**1. 配置文件的作用**

保存系统或程序运行时需要加载的运行参数。

**2. 常用格式**

* 格式一：

#注释行

[Section]

key=value

* 格式二：

Xml格式

**3. C++文件操作函数**

* C++主要通过三个类支持文件的输入输出：

ofstream: 写操作（输出）的文件类

ifstream: 读操作（输入）的文件类

fstream: 可同时读写操作的文件类

* 打开文件：

构造函数

open函数

is\_open函数

* 读取流指针：

tellg函数

tellp函数

* 改变流指针：

seekg函数

seekp函数

* 读写文件：

<<和>>

get函数和put函数

read函数和write函数

getline函数

**1. 国家密码管理局**

国家密码管理局全称为国家商用密码管理办公室，简称为国密局，是我国关于密码工作的管理机构。由国密局组织设计并发布的密码算法称为国密算法。

**2. 密钥**

密钥是一种参数，它是在明文转换为密文或将密文转换为明文的算法中输入的参数。

**3. 对称密码算法**

加密密钥和解密密钥相同的加密算法。

常见的对称算法有DES、3DES、AES、SM1、SM4等。

**4. 非对称密码算法**

加密密钥和解密密钥不同的加密算法，非对称算法需要两个密钥：公钥（public key）和私钥（private key），公钥和私钥是一对，如果用公钥加密，只有用对应的私钥才能解密；如果用私钥加密，那么只有用对应的公钥才能解密，通常我们把私钥加密、公钥解密的过程称为签名和验签。

常见的非对称算法有RSA、ECC、SM2等。

**5. 分组加密** 对称密码算法只能把特定长度的明文消息加密成相同长度的密文，如果要加密的消息其长度超过这个特定长度，就要按这个长度把消息分成若干个分组，并分别对每个分组进行加密，如果最后一个分组长度不够特定长度，则需要用字符补足长度。

**6. ECB方式和CBC方式**

电子密码本（ECB）方式使用同一个密钥简单地将每个明文分组一个接一个地进行加密；在密码分组链接（CBC）方式中，每个明文分组在加密前先与前一个分组的密文进行“异或”运算，从而增加了复杂程度。

**7. 哈希算法**

又称为散列算法，国密局称为杂凑算法，哈希算法将任意长度的二进制值映射为较短的固定长度的二进制值，这个小的二进制值称为哈希值。哈希值是一段数据唯一且极其紧凑的数值表示形式。如果散列一段明文而且哪怕只更改该段落的一个字母，随后的哈希都将产生不同的值。

常见的哈希算法有MD5、SHA-1、SHA-2（SHA-224, SHA-256, SHA-384, SHA-512）、SM3等算法。

**1. 第三方库libxml2**

官方网站：<http://www.xmlsoft.org/>

**2. 下载安装包**

从官方网站下载libxml2的tar包，由于编译libxml2的时候会用到libiconv和zlib两个第三方库，所以还需要下载libiconv和zlib。

libiconv官方网站：<https://www.gnu.org/software/libiconv/>

zlib官方网站：<http://zlib.net/>

**界面布局**

主界面上至少包括菜单、静态文本控件（显示动态口令）、时间进度条控件。

**安全要求**

* 用户输入密码时应显示为\*号；
* 保存密码时应保存哈希值，不能直接保存明文；
* 修改密码时应要求用户输入两遍；
* 弱密码检测。
* 需要实现socket通讯的客户端部分
* 需要实现对配置文件的操作

**什么是守护进程？**

守护进程是脱离于终端并且在后台运行的进程，周期性地执行某种任务或者等待处理某些特定的事件

**1. 什么是多线程？**

多线程（Multi-Threading），是指从软件或者硬件上实现多个线程并发执行的技术，可以提升整体处理性能。

**2. 线程函数**

**创建线程**

int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr, void\*（\*start\_routine)(void\*), void \*arg);

参数：

* 第一个参数为指向线程标识符的指针。
* 第二个参数用来设置线程属性。
* 第三个参数是线程运行函数的起始地址。
* 最后一个参数是运行函数的参数。

返回值：

* 成功则返回0
* 失败返回出错误编号

**线程退出**

void pthread\_exit(void \*retval);

参数：

* 线程的返回值，可以用pthread\_joid函数获取。

**等待线程结束**

int pthread\_join(pthread\_t thread, void \*\*retval);

参数：

* 第一个参数为被等待的线程标识符。
* 第二个参数为一个用户定义的指针，它可以用来存储被等待线程的返回值。

返回值：

* 成功则返回0
* 失败返回出错误编号

**线程分离**

int pthread\_detach(pthread\_t thread);

参数：

* 第一个参数为被等待的线程标识符。

返回值：

* 成功则返回0
* 失败返回出错误编号

在任何一个时间点上，线程是可结合的（joinable）或者是分离的（detached）。一个可结合的线程能够被其他线程收回其资源和杀死。在被其他线程回收之前，它的存储器资源（例如栈）是不释放的。相反，一个分离的线程是不能被其他线程回收或杀死的，它的存储器资源在它终止时由系统自动释放。

默认情况下，线程被创建成可结合的。为了避免存储器泄漏，每个可结合线程都应该要么被显示地回收，即调用pthread\_join；要么通过调用pthread\_detach函数被分离。

**3. 什么是互斥锁**

互斥锁（Mutual exclusion，缩写 Mutex）是一种用于多线程编程中，防止两条线程同时对同一公共资源（比如全局变量）进行读写的机制。

**4. 互斥锁函数**

* 互斥锁定义类型 pthread\_mutex\_t
* 创建互斥锁 pthread\_mutex\_init (pthread\_mutex\_t *mutex, const pthread\_mutexattr\_t* attr);
* 销毁互斥锁 pthread\_mutex\_destroy(pthread\_mutex\_t \*mutex);
* 加锁 pthread\_mutex\_lock(pthread\_mutex\_t \*mutex);
* 解锁 pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);

**5. 什么是信号量**

信号量（Semaphore）的使用主要是用来保护共享资源，使得资源在一个时刻只有一个线程所拥有。

**6. 信号量函数**

* 信息号定义类型 sem\_t
* 创建信号量 sem\_init(sem\_t \*sem, int pshared, unsigned int value);

参数pshared为0时，信号量在线程间共享，非0时，信号量在进程间共享。

* 销毁信号量 sem\_destroy(sem\_t \*sem);
* 申请信号量 sem\_wait(sem\_t \*sem);
* 申请信号量 sem\_trywait(sem\_t \*sem);
* 释放信号量 sem\_post(sem\_t \*sem);

sem\_wait：阻塞模式，若sem>0，那么它减1并立即返回；若sem==0，则睡眠直到>0，此时立即减1，然后返回。 sem\_trywait：非阻塞模式，若sem>0，那么它减1并立即返回；若sem==0，不是睡眠，而是返回一个错误EAGAIN。

**1. 什么是socket？**

套接字（socket）用于描述IP地址和端口，是一个通信链的句柄，可以用来实现不同计算机之间的通信。

**2. 什么是TCP？**

TCP（Transmission Control Protocol 传输控制协议）是一种面向连接的、可靠的、基于字节流的传输层通信协议，在简化的计算机网络OSI模型中，它完成第四层传输层所指定的功能，用户数据报协议（UDP）是同一层内另一个重要的传输协议。

**3. socket函数**

**创建socket描述符**

int socket(int domain, int type, int protocol);

参数：

* 第一个参数是协议域，又称为协议族（family）。常用的协议族有，AF\_INET、AF\_INET6、AF\_LOCAL（或称AF\_UNIX，Unix域socket）、AF\_ROUTE等等。
* 第二个参数指定socket类型。常用的socket类型有，SOCK\_STREAM、SOCK\_DGRAM、SOCK\_RAW、SOCK\_PACKET、SOCK\_SEQPACKET等等。
* 第三个参数指定协议。常用的协议有，IPPROTO\_TCP、IPPTOTO\_UDP、IPPROTO\_SCTP、IPPROTO\_TIPC等，它们分别对应TCP传输协议、UDP传输协议、STCP传输协议、TIPC传输协议。

返回值：

* 成功返回socket描述符
* 失败返回-1

**绑定特定地址给socket描述符**

int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

参数：

* 第一个参数是socket函数创建的socket描述符。
* 第二个参数是指向要绑定给sockfd的地址结构的指针。
* 第三个参数是地址结构的长度。

返回值：

* 成功返回0
* 失败返回-1

**监听socket**

int listen(int sockfd, int backlog);

参数：

* 第一个参数是要监听的socket描述符。
* 第二个参数是相应socket可以排队的最大连接个数。

返回值：

* 成功返回0
* 失败返回-1

**客户端发出连接请求**

int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

参数：

* 第一个参数是客户端socket函数创建的socket描述符。
* 第二个参数是指向服务器地址结构的指针。
* 第三个参数是地址结构的长度。

返回值：

* 成功返回0
* 失败返回-1

**服务器接受连接请求**

int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);

参数：

* 第一个参数是监听的socket描述符。
* 第二个参数是返回的指向发出连接请求的客户端的地址结构的指针。
* 第三个参数是返回的客户端地址结构的指针。

返回值：

* 成功返回实际与客户端发生连接的socket描述符
* 失败返回-1

Accept缺省是阻塞模式，直到接收到客户端的连接请求才会返回，可以使用fcntl函数修改为非阻塞模式。

**接收**

ssize\_t recv(int sockfd, void \*buf, size\_t len, int flags);

参数：

* 第一个参数是连接的socket描述符。
* 第二个参数是接收数据的缓冲区地址。
* 第三个参数是接收缓冲区的字节数。
* 第四个参数一般置0。

返回值：

* 成功返回实际接收数据的长度
* 失败返回-1
* 如果对方关闭socket连接返回0。

**发送**

ssize\_t send(int sockfd, const void \*buf, size\_t len, int flags);

参数：

* 第一个参数是连接的socket描述符。
* 第二个参数是发送数据的缓冲区地址。
* 第三个参数是要发送数据的字节数。
* 第四个参数一般置0。

返回值：

* 成功返回实际接收数据的长度
* 失败返回-1

**关闭socket连接**

int close(int fd);//Linux

int closesocket(int fd);//Windows

在Windows系统上的使用socket函数时，首先必须调用WSAStartup 函数完成对winsock服务的初始化。

WSADATA wsaData;

WSAStartup(MAKEWORD(2,2),&wsaData);

结束前调用WSACleanup函数释放占用的资源。

WSACleanup();

**1. 通讯报文的定义**

报文是网络传输的基本单位，通常由报文头和报文体组成，报文头通常包含报文类型、报文版本、报文长度等信息，报文体由交易码和交易要处理的信息组成。

**2. 报文格式**

* 自定义格式：发送方和接收方约定的格式化字符串；
* TLV格式：每个报文域由Tag、Length、Value组成；
* XML格式：符合XML规范的报文
* http格式：符合http规范的报文

应实现4种业务的处理：

* 创建令牌
* 注销令牌
* 验证动态口令
* 同步动态口令

**1. 使用日志线程保证日志文件的完整性**

如果在每个线程中写日志文件，日志文件也就成为多线程的竞争资源，当高并发时，会导致性能下降或者丢失日志信息。为了解决这个问题，使用一个单独的线程，用于操作日志文件，设置一个日志缓冲区，当处理交易的工作线程记日志时，只要把日志信息写入日志缓冲区即可，日志线程一旦发现日志缓冲区内有新信息，就把新信息写入日志文件。

**2. 使用日志MAC保证日志文件的正确性**

日志信息中记录了交易现场的信息，在查找问题时会提供重要的线索，为了防止人为篡改日志文件，可以针对每条日志信息产生一个MAC信息（从完整的MAC中截取一段即可），用于对日后对日志信息进行校验，保证日志的正确性。