华中科技大学

“计算机网络安全”实验报告

——VPN 实验

院　　系：网络空间安全学院

专业班级：网安 2002 班

姓　　名：汪闻韵

学　　号：U202012056

日　　期：2023 年 5 月 23 日

评分项	实验报告评分（50%）	检查单分数（50%）	综合得分	教师签名
得分				

实验报告评分表

评分项目	分值	评分标准	得分
实验原理	10	10-8：原理理解准确，说明清晰完整 7-5：原理理解基本准确，说明较为清楚 4-0：说明过于简单	
VPN 系统设计	25	25-19：系统架构和模块划分合理，详细设计说明详实准确 18-11：系统架构和模块划分基本合理，详细设计说明较为详实准确 10-0：系统架构和模块划分不恰当，详细设计说明过于简单	
VPN 实现细节	25	25-19：文字表达清晰流畅，实现方法技术优良，与设计实现及代码一致 18-11：文字表达较清晰流畅，实现方法一般，与设计实现及代码有偏差 10-0：文字表达混乱，实现方法过于简单	
测试结果与分析	20	20-15：功能测试覆盖完备，测试结果理想，分析说明合理可信 14-9：功能测试覆盖基本完备，测试结果基本达标，分析说明较少 8-0：功能测试覆盖不够，测试未达到任务要求，缺乏分析说明	
体会与建议	10	10-8：心得体会真情实感，意见中肯，建议合理可行，体现了个人的思考 7-5：心得体会较为真实，意见建议较为具体 4-0：过于简单敷衍	
格式规范	10	图、表的说明，行间距、缩进、目录等不规范相应扣分	
总 分			

目　　录

1 实验原理

1.1 网络拓扑

在本次实验中，实验环境的网络拓扑如图 1-1 所示，图中展示了实验环境中的网络布局。

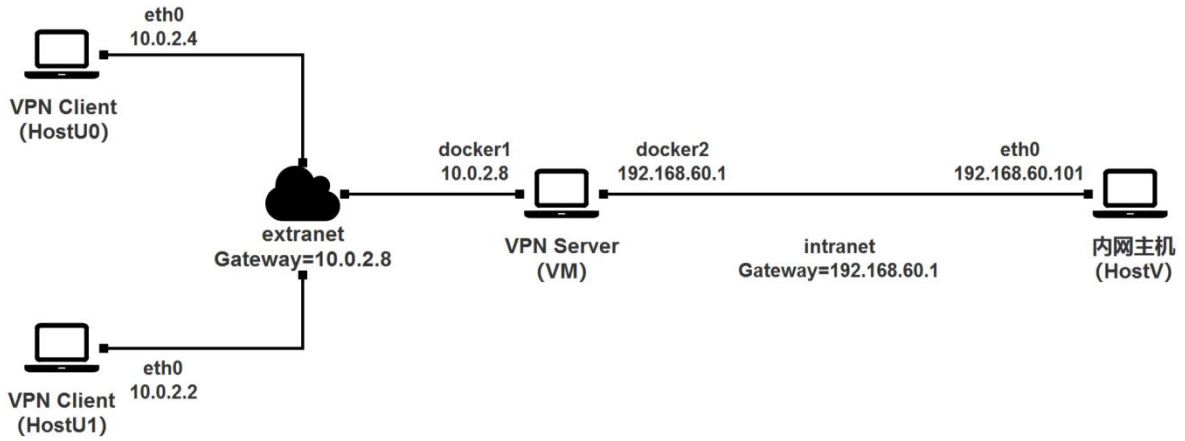

图 1-1 实验环境网络拓扑图

实验环境中包括了 HostU0 和 HostU1 两台主机，它们分别运行着 miniVPNclient 软件。另外，我们还有一台名为 VM 的虚拟机，它上面安装了 miniVPNserver 软件。值得注意的是，为了模拟外网和内网的环境，我们使用了 docker 创建的虚拟网桥来实现 extranet 和 intranet。初始情况下，HostU 和 HostV 可以相互通信，无法真实模拟出 HostU 位于外网、HostV 位于内网且彼此无法直接通信的情况。因此，我们需要在每台机器上执行命令"route del default"，以删除默认的路由表，使它们在形式上无法互通。同样地，HostV 也需要进行相同的操作步骤来构建实验环境。

1.2 通信机制

本次实验中，我们使用了 TUN 技术来实现 miniVPN，并设计了如图 1-2 所示的通信机制。在图 1-2 中，描述了 extranet 中的主机向 intranet 中的主机发送数据的流程。其中，图中的三个黄色方块表示三台独立的主机。

在 HostU 上，我们运行 miniVPNclient 软件。它会自动在/dev/net 目录下创建一个名为 tun 的虚拟网卡，并使用 IP 地址 192.168.53.5 启动该虚拟网卡（tun0）。此外，由于需要与远程子网（192.168.60.0/24）进行通信，miniVPNclient 还会自动在系统中添加一个路由表项，将所有目的 IP 地址在 192.168.60.0/24 范围内的数据包转发到虚拟网卡 tun0。

因此，当用户发起对 HostV（IP 地址为 192.168.60.101）的通信请求时，系统会将这些网络通信数据包写入 tun0 中。miniVPNclient 会检测到 tun0 中有数据包写入，然后将这些数据包读取出来，并封装到一个 TCP 报文的数据部分中。接着，miniVPNclient 通过与运行在 VM 中的 miniVPNserver 建立的 TCP 连接将该 TCP 报文发送出去。

运行在 VM 上的 miniVPNserver 会从 TCP 连接中接收到这个报文，并解封装出数据部分的数据。然后，它将这些数据写入到在 VM 中创建的虚拟网卡 tun0 中。系统会检测到 tun0 中有数据包写入，并通过网络层将其读取出来并发送出去。由于该数据包的目的 IP 地址是 192.168.60.101，所以该数据包会被发送到 docker2，然后 docker2 将其转发给 HostV。

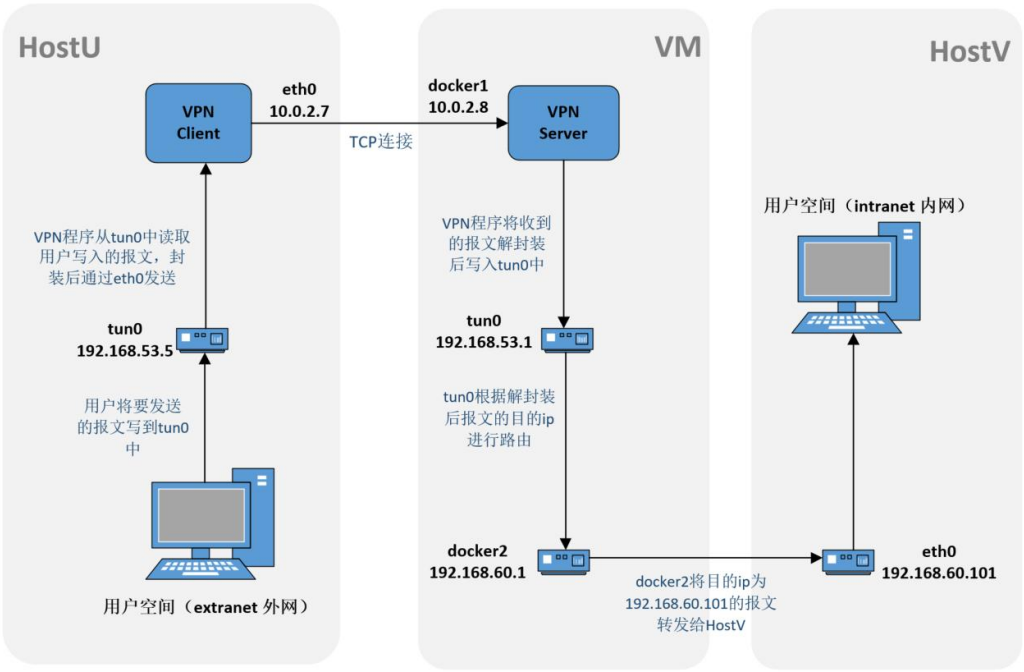

图 1-2 extranet 向 intranet 发送数据示意图

对于 HostU 和 HostV 来说，整个过程是透明的，他们并不察觉到他们发送的网络报文经历了封装、发送、解封装和接收的过程。因此，当 HostV 向 HostU 发送报文时，只需简单地将目的 IP 地址设置为 192.168.53.5 即可。在 HostV 中，我们可以将 192.168.60.1 设为所有报文的网关，这样报文就会被转发给 docker2，而 docker2 会将报文发送给 VM 上的 tun0。

当 miniVPNserver 程序检测到 tun0 接收到报文后，它会将该报文封装到一个 TCP 报文的数据段中，并发送给运行在 HostU 上的 miniVPNclient。miniVPNclient 接收到报文后，会从数据段中提取出原始报文，并将其写入到 HostU 上的 tun0 中。系统会读取该报文，并将其交付给之前发起与 HostV 通信的用户空间程序。

如图 1-3 所示，展示了报文的处理流程，整个过程中，HostU 用户空间程序发出的网络报文好像经过了一个建立在公网上的隧道一样，直接到达了处于内网中的 HostV。这也解释了为什么 VPN 有时被称为隧道。然而，在我看来，更形象地描述这个过程应该是"摆渡"。HostU 用户空间发出的报文就像一个人开车过河，需要将车开到摆渡船上。摆渡船将车运送过河后，车再次下船并继续向目的地行驶。而 miniVPN 实现的正是这个"摆渡船"的功能。HostU 和需要访问的内网中的 HostV 就像是河对岸的关系。通过 miniVPN 的"摆渡"功能以及 Linux 提供的 tun 虚拟网卡功能，我们成功实现了所需的通信机制。

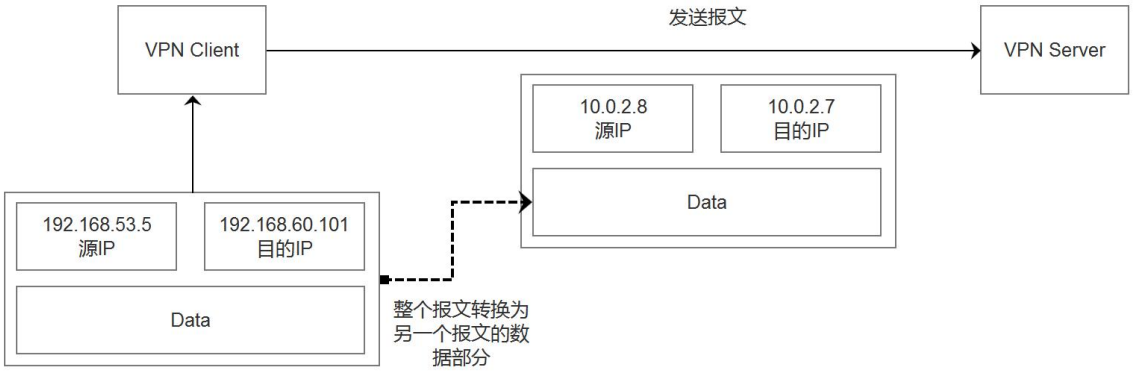

图 1-3 报文的处理流程

1.3 加密原理

在之前介绍的 miniVPN 的"摆渡"过程中，由于仅简单地将原始报文作为数据段进行传输，无法确保通信过程的数据机密性和完整性。因此，我们需要使用能够提供这种保证的 TLS 协议进行传输。具体而言，miniVPNclient 和 miniVPNserver 在建立 TCP 连接后，完成 TLS v1.2 所定义的服务器身份验证和密钥协商过程。随后，在后续的通信过程中，使用协商得到的密钥对经过"摆渡"的报文进行整体加密，到达目标主机后再由 miniVPN 进行解密。

TLS v1.2 的握手流程如下：

1. miniVPNserver 向 miniVPNclient 发送经过 CA 签名的证书，其中包含 CA 私钥对 miniVPNserver 公钥摘要值的签名以及公钥本身。miniVPNclient 收到证书后，使用 CA 的公钥对公钥摘要值进行解密，并计算证书中附带的公钥的摘要值。将这两个摘要值进行比较，当它们相等时，表示服务器身份验证成功。

2. miniVPNclient 基于 AES256-GCM-SHA384 算法生成一个加密密钥，并使用前一步中获得的 miniVPNserver 的公钥对该密钥进行加密。然后将加密结果发送给 miniVPNserver。

3. miniVPNserver 收到加密值后，使用自己的私钥进行解密，得到对称密钥。至此，密钥

协商完成。

4. 在随后的隧道通信中，使用前述对称密钥对用户空间的报文进行整体加密，并将其封装到 TCP 报文的数据段中进行传输。

在这个过程中，首先需要为 miniVPNserver 生成一对公私钥，并为公钥生成证书。由于我们无法向真正的 CA 申请证书，因此我们需要自行生成 CA 的公私钥，并使用 CA 的私钥对 miniVPNserver 公钥的签名请求进行签名。此外，还需要将 CA 的公钥添加到 HostU 中的信任 CA 列表中。否则，由于对 miniVPNserver 公钥进行签名的 CA 并非信任的 CA，身份验证仍无法完成。CA 和 miniVPNserver 的公私钥均基于 2048 位的 RSA 算法。

1.4 认证机制

认证机制涵盖了客户端对服务器的身份验证和服务器对客户端的身份验证两个方面。

客户端对服务器的身份验证是通过使用 CA 进行验证的。在 TLS 握手过程中，服务器会发送自己的证书和公钥给客户端。客户端使用 CA 的公钥来验证服务器的证书是否经过 CA 的真实签名。如果验证成功且证书处于有效期内，那么客户端可以确认对服务器的身份验证是成功的。

服务器对客户端的身份验证发生在 TLS 握手完成后，也就是在客户端认证服务器身份之后。此时，基于 TLS 的隧道已经建立完成，因此可以认为客户端和服务器之间的数据通信是安全的。在这种情况下，服务器可以要求客户端使用账户和密码进行登录。账户和密码的验证是基于服务器上的"/etc/shadow"文件来确定的，客户端需要使用服务器上已存在的账户及其相应的密码进行登录。当服务器验证发现客户端发送的账户和密码与"/etc/shadow"文件中的记录匹配时，即可认为服务器对客户端的身份验证是成功的。

2 VPN 系统设计

2.1 系统概要

miniVPNserver 包含 SSL 通信模块、管道通信模块、TUN 通信模块以及主体模块,四个模块协作完成 miniVPNserver 的工作，如图 2-1 所示。它们主要负责的工作如下。

SSL 通信模块：与 miniVPNclient 建立 TLS 连接、完成服务器身份的证明工作、完成密钥协商并使加解密过程对其他模块透明。

管道通信模块：负责创建命名管道并监听该管道，当从管道中读取到数据之后，将其交付

给 SSL 通信模块。同时，还会根据管道中数据的情况，判断连接是否正常，当连接错误或者结束时，停止对管道的监听，并将管道文件删除。

TUN 通信模块：负责监听虚拟网络设备 tun0，从 tun0 中读取到报文数据后，根据报文的目的 ip 来将其写入对应的管道文件中。

主体模块：负责实现各个模块的初始化、启动过程。在接收到 miniVPNclient 的连接请求时，通过调用 SSL 通信模块来完成 ip 分配的工作。同时，主体模块还为实现 ip 占用信息共享、管道文件描述符等数据的进程间共享提供了共享内存区域初始化以及各自的同步锁（用于对共享内存区域访问进行锁操作）初始化的功能。

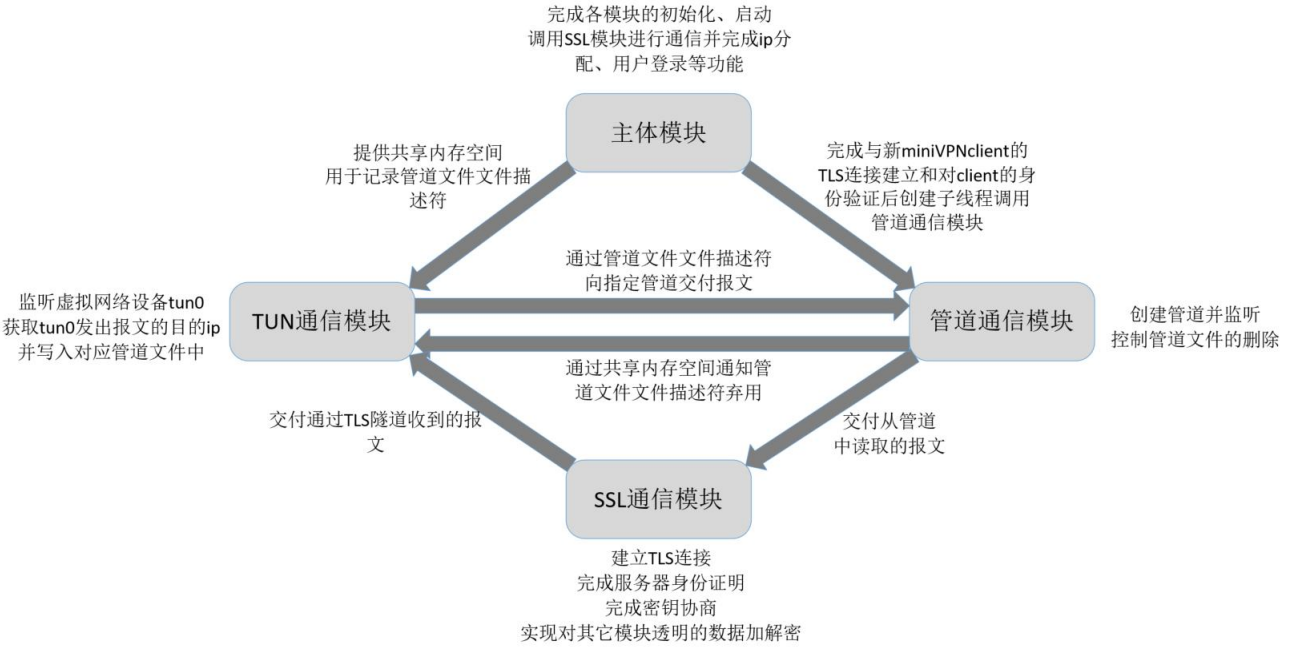

图 2-1 miniVPNserver 整体模块架构

miniVPNclient 仅有一个主体模块，实现与 miniVPNserver 建立 TLS 连接，验证服务器身份，获取虚 IP 的功能，接收来自 TLS 隧道的报文，将数据部分解密后交付给虚拟网卡 tun0，同时还要负责监听虚拟网卡 tun0，从 tun0 中读取到报文后将报文加密封装并通过 TLS 隧道发送给 miniVPNserver。

2.2 miniVPNserver 模块设计

2.2.1 SSL 通信模块详细设计

SSL 通信模块中主要的数据结构如下：

1. SSL 结构体指针 SSL* ssl。

2. 通过共享内存访问的 ip 占用符数组 ipFlag* ipflags。
3. 通过共享内存访问的管道文件描述符表 pipefdTable* pipefdTables。
4. 两片共享内存区域各自的锁变量 mutexWorker ipFlagMutexWorker 和 mutexWorker pipefdTableMutexWorker。

其中 SSL 结构体由 ssl 库提供，其他自定义结构体内容如下图 2-2 所示。

```
typedef struct ipFlag {
    int area[256];
} ipFlag;

typedef struct _pipefdTable {
    int ipCode;
    char* object;
    int value;
    int len;
    struct _pipefdTable* next;
} pipefdTable;

typedef struct _mutexWorker {
    int num;
    pthread_mutex_t mutex;
    pthread_mutexattr_t mutexattr;
} mutexWorker;
```

图 2-2 SSL 信模块自定义结构体

SSL 通信模块并无什么特别设计的算法，毕竟数据的加解密过程都由 openssl 提供的函数为我们完成，下面图 2-3 为 SSL 通信模块的主要工作流程图。

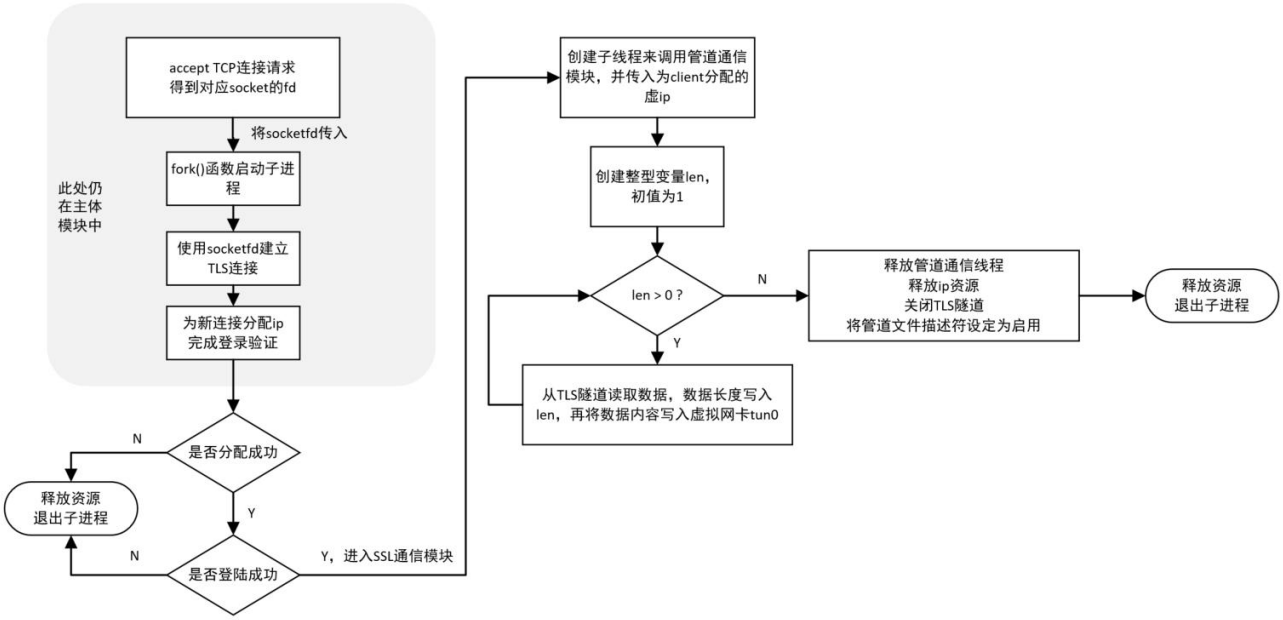

图 2-3 SSL 通信模块主要工作流程图

在图中，我们可以看到一些关键的模块和代码片段，其中主体模块扮演着核心角色。在建立 TLS 连接后，主体模块负责进行 IP 分配和用户验证登录的过程。一旦登录完成，SSL 通信模块启动子线程调用管道通信模块的功能。随后，SSL 通信模块会不断地从 TLS 隧道中读取数据。

在读取数据时，TLS 隧道中的数据是经过加密的，但是 SSL 通信模块会执行解密操作，将解密后的明文数据写入缓冲区（buf），然后将数据从缓冲区写入 tun0 虚拟网卡中。值得注意的是，当 TLS 隧道中没有数据可读时，读取函数会进入阻塞状态，而不是直接返回长度为 0 的数据。这与后面提到的客户端退出导致的读取长度为 0 的情况是不同的。

在该模块中，有两个关键的代码段。首先，在图 2-4 中展示了从 TLS 隧道读取数据并将其写入 tun0 虚拟网卡的实现代码。其次，在图 2-5 中展示了将数据写入共享内存的代码，以便将 IP 解除占用和管道文件描述符启用的信息传递给其他进程。这些关键代码段确保了数据的传递和共享。

需要特别说明的是，在图 2-5 中，ipflags 是指向共享内存空间的指针，用于保存 IP 占用情况的信息。而 pipefdTables 是指向共享内存空间的指针，用于保存管道文件和其文件描述符之间的映射关系。此外，ipFlagMutexWorker 和 pipefdMutexWorker 是包含相应锁变量的结构体。

在代码中，我们可以观察到以下操作顺序：首先，使用 pthread_mutex_lock 函数对锁进行上锁，然后再访问相应的共享内存空间。例如，(ipflags->area)[malloc_ip] = 0 表示将当前分配

的 IP 的占用情况设置为 0，即表示未被占用。另外，pipefdTable_delete(pipefdTables, worker.pipe, strlen(worker.pipe))表示从 pipefdTables 中删除与 worker.pipe 中保存的管道文件名对应的文件描述符，即执行弃用操作。通过这些操作，程序能够更新 IP 的占用情况并删除不再使用的管道文件对应的文件描述符，以确保相关数据的一致性和正确性。

```
int len = 1;
while (len > 0) {
    char buf[BUFF_SIZE];
    len = SSL_read(ssl, buf, sizeof(buf));
    printf("SSL read %d bytes\n", len);
    write(tunfd, buf, len);
    printf("tun written %d bytes\n", len);
}
```

图 2-4 从 TLS 隧道读取数据并交付给 tun0 的代码展示图

```
pthread_mutex_lock(&ipFlagMutexWorker->mutex);
printf("ip %s 解除占用\n", clientip);
(ipflags->area)[malloc_ip] = 0;
pthread_mutex_unlock(&ipFlagMutexWorker->mutex);

pthread_mutex_lock(&pipefdTableMutexWorker->mutex);
printf("管道文件 %s 对应的文件描述符弃用\n", worker.pipe);
pipefdTable_delete(pipefdTables, worker.pipe, strlen(worker.pipe));
pthread_mutex_unlock(&pipefdTableMutexWorker->mutex);
```

图 2-5 进程间信息传递代码展示图

2.2.2 TUN 通信模块详细设计

TUN 通信模块的主要数据结构如下：

1. 管道文件描述符表 pipefdTable* pipefdTables，通过共享内存进行访问。

2. 管道文件描述符表对应的锁变量 mutexWorker pipefdTableMutexWorker，与 SSL 通信模块中的定义相同。

上述数据结构与 SSL 通信模块中的定义相同，并通过共享内存实现对管道文件描述符的访问。

在 TUN 通信模块中，主要算法的任务是根据从虚拟网卡设备 tun0 中读取到的 IP 数据报的目的 IP，在管道文件描述符表（即 pipefdTables）中查询相应 IP 对应的管道文件的文件描述符。

如果查询得到的文件描述符值为 0，则表示在文件描述符表中不存在有效的对应管道文件的文件描述符。

读取目的 IP 的算法相对简单。首先，从 tun0 中将数据读取到缓冲区 buff 中，然后检查首字节是否为 0x45，其中 4 表示报文协议为 IPv4，5 表示 IP 报文头部长度为 5 * 4，即 20 字节。这样我们可以基本确定接收到的数据是一个完整的 IPv4 报文。接下来，直接读取 buff 的第 17 到 20 个字节，因为这部分是 IP 报文头的目的 IP 地址字段，如图 2-6 中所示。

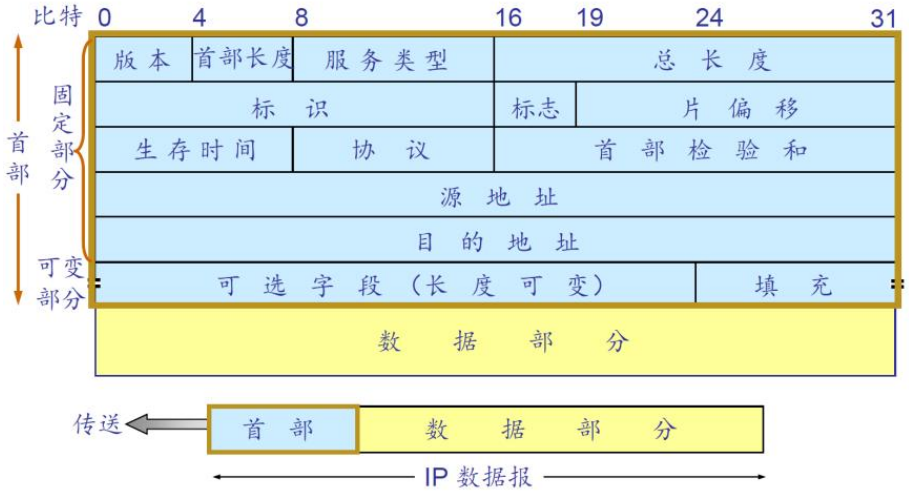

图 2-6 IP 数据报格式展示图

TUN 通信模块的主体函数的执行流程如图 2-7 所示。在图中，程序首先声明一个大小为 4 的整型数组 ipv4，命名为 ipv4，并从指定位置的 buff 中读取目的 IP 的数值。在读取时，必须将数据强制转换为 unsigned char 类型后再读取，这是因为 IP 地址的数值是无符号的，必须使用无符号数据类型进行读取。由于数据是按字节读取的，所以使用 unsigned char 来进行强制转换。读取目的 IP 值完成后，程序使用 sprintf 函数构造包含管道文件完整路径的字符串，并将该字符串存储在变量 ipdst 中，同时还将目的 IP 地址的字符串形式存储在变量 ipdstStr 中。

TUN 通信模块的重点代码在图 2-7 中都有详细给出。其中，特别重要的部分包括从 IP 数据报中读取目的 IP、在文件描述符表中查找记录项、对文件描述符表进行更新以及在访问文件描述符表之前使用跨进程的锁变量进行上锁。这四部分内容对于 TUN 通信模块的正常运行至关重要。

在 TUN 通信模块的代码实现中，图 2-7 中已经提供了以下重要部分：

1. 从 IP 数据报中读取目的 IP 地址。

2. 在文件描述符表中查找相应记录项。

3. 对文件描述符表进行更新。

4. 在访问文件描述符表之前使用跨进程的锁变量进行上锁。

这四部分内容是 TUN 通信模块中的关键代码，它们负责实现从接收到的 IP 数据报中提取目的 IP 地址、查询和更新文件描述符表的相关操作，并使用跨进程的锁变量确保对文件描述符表的访问的互斥性。

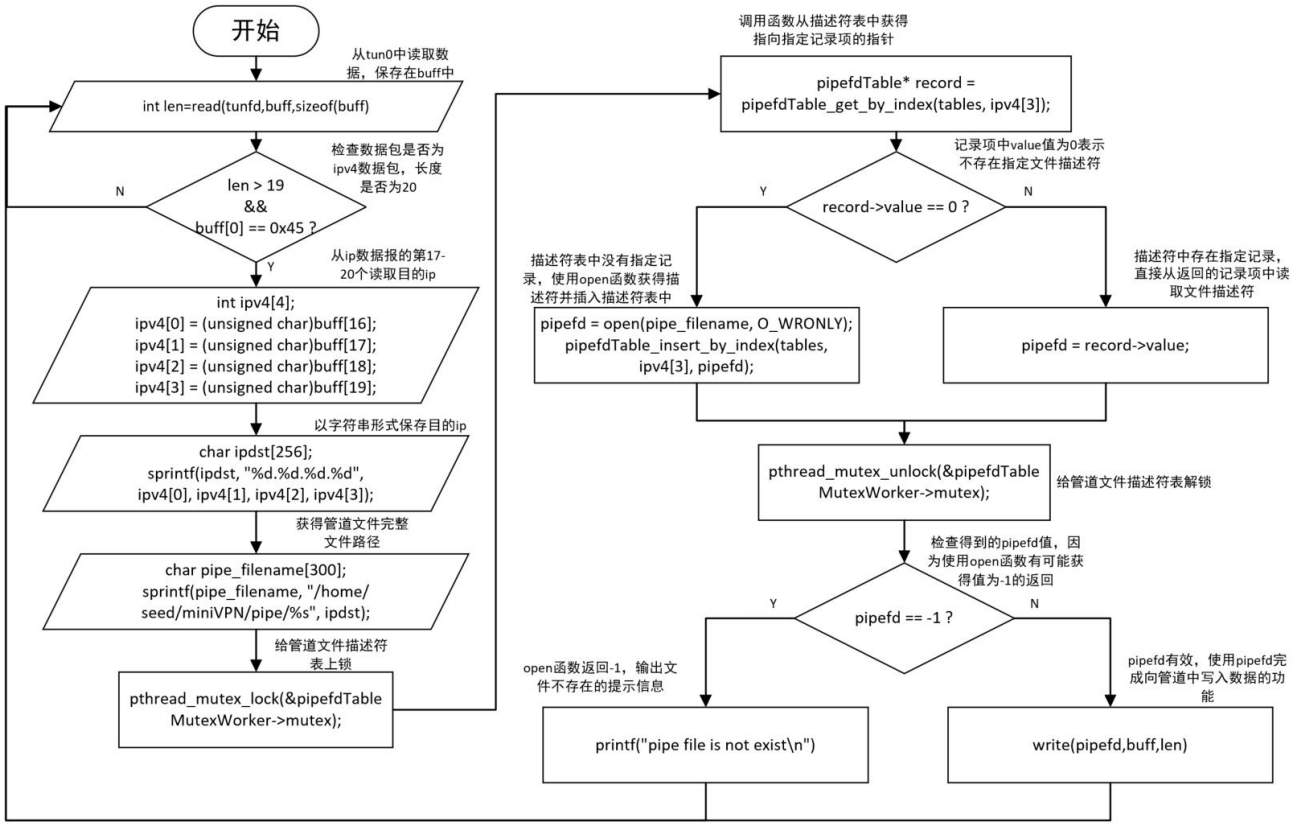

图 2-7 TUN 通信模块主体函数执行流程图

2.2.3 管道通信模块详细设计

管道通信模块主要的数据结构如下：

1. 线程变量 pthread_t pipeThread。

2. 用于承载数据信息进入子线程的结构体 pipeWorker worker。

3. 指向子线程任务函数的函数指针 void* (*pipeListen)(void* _worker)。

pthread_t 结构体由 pthread 库提供，其他自定义结构体如下图 2-8 所示。

```
typedef struct _pipeWorker {
    char pipe[512];
    SSL* ssl;
} pipeWorker;
```

图 2-8 管道通信模块自定义结构体

管道通信模块负责持续监听指定的管道文件，并使用 pipeWorker 结构体中指定的 SSL 结构体将数据发送出去。主要的函数执行流程如图 2-9 所示。

管道通信模块由主体模块启动，主体模块首先创建一个 pipeWorker 结构体变量，并将其中的 SSL 结构体指针和管道文件的完整绝对路径写入。然后，主体模块调用管道通信模块的进入函数 createSubThread，该函数接收一个 pthread_t 线程变量指针和一个 pipeWorker 结构体变量指针作为参数。

一旦进入 createSubThread 函数，就进入了管道通信模块。首先，它尝试使用 Linux 内核提供的 mkfifo 函数创建管道文件，传入参数为指向管道文件完整绝对路径字符串的指针和创建后的权限属性 0666。如果管道文件创建失败（mkfifo 函数返回值为-1），createSubThread 函数会直接退出，并结束管道通信模块的运行。如果创建成功，它将使用 pthread 库提供的 pthread_create 函数启动子线程，并传入 pipeListen 函数的指针作为子线程要执行的函数。

createSubThread 函数执行完毕后，流程图中的执行流程指向 pipeListen 函数。pipeListen 函数的执行流程比较简单。它首先通过管道文件的绝对路径获得文件描述符，然后创建用于暂存从管道中读取数据的缓冲区 buff。接下来，它使用 len 变量作为循环变量，进入读取-发送的循环。在循环体中，len 的值由读取情况决定。当正常读取数据时，len 的值为读取的数据长度。当管道关闭时，read 函数返回值为 0。当读取失败或出错时，read 函数的返回值为-1。如果 len 的值小于等于 0，则循环结束，模块认为通信结束，并使用 remove 函数将管道文件删除。

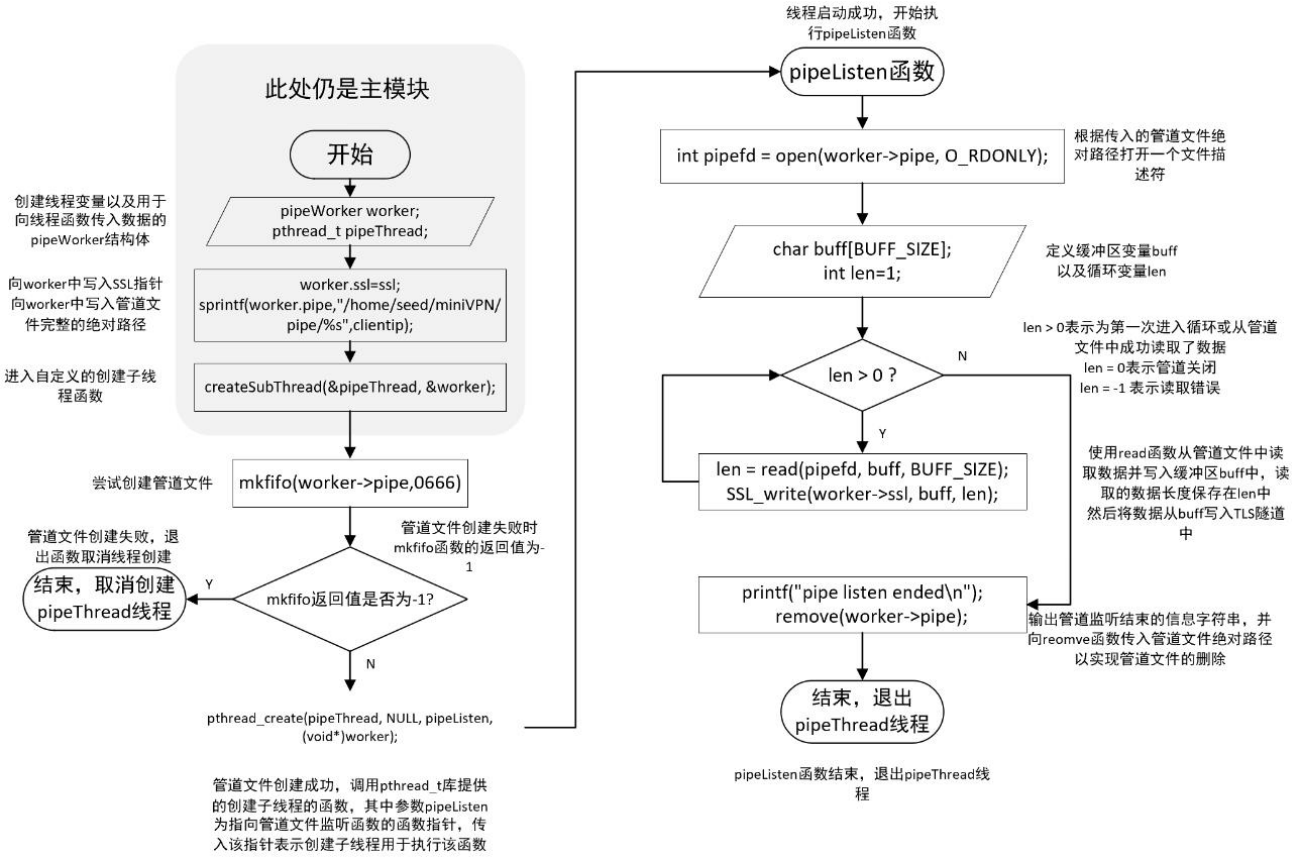

图 2-9 管道通信模块主要函数执行流程图

管道通信模块的关键代码就是 createSubThread 函数中，尝试创建管道文件并根据创建结果决定是否启动子线程的代码，具体代码如图 2-10 所示。

```
void createSubThread(pthread_t* pipeThread, pipeWorker* worker)
{
    printf("try to create pipe %s\n", worker->pipe);
    if (mkfifo(worker->pipe, 0666) == -1) {
        printf("IP %s has been used\n", worker->pipe);
        return;
    } else {
        pthread_create(pipeThread, NULL, pipeListen, (void*)worker);
    }
}
```

图 2-10 createSubThread 函数代码展示图

2.2.4 主体模块详细设计

主体模块中主要的数据结构如下：

1. 两个共享内存区域的 shmget 函数返回的 id，分别是 int ipFlagShareMemoryId 和 int

pipefdTableShareMemoryId。

2. 指向共享内存区域的 IP 占用符数组指针 ipFlag* ipflags。
3. 指向共享内存区域的管道文件描述符表数组指针 pipefdTable* pipefdTables。
4. 两片共享内存区域各自的锁变量 mutexWorker ipFlagMutexWorker 和 mutexWorker pipefdTableMutexWorker。
5. SSL 结构体指针 SSL* ssl。
6. 一组用于启动 TUN 通信模块的变量，包括：用于启动 TUN 通信模块工作线程的线程变量 pthread_t tunListenThread；用于传递数据给子线程的结构体 tunListenWorker tunWorker，其中包含 tunfd（tun0 虚拟网卡的文件描述符）和 pipefdTableShareMemoryId（管道文件描述符表共享内存区域的 id）；指向子线程任务函数的函数指针 void* (*tunListen)(void* _worker)；tun0 虚拟网卡的文件描述符 int tunfd。
7. TCP Socket 监听使用的 socket 号 int tcpListenSock。
8. TCP 建立连接后返回的 socket 号 int tcpClientConnectionSock。

在主体模块中，主要的算法有两个，工作流程如图 2-11 所示。一个是实现为客户端分配 IP 的算法，另一个是实现客户端基于服务器上/etc/shadow 文件登录的算法。

分配 IP 的算法中只需在 ipflags 数组中顺序遍历，直到找到一个满足条件的 IP（即 ipflags[i] 等于 0），表示 IP 地址 192.168.53.i 可用。登录算法中，miniVPN 客户端在获得分配的 IP 信息后，会要求用户输入用于登录的账户密码。主体模块发送完可用的 IP 信息后，开始调用 SSL_read 函数读取 miniVPN 客户端发送的账户密码，并使用 Linux 提供的 shadow 库进行验证。验证结果有三种：登录成功、密码错误、用户用于登录的账户不存在。主体模块使用 1、-1、-2 三个值来代表这三种状态，并在相应时刻将其返回给客户端，以使客户端了解登录结果。

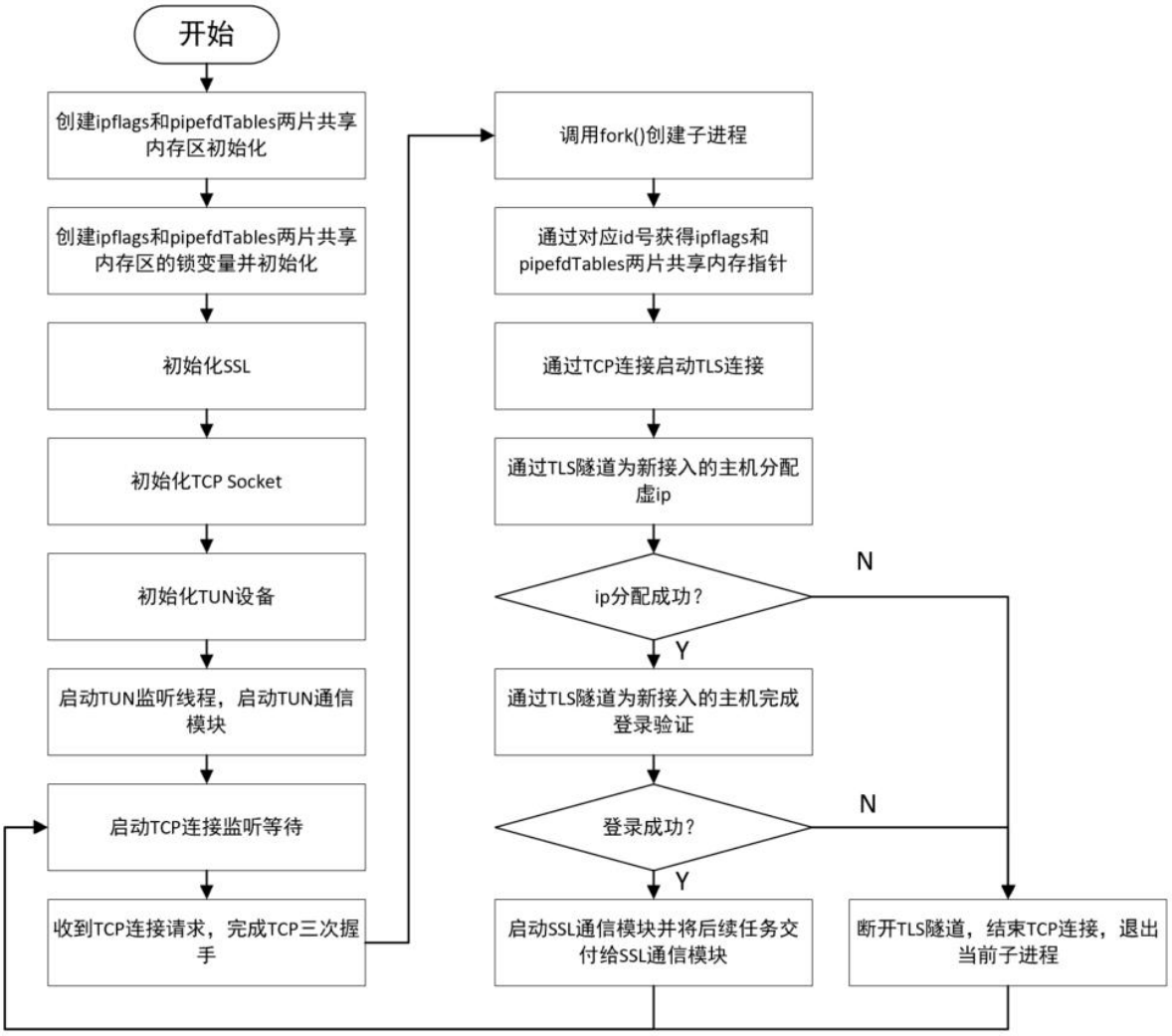

图 2-11 主体模块工作流程图

主体模块负责执行多个任务，并且流程图中通过文字描述了这些任务的工作流程。以下是主体模块中几个重要部分代码的概述。

首先，通过使用 shmget 和 shmat 函数，主体模块声明和访问了共享内存区域。这些函数允许主体模块与其他模块之间共享数据。此外，为了确保数据的同步和访问安全，主体模块还创建了锁变量。这些代码的详细实现可以参考图 2-12。

其次，主体模块包含了一个函数，用于为新接入的客户端分配 IP 地址。这个函数的作用是按顺序遍历 IP 占用符数组，并找到一个可用的 IP 地址分配给客户端。通过这个函数，主体模块能够动态地管理 IP 地址的分配。具体的代码实现可以参考图 2-13。

最后，主体模块还包含了一个用于新接入客户端登录验证的函数。在这个函数中，主体模块接收客户端发送的账户密码信息，并利用 Linux 提供的 shadow 库进行验证。验证结果将分为三种情况：登录成功、密码错误和用户账户不存在。主体模块将使用特定的返回值来表示这

些状态，并及时将结果返回给客户端，使其能够了解登录的结果。图 2-14 中给出了相应的代码实现。

通过上述的代码实现，主体模块能够有效地处理共享内存的声明和访问、为新客户端分配 IP 地址以及进行登录验证等任务。这些功能的实现为整个系统的正常运行提供了基础支持。

```
    //完成共享内存的创建
    int   ipFlagShareMemoryId   =   shmget(IPC_PRIVATE,   sizeof(ipFlag),
IPC_EXCL | 0666);
    ipFlag* ipflags = (ipFlag*)shmat(ipFlagShareMemoryId, NULL, 0);

    //完成进程同步锁的创建，用于对共享内存访问过程上锁
    ipFlagMutexWorker  =  mmap(NULL,  sizeof(mutexWorker),  PROT_READ  |
PROT_WRITE, MAP_SHARED | MAP_ANON, -1, 0);
    memset(ipFlagMutexWorker, 0, sizeof(mutexWorker));
    mutexWorkerInit(ipFlagMutexWorker);
```

图 2-12 共享内存声明及创建锁变量代码展示图

```
int getAvailableIP(char* clientip, ipFlag* flags)
{
    pthread_mutex_lock(&ipFlagMutexWorker->mutex);
    for (int i = 3; i < 255; ++i) {
        if ((flags->area)[i] == 0) {
            sprintf(clientip, "192.168.53.%d", i);
            (flags->area)[i] = 1;
            pthread_mutex_unlock(&ipFlagMutexWorker->mutex);
            return i;
        }
    }
    pthread_mutex_unlock(&ipFlagMutexWorker->mutex);
    return 0;
}
```

图 2-13 为新接入客户端分配 IP 函数代码展示图

```c
int login(SSL* ssl)
{
    char userName[256];
    char passwd[1024];
    char* invalid_user = "-1";
    char* invalid_passwd = "-2";
    char* success = "1";
    char* fail = "-3";

    int len;
    int threeChance = 3;
    while (threeChance != 0) {
        len = SSL_read(ssl, userName, sizeof(userName));
        userName[len - 1] = '\0';
        len = SSL_read(ssl, passwd, sizeof(passwd));
        passwd[len - 1] = '\0';
        struct spwd* user_profile;
        char* hash_passwd;
        user_profile = getspnam(userName);
        if (user_profile == NULL) {
            printf("password of %s not found or it is not a valid user\n", userName);
            SSL_write(ssl, invalid_user, strlen(invalid_user) + 1);
            //登录失败 原因是账户没有密码或者没有这个账户
            threeChance--;
            continue;
        }
        hash_passwd = crypt(passwd, user_profile->sp_pwdp);
        if (strcmp(hash_passwd, user_profile->sp_pwdp) != 0) //不为0表示两个字符串不相等
        {
            printf("Invalid password\n");
            SSL_write(ssl, invalid_passwd, strlen(invalid_passwd) + 1);
            //登录失败 原因是密码错误
            threeChance--;
            continue;
        }
        SSL_write(ssl, success, strlen(success) + 1);
        return 1;
    }
    SSL_write(ssl, fail, strlen(fail) + 1);
    return -3;
}
```

图 2-14 登陆验证服务函数代码展示图

2.3 miniVPNclient 详细设计

主体模块中主要的数据结构图下：

1. 指向服务器域名字符串的指针 char* hostname.
2. 保存 miniVPNserver 在服务端监听端口号的整型值 int port.
3. SSL 结构体指针 SSL* ssl.

主体模块的主要任务是与 miniVPNserver 建立 TCP 连接，进行身份验证和密钥协商，并建立 TLS 隧道，获取服务端分配的虚拟 IP，并实现登录验证功能。完成这些任务后，主体模块开始进行隧道通信。整个工作流程如图 2-15 所示。

模块中，重要的代码分别有两处，一处是实现登录验证部分的代码，其内容如图 2-16 所示，一处是实现用户登录时输入密码不回显的代码，这一处代码在后面有展示。

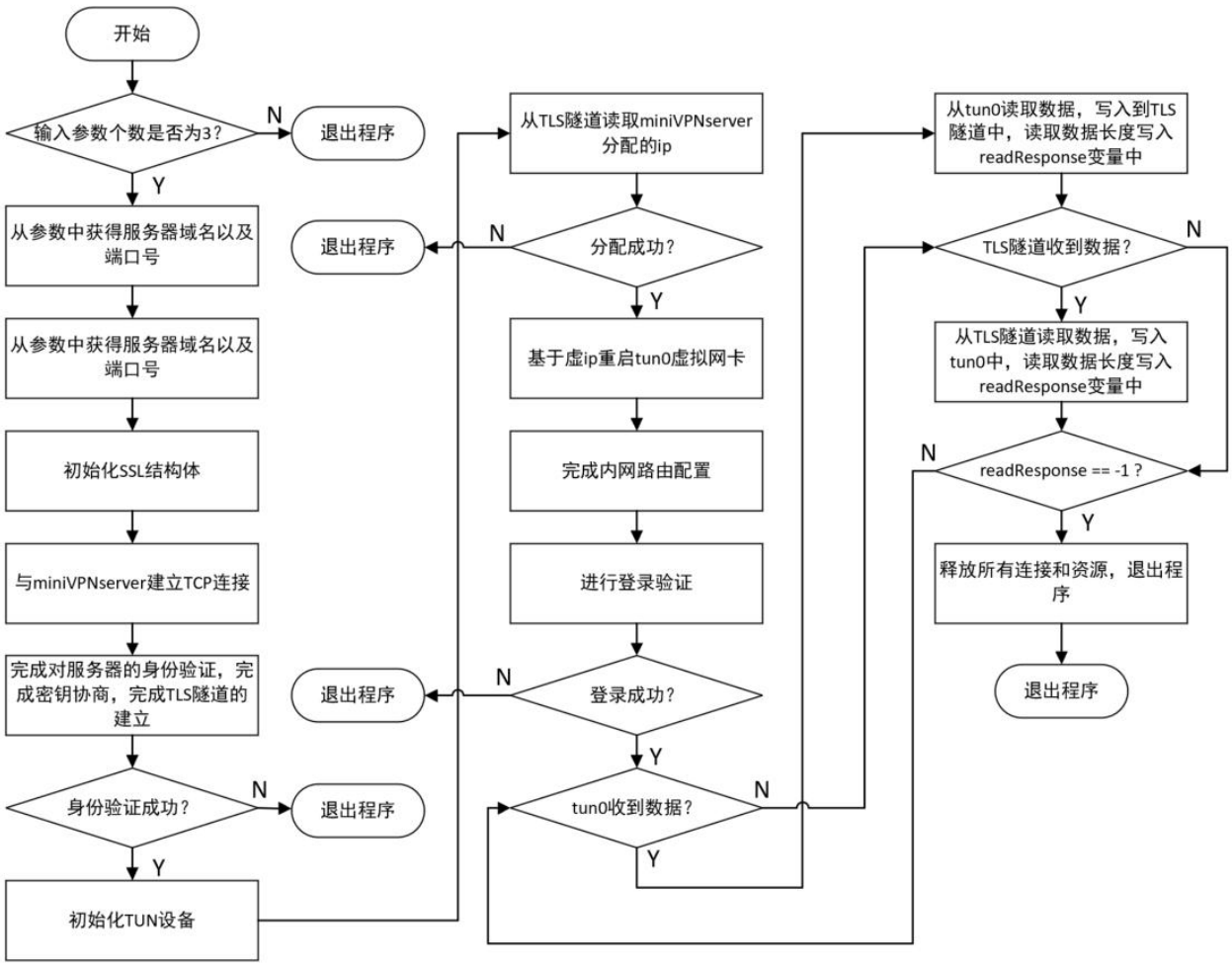

图 2-15 miniVPNclient 主体模块工作流程图

```
//下面完成登录操作,有三次尝试的机会
char user[256];
char passwd[256];
char loginBuff[256];
SSL_read(ssl, loginBuff, sizeof(loginBuff));
for (int i = 0; i < 3; ++i) {
    printf("Please input username:");
    fgets(user, 199, stdin);
    user[strlen(user) - 1] = '\0';
    printf("Please input password:");
    //关闭输入时的回显，保护密码的安全
    set_disp_mode(STDIN_FILENO, 0);
    fgets(passwd, 199, stdin);
    set_disp_mode(STDIN_FILENO, 1);
    //密码输入完成后重新启用回显
    passwd[strlen(passwd) - 1] = '\0';
    SSL_write(ssl, user, sizeof(user));
    SSL_write(ssl, passwd, sizeof(passwd));
    int replen = SSL_read(ssl, loginBuff, sizeof(loginBuff));
    if (replen <= 0) {
        printf("Server response error.Maybe server has exited.\n");
        printf("response length: %d\n", replen);
        printf("This client is going to exit.\n");
        exit(0);
    }
    printf("\nlogin response: length is %d,content is %s\n", replen, loginBuff);
    int response = atoi(loginBuff);
    if (response == 1) {
        printf("login successfully\n");
        break;
    } else if (response == -1) {
        printf("password of %s not found or it is not a valid user\n", user);
    } else if (response == -2) {
        printf("Invalid password\n");
    }
    if (i == 2) {
        printf("Login failed.miniVPN is going to exit.\n");
        exit(1);
    }
}
```

图 2-16　miniVPNclient 实现登录验证代码片段展示图

3　VPN 实现细节

（1）　回答实验指导手册 4.2 第 6 步提出的问题：在 HostU 上，telnet 到 HostV。在保持 telnet 连接存活的同时，断开 VPN 隧道。然后我们在 telnet 窗口中输入内容，并报告观察到的内容。然后我们重新连接 VPN 隧道。需要注意的是，重新连接 VPN 隧道，需要将 vpnserver 和 vpnclient 都退出后再重复操作，请思考原因是什么。正确重连后，telnet 连接会发生什么？会被断开还是继续？请描述并解释你的观察结果。

观察到，在 telnet 连接窗口中执行命令时，并没有任何反应或变化。然而，当我们重新启动服务器程序和客户端程序，并重新连接 VPN 隧道后，我们可以观察到之前输入的命令会在原 telnet 连接窗口中执行一次，而 telnet 连接保持不变。

可能的原因是，当隧道断开时，TCP 连接并未中断。只有重新启动客户端程序或

服务器程序是无法重新连接之前的连接的，因此我们需要双方都退出并重新连接，才能建立新的隧道。在隧道断开期间，TCP 会将无法执行的命令缓存，并不断尝试执行。一旦隧道重新连接，并且指令信息能够发送出去，那么指令就可以成功执行。

（2） 你的 VPN 客户端，是如何知道并配置自己的虚 IP 和服务器保护子网路由的？

虚拟 IP：每个 miniVPNclient 在启动时，使用保留的未使用 IP 地址 192.168.53.2 来配置 tun0 网卡。TLS 隧道建立后，miniVPNserver 和 miniVPNclient 之间的通信实际上是在它们所在机器的真实物理网卡上进行的，这些网卡都有唯一确定的 IP 地址，不依赖于 tun0 虚拟网卡的 IP。因此，即使同时有两个 miniVPNclient 使用 192.168.53.2 连接到 miniVPNserver，报文的源机器也能够正确识别它们。这是因为 tun0 虚拟网卡的 IP 地址在客户端与内网机器进行通信时才会生效，而 IP 地址的分配是在 miniVPNclient 和 miniVPNserver 之间进行的。因此，所有的 miniVPNclient 都可以使用 192.168.53.2 作为初始启动 tun0 虚拟网卡的 IP 地址，但是 miniVPNserver 在分配 IP 地址时必须将这个 IP 保留不使用，否则当有一个 miniVPNclient 实际被分配到这个 IP 时，会导致报文转发错误的情况发生。

之后，miniVPNclient 使用 192.168.53.2 启动 tun0 虚拟网卡后，会向 miniVPNserver 发起 TLS 连接。连接成功后，在隧道中写入字符串"192.168.53.2"。miniVPNserver 读取到这个 IP 后，会知道需要为其分配一个 IP 地址。然后，miniVPNserver 的 SSL 通信模块会查询可用的 IP 地址，并将其标记为已占用。然后，它会以完整的 IP 字符串形式将这个 IP 写入 TLS 隧道，例如"192.168.53.3"。miniVPNclient 收到这个 IP 后，会使用它重新启动 tun0 虚拟网卡。

子网路由：miniVPNclient 会自动为设备配置到内部子网的路由，这是通过在程序中添加代码 system(“route add -net 192.168.60.0/24 tun0”)实现的，这一行代码将在完成 ip 分配之后才执行，否则 ip 分配过程中的 tun0 虚拟网卡重启的操作将导致导致路由失效。

（3） 你的 VPN 客户端用户名口令认证是在隧道协商的什么阶段实现的？认证失败时双方是如何处理的？

miniVPNclient 的用户名口令认证是在 TLS 隧道已经建立完成，ip 分配成功之后，程序启动对 tun0 虚拟网卡的监听之前实现的。也就是当客户端用户名口令认证成功后，才启动对 tun0 虚拟网卡监听，在这之前，虽然从客户端设备到内部子网的路由已经添

加，但是由于写入到 tun0 虚拟网卡中的报文不会被读取，导致从客户端设备到内部子网的网络通信实际上仍然是阻塞的。认证将有 3 次机会，失败时，miniVPNclient 会关闭 TLS 隧道，并且自动结束程序，而 miniVPNserver 由于此时已经完成了 ip 分配，因此还需要将 ipflags 中这次连接使用的 ip 改为空闲，然后再结束为这一次隧道通信创建的子进程。

（4） 你的 VPN 服务器支持多客户端采用的什么技术？VPN 服务器收到保护子网主机的应答报文时，如何判断应发送给哪个 VPN 客户端的隧道？

支持多客户端采用的是多进程架构，miniVPNserver 会使用 fork()函数为每一个连接启动一个单独的进程。进程间通信采用的是共享内存技术，同时为了保证同一时间只有一个进程访问这片内存空间，程序还使用了可在进程之间使用的锁机制。

miniVPNserver 收到来自内网主机的报文之后，会读取报文的第 17 到第 20 个字节，这一部分正是 ipv4 报文的目标 ip 地址部分，得到目的 ip 之后即可将这个报文写入与 ip 同名的管道文件中。即 miniVPNserver 创建管道文件时，根据 miniVPNclient 连接后分配的虚 ip 来创建管道文件，如当 miniVPNclient 获得的虚 ip 为 192.168.53.5 时，就会创建管道文件/home/seed/miniVPN/pipe/192.168.53.5。故可根据内网主机所发送报文的目的 ip 来确定应当将报文写入哪一个管道文件中。

（5） 你的 VPN 客户端退出时，VPN 服务器如何发现？如何处理？

miniVPNclient 退出时，会在 TLS 隧道中写入一个长度为 0 的数据，这在正常通信过程中是不会出现的，因此当 miniVPNserver 从 TLS 隧道中读到一个长度为 0 的数据时，会判断对端 miniVPNclient 已经退出，于是结束通信，将为该 miniVPNclient 分配的虚 IP 改为空闲后退出子进程。

4 测试结果与分析

4.1 连通性测试

（1） 单客户端

首先在 VM 上启动 miniVPNserver，然后在 HostU 上启动 miniVPNclient，再另起一个 HostU 的命令行，然后尝试 ping HostV（192.168.60.101），结果如图 4-1 所示。在图中可以看到客户端在验证服务端证书时输出了证书中的详细信息。

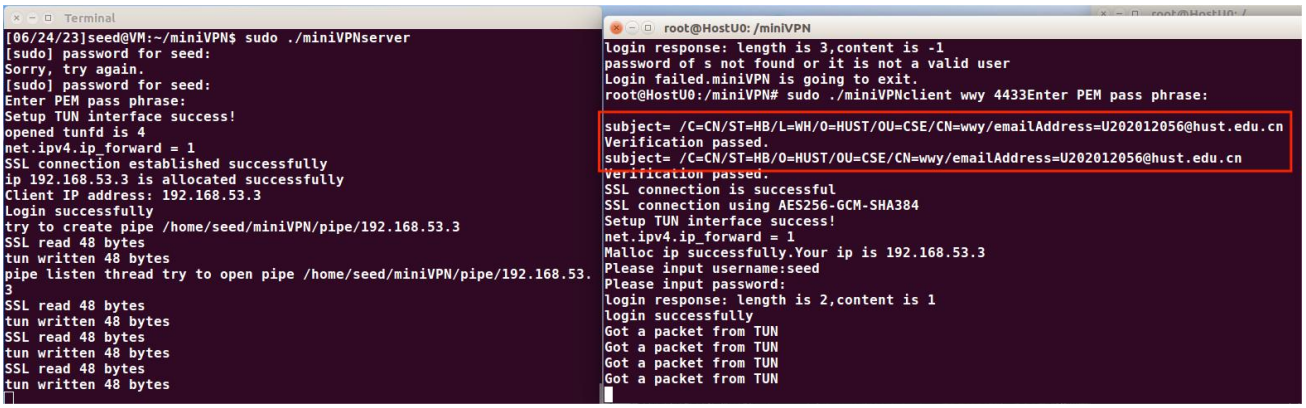

图 4-1 基础连通性测试结果图

（2） 多客户端

新增一个客户端运行在 HostU1 上，两个客户端连通之后同时进行 ping 操作。测试结果如图 4-2 所示。可以在图中看到 HostU1 和 HostU0 都能够正常的 ping 通，HostX 的 ip 也自动分配为 192.168.53.4，服务器端交替显示来自两个客户端的活动信息。

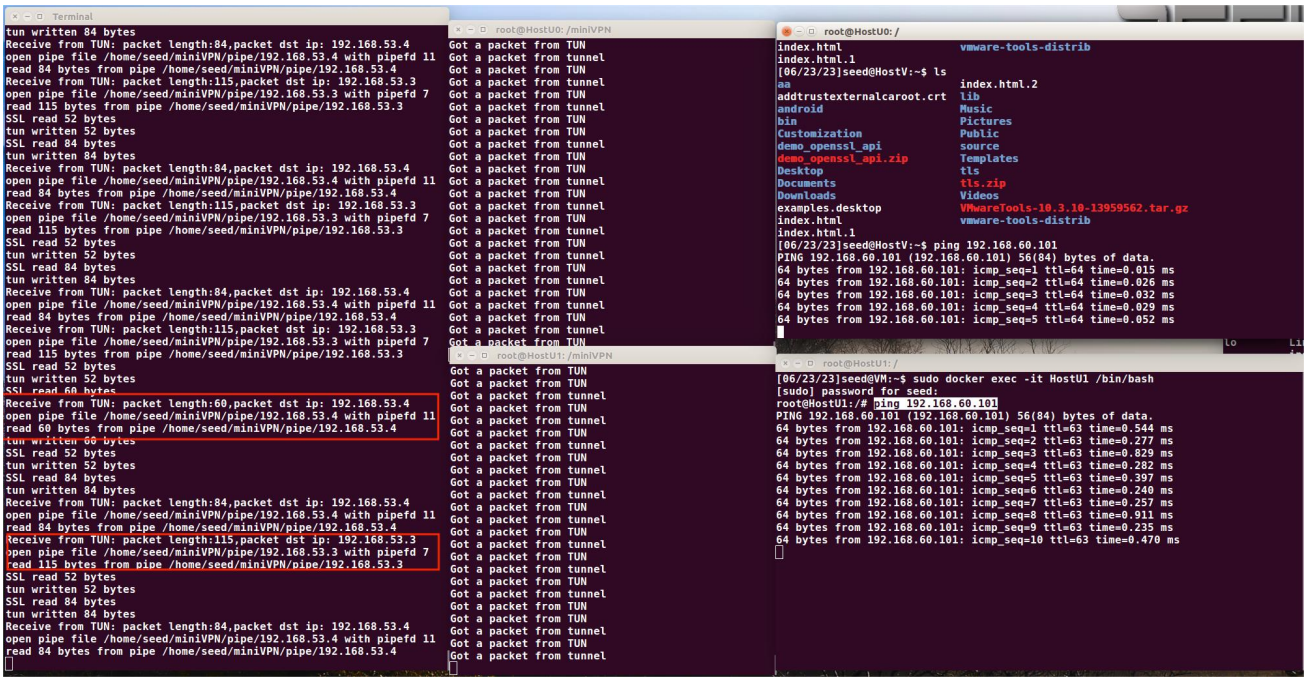

图 4-2 双客户端连通测试

4.2 安全性测试

（3） 用 openssl 检查 VPN 服务器证书信息

在 seed@VM 中运行指令：openssl x509 -noout -text -in cert_server/server-wwy-crt.pem。如下图 4-3 所示，证书主题包含个人信息，符合要求。

```
[06/24/23]seed@VM:~/miniVPN$ openssl x509 -noout -text -in cert_server/server-wwy-crt.pem
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4096 (0x1000)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=CN, ST=HB, L=WH, O=HUST, OU=CSE, CN=wwy/emailAddress=U202012056@hust.edu.cn
        Validity
            Not Before: Jun 22 16:56:58 2023 GMT
            Not After : Jun 21 16:56:58 2024 GMT
        Subject: C=CN, ST=HB, O=HUST, OU=CSE, CN=wwy/emailAddress=U202012056@hust.edu.cn
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (1024 bit)
```

图 4-3 用 openssl 检查 VPN 服务器证书信息

（4） 使用错误的账户密码进行登录

在图 4-4 中可以看到，登录的第一次尝试中使用了错误的密码，程序返回了密码无效的结果，拒绝了我们的登录。第二次尝试中使用错误账户 s 登录，程序返回了账户无效的结果，再次拒绝了我们的登录。尝试三次过后，服务器依次显示用户登陆状态，客户端则自动退出。

```
SSL connection established successfully
ip 192.168.53.3 is allocated successfully
Client IP address: 192.168.53.3
Invalid password
password of s not found or it is not a valid user
password of s not found or it is not a valid user
ip 192.168.53.3解除占用
root@HostU0: /miniVPN
subject= /C=CN/ST=HB/L=WH/O=HUST/OU=CSE/CN=wwy/emailAddress=U202012056@hust.edu.cn
Verification passed.
subject= /C=CN/ST=HB/O=HUST/OU=CSE/CN=wwy/emailAddress=U202012056@hust.edu.cn
Verification passed.
SSL connection is successful
SSL connection using AES256-GCM-SHA384
Setup TUN interface success!
net.ipv4.ip_forward = 1
Malloc ip successfully.Your ip is 192.168.53.3
Please input username:seed
Please input password:
login response: length is 3,content is -2
Invalid password
Please input username:s
Please input password:
login response: length is 3,content is -1
password of s not found or it is not a valid user
Please input username:s
Please input password:
login response: length is 3,content is -1
password of s not found or it is not a valid user
Login failed.miniVPN is going to exit.
```

图 4-4 使用错误账户和错误密码登录尝试示意图

（5） 证书过期

由于证书的有效期为一年，这里将容器的时间改为 2024 年使得服务器的证书过期，测试结果如图 4-5 所示。可以在图中清楚的看见，客户端在验证证书时发现证书过期，于是直接退出了程序，结束了和服务端的连接。

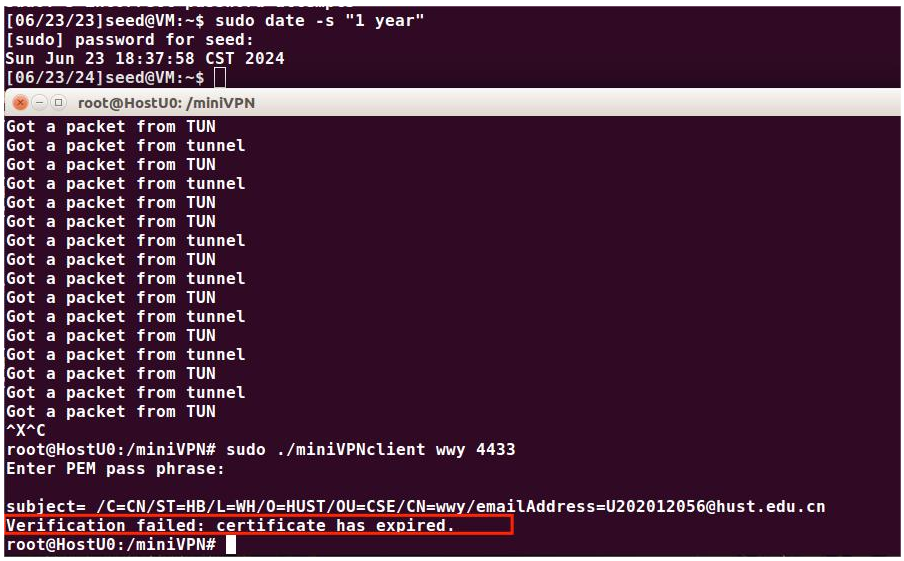

图 4-5 证书过期测试结果图

（6） 加密通道测试

VPN 客户端 ping 内网主机，wireshark 在 VPN 服务器外口截包检查。如下图 4-6 所示，VPN 客户端 ping 内网主机，并在 wireshark 选择网卡 docker1 进行截包检查，发现能通信、经隧道封装、隧道为 TLS，符合要求。

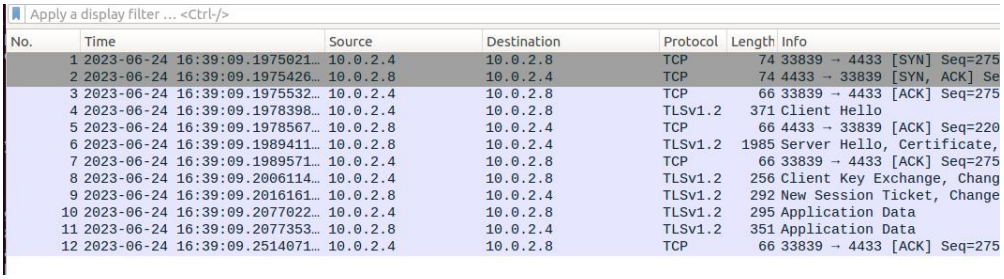

图 4-6 SSL 隧道加密通信测试结果图

4.3 稳定性测试

这一部分的设计受益于文件描述符表的功能，使得文件描述符能够得到复用，从而使程序能够在大量的数据包冲击下正常工作。这样的设计避免了重复获取文件描述符和打开管道文件的过程，防止程序耗尽文件描述符资源。在 Linux 系统中，每个进程最多持有 1024 个文件描述符，文件描述符的作用类似于 Windows 系统中的句柄。

如果程序没有实现文件描述符的复用功能，它将无法通过这一部分的测试。大多 VPN 程序的设计在获取文件描述符时会重新打开管道文件来获取新的文件描述符，并且不会释放之前使用的文件描述符，而这样的程序无法通过这一测试。

这项测试的方法是同时在 HostU0 和 HostU1 上执行命令"ping 192.168.60.101 -f"，其中参数

"-f"使得程序每 1 秒发送 100 个 ICMP echo 数据包，而不是正常情况下的 1 秒 1 个。

测试结果如图 4-7 所示，HostU 发送了近 7000 个数据包，HostX 发送了近 9000 个数据包，而 VPN 程序仍然能够正常运行，并且平均延迟保持在 3ms 左右，该程序成功通过了这一测试。

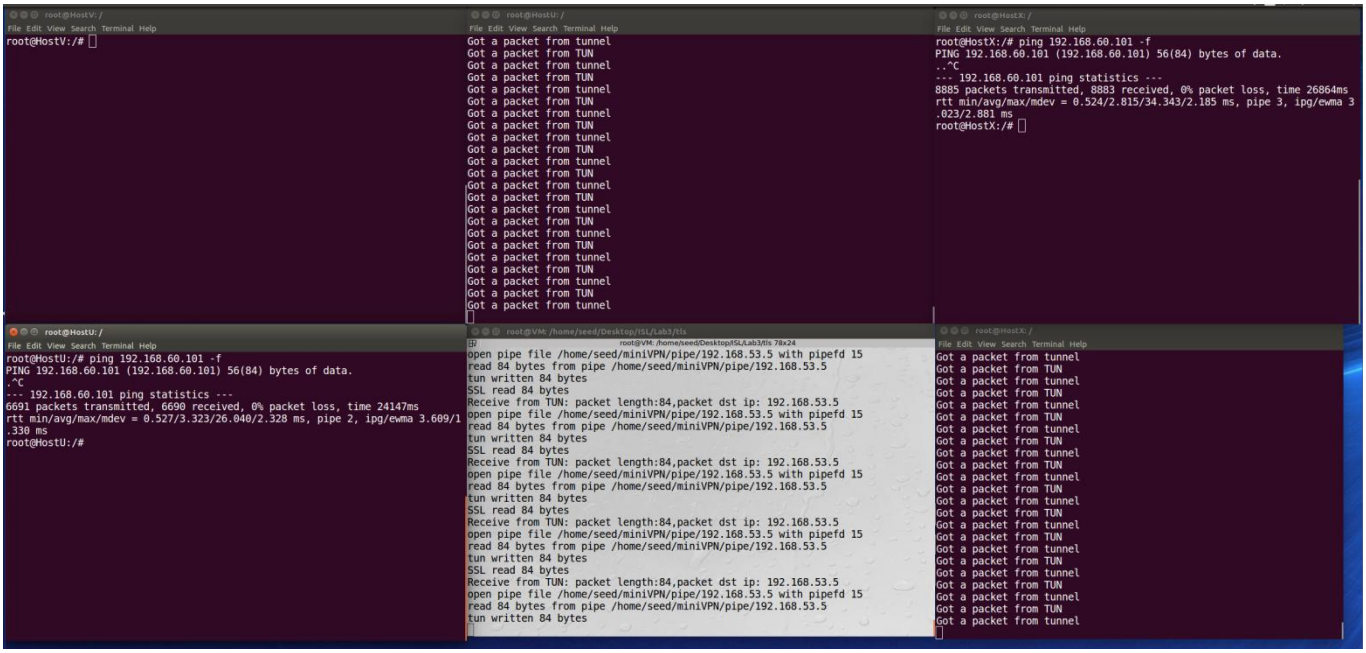

图 4-7 ping -f 命令测试

5 体会与建议

5.1 心得体会

本次实验使我深入了解了 TLS VPN 的工作流程，同时为了实现证书的验证和证书的生成，我比过去更加深入的了解了公钥基础设施的工作流程，了解了数字签名的工作流程。同时，为了实现进程间的通信，我第一次接触了 Linux 环境下的共享内存和跨进程的锁机制，这些都使我受益匪浅，坦白说，Linux 环境下的共享内存机制相比 Windows 环境下的共享内存要更加方便易用。

在这次实验过程中，我因为实验做到一半跑去上课，才意外发现了原先文件描述符获取机制的漏洞，发现了这个使得程序不能长时间工作的巨大缺陷，经过长时间的思考以及在脑海中模拟程序的运行逻辑，才终于怀疑可能是 open 函数重复打开同一个文件时获取到的文件描述符可能不是同一个，然后又在脑海中类比文件描述符与 Windows 环境下句柄的关系，才得出可能是文件描述符资源耗尽导致的这个问题，经过查找资料了解到 Linux 确实会限制每个进程

获取文件描述符个数之后，我查看了程序无法正常运行时和正常运行时所持有的文件描述符的个数，终于定位了这个问题。这才使得我决定设计一个文件描述符复用的机制，来保证程序不会重复的获取文件描述符导致程序资源耗尽而无法正常运行。

在本次实验里，除了网络通信的问题以外，绝大多数问题都是由程序的并发导致的，因此这次程序的编写也使我学习了很多并发相关的知识。

5.2 意见建议

希望老师能更多的讲解公钥基础设施相关的知识，这一部分在程序编写初期使我困惑良久。