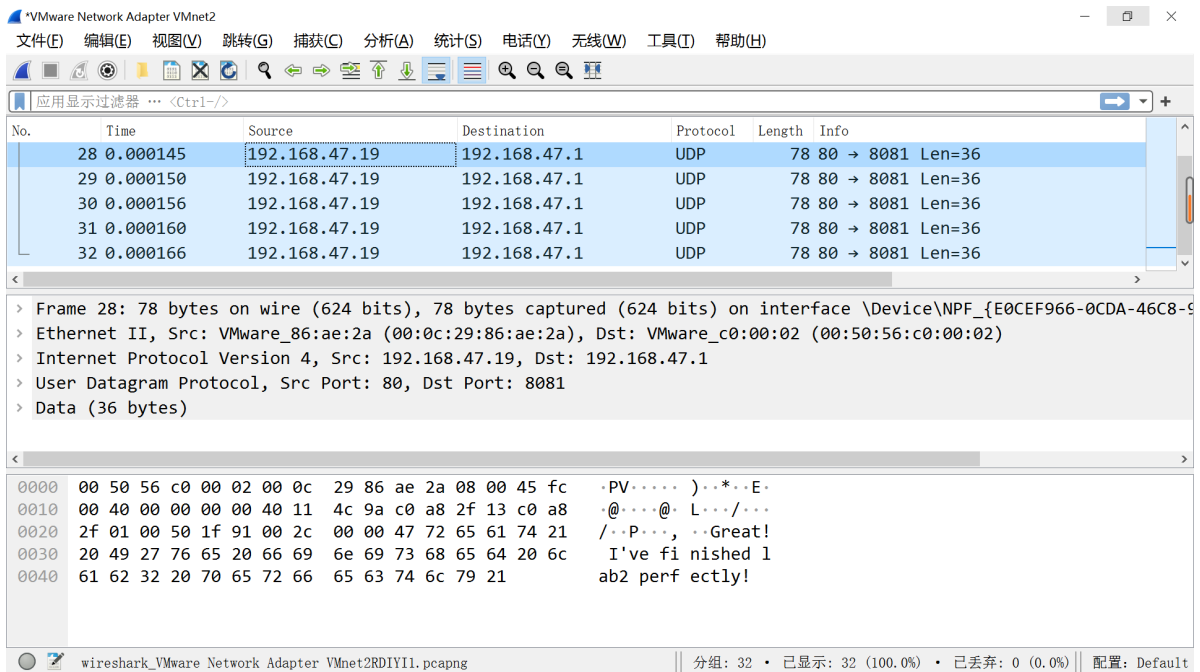
rte_mbuf 可用作存储消息缓存区，它被存储在 mempool 中，其结构如下图所示：



一些指针、成员或函数的内容如下所示，mbuf 结构体的指针简写为 `m`

- 指针 `next`：指向下一个 `rte_mbuf`
- `headroom`：一般用来存放用户自己针对于 mbuf 的一些描述信息，可以通过修改 mbuf 头文件改变 `headroom` 的大小
- 数据 `data`：地址区间在 `buf_addr + data_off` 到 `buf_addr + data_off + data_len`，用于存放数据
- `tailroom`：一般指的是 `data_len` 还未包含的东西
- `m->buf_addr`：headroom 起始地址
- `m->data_off`：data 起始地址相对于 `buf_addr` 的偏移
- `m->pkt_len`：整个 mbuf 链的 data 总长度
- `m->data_len`：实际 data 的长度
- `m->buf_addr+m->data_off`：实际 data 的起始地址
- `rte_pktmbuf_mtod(m)`：同上
- `rte_pktmbuf_data_len(m)`：同 `m->data_len`
- `rte_pktmbuf_pkt_len`：同 `m->pkt_len`
- `rte_pktmbuf_data_room_size`：同 `m->buf_len`
- `rte_pktmbuf_headroom`：headroom 长度

Part 2: send packets with DPDK



在程序中，我们对于MAC地址、IP地址和UDP端口号的设置分别为：

```
struct rte_ether_addr d_addr = {{0x00, 0x50, 0x56, 0xc0, 0x00, 0x02}};
struct rte_ether_addr s_addr = {{0x00, 0x0c, 0x29, 0x86, 0xae, 0x2a}};
ipv4_hdr->src_addr = rte_cpu_to_be_32(RTE_IPV4(192, 168, 47, 19));
ipv4_hdr->dst_addr = rte_cpu_to_be_32(RTE_IPV4(192, 168, 47, 1));
udp_hdr->src_port = rte_cpu_to_be_16(80);
udp_hdr->dst_port = rte_cpu_to_be_16(8081);
```

可以看到，wireshark中捕捉到包的MAC地址、IP地址和UDP端口号均与上述代码中的分别相同。

而捕捉到包的数量，也恰为发送的32个。

而我们发送的内容为：

```
char *data = (char *) (rte_pktmbuf_mtod(bufs[i], char *) + sizeof(struct
rte_ether_hdr) + sizeof(struct rte_ipv4_hdr) + sizeof(struct rte_udp_hdr));
strcpy(data, "Great! I've finished lab2 perfectly!");
```

显然，捕捉到包中的内容"Great! I've finished lab2 perfectly!"与我们想要发送的完全相同。

综上所述，我们的程序是完全正确的！

PS:

我编写了shell脚本用于进行程序运行前的一系列准备工作及方便测试。

若想运行这些脚本，请将本文件夹 `dpdk_lab\` 与文件夹 `dpdk\` 置于同一父目录下：



而后使用 `su` 进入root账户（主要是为了拥有关键开启hugepage），

执行 `chmod +x init.sh` 和 `chmod +x test.sh` 赋予shell脚本可执行权限

而后键入 `.\init.sh` 完成前期配置，再键入 `.\test.sh` 即可进行测试

NOTE:

1. IP包头中的TTL经查询后发现设置为64较为科学
2. 网络字节序是高地址存低位，低地址存高位，是**big-endian**
主机字节序是高地址存高位，低地址存低位，是**little-endian**
这点务必注意，起初给我带来了很大麻烦！
从**little-endian**到**big-endian**的转换，在编写代码时可使用 `rte_cpu_to_be_16` 等进行实现
3. [IP包头结构详解 - 御用闲人 - 博客园\(cnblogs.com\)](http://cnblogs.com/yym579167793/)对于IP包头结构的解释给了我很大帮助

519021910594

陶昱丞

taoyucheng@sjtu.edu.cn