计算机网络实验 11 报告

学号　2017K8009929059
姓名　於修远

网络传输机制实验一

一、实验内容

- 补全实验代码，实现对 TCP 状态机的状态维护和更新。
- 运行给定网络拓扑，用两个节点分别作为服务器节点和客户端节点，验证握手机制的正确性。

二、实验流程

（一）TCP 状态机切换

TCP 状态机切换的函数是本次实验中被动响应和切换 TCP 传输状态的核心。如下图所示，代码中考虑了状态信号为 SYN 和 SYN|ACK 和 ACK 和 ACK|FIN 和 FIN 共计 5 种可能的情况；每种情况又根据本地 TCP 状态进行分类，使得状态可以被正确切换。

```
tsk->snd_una = cb->ack;
tsk->rcv_nxt = cb->seq_end;
switch (cb->flags) {
    case TCP_SYN:
        if (tsk->state == TCP_LISTEN) {...} else printf( form
        break;
    case (TCP_SYN | TCP_ACK):
        if (tsk->state == TCP_SYN_SENT) {...}
        break;
    case TCP_ACK:
        switch (tsk->state) {...}
        break;
    case (TCP_ACK | TCP_FIN):
        if (tsk->state == TCP_FIN_WAIT_1) {...} else printf(
        break;
    case TCP_FIN:
        switch (tsk->state) {...}
        break;
    default: printf( format: "Unset flag %d\n", cb->flags);
}
```

以最为复杂的接收到 ACK 消息的情况为例进行分析，共有三种状态可能会接收到 ACK 消息，其他状态接收到时输出错误信息。对于 SYN_RECV 状态，需要唤醒待响应的队列，并切换状态至 ESTABLISHED；对于 FIN_WAIT_1 状态，只需切换状态至 FIN_WAIT_2；对于 LAST_ACK 状态，接收到 ACK 消息意味着传输结束，所以置状态为 CLOSED，并释放资源。接收到其他消息的情况不赘述，具体可见代码实现。

```
case TCP_ACK:
    switch (tsk->state) {
        case TCP_SYN_RECV:
            tcp_sock_accept_enqueue(tsk);
            wake_up(tsk->parent->wait_accept);
            tcp_set_state(tsk, state: TCP_ESTABLISHED);
            break;
        case TCP_FIN_WAIT_1:
            tcp_set_state(tsk, state: TCP_FIN_WAIT_2);
            break;
        case TCP_LAST_ACK:
            tcp_set_state(tsk, state: TCP_CLOSED);
            if (!tsk->parent) tcp_bind_unhash(tsk);
            tcp_unhash(tsk);
            break;
        default: printf( format: "Unset state for ACK %d\n", tsk->state);
    }
    break;
```

（二）超时中断函数

本部分的核心函数是`tcp_scan_timer_list`函数，用于定时唤醒扫描超时队列。这一队列中的 TCP 状态描述符在进入 TIME_WAIT 状态时加入队列，达到两倍 MSL 时间后释放资源。

```
void tcp_scan_timer_list()
{
    // TODO: implement %s please.\n, __FUNCTION__
    struct tcp_timer *pos, *q;
    list_for_each_entry_safe(pos, q, head: &timer_list, list) {
        pos->timeout -= TCP_TIMER_SCAN_INTERVAL;
        if (pos->timeout <= 0) {
            list_delete_entry(&pos->list);
            struct tcp_sock *tsk = timewait_to_tcp_sock(pos);
            tcp_set_state(tsk, state: TCP_CLOSED);
            if (!tsk->parent) tcp_bind_unhash(tsk);
            tcp_unhash(tsk);
        }
    }
}
```

（三）socket 管理函数

1、连接建立

。

```
tsk->sk_dip = ntohl(skaddr->ip);
tsk->sk_dport = ntohs(skaddr->port);
tsk->sk_sip = ((iface_info_t*)(instance->iface_list.next))->ip;
if (tcp_sock_set_sport(tsk, port: 0) < 0) {
    printf( format: "No available port!\n");
    return -1;
}
//printf("%u    %u    %u    %u\n", tsk->sk_sip, tsk->sk_dip, ts
tsk->snd_nxt = tsk->iss = tcp_new_iss();
tcp_set_state(tsk, state: TCP_SYN_SENT);
tcp_hash(tsk);

tcp_send_control_packet(tsk, TCP_SYN);
sleep_on(tsk->wait_connect);

tcp_set_state(tsk, state: TCP_ESTABLISHED);
tcp_send_control_packet(tsk, TCP_ACK);

return 0;
```

2、连接关闭

如下图所示，根据触发该函数时的状态区分被动建立方和主动建立方，并分类讨论。

```c
void tcp_sock_close(struct tcp_sock *tsk)
{
    // TODO: implement %s please.\n, __FUNCTION__
    if (tsk->parent) {
        while (tsk->state != TCP_CLOSE_WAIT);
    }
    if (tsk->state == TCP_ESTABLISHED) {
        tcp_set_state(tsk, state: TCP_FIN_WAIT_1);
        tcp_send_control_packet(tsk, TCP_FIN);
    }
    else if (tsk->state == TCP_CLOSE_WAIT) {
        tcp_set_state(tsk, state: TCP_LAST_ACK);
        tcp_send_control_packet(tsk, TCP_FIN);
    }
}
```

三、结果分析

（一）运行结果

如下图所示，执行` tcp_topo.py `脚本后，将 h1 节点作为 server 端，将 h2 节点作为 client 端，建立 tcp 连接。两节点间的交互结果如下图所示，可见连接正常。

```
"Node: h1"
root@cod-VirtualBox:~/workspace/ucas_network_2020/prj11# ./tcp_stack server 100
01
DEBUG: find the following interfaces:  h1-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 0.0.0.0:10001 switch state, from CLOSED to LISTEN.
DEBUG: listen to port 10001.
167772161     167772162     10001     12345
DEBUG: 10.0.0.1:10001 switch state, from CLOSED to SYN_RECV.
167772161     167772162     10001     12345
DEBUG: 10.0.0.1:10001 switch state, from SYN_RECV to ESTABLISHED.
DEBUG: accept a connection.
167772161     167772162     10001     12345
DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to LAST_ACK.
167772161     167772162     10001     12345
DEBUG: 10.0.0.1:10001 switch state, from LAST_ACK to CLOSED.
```

```
"Node: h2"
root@cod-VirtualBox:~/workspace/ucas_network_2020/prj11# wireshark&
[1] 4628
root@cod-VirtualBox:~/workspace/ucas_network_2020/prj11# ./tcp_stack client 10.
0.0.1 10001
DEBUG: find the following interfaces:  h2-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 10.0.0.2:12345 switch state, from CLOSED to SYN_SENT.
167772162     167772161     12345     10001
DEBUG: 10.0.0.2:12345 switch state, from SYN_SENT to ESTABLISHED.
DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1.
167772162     167772161     12345     10001
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-1 to TIME_WAIT.
DEBUG: 10.0.0.2:12345 switch state, from TIME_WAIT to CLOSED.
```

通过 wireshark 查看节点 h2 的收发包情况，可以看到 TCP 连接正常建立到关闭的过程。

4	0.020797157	10.0.0.2	10.0.0.1	TCP	54	12345 → 10001 [SYN] Seq=0
5	0.031225382	10.0.0.1	10.0.0.2	TCP	54	10001 → 12345 [SYN, ACK] S
6	0.041948614	10.0.0.2	10.0.0.1	TCP	54	12345 → 10001 [ACK] Seq=1
7	1.042122389	10.0.0.2	10.0.0.1	TCP	54	12345 → 10001 [FIN] Seq=1
8	1.052429576	10.0.0.1	10.0.0.2	TCP	54	10001 → 12345 [FIN, ACK] S
9	1.062787131	10.0.0.2	10.0.0.1	TCP	54	12345 → 10001 [ACK] Seq=2