Bk3436 tx 数据之后的代码执行流程

本文以bk3436 rw 5.2 协议栈分析tx 的流程。

其相关必要初始化以及函数的注册可以参考2021\_05\_20\_3436\_rx\_flow.docx 里面说明，此文不再说明。

同样本文以fee0s srever为例，以fee4 的ntf 向手机端发送数据。

1. app\_fee0中app\_fee4\_send\_ntf (uint8\_t conidx,uint16\_t len,uint8\_t\* buf)

2. fee0中 fee0s\_exe\_operation🡪

 status = gatt\_srv\_event\_mtp\_send(conidx\_bf, p\_fee0s\_env->user\_lid,

                                                 (p\_buf\_meta->conidx == GAP\_INVALID\_CONIDX),(p\_buf\_meta->att\_idx == FEE0S\_IDX\_FEE4\_VAL\_VAL) ? GATT\_NOTIFY :GATT\_INDICATE,

                                                 p\_buf\_meta->handle, p\_buf, true);

1. gatt\_srv\_event\_mtp\_send-->gatt\_srv\_event\_proc\_create(conidx, user\_lid, dummy, evt\_type, 1, &att\_info, p\_data, &p\_proc);

3.🡪  status = gatt\_proc\_create(conidx, user\_lid, dummy,

                                      (evt\_type == GATT\_NOTIFY ? GATT\_PROC\_NOTIFY: GATT\_PROC\_INDICATE), tx\_length,

                                      (sizeof(gatt\_srv\_event\_proc\_t) + (sizeof(gatt\_att\_t) \* nb\_att)),

                                      (gatt\_proc\_cb) gatt\_srv\_event\_proc\_continue, (gatt\_proc\_t\*\*) &p\_proc);

4.gatt\_proc\_push(conidx, (gatt\_proc\_t\*) p\_proc);🡪  gapc\_sdt\_prepare(&(p\_proc->defer\_timer), conidx, GAPC\_SDT\_GATT\_PROC);

5. gatt\_proc\_continue(conidx, p\_proc, GATT\_PROC\_START, GAP\_ERR\_NO\_ERROR);

6.gapc\_sdt\_defer(&(p\_proc->defer\_timer), status);🡪 co\_djob\_reg(&(gapc\_env.sdt\_job));// reg 之后则 会调用到gapc\_sdt\_djob\_handler

7. gapc\_sdt\_handlers[GETF(p\_hdl->info\_bf, GAPC\_SDT\_CLIENT)].defer;

取到 gatt\_proc\_continue\_defer\_cb

8.gatt\_proc\_continue\_defer\_cb🡪gatt\_proc\_continue\_exe(conidx, p\_proc, state, status);🡪 p\_proc->cb\_continue(conidx, p\_proc, state, status);

9. p\_proc->cb\_continue 指向gatt\_srv\_event\_proc\_continue

10. gatt\_srv\_event\_proc\_continue—>gatt\_proc\_pdu\_send🡪gatt\_bearer\_pdu\_send(conidx, p\_proc->bearer\_lid, p\_proc->token, p\_pdu, p\_data);🡪 l2cap\_chan\_sdu\_send(conidx, token, chan\_lid, 0, p\_sdu);--> l2cap\_chan\_segment\_send(conidx, chan\_lid, &nb\_buffer)

🡪hci\_send\_2\_controller(data\_tx);

10.  p\_chan->p\_cb->cb\_sdu\_sent(conidx, dummy, chan\_lid, GAP\_ERR\_NO\_ERROR, p\_sdu);

\_\_STATIC const l2cap\_chan\_cb\_t gatt\_bearer\_fix\_chan\_cb =

{

        .cb\_sdu\_rx   = gatt\_bearer\_sdu\_rx\_cb,

        .cb\_sdu\_sent = gatt\_bearer\_sdu\_sent\_cb,

};

11. gatt\_bearer\_sdu\_sent\_cb 🡪 gatt\_proc\_continue(conidx, p\_proc, GATT\_PROC\_PDU\_PUSHED\_TO\_LL, status);

🡪 gatt\_proc\_continue\_exe🡪 p\_proc->cb\_continue(conidx, p\_proc, state, status);

12. cb\_continue🡪 gatt\_srv\_event\_proc\_continue 此时 case GATT\_PROC\_PDU\_PUSHED\_TO\_LL:

if(proc\_id == GATT\_PROC\_NOTIFY)

            {

                p2p\_finished = true;

            }

13. p2p\_finished  为 true 则p\_user->p\_cb->srv.cb\_event\_sent(conidx, p\_proc->hdr.user\_lid, p\_proc->hdr.dummy, status);

通知app层 cb\_event\_sent == fee0s\_cb\_event\_sent

14.gatt\_srv\_event\_proc\_continue处理完cb\_event\_sent之后—>  gatt\_proc\_pop(conidx, (gatt\_proc\_t\*) p\_proc, proc\_finished);🡪 gatt\_bearer\_release(conidx, p\_proc);🡪  gatt\_proc\_check\_grant(conidx, p\_proc->bearer\_lid, CO\_BIT(GATT\_PROC\_TYPE\_GET(p\_proc->proc\_id)));

15.到这里 host 层对数据的处理已经结束了，在上面调用了l2cap\_chan\_segment\_send(conidx, chan\_lid, &nb\_buffer)

🡪hci\_send\_2\_controller(data\_tx);把数据送到了control 层，

16.在l2cap\_chan\_segment\_send 函数中，uint16\_t buf\_ptr = ble\_util\_buf\_acl\_tx\_alloc();

                data\_tx->buf\_ptr = buf\_ptr;

17.在 hci\_send\_2\_controller 之后—>hci\_acl\_data\_handler

18.接下来我们到control 层看数据是如何被硬件取到发送出去的。

19.  hci\_acl\_data\_handler

🡪struct ble\_em\_acl\_buf\_elt\* tx\_elt = ble\_util\_buf\_acl\_tx\_elt\_get((uint16\_t) param->buf\_ptr);

从buf\_ptr 中拿到elt 信息。🡪 lld\_con\_data\_tx(link\_id, tx\_elt);

20. co\_list\_push\_back(&con\_par->queue\_acl\_tx, &buf\_elt->hdr);🡪 lld\_con\_tx\_prog(link\_id);

21.在lld\_con\_tx\_prog 在告诉硬件数据位置，填充相关信息。

22.之后硬件在完成tx 之后，在tx isr 中

lld\_con\_tx 🡪 ble\_util\_buf\_acl\_tx\_free(acl\_buf\_elt->buf\_ptr);

会把相关buf free 掉。并发送 ke\_msg\_send\_basic(LLD\_ACL\_TX\_CFM, KE\_BUILD\_ID(TASK\_LLC, link\_id), TASK\_NONE);

int lld\_acl\_tx\_cfm\_handler(ke\_msg\_id\_t const msgid, void \*param,

                           ke\_task\_id\_t const dest\_id, ke\_task\_id\_t const src\_id)

{

    DBG\_FUNC\_ENTER(lld\_acl\_tx\_cfm\_handler);

    uint8\_t link\_id = KE\_IDX\_GET(dest\_id);

    // Inform host that buffer is available

    llc\_hci\_nb\_cmp\_pkts\_evt\_send(link\_id, 1);

    DBG\_FUNC\_EXIT(lld\_acl\_tx\_cfm\_handler);

    return (KE\_MSG\_CONSUMED);

}

23. llc\_hci\_nb\_cmp\_pkts\_evt\_send(link\_id, 1); 通知buf 可以release了。🡪 l2cap\_hci\_nb\_cmp\_pkts\_evt\_handler🡪 l2cap\_chan\_ll\_buf\_release(conidx, p\_event->nb\_comp\_pkt[i]);

24.如果还有数据需要发送则根据if(l2cap\_env.tx\_con\_bf != 0)

        {

            // mark that transmission can be started

            co\_djob\_reg(&(l2cap\_env.tx\_bg\_job));🡪 l2cap\_chan\_tx\_handler

        }

到此，一个 TX 流程就到此结束了。