

私有协议文件管理服务器

—— 项目需求文档(类似百度网盘)

一期功能

功能概述

1. 服务器端实现命令解析。具体操作步骤如下：

- 首先启动服务器端
- 然后启动客户端
- 通过客户端可以输入以下命令，与服务器进行交互：
 - (1) cd 进入服务器对应目录
 - (2) ls 列出服务器上相应的目录和文件
 - (3) pwd 显示目前所在路径
 - (4) puts filename 将本地文件上传至服务器
 - (5) gets filename 文件名 下载服务器文件到本地
 - (6) rm filename 删除服务器上的某文件
 - (7) mkdir dirname 创建文件夹
 - (8) ... 可自行扩展其他命令
 - (9) 非法命令 不响应

2. 项目启动方法

```
$ ./NetDiskServer ./conf/server.conf
```

3. 客户端启动方法

```
$ ./client ip port
```

二期功能

1. 密码验证

二期功能暂时不用写注册功能，我们直接利用 Linux 的用户系统，实现登录功能。所以必须要了解 Linux 的密码验证功能。

对于 Linux 的登陆验证，我想大家可能都知道，是将输入的密码进行相应编码加密后与 /etc/shadow 文件中对应的密码部分进行比较，如果相同则验证通过，如果不同则表明密码错

误，但是问题是我们要如何将用户输入的密码加密然后进行比较呢？

为了提高安全性，Linux 引入了 salt，所谓 salt 即为一个随机数，引入的时候为一个 12bit 的数值，当用户设置密码时，会随机生成一个 salt 与用户的密码一起加密，得到一个加密的字符串(salt 以明文形式包含在该字符中)，存储到密码文件中。crypt 函数可以将用户输入的密码和 salt 一起适应某种算法进行加密，该加密后的字符串就是我们需要的与密码文件中密码对比的加密字符串。

crypt 为支持不同的方式将 salt 格式化为

`idsalt$encode`

其中 id 代表不同的算法

- 1 | MD5
- 2a | Blowfish (not in mainline glibc; added in some
| Linux distributions)
- 5 | SHA-256 (since glibc 2.7)
- 6 | SHA-512 (since glibc 2.7) **当前我们 Linux 采用的加密算法**

这样我们就可以利用 crypt 函数将用户输入的字符加密，然后与密码文件中的密码进行对比了，有一个函数 getspnam 可以根据用户名从/etc/shadow 中得到密码，函数返回的数据结构为

```
struct spwd {
    char *sp_namp;    /* Login name */
    char *sp_pwdp;    /* Encrypted password */
    long  sp_lstchg;   /* Date of last change (measured
                        in days since 1970-01-01 00:00:00 +0000 (UTC)) */
    long  sp_min;      /* Min # of days between changes */
    long  sp_max;      /* Max # of days between changes */
    long  sp_warn;     /* # of days before password expires
                        to warn user to change it */
    long  sp_inact;    /* # of days after password expires
                        until account is disabled */
    long  sp_expire;   /* Date when account expires (measured
                        in days since 1970-01-01 00:00:00 +0000 (UTC)) */
    unsigned long sp_flag; /* Reserved */
};
```

我们现在创建一个 test 用户，并将密码设置为 1234 来做个测试看一下

这是创建之后从/etc/shadow 中取出来的密码部分

`6qOrvsN41$gGlv8z7P1sAKuyaTAY03AvCn9/Z6Ygc4DJ9Uwe0RVzNAI6TQsTfsdihPnIh3lSZV2C02HjW.9bvJNdep3k.ER.`

可以看到这里的 salt 为 \$6\$qOrvsN41

最后写个程序做下验证

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <shadow.h>
#include <errno.h>

void help(void) {
    printf("用户密码验证程序\n 第一个参数为用户名!\n");
    exit(-1);
}

void error_quit(char *msg) {
    perror(msg);
    exit(-2);
}

void get_salt(char *salt, char *passwd) {
    int i, j;
    //取出 salt, i 记录密码字符下标, j 记录$出现次数
    for(i=0, j=0; passwd[i] && j != 3; ++i) {
        if(passwd[i] == '$') ++j;
    }
    strncpy(salt, passwd, i-1);
}

int main(int argc, char **argv) {
    struct spwd *sp;
    char *passwd;
    char salt[512] = {0};
    if(argc != 2) help();
    passwd = getpass("请输入密码:"); //输入用户名密码
    //得到用户名以及密码, 命令行参数的第一个参数为用户名
    if((sp = getspnam(argv[1])) == NULL)
        error_quit("获取用户名和密码");
    //得到 salt, 用得到的密码作参数
    get_salt(salt, sp->sp_pwdp);
    //进行密码验证
    if(strcmp(sp->sp_pwdp, crypt(passwd, salt)) == 0)
        printf("验证通过!\n");
    else
        printf("验证失败!\n");
    return 0;
}
```

注意:

1. gcc 编译该程序时需要加入 -lcrypt
2. 进行登录密码验证后, 客户端只能看到自己的文件, 不能看到其他用户的文件
3. 参考博客: <https://www.cnblogs.com/xu2shuang97664/p/5093183.html>

2.日志记录

- (1) 服务器日志记录客户端请求信息及客户端连接时间。
- (2) 服务器日志记录客户端操作记录及操作时间。
- (3)

3.断点续传

断点续传: 该功能支持从文件上次中断的地方开始传送数据, 而并非是从文件开头传送。

举例: 进行 gets hello.avi 时候, 首先判断本地是否存在 hello.avi

- 如果存在, 则通过 stat 函数获取本地文件 hello.avi 的大小, 然后传递给服务器的是 gets hello.avi 1000
- 如果不存在, 则直接进行下载即可。

4.大文件传输优化

mmap 将大文件映射入内存, 进行网络传递。当发现文件大于 100M, 就使用 mmap 方式映射该文件, 然后再传输。

三期功能

1. 用户注册

用户表设计

通过关系型数据库 MySQL 存储用户信息, 包括用户名, salt 值(采用随机字符串生成), 密码(密文形式存储)。

字段名	字段类型	注释
id	int(自增)	用户 id, 主键
username	varchar	用户名
salt	char	盐值
cryptpasswd	varchar	加密密码
pwd	varchar	当前工作目录

用户表举例：

id	username	salt	cryptpasswd	pwd
1	niquiu	xx	fafdfafd	/niquiu/
2	yiming	yy	ljlkjkljfafa	/yiming/

如何生成随机字符串呢？请参考下面的 2 组代码：

代码一：

#define STR_LEN 10 //定义随机输出的字符串长度。

```
char *GenerateStr() {
    char str[STR_LEN + 1] = {0};
    int i, flag;
    srand(time(NULL));
    for(i = 0; i < STR_LEN; i++) {
        flag = rand()%3;
        switch(flag) {
            case 0: str[i] = rand()%26 + 'a'; break;
            case 1: str[i] = rand()%26 + 'A'; break;
            case 2: str[i] = rand()%10 + '0'; break;
        }
    }
    printf("%s\n", str); //输出生成的随机数。
    return str;
}
```

代码二：

```
char* GenRandomString(int length) {
    int flag, i;
    char* string;
    srand((unsigned) time(NULL));
    if ((string = (char*) malloc(length)) == NULL) {
        printf("malloc failed!flag:14\n");
        return NULL;
    }
    for (i = 0; i < length+1; i++) {
        flag = rand() % 3;
        switch (flag) {
            case 0: string[i] = 'A' + rand() % 26; break;
            case 1: string[i] = 'a' + rand() % 26; break;
            case 2: string[i] = '0' + rand() % 10; break;
            default: string[i] = 'x'; break;
        }
    }
    string[length] = '\0';
}
```

```

    return string;
}

```

代码二的生成方法用到了 `malloc` 内存之后没有释放，存在内存泄漏的风险，不建议使用下面的方法，想想如何改进呢？（想想 MySQL 的资源销毁）

2. 用户登录

在二期功能中，我们已经了解了 Linux 系统的密码验证功能。接下来，我们将其应用在自定义的用户登录的功能中，服务器先传输 salt 值给客户端，客户端 crypt 加密后，传输密码明文给服务器，服务器进行匹配后，判断是否成功。

3. 虚拟文件表

前面的二期功能实现中，客户端登录后，直接获取服务器的目录结构，但只能针对单个用户有效，如果是多用户后，就不能这样去实现了。因为当有多用户时，每个用户只能看到自己上传的文件，而看不到其他用户的文件，这说明每个用户需要拥有自己独立的目录结构。那目录结构该怎么设计呢？用文件存储，还是用数据库存储呢？

这里我们采用数据库方案进行存储。

服务器：由数据库记录每一个用户的目录结构和文件，但每个文件的内容本身存储在服务器磁盘上，并以 MD5 码进行命名。所有用户上传的文件都存在服务器上的某一个目录里，如 `/netdisk/`

客户端：在客户端，用户登录服务器后，每个用户只能看到自己的文件，不能看到其他人的文件。然后可以执行各种操作命令，包括查看文件信息，上传文件，下载文件，删除文件，创建文件夹等，即一期功能。

那文件的数据库表该如何设计呢？

这里我们直接给出方案。

虚拟文件表

字段名	字段类型	注释
id	int	文件名主键
parent_id	int	所在上一层目录(0 表示无父目录)
filename	varchar	文件名
owner_id	int	文件属与某一个用户的 id
md5	char	文件对应的 MD5 码
filesize	int	文件大小
type	int	文件类型 (d 目录, f 普通文件)

虚拟文件表举例：

id	parent_id	filename	owner_id	md5	filesize	type
1	0	dir1	1	0	0	d
2	1	file1	1	8d433f	10K	f
3	1	file2	2	1s43df	200M	f
4	0	file2	2	8d433f	10K	f

4. 文件秒传

如果有两个用户上传的文件都是同一个文件，那需要在服务器端保留两份文件吗？或者说如何实现文件的秒传功能呢？只需要在服务器上查找是否有文件与待上传的文件有相同的MD5即可。如果相同，则只需要在数据库中添加一条记录，而不需要真的上传文件。

文件 MD5 码的生成方式

通过 C 接口，得到每一个文件的 MD5，可以参考博客的例子

https://blog.csdn.net/a_ran/article/details/40897159

4.1 在线安装

```
$ sudo apt-get install libssl-dev
```

使用下面的测试程序验证是否安装成功

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<openssl/md5.h>
int main(int argc, char* argv[])
{
    char *buf = "helloworld";
    char *md;
    int i;
    md = MD5(buf,strlen(buf),NULL);
    printf("%s\n",md);
    for (i = 0; i < strlen(md); i++)
        printf("%x", md[i]);
    printf("\n");
    return 0;
}
```

注意：编译程序时请加 `-lcrypto` 或者 `-lssl`

3.2.2 离线安装

如果在线安装可以使用 MD5，则无需阅读下面的离线安装

- 1) 下载 openssl-1.1.1-pre6.tar.gz，

- 2) 解压以后执行 ./config 生成 Makefile,
- 3) 执行 make,然后再执行 make install,
- 4) 把两个动态库 libcrypto.so 和 libssl.so 全部复制到/usr/lib 下 (包括两个软链接一并复制)

四期功能

1. 实现长短命令分离

为什么要实现长短命令分离呢?

我们考虑一下, 当使用 gets/puts 命令进行下载与上传任务时, 由于需要进行大文件的传输, 那在传输过程中, 无法执行其他操作。如果此时希望能执行其他的命令, 如查看其他文件, 或者删除某个文件, 或者切换到其他文件夹, 则只能等待下载或上传操作执行完毕后, 才能进行其他命令。

长命令: 不能立即执行完毕的命令, 需要一段时间来等待命令执行完毕。如上传下载等。

短命令: 立即执行完毕的命令。如 cd, pwd, mkdir 等。

由此就带来了新的需求, 需要对长短命令的操作进行分离。可以考虑以下**方案**:

- (1) **服务器**: 短命令的执行由主线程负责, 长命令的执行由子线程负责;
- (2) **客户端**: 短命令的执行由主线程负责, 长命令的执行由子线程负责。

2. 用户身份验证的令牌 ---- Token

在实现长短命令分离的过程中, 会面临一个身份验证的问题。客户端用子线程处理长命令, 还能用原来的连接进行数据的接收吗? 答案是否定的, 因为会影响短命令的操作, 此时客户端需要与服务器再起一个新连接进行操作。**那再起连接之后, 客户端还需要重新进行登录吗? 从逻辑上来说, 很显然是不需要的, 因为都是同一个客户端在进行操作。**

那服务器怎么区分再起的连接就是刚刚已经登录过的客户端呢, 即如何对用户身份进行登录验证呢? 要解决这个问题, 目前技术实践上是有多方案的, 比如 Session 和 Token. 在我们的方案中, 推荐使用 **Token**, 这也是目前最通用的一种方式。通过 **Token** 实现服务器的认证, 之后再进行大文件的传输工作。

这里我们稍做扩展学习, 聊聊 Session 和 Token 的区别。(面试专用)

Session

Session 从字面上讲, 就是会话。这个就类似于你和一个人交谈, 你怎么知道当前和你交谈的是张三而不是李四呢? 对方肯定有某种特征(长相等)表明他就是张三。Session 也是类似的道理, 服务器要知道当前发请求给自己的是谁。为了做这种区分, 服务器就要给每个客

户端分配不同的“身份标识”，然后客户端每次向服务器发请求的时候，都带上这个“身份标识”，服务器就知道这个请求来自于谁了。

在 Token 出现之前，服务器程序都是通过**在服务端存储的登录信息**来辨别请求的。这种方式一般都是通过存储 Session 来完成。服务器采用 Session 把用户的信息临时保存在了服务器上，用户离开后 Session 会被销毁。**由于服务器需要存储 Session 信息，所以是有状态的。**

随着 Web，应用程序以及移动端的兴起，这种验证的方式逐渐暴露出了问题。尤其是在可扩展性方面。问题如下：

- (1) **内存开销**：每次认证用户发起请求时，服务器需要去创建一个记录来存储信息。当越来越多的用户发请求时，内存的开销也会不断增加。
- (2) **可扩展性**：在服务端的内存中使用 Session 存储登录信息，伴随而来的是可扩展性问题。
- (3) **CORS(跨域资源共享)**：当我们需要让数据跨多台移动设备上使用时，跨域资源的共享会是一个让人头疼的问题。
- (4) **CSRF(跨站请求伪造)**：用户在访问银行网站时，他们很容易受到跨站请求伪造的攻击，并且能够被利用其访问其他的网站。

Token

所谓 Token，就是服务端生成的一串加密字符串，以此作为客户端进行请求的一个“令牌”。当用户第一次使用账号密码成功进行登录后，服务器便生成一个 Token，并将此返回给客户端，该 Token 是有时间限制的，超时便是无效的。若成功登录，以后客户端只需在有效期内带上该 Token 来请求数据即可，无需再次带上用户名和密码。

基于 Token 的身份验证是无状态的，我们不将用户信息存在服务器或 Session 中。这种概念解决了在服务端存储信息时的许多问题。NoSession 意味着你的程序可以根据需要去增减机器，从而方便地实现扩展的需求，而不用担心用户是否登录。

大家熟悉的 HTTP 协议就是无状态的。**无状态**是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

基于 Token 的身份验证过程如下：

- (1) 客户端通过用户名和密码发送登录请求
- (2) 服务器收到请求，去验证用户名和密码
- (3) 服务器验证成功后，返回一个签名的 Token 给客户端
- (4) 客户端存储 Token
- (5) 当客户端再次向服务器请求资源时，携带 Token
- (6) 服务器验证 Token，如果验证成功，则返回请求的数据

Token 几个问题

在这样的流程下，我们需要考虑以下几个问题：

- (1) 服务器端如何根据 Token 获取用户的信息?
- (2) 服务器端如何确保识别伪造的 Token?
- (3) 如何应对冒充的情况呢?

用户匹配

对于问题 1，服务端在生成 token 时，加入少量的用户信息，比如用户的 id。服务端接收到 token 之后，可以解析出这些数据，从而将 token 和用户关联了起来。

防伪造

对于问题 2，一般情况下，建议放入 token 的数据是不敏感的数据，这样只要服务端使用私钥对数据生成签名，然后和数据拼接起来，作为 token 的一部分即可，比如 JWT。

防冒充

方式有多种：

- 加干扰码

服务端在生成 Token 时，使用了客户端的 UA(User Agent - 用户代理，Web 开发中指浏览器) 作为干扰码对数据加密，客户端进行请求时，会同时传入 Token、UA，服务端使用 UA 对 Token 解密，从而验证用户的身份。

如果只是把 Token 拷贝到另一个客户端使用，不同的 UA 会导致在服务端解析 Token 失败，从而实现了一定程度的防冒充。但是攻击者如果猜到服务端使用 UA 作为加密钥，他可以修改自己的 UA。

- 设置有效期

给 Token 加上有效期，即使被冒充也只是在一定的时间段内有效。这不是完美的防御措施，只是尽量减少损失。服务端在生成 Token 时，加入有效期。每次服务端接收到请求，解析 Token 之后，判断是否已过期，如果过期就拒绝服务。

Token 刷新策略

如果 token 过期了，客户端应该对 token 续期或者重新生成 token。这取决于 token 的过期机制：

- (1) 服务器缓存 token 及对应的过期时间。这个时候就可以采用续期的方式，服务器更新过期时间，token 再次有效。
- (2) token 中含有过期时间。这个时候需要重新生成 token。

在 token 续期或者重新生成 token 的时候，需要额外加入数据来验证身份。因为 token 已经过期了，即 token 已经不能用来验证用户的身份了。此时，可以采用的方式一是请求用户重新输入账号和密码，但是用户体验稍差。

另一种方式是使用摘要。服务端生成 Token，同时生成 Token 的摘要，然后一起返回给客户端。客户端保存摘要，一般请求只需要用到 Token，在刷新 Token 时，才需要用到摘要。服务端验证摘要，来验证用户的身份。因为摘要不会频繁的在客户端和服务端之间传输，所以

被截取的概率较小。

JWT

JWT 的全称是 JSON Web Token，它是一个非常轻巧的规范。这个规范允许我们使用 JWT 在用户和服务器之间传递安全可靠的信息，具体的用法可以参考以下文章：

《JSON Web Token - 在 Web 应用间安全地传递信息》

<http://blog.leapoahead.com/2015/09/06/understanding-jwt/>

实现 JWT 的开源库有很多，我们可以参考 JWT 的官网 <https://jwt.io/#libraries> 查看 JWT 有哪些开源库支持。由于我们是用 C 语言进行开发，所以可以选择一个 C 库完成我们的需求。在此，我们推荐使用下面这个开源库 <https://github.com/GlitchedPolygons/l8w8jwt.git>

l8w8jwt 库的安装和使用

以下攻略参考该项目在 github 上的文档

(1) 使用 git 工具下载 l8w8jwt

\$ git clone --recursive <https://github.com/GlitchedPolygons/l8w8jwt.git>

注意：--recursive 一定要有，需要递归下载其子项目，涉及到的子项目在 lib 文件夹中，否则下载会不全，后续在编译时会出现找不到库的 bug

```
lwh@ubuntu:lib$ pwd
/home/lwh/documents/l8w8jwt/lib
lwh@ubuntu:lib$ ls -l
total 24
drwxrwxr-x  4 lwh lwh 4096 Mar 16 10:42 acutest
drwxrwxr-x  6 lwh lwh 4096 Mar 16 15:10 checknum
drwxrwxr-x  7 lwh lwh 4096 Mar 16 15:10 chillbuff
drwxrwxr-x  4 lwh lwh 4096 Mar 16 10:42 ed25519
drwxrwxr-x  4 lwh lwh 4096 Mar 16 15:10 jsmn
drwxrwxr-x 14 lwh lwh 4096 Mar 16 10:42 mbedtls
```

递归下载子项目的过程中，如果断开了，则需要再次执行

\$ git clone --recursive <https://github.com/GlitchedPolygons/l8w8jwt.git>

\$ cd l8w8jwt/lib/

\$ rm -rf 空文件夹 #删除 lib 下的空文件夹

\$ cd ../ # 回到 l8w8jwt 项目的根目录

\$ git submodule update --init --recursive

(2) 编译静态库（前提条件是已经安装好了 cmake 工具）

\$ cd l8w8jwt/

\$ mkdir -p build && cd build

\$ cmake -DBUILD_SHARED_LIBS=Off -DL8W8JWT_PACKAGE=On
-DCMAKE_BUILD_TYPE=Release ..

\$ cmake --build . --config Release

这 4 条命令建议直接复制执行，自己敲很容易打错字

这一步执行完，在 `18w8jwt` 项目的根目录下有个 `build` 文件夹，里面我们需要的头文件和库文件，如下图所示，

```
lwh@ubuntu:build$ pwd
/home/lwh/documents/l8w8jwt/build
lwh@ubuntu:build$ ls -l
total 88
drwxrwxr-x 3 lwh lwh 4096 Mar 16 15:11 chillbuff
-rw-rw-r-- 1 lwh lwh 14696 Mar 16 15:11 CMakeCache.txt
drwxrwxr-x 5 lwh lwh 4096 Mar 16 15:12 CMakeFiles
-rw-rw-r-- 1 lwh lwh 1914 Mar 16 15:11 cmake_install.cmake
drwxrwxr-x 4 lwh lwh 4096 Mar 16 15:12 l8w8jwt
-rw-rw-r-- 1 lwh lwh 33702 Mar 16 15:12 l8w8jwt-2.0.0-linux-x86\_64.tar.gz
-rw-rw-r-- 1 lwh lwh 13412 Mar 16 15:11 Makefile
drwxrwxr-x 6 lwh lwh 4096 Mar 16 15:11 mbedtls
```

(3) 我们所需要的头文件和库文件，就放在上图所示 18w8jwt 目录中，

```
lwh@ubuntu:18w8jwt$ pwd
/home/lwh/documents/18w8jwt/build/18w8jwt
lwh@ubuntu:18w8jwt$ tree
.
├── bin
│   └── release
│       └── lib18w8jwt.a
├── include
│   └── 18w8jwt
│       ├── algs.h
│       ├── base64.h
│       ├── claim.h
│       ├── decode.h
│       ├── encode.h
│       ├── retcodes.h
│       ├── util.h
│       └── version.h
└── NOTICE
```

对于静态库的使用，有两种方式，我们可以直接把头文件和库文件放在自己的项目中，也可以直接放在系统路径下。比如头文件放在/usr/local/include 下，库文件放在/usr/local/lib/目录下

```
$sudo cp -r libl8w8jwt.a /usr/local/lib/
```

```
$sudo cp -r include/18w8jwt/ /usr/local/include/
```

当然，如果考虑项目的可移植性，建议直接放在自己的项目中。

(4) 测试。采用 github 项目文档的例子来进行，一个例子是编码并生成一个 Token，另一个是解码并验证 Token 的例子。

Encoding and signing a token

```
#include "18w8jwt/encode.h"

int main(void)
{
    char* jwt;
    size_t jwt_length;

    struct 18w8jwt_encoding_params params;
    18w8jwt_encoding_params_init(&params);
```

Decoding and verifying a token

```
#include "l8w8jwt/decode.h"  
  
static const char KEY[] = "YOUR sUpEr S3krEt 1337 HMAC kI";  
static const char JWT[] = "eyJhbGciOiJIUzUxMiIsInR5cCI6IWIzbnQ9ImFkbGkiLCJ0eXAiOiJKV1QiLCJkaXI6ImNpdCkiLCJpcyI6ImR5bmFudGVkaSIsImVudCI6ImNpdCk9e30=  
  
int main(void)  
{  
    struct l8w8jwt_decoding_params params;  
    l8w8jwt_decoding_params_init(&params);
```

源码直接复制过去，分别放入 encode.c 和 decode.c 文件中。编译时需要带上库 libl8w8jwt，这

一个还不够，因为 l8w8jwt 库还用到了 mbedtls 库，在(2)中如图所示的目录下有一个文件夹 mbedtls，其中有 3 个库文件


```
lwh@ubuntu:library$ pwd
/home/lwh/documents/l8w8jwt/build/mbedtls/library
lwh@ubuntu:library$ ls -l
total 1292
drwxrwxr-x 6 lwh lwh 4096 Mar 16 15:11 CMakeFiles
-rw-rw-r-- 1 lwh lwh 2218 Mar 16 15:11 cmake_install.cmake
-rw-rw-r-- 1 lwh lwh 729904 Mar 16 15:12 libmbedcrypto.a
-rw-rw-r-- 1 lwh lwh 319730 Mar 16 15:12 libmbedtls.a
-rw-rw-r-- 1 lwh lwh 161010 Mar 16 15:12 libmbedx509.a
-rw-rw-r-- 1 lwh lwh 92532 Mar 16 15:11 Makefile
```


```
$ gcc -o encode encode.c -ll8w8jwt -lmbedcrypto
```

```
$ gcc -o decode decode.c -ll8w8jwt -lmbedcrypto -lmbedx509
```

如果编译成功就能得到两个可执行文件 encode 和 decode。如果能正确执行，则 l8w8jwt 库测试成功。

(5) 为了大家方便，这里提供一个已经编译好的，大家可以直接复制过去，就可以用了，放在项目资料中。

 JWT开源库

 百度网盘项目需求.doc

Token 生成策略

回到我们上面遇到的问题，那服务器怎么区分再起的连接就是刚刚已经登录过的客户端呢，即如何对用户身份进行登录验证呢？

使用 Token，可以保护用户的用户名及密码不被多次提交，以防密码泄露。其操作流程如下：

- (1) 未登录时，用户提交“用户名”和“密码”，进行登录操作
- (2) 登录成功后，服务器端生成一个 user_token，发送给客户端。user_token 表设计如下：

字段名	字段类型	注释
id	int	文件名主键
user_id	int	用户 id
token	varchar	文件名
expire	date	文件名

user_token 表记录举例：

id	user_id	token	expire
1	2	yming12514	2021/3/16
2	1	niqiu12345	2021/3/17

- (3) 客户端存储 user_token，后续客户端每次请求时，如果需要用户登录才能访问，则需要把 user_id 与 user_token 一起发送给服务器。
- (4) 服务器接收到这 2 个参数后，需要进行如下操作：
 - a. 删除过期的 user_token 表记录
 - b. 根据 user_id、user_token 获取记录，如果记录不存在，直接返回错误；如果记录存

在，则进行下一步

- c. 更新 `user_token` 的过期时间

3. 超时断开链接

针对已经连接上的客户端，如果超过 30 秒未发送任何请求，那必须关闭其描述符。

可以采用的算法是**环形队列法（轮盘法）**，具体请参考如下文章

《10w 定时任务，如何高效触发超时》

https://mp.weixin.qq.com/s?__biz=MjM5ODYxMDA5OQ==&mid=2651959957&idx=1&sn=a82bb7e8203b20b2a0cb5fc95b7936a5&chksm=bd2d07498a5a8e5f9f8e7b5aea5bd8585a0ee4bf470956e7fd0a2b36d132eb46553265f4eaf&mpshare=1&scene=23&srcid=0718Qlp4AVKnZq1E1f144pE6#rd

五期功能

多点下载

多点下载是指客户端通过多个 `sfd` 连接不同的服务器，将一份大文件拆分为几段，不同段从不同的服务器进行下载。

分为三级难度或者说三套方案：

- **方案 A**：固定从某几台服务器下载，而且文件多点下载一定一次性下载完毕。
- **方案 B**：从某台 IP 服务器上得到具有数据源的服务器地址，然后再从对应的地址进行下载，而且文件一次性下载完毕
- **方案 C**：如果多点下载过程中，客户端断开，那么下次重新下载时，需要再次进行多点下载，所以客户端本地需要使用文件，或者数据库存储多点下载，每个位置下载到什么地方

想研究实际 FTP 协议，详见

<http://blog.csdn.net/yxyhack/article/details/1826256>

P2P 协议

<http://www.52im.net/thread-50-1-1.html>

面试相关

面试时询问和 `ftp` 有什么区别，个性化功能回答

- 1、多级权限管理

- 2、某一级主管可以分享给自己的下属小组查看文件
- 3、文件分为可下载和在线阅读，读后即焚等