计算机网络研讨课实验报告

冯吕 2015K8009929049

2018 年 7 月 6 日

1 实验题目

网络传输机制实验三

2 实验内容

在本次实验中，需要通过超时重传机制实现有丢包环境下的可靠传输;

3 实验流程

3.1 增加 *tcp_sock* 结构

在有丢包环境下，当一个包发送出去之后，不能直接丢弃，而需要先保存下来，直到 ACK 之后才能删除，同时，需要进行超时重传，因此，在 *tcp_sock* 中增加如下三个结构:

```
struct tcp_sock{
  ...
  struct tcp_timer retrans_timer;
  struct list_head send_buf;
  struct list_head ofo_buf;
  ...
};
```

所有发送出去未确认的包需要先保存在 *send_buf* 中，同时，当收到不连续的包时，先保存到 *ofo_buf* 中，当发送一个包出去之后，启动定时器。

存储于 *send_buf* 和 *ofo_buf* 中的数据结构如下:

```
struct send_packet{
        struct list_head list;
        char *packet;
        int len;
};

struct ofo_packet{
        struct list_head list;
        char *packet;
        int len;
```

```
11          int seq_num;
12  };
```

3.2 修改 *tcp_send_packet*

在发送包的时候，需要将包保存到 *send_buf* 中，因此，需要修改 *tcp_send_packet* 函数，将包保存下来，同时启动定时器：

```
1  void tcp_send_packet(struct tcp_sock *tsk, char *packet, int len)
2  {
3      ...
4
5          struct send_packet *send_pkt = (struct send_packet
6          *)malloc(sizeof(struct send_packet));
7          send_pkt->packet = (char *)malloc(len);
8          send_pkt->len = len;
9          memcpy(send_pkt->packet, packet, len);
10         list_add_tail(&send_pkt->list, &tsk->send_buf);
11         tcp_set_retrans_timer(tsk);
12
13         ip_send_packet(packet, len);
14 }
```

相应地，*tcp_send_control_packet* 函数也需要修改：如果发送的包为 $TCP\_SYN \mid TCP\_ACK$ 包，则将其保存到 *send_buf* 中，启动定时器；

3.3 删除 *ACK* 数据

当收到 *ACK* 数据包之后，将 *send_buf* 中 $seq\_end < ack$ 的数据包删除，同时关闭定时器：

```
1  void remove_ack_data(struct tcp_sock *tsk, int ack_num){
2          tcp_remove_retrans_timer(tsk);
3          struct send_packet *pos, *q;
4          list_for_each_entry_safe(pos, q, &tsk->send_buf, list){
5                  struct tcphdr *tcp = packet_to_tcp_hdr(pos->packet);
6                  struct iphdr *ip = packet_to_ip_hdr(pos->packet);
7                  if (ack_num >= ntohl(tcp->seq)){
8                          tsk->snd_wnd += (ntohs(ip->tot_len) - IP_HDR_SIZE(ip) - TCP_H
9                          free(pos->packet);
10                         list_delete_entry(&pos->list);
11                 }
12         }
13         if (!list_empty(&tsk->send_buf)){
14                 tcp_set_retrans_timer(tsk);
15         }
16 }
```

3.4 重传实现

本次实验中，需要通过定时器重传，超时之后，重新发送，如果重传次数超过三次依旧失败，则关闭连接。

设置定时器与关闭定时器：

```
void tcp_set_retrans_timer(struct tcp_sock *tsk){
        struct tcp_timer *timer = &tsk->retrans_timer;

        timer->type = 1;
        timer->timeout = TCP_RETRANS_INTERVAL;
        timer->retrans_number = 0;

        list_add_tail(&timer->list, &timer_list);
}

void tcp_remove_retrans_timer(struct tcp_sock *tsk){
        list_delete_entry(&tsk->retrans_timer.list);
}
```

在 *tcp_timer* 结构中增加一项记录重传次数，发送数据包时启动定时器。需要修改 *tcp_scan_timer_list*:

```
void tcp_scan_timer_list()
{
        struct tcp_sock *tsk;
        struct tcp_timer *t, *q;
        list_for_each_entry_safe(t, q, &timer_list, list) {
                t->timeout -= TCP_TIMER_SCAN_INTERVAL;
                if (t->timeout <= 0 && t->type == 0) {
                        list_delete_entry(&t->list);

                        // only support time wait now
                        tsk = timewait_to_tcp_sock(t);
                        if (! tsk->parent)
                                tcp_bind_unhash(tsk);
                        tcp_set_state(tsk, TCP_CLOSED);
                        free_tcp_sock(tsk);
                }
                else if(t->timeout <= 0 && (t->type == 1)){
                        tsk = retranstimer_to_tcp_sock(t);
            if(t->retrans_number == 3){
                tcp_sock_close(tsk);
                //free_tcp_sock(tsk);
                return ;
            }
            struct send_packet *buf_pac = list_entry(tsk->send_buf.next, struct send_
```

```
26              if(!list_empty(&tsk->send_buf)){
27                  char *packet = (char *)malloc(buf_pac->len);
28                  memcpy(packet, buf_pac->packet, buf_pac->len);
29                  /*struct tcphdr *tcp = packet_to_tcp_hdr(packet);*/
30                  /*struct iphdr *ip = packet_to_ip_hdr(packet);*/
31
32                  ip_send_packet(packet, buf_pac->len);
33                  t->timeout = TCP_RETRANS_INTERVAL;
34
35                          for(int i = 0; i <= t->retrans_number; i++){
36                              t->timeout *= 2;
37                          }
38                          t->retrans_number ++;
39                      }
40                  }
41          }
42  }
```

- (1) 判断定时器类型，如果类型为 *wait*，则根据是否 *timeout* 关闭即可，如果类型为 *retrans*，转 (2);
- (2) 判断重传次数是否小于 3，如果不小于，则关闭该 *timer* 对应的 *socket*，否则转 (3);
- (3) 重传 *send_buf* 中的第一个包，更新定时器：$timeout * 2, retrans\_number + 1$

4 实验结果

能够建立连接并传输数据，但有时候会传输失败。

5 结果分析

可能会出现连续三次都丢包的情况，导致数据传输失败。