第十四周实验报告

网络传输机制实验（第一部分）

2015K8009922021
李一苇

一、实验内容

- 实现TCP数据包处理
  - 如何建立连接、关闭连接、处理异常情况
- 实现tcp_sock连接管理函数
  - 类似于socket函数，能够绑定和监听端口，建立和关闭连接

二、实验流程

第一步，实现`tcp_process`函数

1. 收到包flags的RESET置1，需要立即转为`TCP_CLOSED`状态且释放established_list表项资源
2. 在TCP_LISTEN状态过后，开始记录`tsk->snd_una`和`tsk->rcv_next`

   再过一次握手，用`is_tcp_seq_valid`开始检查收到的ack和seq是否合理

```
if (state != TCP_CLOSED && state != TCP_LISTEN) {
        //After TCP_LISTEN, we start record seq_num
        tsk->snd_una = cb->ack;
        tsk->rcv_nxt = cb->seq_end;
        //After TCP_SYN_SENT, we start checking seq_num's validity
        if ((state != TCP_SYN_SENT) && (!is_tcp_seq_valid(tsk, cb))) {
            log(ERROR, "tcp_process(): received packet with invalid seq, drop
it.");
            return ;
        }
    }
```

3. 处理TCP状态机转换，采用下面的形式：

```
    switch (state) {
        case TCP_CLOSED: //CLOSED -> X
            invalid_state(tsk, cb);
            break;
        case TCP_LISTEN: //LISTEN -> SYN_RCVD
            if (cb->flags & TCP_SYN) {
                struct tcp_sock *csk = set_up_child_tsk(tsk, cb);
                tcp_set_state(csk, TCP_SYN_RECV);
                tcp_hash(csk);
```

```
                tcp_send_control_packet(csk, TCP_SYN | TCP_ACK);
            } else invalid_state(tsk, cb);
            break;
        case TCP_SYN_RECV: //SYN_RECV -> ESTABLISHED
            if (cb->flags & TCP_ACK) {
                tcp_set_state(tsk, TCP_ESTABLISHED);
                tcp_sock_listen_dequeue(tsk);
                tcp_sock_accept_enqueue(tsk);
                wake_up(tsk->parent->wait_accept);
            } else invalid_state(tsk, cb);
            break;
        case TCP_SYN_SENT: //SYN_SENT -> ESTABLISHED
            if (cb->flags & (TCP_ACK | TCP_SYN)) {
                tcp_set_state(tsk, TCP_ESTABLISHED);
                wake_up(tsk->wait_connect);
                tcp_send_control_packet(tsk, TCP_ACK);
            } else invalid_state(tsk, cb);
            break;
        case TCP_ESTABLISHED: //ESTABLISHED -> ESTABLISHED or CLOSE_WAIT
            if (cb->flags & TCP_FIN) {
                tcp_set_state(tsk, TCP_CLOSE_WAIT);
                tcp_send_control_packet(tsk, TCP_ACK);
            } else if (cb->flags & TCP_ACK) {
                tcp_send_control_packet(tsk, TCP_ACK);
            } else invalid_state(tsk, cb);
            break;
        case TCP_CLOSE_WAIT: //should not recv packet in CLOSE_WAIT
            log(ERROR, "tcp_process(): peer should not send tcp packet when I'm in
TCP_CLOSE_WAIT.\n");
            invalid_state(tsk, cb);
            break;
        case TCP_LAST_ACK: //LAST_ACK -> CLOSED
            if ((cb->ack == tsk->snd_nxt) && (cb->flags & TCP_ACK)) {
                tcp_unhash(tsk);
                tcp_bind_unhash(tsk);
                tcp_set_state(tsk, TCP_CLOSED);
            }
            else invalid_state(tsk, cb);
            break;
        case TCP_FIN_WAIT_1: //FIN_WAIT_1 -> FIN_WAIT_2
            if ((cb->ack == tsk->snd_nxt) && (cb->flags & TCP_ACK))
                tcp_set_state(tsk, TCP_FIN_WAIT_2);
            else invalid_state(tsk, cb);
            break;
        case TCP_FIN_WAIT_2: //FIN_WAIT_2 -> TIME_WAIT
            if ((cb->ack == tsk->snd_nxt) && (cb->flags & (TCP_FIN | TCP_ACK))) {
                tcp_set_state(tsk, TCP_TIME_WAIT);
                tcp_set_timewait_timer(tsk);
                tcp_send_control_packet(tsk, TCP_ACK);
            }
            else invalid_state(tsk, cb);
            break;
```

```
        case TCP_CLOSING: case TCP_TIME_WAIT: //should not recv packet in CLOSING or
TIME_WAIT
            invalid_state(tsk, cb);
            break;
        default:
            invalid_state(tsk, cb);
            break;
    }
```

第二步，实现类tcp_socket的六个原语

- `struct tcp_sock *alloc_tcp_sock();` 已经由老师实现
- `int tcp_sock_bind(struct tcp_sock *, struct sock_addr *);` 已经由老师实现
- `int tcp_sock_listen(struct tcp_sock *, int);`
  - 设置backlog
  - 转入TCP_LISTEN态
  - 把tsk放入listen_table
- `int tcp_sock_connect(struct tcp_sock *, struct sock_addr *);`
  - 获取本机ip和port
  - 转入TCP_SYN_SENT态
  - 放入bind_table表
  - 等待SYN包
- `struct tcp_sock *tcp_sock_accept(struct tcp_sock *);`
  - 从accept队列取出一个sock，如果没有，阻塞进程
- `void tcp_sock_close(struct tcp_sock *);`
  - 对于主动（客户端）关闭：由TCP_ESTABLISHED转入TCP_FIN_WAIT1
  - 对于服务器端的关闭：由TCP_CLOSE_WAIT传入TCP_LAST_ACK

三、查错心得

1. `tcp_process` 函数里需要在TCP_LISTEN状态过后，开始记录 `tsk->snd_una` 和 `tsk->rcv_next`

   再过一次握手，用 `is_tcp_seq_valid` 开始检查收到的ack和seq是否合理。否则，如果始终检查是否合理，则在建立连接过程中因为seq不合理而程序中断

2. 正确释放tcp_sock的资源：涉及ref_cnt的合理增减

   服务器端：

   执行 `(tcp_sock_bind(tsk, &addr) < 0)` 时对父tsk引用增加

   执行 `tcp_sock_listen` 里的 `tcp_hash` 时父tsk引用增加

   在收到SYN包时，生成子csk，执行 `tcp_hash` 放入Established_list时对csk引用增加

   客户端：

   执行 `tcp_sock_connect` 时对tsk引用增加

   发送SYN包时，执行 `tcp_hash` 放入Established_list时对tsk引用增加

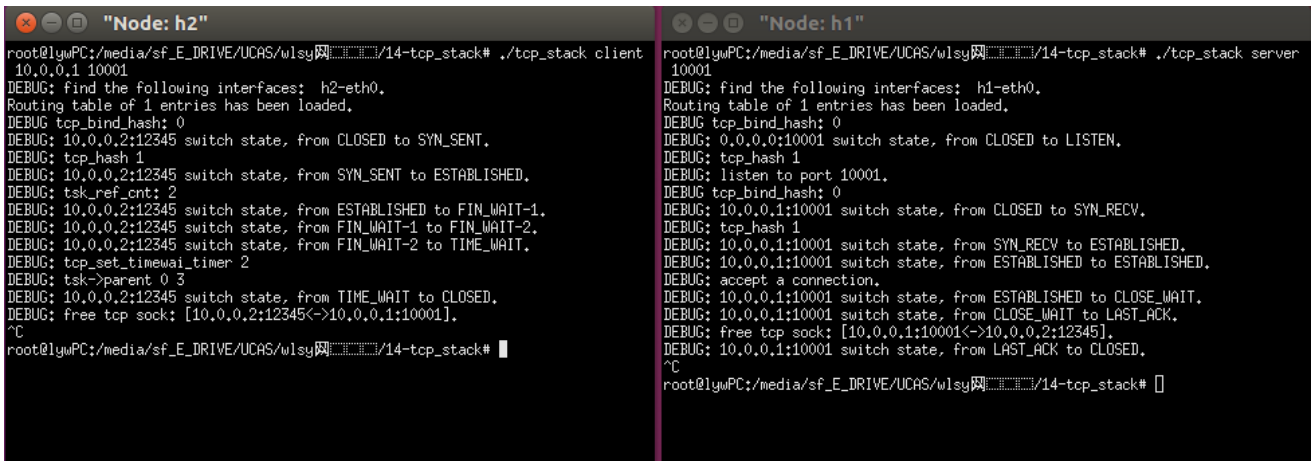
   TIME_WAIT状态时增加计时器对tsk引用增加

四、实验结果和分析

测试方法：运行网络拓扑tcp_topo.py

- 在h1上运行TCP协议栈的服务器模式 `./tcp_stack server 10001`
- 在h2上运行TCP协议栈的客户端模式，连接至h1，显示建立连接成功后自动关闭连接 `./tcp_stack client 10.0.0.1 10001`

截图如下：



用tcp_stack.py替换服务器端或客户端时，另一方仍显示同样的结果，能够证明tcp栈正常工作

抓包的结果如下：

可以看到按照通信过程中的包的flags按照理论课上的流程图变化，侧面验证TCP状态机的正确变化