安全文件传输系统

1 主要功能

实现客户端与服务器之间互传文件，客户端可以浏览本地文件列表和服务器文件列表，可以将本地文件上传至服务器或者从服务器上下载所需文件

2 基础知识

2.1 SLL

SSL(Secure Socket Layer)是netscape公司提出的主要用于web的安全通信标准，可以保证应用层数据在互联网络传输不被监听,伪造和窜改。

SSL是为了加密传输数据而产生的协议，可以这么理解，它是位于应用层和TCP/IP之间的一层，数据经过它流出的时候被加密，再往TCP/IP送，而数据从TCP/IP流入之后先进入它这一层被解密，同时它也能够验证网络连接俩端的身份。

2.2 openssl

简介：openssl是一个功能丰富且自包含的开源安全工具箱。它提供的主要功能有：SSL协议实现(包括SSLv2、SSLv3和TLSv1)、大量软算法(对称/非对称/摘要)、大数运算、非对称算法密钥生成、ASN.1编解码库、证书请求(PKCS10)编解码、数字证书编解码、CRL编解码、OCSP协议、数字证书验证、PKCS7标准实现和PKCS12个人数字证书格式实现等功能

生成数字证书和密钥:

建立SSL连接:

SSL库初始化

载入所有 SSL 算法

载入所有 SSL 错误消息

创建本次会话连接所使用的协议

申请SSL会话的环境 CTX

载入用户的数字证书

载入用户私钥

检查用户私钥是否正确

开启一个 socket 监听

等待客户端连上来

基于 ctx 产生一个新的 SSL

将连接用户的 socket 加入到 SSL

建立 SSL 连接

2.3 线程池

线程池中的线程由系统管理，程序员不需要费力于线程管理，可以集中精力处理应用程序任务。线程池是一种多线程处理形式，处理过程中将任务添加到队列，然后在创建线程后自动启动这些任务。线程池线程都是后台线程.每个线程都使用默认的堆栈大小,以默认的优先级运行,并处于多线程单元中。

当服务器同时接收到多个客户端的连接时，服务器需不停的创建和销毁线程，这将耗费大量系统资源，故采用线程池管理线程，重复利用已创建的线程，在线程池管理中需要使用条件变量来实现线程的阻塞和唤醒。

2.3 条件变量

条件变量是线程可用的另一种同步机制，条件变量给多个线程提供了一个会合的场所，条件变量和互斥量一起使用时，允许线程以无竞争方式等待特定条件发生。

条件本身由互斥量保护，线程在改变条件状态前必须先锁住互斥量。与互斥锁不同，条件变量是用来等待而不是用来上锁的。条件变量用来自动阻塞一个线程，直到某特殊情况发生为止。通常条件变量和互斥锁同时使用。

条件变量是利用线程间共享的全局变量进行同步的一种机制，主要包括两个动作：一个线程等待"条件变量的条件成立"而挂起；另一个线程使"条件成立"。

3 项目实现

3.1 传输协议设计

通信双方需要遵循一定的传输协议，利用一个结构体来实现协议。

```
struct FilePackage{
    char cmd;               //操作命令
    int filesize;           //每次传输数据包大小
    int ack;                //标志位
    char username[50];      //客户端用户名
    char filename[125];     //传输文件名
    char buf[1024];         //传输文件的元数据
}
```

数据包中cmd和ack的具体设计：

登陆：cmd:L

服务器ack	含义	客户端ack	含义
登陆			
0	用户名或密码错误	9	登陆服务器
1	登陆成功		
2	客户端最大连接数		
下载			
0	接受下载，返回待下载文件大小	9	请求下载
2	开始下载	3	接受完毕
4	下载完毕		
上传			
0	接受上传请求	9	请求上传
1	本地磁盘空间不足	2	开始上传文件
3	接收完毕	4	上传完毕
显示文件列表			

3.2 服务器设计

管理员登陆服务器，登陆成功进行相关初始化工作。主函数产生两个线程，一个菜单线程，用于显示菜单及处理用户输入；一个服务线程，先初始化SSL库等一系列工作，然后建立socket套接字，监听网络，等待客户端连接，当收到客户端的连接请求时，利用条件变量唤醒线程池中的一个线程从线程池中分配一个线程处理客户端的连接，接收客户端命令并进行相应操作，客户端与服务器间的通信采用OpenSSL进行加密传输。

```
int main(int argc, char *argv[])
{
    /* 初始化客户端最大连接数，从maxclientnum.txt文件中读取信息 */
    InitMaxClientNum();
    /* 初始化管理员和用户，从admin.txt和user.txt文件中读取信息 */
    InitAU();
    /* 管理员登录 */
    /* 创建主菜单线程 */
    pthread_create(&controlId,NULL,(void *)mainMenu,NULL);
    /* 创建主处理线程 */
    pthread_create(&mainId,NULL,(void *)mainThread,NULL);
    /* 等待主菜单线程退出 */
    pthread_join(controlId,NULL);
    /* 等待主处理线程退出 */
    pthread_join(mainId,NULL);
}
void mainMenu(){
```

```
    /* 打开maxclientnum.txt，admin.txt和user.txt文件 */
    /* 进入死循环 */
    while(1){
        /* 打印菜单 */
        switch(choice)
        {
            /* 服务器配置: 设置最大连接数、增加管理员、增加用户 */
            case 1: ...;
            /* 运行服务器 */
            case 2: ...;
            /* 关闭服务器 */
            case 3: ...;
            /* 打印系统日志 */
            case 4: ...;
        }
    }
}
void mainThread(){
    /* SSL库初始化 */
    /* 载入所有SSL算法 */
    /* 载入所有SSL错误消息 */
    /* 以 SSLV2和V3标准兼容方式产生一个SSL_CTX */
    /* 载入用户的数字证书 */
    /* 载入用户私钥 */
    /* 检查用户私钥是否正确 */
    /* 初始化socket套接字 */
    /* 创建线程池 */
    CreateThreadPool();
    /* 进入死循环，判断各变量进行相应操作 */
    while(1){
        /* 判断是否退出服务器 */
        if(IsExit==1) ...;
        /* 判断服务器是否正常运行 */
        if(IsRun==1){
            /* 监听客户端的连接，阻塞直到客户端请求连接 */
            /* 改变条件变量状态，系统唤醒线程池中的一个等待线程处理客户端连接 */
            pthread_cond_signal(&pthreadCond);
        }
    }
    /* 释放CTX */
    /* 等待处理线程退出 */
    pthread_join(id,NULL);
}
void CreateThreadPool(){
    for(...){
        /* 创建线程池中的线程 */
        pthread_create(&tid,NULL,(void *)process,NULL);
    }
}
void process(){
tap:
    /* 对互斥量上锁 */
    pthread_mutex_lock(&pthreadMutex);
    /* 进入阻塞状态，表明处于空闲，等待条件变量改变 */
    pthread_cond_wait(&pthreadCond,&pthreadMutex);
    /* 解锁互斥量，表明处于忙碌 */
    pthread_mutex_unlock(&pthreadMutex);
    /* 基于ctx产生一个新的SSL */
```

```
    /* 将连接客户端的socket加入到SSL */
    /* 建立SSL连接 */
    /* 若客服端连接数达到最大，则断开连接*/
    if(CurrentClientNum>MaxClientNum) ...;
    /* 进入死循环，处理客户端发送的数据包 */
    while(1){
        /* 接收客户端的数据包 */
        SSL_read(NewFd,&buff,sizeof(struct FilePackage));
        /* 客户端退出，关闭连接 */
        if(buff.cmd=='Q' && buff.ack=='0') ...;
        /* 数据包解包并进行相应处理 */
        else{
            sendPackage=unpack(NewFd,buff);
        }
    }
    --CurrentClientNum;
    goto tap;
}
struct FilePackage unpack(SSL *NewFd,struct FilePackage tpack){
    switch(tpack.cmd)
    {
        /* 用户登录 */
        case 'L': ...;
        /* 上传文件 */
        case 'U': ...;
        /* 显示文件列表 */
        case 'S': ...;
        /* 下载文件 */
        case 'D':
            /* 检查文件是否存在 */
            /* 发送数据包告知客户端文件大小 */
            /* 发送文件，每次发送1024字节即1k数据 */
            while((count=read(Fd,(void *)buf,1024))>0){
                SSL_write(NewFd,&sendPack,sizeof(struct FilePackage));
            }
            /* 发送数据包告知客户端发送完毕 */
            /* 接收数据包表明客户端接收完毕 */
            /* 写入日志 */
            break;
    }
}

```

3.3 客户端设计

```
int main(int argc,char *argv[])
{
    /* SSL库初始化 */
    /* 连接到服务器 */
    connectto(argc,args);
    /* 用户登录 */
    login(username,userpasswd);
    /* 显示主菜单 */
    mainMenu();
}
```

```c
int connectto(int argc, char *argv[]){
    /* 初始化socket */
    /* 基于ctx产生一个新的SSL */
    /* 建立SSL连接 */
    SSL_connect(ssl);
    /* 显示数字证书 */
    ShowCerts(ssl);
}
int login(char username[20],char userpasswd[10]){
    /* 输入用户名和密码 */
    /* 发送登录请求 */
    /* 用户名或密码错误 */
    if(data.cmd == 'L' && data.ack == 0) ...;
    /* 登录成功 */
    if(data.cmd == 'L' && data.ack == 1) ...;
    /* 连接数达到最大 */
    if(data.cmd == 'L' && data.ack == 2) ...;
}
void mainMenu(){
    /* 进入死循环 */
    while(1){
        switch(temp){
            /* 上传文件 */
            case 1:
                /* 实现同时上传多个文件，每上传一个文件就建立一个连接 */
                while(Files[count]!='\0' && Files[count]!='\n'){
                    pthread_create(&pthreadt,NULL,UpdateF,(void
*)&Files[temp1]);
                }
                break;
            /* 下载文件 */
            case 2: ...;
            /* 退出 */
            case 3: ...;
        }
    }
    /* 打印菜单 */

}
void UpdateF(void *filename){
    usleep(500);
    /* 初始化socket */
    /* 建立SSL连接 */
    /* 上传文件 */
}
void * DownloadF(void *filename){
    /* 下载文件 */
}
```