DS-lab2

part1

What's the purpose of using hugepage?

- TLB大小有限，使用大页，在需要相同内存的情况下减少页表项,从而能减少TLB miss的概率
- 如果采用MB作为分页的基本单位，那么只需要一个表项就可以保证不出现TLB不命中的情况；而对于GB为单位的大型程序，可以采用1GB为单位作为分页的基本单位，减少TLB不命中的概率。

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

- 初始化基础运行环境
- 失败报错 or 成功则多核运行初始化
- 运行工作函数线程
- 等待所有线程执行结束

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

- 收包
  - rte_eth_rx_burst函数
- 发包
  - rte_eth_tx_burst
- 两者的参数均为端口号,队列号,缓冲区,收发包数。

Describe the data structure of 'rte_mbuf'.

- 代码如下

```
    struct rte_mbuf *m = _m;
    uint32_t buf_len = mp->elt_size - sizeof(struct rte_mbuf);

    RTE_MBUF_ASSERT(mp->elt_size >= sizeof(struct rte_mbuf));

    memset(m, 0, mp->elt_size);

    /* start of buffer is just after mbuf structure */
    m->buf_addr = (char *)m + sizeof(struct rte_mbuf);
    m->buf_physaddr = rte_mempool_virt2phy(mp, m) +
            sizeof(struct rte_mbuf);
    m->buf_len = (uint16_t)buf_len;

    /* keep some headroom between start of buffer and data */
    m->pkt.data = (char*) m->buf_addr + RTE_MIN(RTE_PKTMBUF_HEADROOM, m-
>buf_len);

    /* init some constant fields */
    m->type = RTE_MBUF_PKT;
```

```
    m->pool = mp;
    m->pkt.nb_segs = 1;
    m->pkt.in_port = 0xff;
```

- rte_mbuf存在buf_addr,buf_len,data_off,pkt,type,pool等字段
  - buf_addr：当前mbuf的虚拟地址
  - buf_len：HEADROOM、DATA、TAILROOM长度相加的数值
  - data_off：标识mbuf的dataroom开始地址到报文起始位置的偏移
- 在结构体后紧接着分配HEADROOM、DATA、TAILROOM
- buf_addr指向HEADROOM，pkt.data则指向DATA

Part2

- 在skeleton原有代码上进行修改，组装出符合要求的UDP数据
- 需要创建二维数组来储存packets，再构造出struct rte_ether_hdr、struct rte_ipv4_hdr和struct rte_udp_hdr的数据，结合起来完成packets的构造
- 核心代码如下

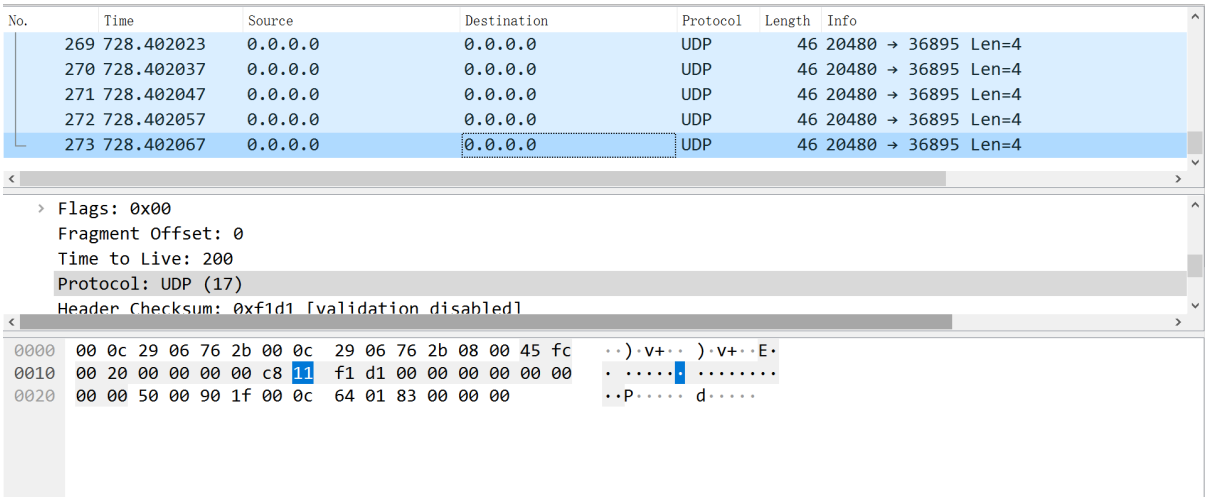
```
        bufs[i] = rte_pktmbuf_alloc(mbuf_pool);
        struct rte_ether_hdr* eth_hdr = rte_pktmbuf_mtod(bufs[i], struct
rte_ether_hdr *);
        struct rte_ipv4_hdr* ip_hdr = (struct rte_ipv4_hdr* )
(rte_pktmbuf_mtod(bufs[i], char *) + sizeof(struct rte_ether_hdr));
        struct rte_udp_hdr* udp_hdr = (struct rte_udp_hdr* )
(rte_pktmbuf_mtod(bufs[i], char *) + sizeof(struct rte_ether_hdr) +
sizeof(struct rte_ipv4_hdr));
        int* data = (int* )(rte_pktmbuf_mtod(bufs[i], char *) +
sizeof(struct rte_ether_hdr) + sizeof(struct rte_ipv4_hdr)+sizeof(struct
rte_udp_hdr));
        struct rte_ether_addr s_addr, d_addr;
        rte_eth_macaddr_get(0, &s_addr);
        rte_eth_macaddr_get(0, &d_addr);

        eth_hdr->d_addr = d_addr;
        eth_hdr->s_addr = s_addr;
        eth_hdr->ether_type = 0x0008;//


        ip_hdr-> version_ihl = RTE_IPV4_VHL_DEF;//
        ip_hdr-> type_of_service = RTE_IPV4_HDR_DSCP_MASK;//
        ip_hdr-> total_length = 0x2000;//
        ip_hdr-> packet_id = 0;
        ip_hdr-> fragment_offset = 0;//
        ip_hdr-> time_to_live = 200;
        ip_hdr-> src_addr = 0;
        ip_hdr-> next_proto_id = 17; //
        ip_hdr-> dst_addr= 0; //
        ip_hdr-> hdr_checksum = rte_ipv4_cksum(ip_hdr);//

        udp_hdr->src_port = 80;
```

```
        udp_hdr->dst_port = 8080;
        udp_hdr->dgram_len = 0x0c00;//
        udp_hdr-> dgram_cksum = 356;
        *data = i + 100;
        bufs[i]->data_len = sizeof(struct rte_ether_hdr) + sizeof(struct
rte_ipv4_hdr)+sizeof(struct rte_udp_hdr) + sizeof(int);
        bufs[i]->pkt_len = sizeof(struct rte_ether_hdr) + sizeof(struct
rte_ipv4_hdr)+sizeof(struct rte_udp_hdr) + sizeof(int);
```

- 其中rte_eth_macaddr_get来获取设备端口对应的MAC地址
- 其中的ether、ip、udp三层，都需要组装，一层都不能少
- udp_hdr-> dgram_cksum 在UDP中没有对应的接口，因此写了一个随机数356；
- 要注意大小端问题
- 结果如下（截取了两次UDP的结果）
  - 

  -