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发包程序
3. 理解DPDK实现高性能收发包的原理
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，观察各协议头的字段值，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
5.
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 11 月 2日23:59之前上传到http://202.38.75.243:5050/。11月5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，以备报告。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（linfei@mail.ustc.edu.cn），并抄送一份给主讲老师。

1. **附录**

首先在example目录下新建一个目录用于本实验。可以看到，example/skeleton目录中有Makefile和meson.build这两个文件，各拷贝一份到新建的目录中，将其中的文件名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

1. 某些字段值可使用宏定义，如IPVERSION
2. 可以使用的常用函数如inet\_addr()
3. 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. \*/
4.
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>
14. #include <rte\_ip.h>
15.
16. #include <pcap/pcap.h>
17. #include <netinet/ip.h>
18. #include <netinet/in.h>
19. #include <rte\_ether.h>
20. #include <rte\_udp.h>
21. #include <arpa/inet.h>
22.
23. #define RX\_RING\_SIZE 1024
24. #define TX\_RING\_SIZE 1024
25.
26. #define NUM\_MBUFS 8191
27. #define MBUF\_CACHE\_SIZE 250
28. #define BURST\_SIZE 32
29.
30. **struct** rte\_mempool \*mbuf\_pool;
31.
32. **static** **const** **struct** rte\_eth\_conf port\_conf\_default = {
33. .rxmode = {
34. .max\_rx\_pkt\_len = ETHER\_MAX\_LEN,
35. },
36. };
37.
38. **static** **inline** **int**
39. port\_init(uint16\_t port, **struct** rte\_mempool \*mbuf\_pool)
40. {
41. **struct** rte\_eth\_conf port\_conf = port\_conf\_default;
42. **const** uint16\_t rx\_rings = 1, tx\_rings = 1;
43. uint16\_t nb\_rxd = RX\_RING\_SIZE;
44. uint16\_t nb\_txd = TX\_RING\_SIZE;
45. **int** retval;
46. uint16\_t q;
47. **struct** rte\_eth\_dev\_info dev\_info;
48. **struct** rte\_eth\_txconf txconf;
49.
50. **if** (!rte\_eth\_dev\_is\_valid\_port(port))
51. **return** -1;
52.
53. rte\_eth\_dev\_info\_get(port, &dev\_info);
54. **if** (dev\_info.tx\_offload\_capa & DEV\_TX\_OFFLOAD\_MBUF\_FAST\_FREE)
55. port\_conf.txmode.offloads |=
56. DEV\_TX\_OFFLOAD\_MBUF\_FAST\_FREE;
57.
58. /\* Configure the Ethernet device. \*/
59. retval = rte\_eth\_dev\_configure(port, rx\_rings, tx\_rings, &port\_conf);
60. **if** (retval != 0)
61. **return** retval;
62.
63. retval = rte\_eth\_dev\_adjust\_nb\_rx\_tx\_desc(port, &nb\_rxd, &nb\_txd);
64. **if** (retval != 0)
65. **return** retval;
66.
67. /\* Allocate and set up 1 RX queue per Ethernet port. \*/
68. **for** (q = 0; q < rx\_rings; q++) {
69. retval = rte\_eth\_rx\_queue\_setup(port, q, nb\_rxd,
70. rte\_eth\_dev\_socket\_id(port), NULL, mbuf\_pool);
71. **if** (retval < 0)
72. **return** retval;
73. }
74.
75. txconf = dev\_info.default\_txconf;
76. txconf.offloads = port\_conf.txmode.offloads;
77. /\* Allocate and set up 1 TX queue per Ethernet port. \*/
78. **for** (q = 0; q < tx\_rings; q++) {
79. retval = rte\_eth\_tx\_queue\_setup(port, q, nb\_txd,
80. rte\_eth\_dev\_socket\_id(port), &txconf);
81. **if** (retval < 0)
82. **return** retval;
83. }
84.
85. /\* Start the Ethernet port. \*/
86. retval = rte\_eth\_dev\_start(port);
87. **if** (retval < 0)
88. **return** retval;
89.
90. /\* Display the port MAC address. \*/
91. **struct** ether\_addr addr;
92. rte\_eth\_macaddr\_get(port, &addr);
93. printf("Port %u MAC: %02" PRIx8 " %02" PRIx8 " %02" PRIx8
94. " %02" PRIx8 " %02" PRIx8 " %02" PRIx8 "\n",
95. port,
96. addr.addr\_bytes[0], addr.addr\_bytes[1],
97. addr.addr\_bytes[2], addr.addr\_bytes[3],
98. addr.addr\_bytes[4], addr.addr\_bytes[5]);
99.
100. /\* Enable RX in promiscuous mode for the Ethernet device. \*/
101. rte\_eth\_promiscuous\_enable(port);
102.
103. **return** 0;
104. }
105.
106. **static** **void**
107. build\_udp\_packet(**struct** rte\_mbuf\* worker)
108. {
109. //DPDK
110. **char** \* payload = (**char**\*)rte\_pktmbuf\_append(worker, 14);
111.
112. \*(payload +  0) = 'h';
113. \*(payload +  1) = 'e';
114. \*(payload +  2) = 'l';
115. \*(payload +  3) = 'l';
116. \*(payload +  4) = 'o';
117. \*(payload +  5) = ',';
118. \*(payload +  6) = ' ';
119. \*(payload +  7) = 'w';
120. \*(payload +  8) = 'o';
121. \*(payload +  9) = 'r';
122. \*(payload + 10) = 'l';
123. \*(payload + 11) = 'd';
124. \*(payload + 12) = '.';
125. \*(payload + 13) = '\0';
126.
127. /\* add your code here \*/
128. }
129.
130. /\*
131. \* The lcore main. This is the main thread that does the work, construct a
132. \* packet and deliver it.
133. \*/
134. **static** \_\_attribute\_\_((**noreturn**)) **void**
135. lcore\_main(**void**)
136. {
137. uint16\_t port;
138.
139. /\*
140. \* Check that the port is on the same NUMA node as the polling thread
141. \* for best performance.
142. \*/
143. RTE\_ETH\_FOREACH\_DEV(port)
144. **if** (rte\_eth\_dev\_socket\_id(port) > 0 &&
145. rte\_eth\_dev\_socket\_id(port) !=
146. (**int**)rte\_socket\_id())
147. printf("WARNING, port %u is on remote NUMA node to "
148. "polling thread.\n\tPerformance will "
149. "not be optimal.\n", port);
150.
151. printf("\nCore %u is running. [Ctrl+C to quit]\n",
152. rte\_lcore\_id());
153.
154. /\* Run until the application is quit or killed. \*/
155. **for** (;;) {
156. **int** ret;
157. **struct** rte\_mbuf \*worker;
158.
159. **do** {
160. worker = rte\_pktmbuf\_alloc(mbuf\_pool);
161. } **while** (unlikely(worker == NULL));
162. worker->nb\_segs = 1;
163. worker->next = NULL;
164. build\_udp\_packet(worker);
165.
166. ret = rte\_eth\_tx\_burst(0, 0, &worker, 1);
167.
168. /\* Free unsent packet. \*/
169. **if** (unlikely(ret < 1)) {
170. rte\_pktmbuf\_free(worker);
171. }
172. }
173. }
174.
175. /\*
176. \* The main function, which does initialization and calls the per-lcore
177. \* functions.
178. \*/
179. **int**
180. main(**int** argc, **char** \*argv[])
181. {
182.
183. unsigned nb\_ports;
184. uint16\_t portid;
185.
186. /\* Initialize the Environment Abstraction Layer (EAL). \*/
187. **int** ret = rte\_eal\_init(argc, argv);
188. **if** (ret < 0)
189. rte\_exit(EXIT\_FAILURE, "Error with EAL initialization\n");
190.
191. argc -= ret;
192. argv += ret;
193.
194. nb\_ports = rte\_eth\_dev\_count\_avail();
195.
196. /\* Creates a new mempool in memory to hold the mbufs. \*/
197. mbuf\_pool = rte\_pktmbuf\_pool\_create("MBUF\_POOL", NUM\_MBUFS \* nb\_ports,
198. MBUF\_CACHE\_SIZE, 0, RTE\_MBUF\_DEFAULT\_BUF\_SIZE, rte\_socket\_id());
199.
200. **if** (mbuf\_pool == NULL)
201. rte\_exit(EXIT\_FAILURE, "Cannot create mbuf pool\n");
202.
203. /\* Initialize all ports. \*/
204. RTE\_ETH\_FOREACH\_DEV(portid)
205. **if** (port\_init(portid, mbuf\_pool) != 0)
206. rte\_exit(EXIT\_FAILURE, "Cannot init port %"PRIu16 "\n",
207. portid);
208.
209. **if** (rte\_lcore\_count() > 1)
210. printf("\nWARNING: Too many lcores enabled. Only 1 used.\n");
211.
212. /\* Call lcore\_main on the master core only. \*/
213. lcore\_main();
214.
215. **return** 0;
216. }