实验二 利用DPDK构造并发送数据包

1. **实验目的**
2. 学会编写和测试DPDK数据包发送程序。
3. 理解DPDK实现高性能数据包处理的原理。
4. **实验内容**
5. 编写一个DPDK程序，实现如下功能：直接构造一个数据包(内容任意)，要求数据包的协议头符合UDP/IP/Ethernet协议规范，并将其发送出去。该程序基于skeleton程序修改得到，附录中给出一个供参考的编程框架。
6. 使用tcpdump命令，抓取数据包，观察各协议字段值，确认实际输出与程序意图是否一致。

过程举例：

1. //在命令行窗口运行如下命令
2. //抓取经过指定网卡含有特定端口号的数据包
3. sudo tcpdump -vv -A -e -i <网卡名称> port <端口号>
4. //然后在另一窗口运行DPDK程序
6. //注意：tcpdump只能在内核驱动网卡上抓取数据包
7. //tcpdump的更多用法可参阅相关资料
8. **回答问题**

官方文档：[DPDK documentation](http://doc.dpdk.org/guides-18.11/), [DPDK API](http://doc.dpdk.org/api-18.11/)

注意：不同版本DPDK对应的文档内容也不完全一样，应对应查看

1. UDP协议头的cksum是必须的吗？不是的话应当如何设置？
2. DPDK为IP协议头的cksum计算提供了专用函数，请在相关文档中找到它，并应用在程序中，调用该函数的注意事项是什么？
3. DPDK实现高性能收发包的原理是什么？
4. **思考题**

参照本实验，一个接收数据包并处理其中数据的DPDK程序应该如何实现？

1. **进展报告**

本实验的完成情况以及对以上3个问题的回答作为第二阶段的进展报告，于 10 月 22日之前提交给助教。10月24日课堂讨论。

进展报告中需注明小组成员（组长排在第一个），以及每位成员对该报告的贡献比例。上课前各小组做好PPT，每个小组派一位同学上台主讲，其余同学可以补充。文件命名及格式：第二阶段\_组长姓名.pdf。

实验过程中遇到任何问题，请及时发邮件给助教（mig@mail.ustc.edu.cn），并抄送一份给主讲老师。

1. **附录**

首先，可以在example文件夹下新建一个文件夹用于本实验，可以看到，example文件夹下其它文件夹下还有Makefile和meson.build这两个文件，各拷贝一份，还需要将其中的文件名对应修改为你的主程序文件名。

注意：

* 1. 主机字节顺序和网络字节顺序的转换
  2. 某些字段值可使用宏定义，如IPVERSION
  3. 可以使用的常用函数如inet\_addr()
  4. 各协议头对应的结构体有不同的版本，既可以使用经典版本(如[Ethernut](http://www.ethernut.de/api/annotated.html))，也可以使用DPDK自带的版本([DPDK\_Data\_Structure](http://doc.dpdk.org/api-18.11/annotated.html))，结构体中包含的数据项可以参阅相关文档

，

以下程序代码基于skeleton修改得来，主要是增加了一个build\_udp\_packet函数，并需要在其中补充相关代码。

1. /\* SPDX-License-Identifier: BSD-3-Clause
2. \* Copyright(c) 2010-2015 Intel Corporation
3. \*/
5. #include <stdint.h>
6. #include <unistd.h>
7. #include <stdbool.h>
8. #include <inttypes.h>
9. #include <rte\_eal.h>
10. #include <rte\_ethdev.h>
11. #include <rte\_cycles.h>
12. #include <rte\_lcore.h>
13. #include <rte\_mbuf.h>
15. #include <pcap/pcap.h>
16. #include <netinet/ip.h>
17. #include <netinet/in.h>
18. #include <netinet/udp.h>
19. #include <arpa/inet.h>
21. #define RX\_RING\_SIZE 1024
22. #define TX\_RING\_SIZE 1024
24. #define NUM\_MBUFS 8191
25. #define MBUF\_CACHE\_SIZE 250
26. #define BURST\_SIZE 32
28. **struct** rte\_mempool \*mbuf\_pool;
30. **static** **const** **struct** rte\_eth\_conf port\_conf\_default = {
31. .rxmode = {
32. .max\_rx\_pkt\_len = ETHER\_MAX\_LEN,
33. },
34. };
36. **static** **inline** **int**
37. port\_init(uint16\_t port, **struct** rte\_mempool \*mbuf\_pool)
38. {
39. **struct** rte\_eth\_conf port\_conf = port\_conf\_default;
40. **const** uint16\_t rx\_rings = 1, tx\_rings = 1;
41. uint16\_t nb\_rxd = RX\_RING\_SIZE;
42. uint16\_t nb\_txd = TX\_RING\_SIZE;
43. **int** retval;
44. uint16\_t q;
45. **struct** rte\_eth\_dev\_info dev\_info;
46. **struct** rte\_eth\_txconf txconf;
48. **if** (!rte\_eth\_dev\_is\_valid\_port(port))
49. **return** -1;
51. rte\_eth\_dev\_info\_get(port, &dev\_info);
52. **if** (dev\_info.tx\_offload\_capa & DEV\_TX\_OFFLOAD\_MBUF\_FAST\_FREE)
53. port\_conf.txmode.offloads |=
54. DEV\_TX\_OFFLOAD\_MBUF\_FAST\_FREE;
56. /\* Configure the Ethernet device. \*/
57. retval = rte\_eth\_dev\_configure(port, rx\_rings, tx\_rings, &port\_conf);
58. **if** (retval != 0)
59. **return** retval;
61. retval = rte\_eth\_dev\_adjust\_nb\_rx\_tx\_desc(port, &nb\_rxd, &nb\_txd);
62. **if** (retval != 0)
63. **return** retval;
65. /\* Allocate and set up 1 RX queue per Ethernet port. \*/
66. **for** (q = 0; q < rx\_rings; q++) {
67. retval = rte\_eth\_rx\_queue\_setup(port, q, nb\_rxd,
68. rte\_eth\_dev\_socket\_id(port), NULL, mbuf\_pool);
69. **if** (retval < 0)
70. **return** retval;
71. }
73. txconf = dev\_info.default\_txconf;
74. txconf.offloads = port\_conf.txmode.offloads;
75. /\* Allocate and set up 1 TX queue per Ethernet port. \*/
76. **for** (q = 0; q < tx\_rings; q++) {
77. retval = rte\_eth\_tx\_queue\_setup(port, q, nb\_txd,
78. rte\_eth\_dev\_socket\_id(port), &txconf);
79. **if** (retval < 0)
80. **return** retval;
81. }
83. /\* Start the Ethernet port. \*/
84. retval = rte\_eth\_dev\_start(port);
85. **if** (retval < 0)
86. **return** retval;
88. /\* Display the port MAC address. \*/
89. **struct** ether\_addr addr;
90. rte\_eth\_macaddr\_get(port, &addr);
91. printf("Port %u MAC: %02" PRIx8 " %02" PRIx8 " %02" PRIx8
92. " %02" PRIx8 " %02" PRIx8 " %02" PRIx8 "\n",
93. port,
94. addr.addr\_bytes[0], addr.addr\_bytes[1],
95. addr.addr\_bytes[2], addr.addr\_bytes[3],
96. addr.addr\_bytes[4], addr.addr\_bytes[5]);
98. /\* Enable RX in promiscuous mode for the Ethernet device. \*/
99. rte\_eth\_promiscuous\_enable(port);
101. **return** 0;
102. }

105. **static** **bool**
106. build\_udp\_packet(**char** \*buf, **int** \*pkt\_size)
107. {
108. //若使用DPDK的协议头结构体，此处结构体名需对应修改
109. **struct** ether\_hdr \*eh = (**struct** ether\_hdr \*)buf;
110. **struct** ip \*ip = (**struct** ip \*)(eh + 1);
111. **struct** udphdr \*udp = (**struct** udphdr \*)(ip + 1);
112. **char** \*payload = (**char** \*)(udp + 1);
114. /\*add your code here\*/
116. \*(payload +  0) = 'h';
117. \*(payload +  1) = 'e';
118. \*(payload +  2) = 'l';
119. \*(payload +  3) = 'l';
120. \*(payload +  4) = 'o';
121. \*(payload +  5) = ',';
122. \*(payload +  6) = ' ';
123. \*(payload +  7) = 'w';
124. \*(payload +  8) = 'o';
125. \*(payload +  9) = 'r';
126. \*(payload + 10) = 'l';
127. \*(payload + 11) = 'd';
128. \*(payload + 12) = '.';
129. \*(payload + 13) = '\0';
131. **return** (**true**);
132. }
134. /\*
135. \* The lcore main. This is the main thread that does the work, construct a
136. \* packet and deliver it.
137. \*/
138. **static** \_\_attribute\_\_((**noreturn**)) **void**
139. lcore\_main(**void**)
140. {
141. uint16\_t port;
143. /\*
144. \* Check that the port is on the same NUMA node as the polling thread
145. \* for best performance.
146. \*/
147. RTE\_ETH\_FOREACH\_DEV(port)
148. **if** (rte\_eth\_dev\_socket\_id(port) > 0 &&
149. rte\_eth\_dev\_socket\_id(port) !=
150. (**int**)rte\_socket\_id())
151. printf("WARNING, port %u is on remote NUMA node to "
152. "polling thread.\n\tPerformance will "
153. "not be optimal.\n", port);
155. printf("\nCore %u is running. [Ctrl+C to quit]\n",
156. rte\_lcore\_id());
158. /\* Run until the application is quit or killed. \*/
159. **for** (;;) {
160. **int** ret;
161. int32\_t pkt\_size = 60;  //为简单起见，这里直接设为一个固定值，也可以改//为由build\_udp\_packet函数返回实际数据包大小
162. **struct** rte\_mbuf \*worker;
164. **do** {
165. worker = rte\_pktmbuf\_alloc(mbuf\_pool);
166. } **while** (unlikely(worker == NULL));
167. worker->nb\_segs = 1;
168. worker->next = NULL;
169. worker->pkt\_len = pkt\_size;
170. worker->data\_len = pkt\_size;
171. build\_udp\_packet(rte\_pktmbuf\_mtod(worker, **void** \*), &pkt\_size);
173. ret = rte\_eth\_tx\_burst(0, 0, &worker, 1);
175. /\* Free unsent packet. \*/
176. **if** (unlikely(ret < 1)) {
177. rte\_pktmbuf\_free(worker);
178. }
179. }
180. }
182. /\*
183. \* The main function, which does initialization and calls the per-lcore
184. \* functions.
185. \*/
186. **int**
187. main(**int** argc, **char** \*argv[])
188. {
190. unsigned nb\_ports;
191. uint16\_t portid;
193. /\* Initialize the Environment Abstraction Layer (EAL). \*/
194. **int** ret = rte\_eal\_init(argc, argv);
195. **if** (ret < 0)
196. rte\_exit(EXIT\_FAILURE, "Error with EAL initialization\n");
198. argc -= ret;
199. argv += ret;
201. nb\_ports = rte\_eth\_dev\_count\_avail();
203. /\* Creates a new mempool in memory to hold the mbufs. \*/
204. mbuf\_pool = rte\_pktmbuf\_pool\_create("MBUF\_POOL", NUM\_MBUFS \* nb\_ports,
205. MBUF\_CACHE\_SIZE, 0, RTE\_MBUF\_DEFAULT\_BUF\_SIZE, rte\_socket\_id());
207. **if** (mbuf\_pool == NULL)
208. rte\_exit(EXIT\_FAILURE, "Cannot create mbuf pool\n");
210. /\* Initialize all ports. \*/
211. RTE\_ETH\_FOREACH\_DEV(portid)
212. **if** (port\_init(portid, mbuf\_pool) != 0)
213. rte\_exit(EXIT\_FAILURE, "Cannot init port %"PRIu16 "\n",
214. portid);
216. **if** (rte\_lcore\_count() > 1)
217. printf("\nWARNING: Too many lcores enabled. Only 1 used.\n");
219. /\* Call lcore\_main on the master core only. \*/
220. lcore\_main();
222. **return** 0;
223. }