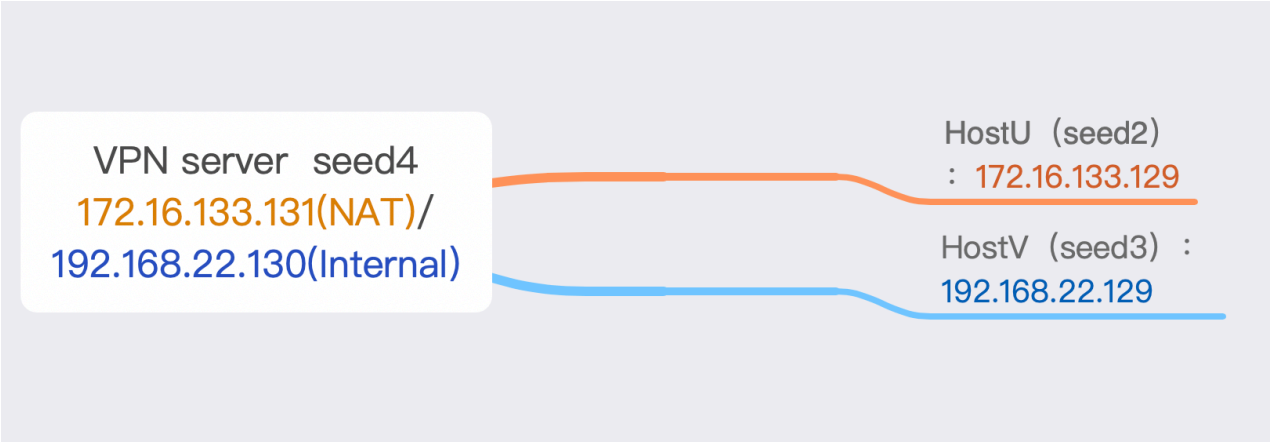
Virtual Private Network (VPN) Lab

18307130089 吴嘉琪

Task 1: VM Setup



Task 2: Creating a VPN Tunnel using TUN/TAP

在Host U 和 server上运行：

```
sudo route add -net 192.168.22.0/24 tun0
```

Step 4: Set Up Routing on Host V

Host U发出的包：

```
  1 2020-12-20 05:48:41.2980084… 192.168.53.5          192.168.22.129
  2 2020-12-20 05:48:42.3221284… 192.168.53.5          192.168.22.129
  3 2020-12-20 05:48:43.3465457… 192.168.53.5          192.168.22.129
  4 2020-12-20 05:48:44.3688587… 192.168.53.5          192.168.22.129
  5 2020-12-20 05:48:45.3936426… 192.168.53.5          192.168.22.129
  6 2020-12-20 05:48:46.4182904… 192.168.53.5          192.168.22.129
```

和上个lab的问题一样，如果将server端的路由配置加入

```
sudo route add -net 192.168.53.0/24 tun0
```

看似能成功，其实因为上述tun0的ip地址是虚拟的，无法ARP查询到mac地址，所以实际上会失败；

于是这里还是采用`sudo iptables -t nat -A POSTROUTING -j MASQUERADE -o ens33`的方法。

Step 5: Test the VPN Tunnel:

在Host U ping192.168.22.129：

成功通过tun0建立连接

```
[12/20/20]seed@VM:~$ ping 192.168.22.129
PING 192.168.22.129 (192.168.22.129) 56(84) bytes of data.
64 bytes from 192.168.22.129: icmp_seq=1 ttl=128 time=0.630 ms
64 bytes from 192.168.22.129: icmp_seq=2 ttl=128 time=1.28 ms
64 bytes from 192.168.22.129: icmp_seq=3 ttl=128 time=0.971 ms
64 bytes from 192.168.22.129: icmp_seq=4 ttl=128 time=0.822 ms
64 bytes from 192.168.22.129: icmp_seq=5 ttl=128 time=0.663 ms
64 bytes from 192.168.22.129: icmp_seq=6 ttl=128 time=0.598 ms
64 bytes from 192.168.22.129: icmp_seq=7 ttl=128 time=0.634 ms
```

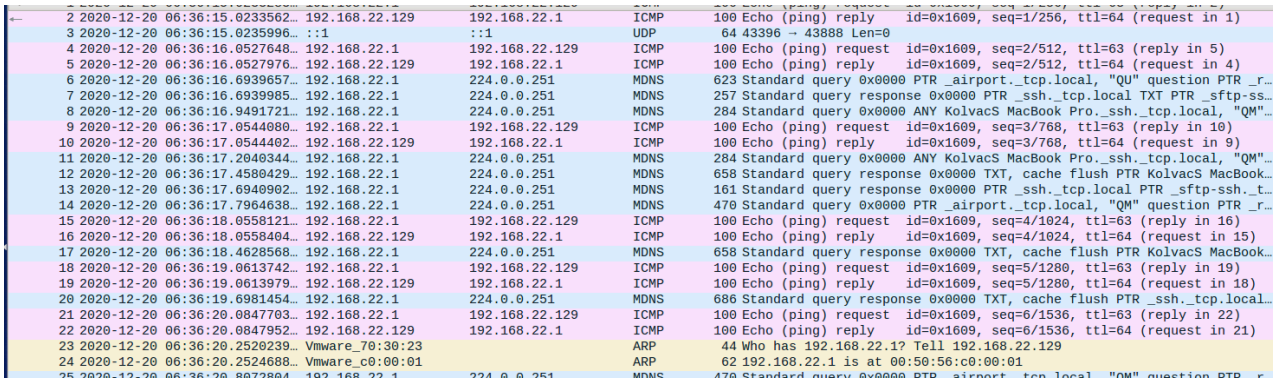
Host U

```
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
```

server：

```
Got a packet from the tunnel
Got a packet from the tunnel
Got a packet from the tunnel
Got a packet from the tunnel
Got a packet from the tunnel
Got a packet from the tunnel
Got a packet from the tunnel
Got a packet from the tunnel
Got a packet from the tunnel
Got a packet from the tunnel
Got a packet from the tunnel
Got a packet from the tunnel
```

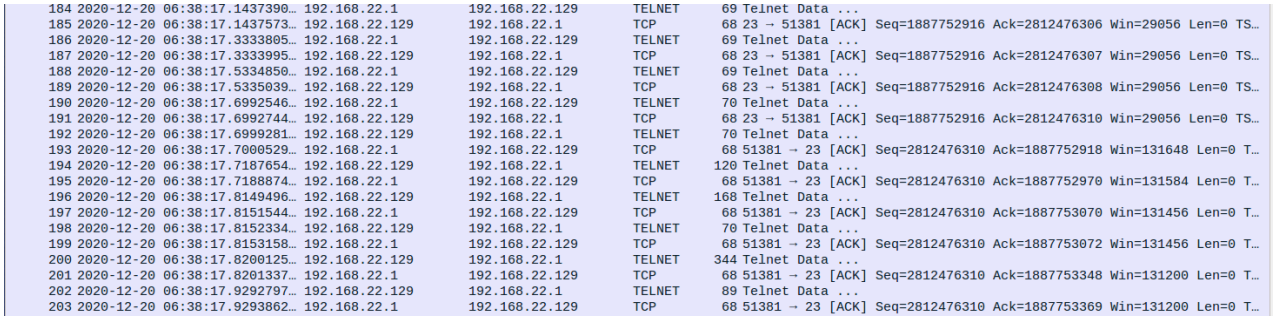
Host V中的 wireshark查看：



←	2	2020-12-20 06:36:15.0233562…	192.168.22.129	192.168.22.1	ICMP	100 Echo (ping) reply id=0x1609, seq=1/256, ttl=64 (request in 1)
	3	2020-12-20 06:36:15.0235996…	::1	::1	UDP	64 43396 → 43888 Len=0
	4	2020-12-20 06:36:16.0527648…	192.168.22.1	192.168.22.129	ICMP	100 Echo (ping) request id=0x1609, seq=2/512, ttl=63 (reply in 5)
	5	2020-12-20 06:36:16.0527976…	192.168.22.129	192.168.22.1	ICMP	100 Echo (ping) reply id=0x1609, seq=2/512, ttl=64 (request in 4)
	6	2020-12-20 06:36:16.6939657…	192.168.22.1	224.0.0.251	MDNS	623 Standard query 0x0000 PTR _airport._tcp.local, "QU" question PTR _r…
	7	2020-12-20 06:36:16.6939985…	192.168.22.1	224.0.0.251	MDNS	257 Standard query response 0x0000 PTR _ssh._tcp.local TXT PTR _sftp-ss…
	8	2020-12-20 06:36:16.9491721…	192.168.22.1	224.0.0.251	MDNS	284 Standard query 0x0000 ANY KolvacS MacBook Pro._ssh._tcp.local, "QM"…
	9	2020-12-20 06:36:17.0544080…	192.168.22.1	192.168.22.129	ICMP	100 Echo (ping) request id=0x1609, seq=3/768, ttl=63 (reply in 10)
	10	2020-12-20 06:36:17.0544402…	192.168.22.129	192.168.22.1	ICMP	100 Echo (ping) reply id=0x1609, seq=3/768, ttl=64 (request in 9)
	11	2020-12-20 06:36:17.2040344…	192.168.22.1	224.0.0.251	MDNS	284 Standard query 0x0000 ANY KolvacS MacBook Pro._ssh._tcp.local, "QM"…
	12	2020-12-20 06:36:17.4580429…	192.168.22.1	224.0.0.251	MDNS	658 Standard query response 0x0000 TXT, cache flush PTR KolvacS MacBook…
	13	2020-12-20 06:36:17.6940902…	192.168.22.1	224.0.0.251	MDNS	161 Standard query response 0x0000 PTR _ssh._tcp.local PTR _sftp-ssh._t…
	14	2020-12-20 06:36:17.7964638…	192.168.22.1	224.0.0.251	MDNS	470 Standard query 0x0000 PTR _airport._tcp.local, "QM" question PTR _r…
	15	2020-12-20 06:36:18.0558121…	192.168.22.1	192.168.22.129	ICMP	100 Echo (ping) request id=0x1609, seq=4/1024, ttl=63 (reply in 16)
	16	2020-12-20 06:36:18.0558404…	192.168.22.129	192.168.22.1	ICMP	100 Echo (ping) reply id=0x1609, seq=4/1024, ttl=64 (request in 15)
	17	2020-12-20 06:36:18.4628568…	192.168.22.1	224.0.0.251	MDNS	658 Standard query response 0x0000 TXT, cache flush PTR KolvacS MacBook…
	18	2020-12-20 06:36:19.0613742…	192.168.22.1	192.168.22.129	ICMP	100 Echo (ping) request id=0x1609, seq=5/1280, ttl=63 (reply in 19)
	19	2020-12-20 06:36:19.0613979…	192.168.22.129	192.168.22.1	ICMP	100 Echo (ping) reply id=0x1609, seq=5/1280, ttl=64 (request in 18)
	20	2020-12-20 06:36:19.6981454…	192.168.22.1	224.0.0.251	MDNS	686 Standard query response 0x0000 TXT, cache flush PTR _ssh._tcp.local…
	21	2020-12-20 06:36:20.0847703…	192.168.22.1	192.168.22.129	ICMP	100 Echo (ping) request id=0x1609, seq=6/1536, ttl=63 (reply in 22)
	22	2020-12-20 06:36:20.0847952…	192.168.22.129	192.168.22.1	ICMP	100 Echo (ping) reply id=0x1609, seq=6/1536, ttl=64 (request in 21)
	23	2020-12-20 06:36:20.2520239…	Vmware_70:30:23		ARP	44 Who has 192.168.22.1? Tell 192.168.22.129
	24	2020-12-20 06:36:20.2524688…	Vmware_c0:00:01		ARP	62 192.168.22.1 is at 00:50:56:c0:00:01

在Host U telnet 192.168.22.129：

```
[12/20/20]seed@VM:~$ telnet 192.168.22.129
Trying 192.168.22.129...
Connected to 192.168.22.129.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Tue Dec 15 06:54:29 EST 2020 on pts/19
/usr/lib/update-notifier/update-motd-fsck-at-reboot:[:59: integer expression exp
ected:            0
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.
```



184	2020-12-20 06:38:17.1437390…	192.168.22.1	192.168.22.129	TELNET	69 Telnet Data ...
185	2020-12-20 06:38:17.1437573…	192.168.22.129	192.168.22.1	TCP	68 23 → 51381 [ACK] Seq=1887752916 Ack=2812476306 Win=29056 Len=0 TS…
186	2020-12-20 06:38:17.3333805…	192.168.22.1	192.168.22.129	TELNET	69 Telnet Data ...
187	2020-12-20 06:38:17.3333995…	192.168.22.129	192.168.22.1	TCP	68 23 → 51381 [ACK] Seq=1887752916 Ack=2812476307 Win=29056 Len=0 TS…
188	2020-12-20 06:38:17.5334850…	192.168.22.1	192.168.22.129	TELNET	69 Telnet Data ...
189	2020-12-20 06:38:17.5335039…	192.168.22.129	192.168.22.1	TCP	68 23 → 51381 [ACK] Seq=1887752916 Ack=2812476308 Win=29056 Len=0 TS…
190	2020-12-20 06:38:17.6992546…	192.168.22.1	192.168.22.129	TELNET	70 Telnet Data ...
191	2020-12-20 06:38:17.6992744…	192.168.22.129	192.168.22.1	TCP	68 23 → 51381 [ACK] Seq=1887752916 Ack=2812476310 Win=29056 Len=0 TS…
192	2020-12-20 06:38:17.6999281…	192.168.22.129	192.168.22.1	TELNET	70 Telnet Data ...
193	2020-12-20 06:38:17.7000529…	192.168.22.1	192.168.22.129	TCP	68 51381 → 23 [ACK] Seq=2812476310 Ack=1887752918 Win=131648 Len=0 T…
194	2020-12-20 06:38:17.7187654…	192.168.22.129	192.168.22.1	TELNET	120 Telnet Data ...
195	2020-12-20 06:38:17.7188874…	192.168.22.1	192.168.22.129	TCP	68 51381 → 23 [ACK] Seq=2812476310 Ack=1887752970 Win=131584 Len=0 T…
196	2020-12-20 06:38:17.8149496…	192.168.22.129	192.168.22.1	TELNET	168 Telnet Data ...
197	2020-12-20 06:38:17.8151544…	192.168.22.1	192.168.22.129	TCP	68 51381 → 23 [ACK] Seq=2812476310 Ack=1887753070 Win=131456 Len=0 T…
198	2020-12-20 06:38:17.8152334…	192.168.22.129	192.168.22.1	TELNET	70 Telnet Data ...
199	2020-12-20 06:38:17.8153158…	192.168.22.1	192.168.22.129	TCP	68 51381 → 23 [ACK] Seq=2812476310 Ack=1887753072 Win=131456 Len=0 T…
200	2020-12-20 06:38:17.8200125…	192.168.22.129	192.168.22.1	TELNET	344 Telnet Data ...
201	2020-12-20 06:38:17.8201337…	192.168.22.1	192.168.22.129	TCP	68 51381 → 23 [ACK] Seq=2812476310 Ack=1887753348 Win=131200 Len=0 T…
202	2020-12-20 06:38:17.9292797…	192.168.22.129	192.168.22.1	TELNET	89 Telnet Data ...
203	2020-12-20 06:38:17.9293862…	192.168.22.1	192.168.22.129	TCP	68 51381 → 23 [ACK] Seq=2812476310 Ack=1887753369 Win=131200 Len=0 T…

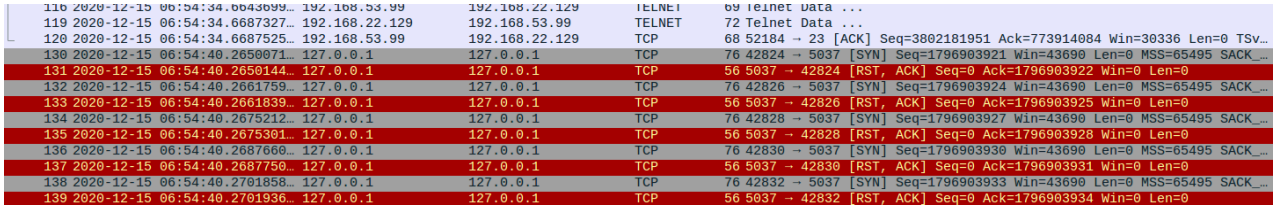
Step 6: Tunnel-Breaking Test.

On Host U, telnet to Host V. While keeping the telnet connection alive, we break the VPN tunnel. We then type something in the telnet window, and report what you observe. We then reconnect the VPN tunnel. What is going to happen to the telnet connection? Will it be broken or resumed? Please describe and explain your observations.

Q：we break the VPN tunnel by stopping the tun client.py or tun server.py program. We then type something in the telnet window. Do you see what you type? What happens to the TCP connection? Is the connection broken?

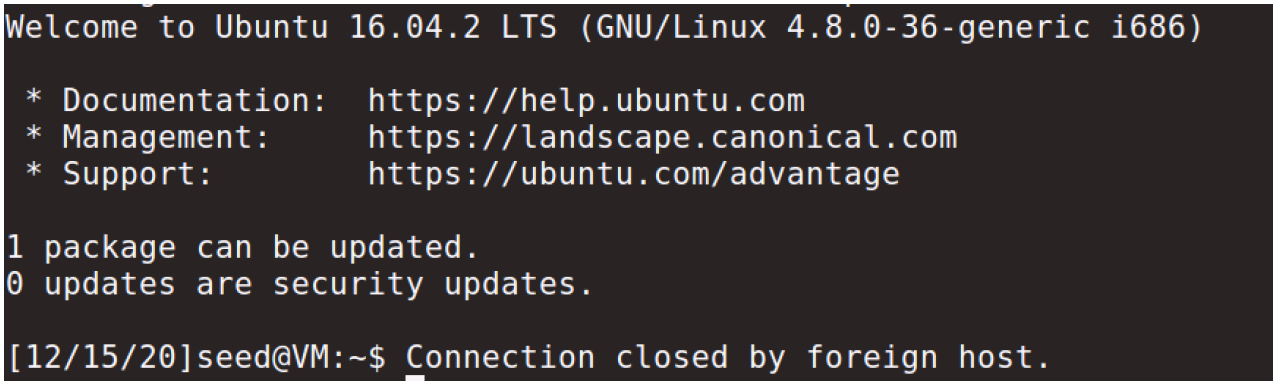
断开链接后，打出的东西无法显示在telnet窗口上；

telnet链接表面上还是链接状态，但是实际上Host U 与 Host V通过tun的链接已经断开；所以无法再传递数据



119	2020-12-15 06:54:34.6687327…	192.168.22.129	192.168.53.99	TELNET	72 Telnet Data ...
120	2020-12-15 06:54:34.6687525…	192.168.53.99	192.168.22.129	TCP	68 52184 → 23 [ACK] Seq=3802181951 Ack=773914084 Win=30336 Len=0 TSv…
130	2020-12-15 06:54:40.2650071…	127.0.0.1	127.0.0.1	TCP	76 42824 → 5037 [SYN] Seq=1796903921 Win=43690 Len=0 MSS=65495 SACK_…
131	2020-12-15 06:54:40.2650144…	127.0.0.1	127.0.0.1	TCP	56 5037 → 42824 [RST, ACK] Seq=0 Ack=1796903922 Win=0 Len=0
132	2020-12-15 06:54:40.2661759…	127.0.0.1	127.0.0.1	TCP	76 42826 → 5037 [SYN] Seq=1796903924 Win=43690 Len=0 MSS=65495 SACK_…
133	2020-12-15 06:54:40.2661839…	127.0.0.1	127.0.0.1	TCP	56 5037 → 42826 [RST, ACK] Seq=0 Ack=1796903925 Win=0 Len=0
134	2020-12-15 06:54:40.2675212…	127.0.0.1	127.0.0.1	TCP	76 42828 → 5037 [SYN] Seq=1796903927 Win=43690 Len=0 MSS=65495 SACK_…
135	2020-12-15 06:54:40.2675301…	127.0.0.1	127.0.0.1	TCP	56 5037 → 42828 [RST, ACK] Seq=0 Ack=1796903928 Win=0 Len=0
136	2020-12-15 06:54:40.2687660…	127.0.0.1	127.0.0.1	TCP	76 42830 → 5037 [SYN] Seq=1796903930 Win=43690 Len=0 MSS=65495 SACK_…
137	2020-12-15 06:54:40.2687750…	127.0.0.1	127.0.0.1	TCP	56 5037 → 42830 [RST, ACK] Seq=0 Ack=1796903931 Win=0 Len=0
138	2020-12-15 06:54:40.2701858…	127.0.0.1	127.0.0.1	TCP	76 42832 → 5037 [SYN] Seq=1796903933 Win=43690 Len=0 MSS=65495 SACK_…
139	2020-12-15 06:54:40.2701936…	127.0.0.1	127.0.0.1	TCP	56 5037 → 42832 [RST, ACK] Seq=0 Ack=1796903934 Win=0 Len=0

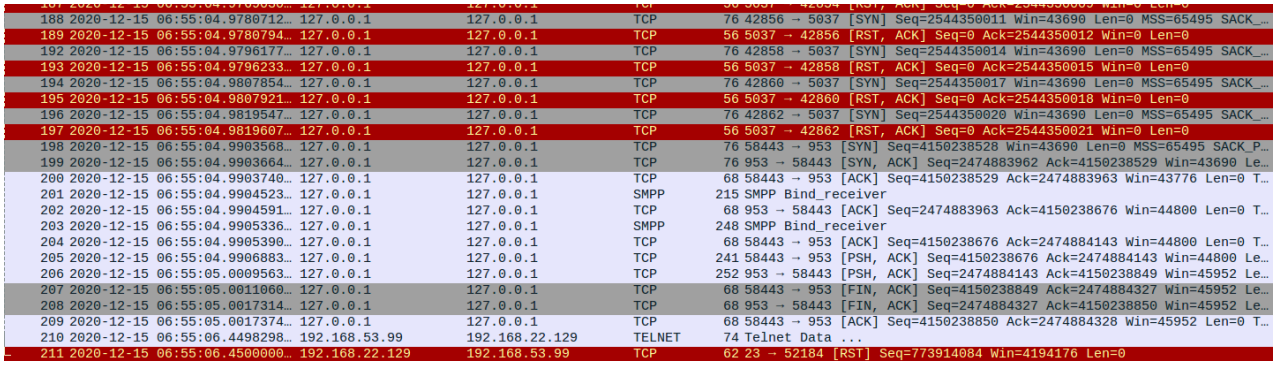
Q：Once the tunnel is re-established, what is going to happen to the telnet connection? Please describe and explain your observations.



```
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.

[12/15/20]seed@VM:~$ Connection closed by foreign host.
```

tunnel重新链接后，telnet connection被彻底断开，显示：connection closed by foreign host.

因为此时当链接重新被建立时，Host U收到了Host V发来的RST报文，TCP链接被重置



```
188 2020-12-15 06:55:04.9780712… 127.0.0.1      127.0.0.1      TCP     76 42856 → 5037 [SYN] Seq=2544350011 Win=43690 Len=0 MSS=65495 SACK_…
189 2020-12-15 06:55:04.9780794… 127.0.0.1      127.0.0.1      TCP     56 5037 → 42856 [RST, ACK] Seq=0 Ack=2544350012 Win=0 Len=0
192 2020-12-15 06:55:04.9796177… 127.0.0.1      127.0.0.1      TCP     76 42858 → 5037 [SYN] Seq=2544350014 Win=43690 Len=0 MSS=65495 SACK_…
193 2020-12-15 06:55:04.9796233… 127.0.0.1      127.0.0.1      TCP     56 5037 → 42858 [RST, ACK] Seq=0 Ack=2544350015 Win=0 Len=0
194 2020-12-15 06:55:04.9807854… 127.0.0.1      127.0.0.1      TCP     76 42860 → 5037 [SYN] Seq=2544350017 Win=43690 Len=0 MSS=65495 SACK_…
195 2020-12-15 06:55:04.9807921… 127.0.0.1      127.0.0.1      TCP     56 5037 → 42860 [RST, ACK] Seq=0 Ack=2544350018 Win=0 Len=0
196 2020-12-15 06:55:04.9819547… 127.0.0.1      127.0.0.1      TCP     76 42862 → 5037 [SYN] Seq=2544350020 Win=43690 Len=0 MSS=65495 SACK_…
197 2020-12-15 06:55:04.9819607… 127.0.0.1      127.0.0.1      TCP     56 5037 → 42862 [RST, ACK] Seq=0 Ack=2544350021 Win=0 Len=0
198 2020-12-15 06:55:04.9903568… 127.0.0.1      127.0.0.1      TCP     76 58443 → 953 [SYN] Seq=4150238528 Win=43690 Len=0 MSS=65495 SACK_P…
199 2020-12-15 06:55:04.9903664… 127.0.0.1      127.0.0.1      TCP     76 953 → 58443 [SYN, ACK] Seq=2474883962 Ack=4150238529 Win=43690 Le…
200 2020-12-15 06:55:04.9903740… 127.0.0.1      127.0.0.1      TCP     68 58443 → 953 [ACK] Seq=4150238529 Ack=2474883963 Win=43776 Len=0 T…
201 2020-12-15 06:55:04.9904523… 127.0.0.1      127.0.0.1      SMPP   215 SMPP Bind_receiver
202 2020-12-15 06:55:04.9904591… 127.0.0.1      127.0.0.1      TCP     68 953 → 58443 [ACK] Seq=2474883963 Ack=4150238676 Win=44800 Len=0 T…
203 2020-12-15 06:55:04.9905336… 127.0.0.1      127.0.0.1      SMPP   248 SMPP Bind_receiver
204 2020-12-15 06:55:04.9905390… 127.0.0.1      127.0.0.1      TCP     68 58443 → 953 [ACK] Seq=4150238676 Ack=2474884143 Win=44800 Len=0 T…
205 2020-12-15 06:55:04.9906883… 127.0.0.1      127.0.0.1      TCP    241 58443 → 953 [PSH, ACK] Seq=4150238676 Ack=2474884143 Win=44800 Le…
206 2020-12-15 06:55:05.0009563… 127.0.0.1      127.0.0.1      TCP    252 953 → 58443 [PSH, ACK] Seq=2474884143 Ack=4150238849 Win=45952 Le…
207 2020-12-15 06:55:05.0011060… 127.0.0.1      127.0.0.1      TCP     68 58443 → 953 [FIN, ACK] Seq=4150238849 Ack=2474884327 Win=45952 Le…
208 2020-12-15 06:55:05.0017314… 127.0.0.1      127.0.0.1      TCP     68 953 → 58443 [FIN, ACK] Seq=2474884327 Ack=4150238850 Win=45952 Le…
209 2020-12-15 06:55:05.0017374… 127.0.0.1      127.0.0.1      TCP     68 58443 → 953 [ACK] Seq=4150238850 Ack=2474884328 Win=45952 Len=0 T…
210 2020-12-15 06:55:06.4498298… 192.168.53.99  192.168.22.129 TELNET  74 Telnet Data ...
211 2020-12-15 06:55:06.4500000… 192.168.22.129 192.168.53.99  TCP     62 23 → 52184 [RST] Seq=773914084 Win=4194176 Len=0
```

Task 3: Encrypting the Tunnel

in your demonstration, you need to use Wireshark to capture the traffic inside the VPN tunnel, and show that the traffic is indeed encrypted.

实验给出的tls文件夹中证书已经过期，有点坑....

首先将tls文件夹共享；

在cert_server文件夹中按以下步骤生成证书：

1. 生成CA：包括两个文件：①cakey.pem，②cacert.pem

   ```
   $ openssl req -new -x509 -keyout cakey.pem -out cacert.pem -config openssl.cnf -days 3650
   ```

2. 生成服务器key：server-key.pem

   ```
   $ openssl genrsa -des3 -out server-key.pem 1024
   ```

3. 生成服务器待签的csr文件：

   ```
   $ openssl req -new -key server-key.pem -out server-csr.pem -config openssl.cnf
   ```

4. 使用CA为生成的服务器csr文件签名：

   ```
   $ openssl ca -in server-csr.pem -out server-cert.pem -cert cacert.pem -keyfile cakey.pem -config openssl.cnf
   ```

生成证书过程中的common name均为 vpnlabserver.com

将cacert放入client文件夹替换掉过期证书

生成证书后，运行tlsclient依旧无法建立连接；

报错：

经过查找，发现是因为客户端验证证书时使用hash查找，要生成cacert.pemhash值的软连接。

查找hash值：

```
[12/21/20]seed@VM:.../ca_client$ openssl x509 -subject_hash -in *cacert.pem*
a1c2ee66
-----BEGIN CERTIFICATE-----
MIICmTCCAgKgAwIBAgIUTkR2m5jn35tCnnpU707ZEzqDmYYwDQYJKoZIhvcNAQEL
BQAwYDELMAkGA1UEBhMCQVUxEzARBgNVBAgMClNvbWUtU3RhdGUxITAfBgNVBAoM
GEludGVybmV0IFdpZGdpdHMgUHR5IEx0ZDEZMBcGA1UEAwwQdnBubGFic2VydmVy
LmNvbTAeFw0yMDEyMTgxMjQ5NDFaFw0zMDEyMTYxMjQ5NDFaMGAxCzAJBgNVBAYT
```

建立symbolic link：

```
-----END CERTIFICATE-----
[12/21/20]seed@VM:.../ca_client$ ln -s cacert.pem a1c2ee66
[12/21/20]seed@VM:.../ca_client$
[12/21/20]seed@VM:.../ca_client$
[12/21/20]seed@VM:.../ca_client$ ln -s cacert.pem a1c2ee66.0
```

> 参考：
>
> https://serverfault.com/questions/453300/openssl-client-authentication-error-tlsv1-alert-unknown-ca-ssl-alert-numbe
>
> https://serverfault.com/questions/639790/postfix-tlsv1-alert-unknown-ca

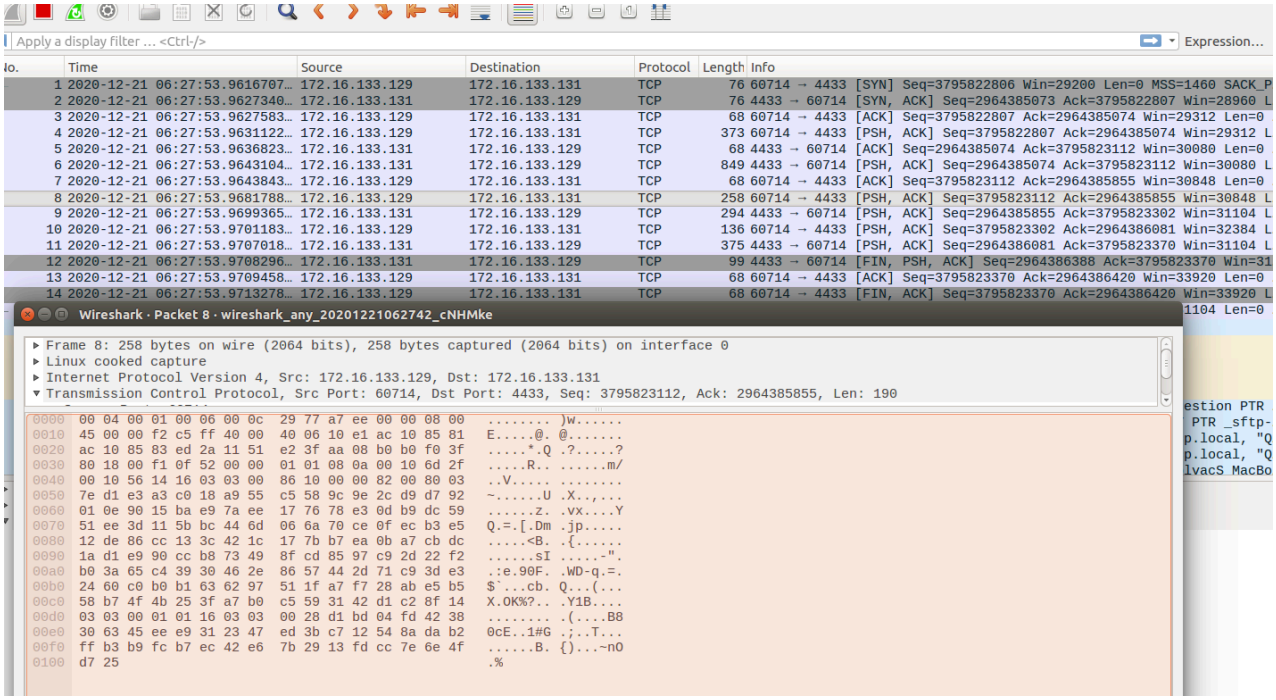
运行`./tlsclient vpnlabserver.com 4433`

```
[12/21/20]seed@VM:.../tls$ ./tlsclient vpnlabserver.com
 4433
SSL connection is successful
SSL connection using AES256-GCM-SHA384
HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE html><html><head><title>Hello World</title></
head><style>body {background-color: black}h1 {font-size
:3cm; text-align: center; color: white;text-shadow: 0 0
 3mm yellow}</style></head><body><h1>Hello, world!</h1>
</body></html>

[12/21/20]seed@VM:.../tls$
```

```
SSL connection established!
Received: GET / HTTP/1.1
Host: vpnlabserver.com
```

成功建立tls链接

wireshark中查看到了双方建立TCP链接的过程，查看数据包，确实被加密。



Task 4: Authenticating the VPN Server

上个task中已经完成了正确的VPN server验证以及证书生成；

There are three important steps in server authentication:

(1) verifying that the server certificate is valid,

```
if(SSL_CTX_load_verify_locations(ctx,NULL, CA_DIR) < 1){
  printf("Error setting the verify locations. \n");
  exit(0);
   }
```

(2) verifying that the server is the owner of the certificate, and

```
if(SSL_CTX_load_verify_locations(ctx,NULL, CA_DIR) < 1){
  printf("Error setting the verify locations. \n");
  exit(0);
   }
```

(3) verifying that the server is the intended server (for example, if the user intends to visit example.com, we need to ensure that the server is indeed example.com, not another site).

```
X509_VERIFY_PARAM *vpm = SSL_get0_param(ssl);
X509_VERIFY_PARAM_set1_host(vpm, hostname, 0);
```

In your demonstration, you need to demonstrate two different cases regarding the third verification: a successful server authentication where the server is the intended server, and a failed server authentication where the server is not the intended server.

Task 5: Authenticating the VPN Client

login.c:

```
 #include <stdio.h>
#include <string.h>
#include <shadow.h>
#include <crypt.h>

int login(char *user, char *passwd)
{
    struct spwd *pw;
    char *epasswd;
    pw = getspnam(user);
    if (pw == NULL) {
        return -1;
    }
    printf("Login name: %s\n", pw->sp_namp);
    printf("Passwd : %s\n", pw->sp_pwdp);
    epasswd = crypt(passwd, pw->sp_pwdp);
    if (strcmp(epasswd, pw->sp_pwdp)) {
        return -1;
    }
    return 1;
}

void main(int argc, char** argv)
{
    if (argc < 3) {
    printf("Please provide a user name and a password\n");
    return;
    }
    int r = login(argv[1], argv[2]);
    printf("Result: %d\n", r);
}
```

编译：

```
[12/28/20]seed@VM:.../tls$ gcc login.c -lcrypt
[12/28/20]seed@VM:.../tls$
[12/28/20]seed@VM:.../tls$
[12/28/20]seed@VM:.../tls$ sudo ./a.out seed dees
Login name: seed
Passwd : $6$wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN/s
fYvDeCAcEo2QYzCfpZoaEVJ8sbCT7hkxXY/
Result: 1
```

设定用户名：seed

密码：dees

tlsserver.c:

定义loginVerify函数

```
int Loginverify(SSL *ssl, int conn) {
  char *input;
  char username[100];
  char password[1000];
  char request[BUFF_SIZE];

  // 初始化
  memset(&username, 0, sizeof(username));
  memset(&password, 0, sizeof(password));
  memset(&request, 0, BUFF_SIZE);

  int len = SSL_read(ssl, request, BUFF_SIZE - 1);
  request[len] = '\0';

  // username
  input = strtok(request, " "); // char *strtok(char *str, const char *delim) 分解字符串 str 为一组字符串，delim 为分隔符。该函数返回被分解的第一个子字符串，如果没有可检索的字符串，则返回一个空指针。
  if (!input) {
    printf("Username invalid!\n");
    return -1;
  }
  strcpy(username, input);


  // password
  input = strtok(NULL, " ");
  if (!input) {
    printf("Password invalid!\n");
    return -1;
  }
  strcpy(password, input);


  // check shadow
  struct spwd *pw;
  char *epasswd;
  pw = getspnam(username);
  if (pw == NULL) {
    printf("Invalid account\n");
    return -1;
  }

  epasswd = crypt(password, pw->sp_pwdp);
  char *fail = "login fail!";
```

```
  char *success = "login success!";
  if (strcmp(epasswd, pw->sp_pwdp)) {
    printf("Username and password do not match\n");
    SSL_write(ssl, fail, strlen(fail));
    return -1;
  }
  SSL_write(ssl, success, strlen(success));
  return 1;
}
```

tlsclient.c:

```
void login(SSL *ssl) {
  char username[NAME_LENGTH];
  char password[PASSWORD_LENGTH];
  char request[BUFF_SIZE];
  char reply[BUFF_SIZE];
  int len;

  printf("Your username:\n");
  scanf("%s", username);
  getchar();
  printf("Your password:\n");
  scanf("%s", password);



  // request
  bzero(request, BUFF_SIZE);
  strcpy(request, username);
  strcat(request, " ");
  strcat(request, password);
  len = strlen(username) + strlen(password) + 1;
  request[len] = '\0';


  SSL_write(ssl, request, len);

  // check reply
  bzero(reply, BUFF_SIZE);
  len = SSL_read(ssl, reply, BUFF_SIZE - 1);
  reply[len] = '\0';

  // fail
  if (strcmp(reply, "success")) {
    printf("Login Failed\n");
    endRequest();
  }
```

```
    // success
}
```

效果：登陆成功：

```
[12/28/20]seed@VM:.../tls$ ./tlsclient vpnlabserver.com 4433
TLS Initialized
TCP Connected
Verification passed.
Verification passed.
SSL connection is successful
SSL connection using AES256-GCM-SHA384
Your username:
seed
Your password:
dees
```

登陆失败：

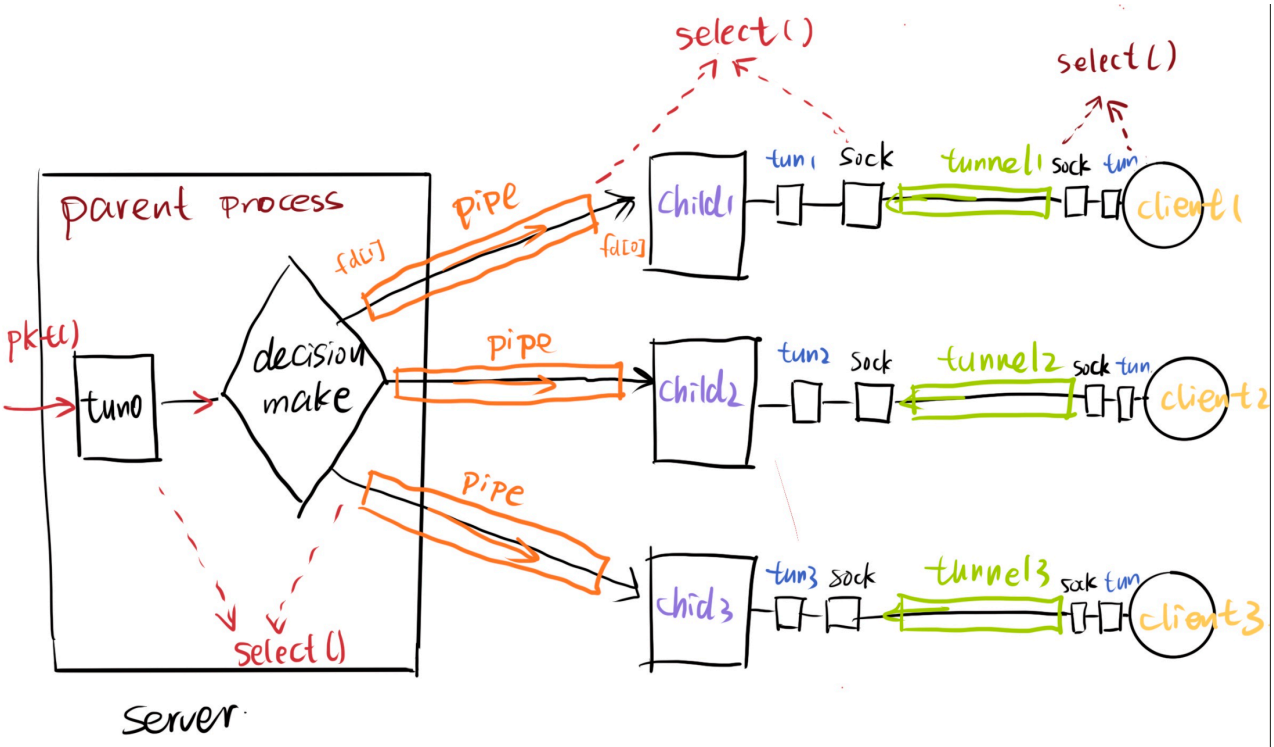
```
[12/28/20]seed@VM:.../tls$ ./tlsclient vpnlabserver.com 4433
TLS Initialized
TCP Connected
Verification passed.
Verification passed.
SSL connection is successful
SSL connection using AES256-GCM-SHA384
Your username:
dees
Your password:
1234
Login Failed
```

Task 6: Supporting Multiple Clients

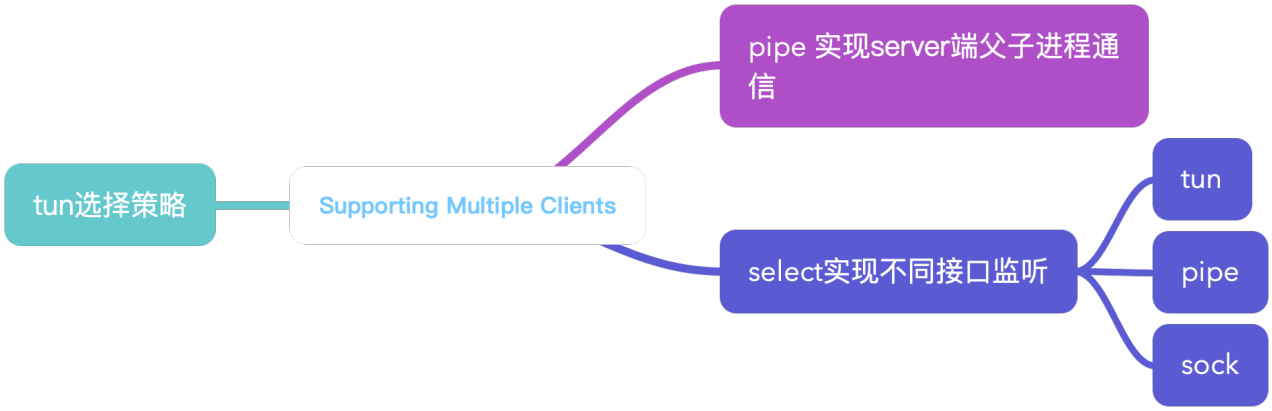
> In a typical implementation, the VPN server process (the parent process) will create a child process for each tunnel (see Figure 4). When a packet comes from the tunnel, its corresponding child process will get the packet, and forward it to the TUN interface. This direction is the same regardless of whether multiple clients are supported or not. It is the other direction that becomes challenging. When a packet arrives at the TUN interface (from the private network), the parent process will get the packet, now it needs to figure out which tunnel this packet should go to. You need to think about how to implement this decision-making logic.

> the parent process will get the packet, now it needs to figure out which tunnel this packet should go to. You need to think about how to implement this decision-making logic.

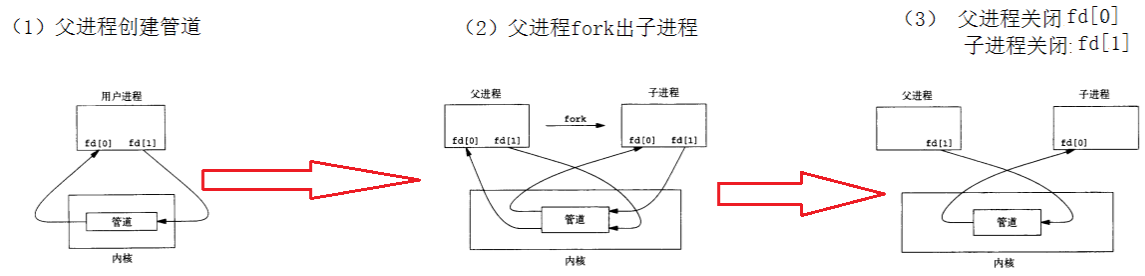
整体结构：



Supporting Multiple Clients，重点是完成三个控制逻辑：



1.pipe来实现server端父子进程单向通信



管道实现通信功能的步骤

> 关于pipe：https://ocaml.github.io/ocamlunix/pipes.html

查找资料发现由于pipe的队列特性，server端父进程在收到internal网络的pipe之后可以用类似广播的方式来传入pipe，从而根据ip地址等信息，来被正确的子进程以及Client接受。

2.server端流程：

```
int main(int argc, char *argv[]) {
  int fd[2];
  int tunfd = createTunDevice();
  printf("createTunDevice\n" );

  pipe(fd);
  if (fork() > 0) //parent
  {
    printf("parent\n");
    close(fd[0]);
    while (1)
    {
      fd_set readFDSet;
      FD_ZERO(&readFDSet);
      FD_SET(tunfd, &readFDSet);
      select(FD_SETSIZE, &readFDSet, NULL, NULL, NULL);

      if (FD_ISSET(tunfd, &readFDSet))
        tun2pipe(tunfd, fd[1]);//将收到的internal网络的packet广播发送给pipe另一端的子进程
    }
    return 0;
  }

  else  //child
  {
    printf("child\n");
    close(fd[1]);

    size_t client_len;
    struct sockaddr_in sa_client;
    int sockfd = setupTCPServer();

    while (1) {
      int conn = accept(sockfd, (struct sockaddr *)&sa_client, &client_len);
      printf("accept\n");
      if (fork() == 0) //每accept一个client连接，就创建子进程
        {
          // child process
          close(sockfd);

          printf("%d: Start\n", getpid());
```

```
            //——————————SSL INIT——————————————————
            SSL_METHOD *meth;
            SSL_CTX* ctx;
            SSL *ssl;
            // Step 0: OpenSSL library initialization
            // This step is no longer needed as of version 1.1.0.
            SSL_library_init();
            SSL_load_error_strings();
            SSLeay_add_ssl_algorithms();

            // Step 1: SSL context initialization
            meth = (SSL_METHOD *)TLSv1_2_method();
            ctx = SSL_CTX_new(meth);
            SSL_CTX_set_verify(ctx, SSL_VERIFY_NONE, NULL);
            // Step 2: Set up the server certificate and private key
            SSL_CTX_use_certificate_file(ctx, "./cert_server/server-cert.pem",
SSL_FILETYPE_PEM);
            SSL_CTX_use_PrivateKey_file(ctx, "./cert_server/server-key.pem",
SSL_FILETYPE_PEM);
            // Step 3: Create a new SSL structure for a connection
            ssl = SSL_new (ctx);


            SSL_set_fd(ssl, conn);
            int err = SSL_accept(ssl);
            int errcode = SSL_get_error(ssl, err);

            printf("%d: Handshake\n", getpid());
            //CHK_SSL(err);
            if (LoginVerify(ssl, conn) != 1) {
              printf("%d: Login Fail! \n", getpid());
              endRequest(ssl, conn);
              return 0;
            }
            printf("%d: Login success!\n", getpid());

            //进入处理逻辑
            processRequest(tunfd, fd[0], conn, ssl);

            printf("%d: Exit\n", getpid());
            endRequest(ssl, conn);
            return 0;
          }
          else
          {
        // parent
        close(conn);
        //close(tunfd);
          }
```

```
    }
  }
}
```

子进程的处理逻辑：重点是通过select监听pipe和sock两个方向的数据流：

```
void processRequest(int tunfd, int pipefd, int sockfd, SSL *ssl) {
  while (1) {
    fd_set readFDSet;

    FD_ZERO(&readFDSet);
    FD_SET(sockfd, &readFDSet);
    FD_SET(pipefd, &readFDSet);
    select(FD_SETSIZE, &readFDSet, NULL, NULL, NULL);

    if (FD_ISSET(pipefd, &readFDSet)) pipe2socket(tunfd, sockfd, ssl);
    if (FD_ISSET(sockfd, &readFDSet)) socket2tun(tunfd, sockfd, ssl);
  }
}
```

3.Client端流程：

```
int main(int argc, char *argv[])
{
  int tunfd;
  tunfd = createTunDevice();

  char *hostname = "vpnlabserver.com";
  int port = 4433;

  if (argc > 1) hostname = argv[1];
  if (argc > 2) port = atoi(argv[2]);

  /*----------------TLS initialization ----------------*/
  ssl = setupTLSClient(hostname);
  printf("TLS Initialized successfully\n");

  /*----------------Create a TCP connection ---------------*/
  int sockfd = setupTCPClient(hostname, port);
  printf("TCP Connected successfully\n");

  /*----------------TLS handshake ---------------------*/
  SSL_set_fd(ssl, sockfd);
  int err = SSL_connect(ssl);
  CHK_SSL(err);
  printf("SSL connection successful\n");
  printf("SSL connection using %s\n", SSL_get_cipher(ssl));
```

```
    //登陆
    login(ssl);
    printf("starting communication......\n");

    //进入通信
    while (1)
    {
      fd_set readFDSet;

      FD_ZERO(&readFDSet);
      FD_SET(sockfd, &readFDSet);
      FD_SET(tunfd, &readFDSet);
      select(FD_SETSIZE, &readFDSet, NULL, NULL, NULL);

      if (FD_ISSET(tunfd, &readFDSet)) tunSelected(tunfd, sockfd, ssl);
      if (FD_ISSET(sockfd, &readFDSet)) socketSelected(tunfd, sockfd, ssl);
    }
}
```

重点也是通过select来监听tun和sock两个端口的数据流

4.效果展示：

client端

```
[01/02/21]seed@VM:.../tls$ ./tlsclient vpnlabserver.com 4433
TLS Initialized successfully
TCP Connected successfully
Verification passed.
Verification passed.
SSL connection successful
SSL connection using AES256-GCM-SHA384
Enter Your username:
seed
Enter Your password:
dees
Login sucessed!
starting communication......
```

server端：

```
[01/02/21]seed@VM:.../tls$ sudo ./tlsserver
TUN setup successfully
createTunDevice.....
parent
child
TCP Setup successfully
accept
9674: Start
Enter PEM pass phrase:
9674: Handshake
username:
password:
request:  seed dees
9674: Login success!
processing request.....
```

```
[01/02/21]seed@VM:.../tls$ sudo ./tlsserver
TUN setup successfully
createTunDevice.....
parent
child
TCP Setup successfully
accept
4974: Start
Enter PEM pass phrase:
4974: Handshake
username:
password:
request:  seed dees
4974: Login success!
processing request.....
accept
4975: Start
Enter PEM pass phrase:
4975: Handshake
username:
password:
request:  seed dees
4975: Login success!
processing request.....
accept
4986: Start
Enter PEM pass phrase:
4986: Handshake
```