

- 网络传输机制实验（二）
 - 实验内容
 - 实验内容一
 - 实验内容二
 - 设计思路
 - 实现数据传输
 - tcp_sock_read 函数
 - handle_recv_data 函数
 - 更新后的 tcp_process 函数
 - tcp_sock_write 函数
 - 结果验证
 - 实验一结果
 - 实验二结果

网络传输机制实验（二）

2020年12月30日

蔡润泽

本实验 [Github](#) 地址

实验内容

实验内容一

- 运行给定网络拓扑(tcp_topo.py)
- 在节点h1上执行TCP程序
 - 执行脚本(disable_tcp_rst.sh, disable_offloading.sh)，禁止协议栈的相应功能
 - 在h1上运行TCP协议栈的服务器模式 (./tcp_stack server 10001)
- 在节点h2上执行TCP程序
 - 执行脚本(disable_tcp_rst.sh, disable_offloading.sh)，禁止协议栈的相应功能
 - 在h2上运行TCP协议栈的客户端模式，连接h1并正确收发数据 (./tcp_stack client 10.0.0.1 10001)
 - client向server发送数据，server将数据echo给client
- 使用tcp_stack.py替换其中任意一端，对端都能正确收发数据

实验内容二

- 修改tcp_apps.c(以及tcp_stack.py)，使之能够收发文件
- 执行create_randfile.sh，生成待传输数据文件client-input.dat

- 运行给定网络拓扑(tcp_topo.py)
- 在节点h1上执行TCP程序
 - 执行脚本(disable_tcp_rst.sh, disable_offloading.sh), 禁止协议栈的相应功能
 - 在h1上运行TCP协议栈的服务器模式 (./tcp_stack server 10001)
- 在节点h2上执行TCP程序
 - 执行脚本(disable_tcp_rst.sh, disable_offloading.sh), 禁止协议栈的相应功能
 - 在h2上运行TCP协议栈的客户端模式 (./tcp_stack client 10.0.0.1 10001)
 - Client发送文件client-input.dat给server, server将收到的数据存储在文件server-output.dat
- 使用md5sum比较两个文件是否完全相同
- 使用tcp_stack.py替换其中任意一端, 对端都能正确收发数据

设计思路

实现数据传输

tcp_sock_read 函数

负责接收TCP数据, 若ring buffer为空, 则睡眠, 当收到数据包时则被唤醒。具体实现如下:

```
int tcp_sock_read(struct tcp_sock *tsk, char *buf, int len) {
    while (ring_buffer_empty(tsk->rcv_buf)) {
        sleep_on(tsk->wait_rcv);
    }
    int rlen = read_ring_buffer(tsk->rcv_buf, buf, len);
    wake_up(tsk->wait_rcv);
    return rlen;
}
```

handle_rcv_data 函数

该函数负责接收TCP数据包中的数据, 根据ACK的值添加进ring buffer。另外, 若ring buffer为满, 则睡眠。具体实现如下:

```
void handle_rcv_data(struct tcp_sock *tsk, struct tcp_cb *cb) {
    while (ring_buffer_full(tsk->rcv_buf)) {
        sleep_on(tsk->wait_rcv);
    }
    write_ring_buffer(tsk->rcv_buf, cb->payload, cb->pl_len);
    wake_up(tsk->wait_rcv);
    tsk->rcv_next = cb->seq + cb->pl_len;
    tsk->snd_una = cb->ack;
    tcp_send_control_packet(tsk, TCP_ACK);
}
```

更新后的 tcp_process 函数

相较于上周的实验，本周的实验需要对该函数进行补充，以支持接收TCP数据包。更新部分的代码如下：

```
if (tsk->state == TCP_ESTABLISHED) {
    if (tcp->flags & TCP_FIN) {
        tcp_set_state(tsk, TCP_CLOSE_WAIT);
        tsk->rcv_nxt = cb->seq + 1;
        tcp_send_control_packet(tsk, TCP_ACK);
    } else if (tcp->flags & TCP_ACK) {
        if (cb->pl_len == 0) {
            tsk->snd_una = cb->ack;
            tsk->rcv_nxt = cb->seq + 1;
            tcp_update_window_safe(tsk, cb);
        } else {
            handle_rcv_data(tsk, cb);
        }
    }
}
```

tcp_sock_write 函数

负责发送TCP数据包。如果对端recv_window允许，则发送数据，每次读取1个数据包大小的数据，即 $\min(\text{data_len}, 1514 - \text{ETHER_HDR_SIZE} - \text{IP_HDR_SIZE} - \text{TCP_HDR_SIZE})$ 。然后封装数据包，通过IP层发送函数，将数据包发出去。具体实现如下：

```
int tcp_sock_write(struct tcp_sock *tsk, char *buf, int len) {
    int single_len = 0;
    int init_seq = tsk->snd_una;
    int init_len = len;

    while (len > 1514 - ETHER_HDR_SIZE - IP_BASE_HDR_SIZE - TCP_BASE_HDR_SIZE) {
        single_len = min(len, 1514 - ETHER_HDR_SIZE - IP_BASE_HDR_SIZE - TCP_BA
        send_data(tsk, buf + (tsk->snd_una - init_seq), single_len);
        sleep_on(tsk->wait_send);
        len -= single_len;
    }

    send_data(tsk, buf + (tsk->snd_una - init_seq), len);
    return init_len;
}
```

其中调用的 send_data 函数如下，负责调用tcp_send_packet函数发送数据包：

```

void send_data(struct tcp_sock *tsk, char *buf, int len) {
    int send_packet_len = ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + TCP_BASE_HDR_SIZE + 1;
    char * packet = (char *)malloc(send_packet_len);
    memcpy(packet + ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + TCP_BASE_HDR_SIZE, buf, len);
    tsk->snd_wnd = len;
    tcp_send_packet(tsk, packet, send_packet_len);
}

```

结果验证

实验一结果

实验结果如下：

| "Node: h2" | "Node: h1" |
|--|---|
| state: ESTABLISHED | Find no established tsk. |
| flags: 0x18 | flags: 0x2 |
| state: ESTABLISHED | state: LISTEN |
| server echoes: 123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ | DEBUG: 0.0.0.0:10001 switch state, from LISTEN to SYN_RECV. |
| flags: 0x10 | flags: 0x10 |
| state: ESTABLISHED | state: SYN_RECV |
| flags: 0x18 | DEBUG: 10.0.0.1:10001 switch state, from SYN_RECV to ESTABLISHED. |
| state: ESTABLISHED | DEBUG: accept a connection. |
| server echoes: 23456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ | flags: 0x18 |
| flags: 0x10 | state: ESTABLISHED |
| state: ESTABLISHED | flags: 0x10 |
| flags: 0x18 | state: ESTABLISHED |
| state: ESTABLISHED | flags: 0x18 |
| server echoes: 3456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ | state: ESTABLISHED |
| flags: 0x10 | flags: 0x10 |
| state: ESTABLISHED | state: ESTABLISHED |
| flags: 0x18 | flags: 0x18 |
| state: ESTABLISHED | state: ESTABLISHED |
| server echoes: 456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ | flags: 0x10 |
| flags: 0x10 | state: ESTABLISHED |
| state: ESTABLISHED | flags: 0x18 |
| flags: 0x18 | state: ESTABLISHED |
| state: ESTABLISHED | flags: 0x10 |
| server echoes: 56789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ | state: ESTABLISHED |
| flags: 0x10 | flags: 0x18 |
| state: ESTABLISHED | state: ESTABLISHED |
| flags: 0x18 | flags: 0x10 |
| state: ESTABLISHED | state: ESTABLISHED |
| server echoes: 6789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ | flags: 0x18 |
| flags: 0x10 | state: ESTABLISHED |
| state: ESTABLISHED | flags: 0x10 |
| flags: 0x18 | state: ESTABLISHED |
| state: ESTABLISHED | flags: 0x18 |
| server echoes: 789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ | state: ESTABLISHED |
| flags: 0x10 | flags: 0x10 |
| state: ESTABLISHED | state: ESTABLISHED |
| flags: 0x18 | flags: 0x18 |
| state: ESTABLISHED | state: ESTABLISHED |
| server echoes: 89abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ | flags: 0x10 |
| flags: 0x10 | state: ESTABLISHED |
| state: ESTABLISHED | flags: 0x18 |
| flags: 0x18 | state: ESTABLISHED |
| state: ESTABLISHED | flags: 0x10 |
| server echoes: 9abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ | state: ESTABLISHED |
| TODO: implement tcp_sock_close here. | flags: 0x18 |
| DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1. | state: ESTABLISHED |
| ***** send FIN ***** | flags: 0x10 |
| flags: 0x10 | state: ESTABLISHED |
| state: FIN_WAIT-1 | flags: 0x11 |
| DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-1 to FIN_WAIT-2. | state: ESTABLISHED |
| | DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT. |

上图可知，可知本次实验结果符合预期。每次能echo正确的值。

另外，若h2用tcp_stack.py运行client，h1不管是运行本设计的server还是利用用tcp_stack.py运行server，结果都是echo出整个字符串“0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ”，结果也符合预期。

实验二结果

实验结果如下：

| "Node: h2" | "Node: h1" |
|---|--|
| remained wlen:3632 update windows remained wlen:2632 update windows remained wlen:1632 update windows remained wlen:632 update windows remained wlen:-368 TODO: implement tcp_sock_close here. DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1. ***** send FIN ***** DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-1 to FIN_WAIT-2. DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-2 to TIME_WAIT. TODO: implement tcp_set_timewait timer here. DEBUG: 10.0.0.2:12345 switch state, from TIME_WAIT to CLOSED. TODO: implement free_tcp_sock here. ^C root@ubuntu:~/code/16-tcp_stack-2# md5sum client-input.dat 8413a9f87645c7edbcd294c782353399 client-input.dat root@ubuntu:~/code/16-tcp_stack-2# md5sum server-output.dat 8413a9f87645c7edbcd294c782353399 server-output.dat root@ubuntu:~/code/16-tcp_stack-2# | ptr:4051000 update windows ptr:4052000 update windows DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT. ptr:4052632 wsize:632, len:632 PbzzJ8lUSZcH45vVzigptdQa340pSoPd0tWQZn/v+4iQv8o UDco2CkTavCcIt3lbYAYrDRXag8pbcnPUNsFE+4Po1S0lt0K0JP8D6q0AKnbxq3SVGN1xak52qBG ozWbeWK56FzPUn900T9FXT0xawMfJU12oDGxBu0V8ynIUXmymPJkKwZ+4i8F3wx4SQpsK6ewkh+g+ gW/43yN0ZBYW33PR/BpqrR7GoqEdsgvhWhgRQn3Q8tDu/7DJ/+WssAvXi5eKCCpNWh90db4IN1ut FwaG4e5P6oRx03YwtC2DA4Bdmhw2BWzHm+LSKSym7L8gEjDa6TtSYXysxdE7qX3JFz4Rx0Vvj6wC 1cEuVIPusJ7MC++GUqOUSxA4hG0JNQZhGIEuU4geHroJwFrhpZmBuKVbbsWZrk6Ac2u7Ry6HIyZ 2S9vKbUq2zFvN6DXgGb/UXacoJpnLfmiW4da8DkEmSnmQ6NdLNGzQ2exbCNS1gQ87XSQ0PUrcsns YPhsJ64+4M0vstMa8R40WtQGMcG9HbHrrhEdmor2tRE0J/SQZMkcziEhnXVwX0Ze7G/jaaQVEwid +XdHdsIezLT2guIc0aEI0jn9qxKauXGir1FBEnxSoVeG DEBUG: close this connection. TODO: implement tcp_sock_close here. DEBUG: 10.0.0.1:10001 switch state, from CLOSE_WAIT to LAST_ACK. DEBUG: 10.0.0.1:10001 switch state, from LAST_ACK to CLOSED. TODO: implement free_tcp_sock here. □ |

上图可知，可知本次实验结果符合预期，客户端发送的文件与服务器端接受的文件一致。