### 相关函数。

#### HANDLE CreateIoCompletionPort(HANDLE, HANDLE, ULONG\_PTR, DWORD) 关联一个已打开的文件实例和新建的或已存在的I/0完成端口，或者创建一个未关联任何文件的I/O完成端口。

第一个参数:关联的文件句柄。

第二个参数:完成端口的句柄， 若NULL,则新建一个IOCP。

第三个参数: 一个32位内存指针，可以将数据存入带入线程或其他处理中,只能在这时候做，以后不能动态改变。

第四个参数:系统允许的最大线程数。线程数= CPU数\*2+2,一般该参数给0,系统根据CPU个数自动确定。

返回值:为NULL则创建IOCP端口失败。

创建一个IOCP端口:

HANDLE CompletionPort = CreateIoCompletionPort(INVALID\_HANDLE\_VALUE, NULL, 0, 0);

if (!CompletionPort)

{ return FALSE;}

#### BOOL GetQueuedCompletionStatus(HANDLE, LPDWORD, PULONG\_PTR, LPOVERLAPPED \*, DWORD) //阻塞函数， 除非发生读取或客户端套接字关闭事件。

第一个参数(输入):完成端口的句柄

第二个参数(传出):传送的实际数据量

第三个参数(传出):与完成端口绑定(关联)的数据

第四个参数(传出):重叠I/O的数据缓冲区, 是OVERLAPPED结构。

第五个参数(输入):等待消息发送到端口的最大时间

if (!GetQueuedCompletionStatus(CompletionPort, &BytesTransferred, (LPDWORD)&socket, (LPOVERLAPPED\*)&PerIoData, INFINITE))

{return FALSE;}

自定义结构指针变量名= (自定义包含OVERLAPPED结构指针)CONTAINING\_RECORD (LPOVERLAPPED, 自定义包含OVERLAPPED结构类型, OVERLAPPED结构的变量名);

#### int WSAAsyncSelect(SOCKET, HWND, unsigned int, long); 将相关套接字设置非阻塞模式并发送相应的消息到窗口。WIN32函数。

#define WM\_SOCKET WM\_USER+0

WSAAsyncSelect(sClient, hwnd, WM\_SOCKET, FD\_READ | FD\_CLOSE); //在发生读取事件之前调用该函数,发生读取事件后向hwnd窗口句柄发送WM\_SOCKET自定义消息。

WSAAsyncSelect(sListen, hwnd, WM\_SOCKET, FD\_ACCEPT); //在发生等待客户端连接之前调用,同上。

hwnd窗口获取该消息方法: LOWORD(lParam)得到FD\_READ或FD\_ACCEPT等。

### 相关例子。

class CFTP

{

public:

CFTPLIB();

~CFTPLIB();

int ftp\_connect(const char \*ip); //连接ftp服务器

int ftp\_login(char \*user, char \*pass); //登陆ftp服务器

int ftp\_pwd(char \*buff); //显示当前目录

int ftp\_cd(char\* dir); //更改目录

int ftp\_cdup(); //返回上层目录

int ftp\_mkdir(char\* dir); //创建目录

int ftp\_rmdir(char\* dir); //删除目录

int ftp\_setpasv(); //数据传输模式

int ftp\_upload(char\* localfile,char\* remotepath,char\* remotefilename); //上传文件

int ftp\_download(char\* localfile,char\* remotefile); //下载文件

int ftp\_quit(); //退出登录

private:

int m\_sockctrl //控制连接socket

int m\_sockdata;//数据连接socket

char m\_cmd[256];//存放指令

char m\_resp[256];//存放返回语句

char m\_ip[64];//保存ip

int ftp\_sendcmd();//发送指令

int ftp\_checkresp(char expresp);//接收返回状态，检测是否成功

int ftp\_mkdirSingle(char\* dir);

};

int CFTP::ftp\_connect(const char \*ip)

{

m\_sockctrl = socket(AF\_INET,SOCK\_STREAM, 0);

if(0 == m\_sockctrl)

return -1;

struct sockaddr\_in addr;

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(21);

addr.sin\_addr.s\_addr = inet\_addr(ip);

int err = connect(m\_sockctrl, (sockaddr\*)&addr,sizeof(addr));

if(err)return -1;

err = ftp\_checkresp('2');

if(err)return -1;

return 0;

}

int CFTP::ftp\_checkresp(char expresp)

{

int len = recv(m\_sockctrl,m\_resp,256,0);

if(-1 == len)return -1;

m\_resp[len]=0;

puts(m\_resp);//应该保存ftp运行日志,此处应该写入日志文件

if(m\_resp[0]!=expresp)return -1;

return 0;

}

int CFTP::ftp\_login(char\* user,char\* pass)

{

sprintf(m\_cmd,"USER %s\r\n",user);

int err = ftp\_sendcmd();

if(err)return -1;

err = ftp\_checkresp('3');

if(err)return -1;

sprintf(m\_cmd,"PASS %s\r\n",pass);

err = ftp\_sendcmd();

if(err)return -1;

err = ftp\_checkresp('2');

if(err)return -1;

return 0;

}

int CFTP::ftp\_sendcmd()

{

int ret = send(m\_sockctrl,m\_cmd,strlen(m\_cmd),0);

if(-1 == ret)return -1;

return 0;

}

int CFTP::ftp\_quit()

{

sprintf(m\_cmd,"QUIT\r\n");

int err = ftp\_sendcmd();

if(err)return -1;

err = ftp\_checkresp('2');

if(err)return -1;

return 0;

}

int CFTP::ftp\_pwd(char\* buff)

{

sprintf(m\_cmd,"PWD\r\n");

int err = ftp\_sendcmd();

if(err)return -1;

err = ftp\_checkresp('2');

if(err)return -1;

char\* p=m\_resp;

while(\*p)

{

if(\*p++ == '"')

while(\*p!='"')

\*buff++=\*p++;

}

\*buff=0;

printf("current work directory is : %s\n",buff);

return 0;

}

int CFTP::ftp\_upload(char\* localfile,char\* remotepath,char\* remotefilename)

{

ftp\_mkdir(remotepath);

int err = ftp\_cd(remotepath);

if(err)return -1;

ftp\_setpasv();

sprintf(m\_cmd,"STOR %s\r\n",remotefilename);

err = ftp\_sendcmd();

if(err)return -1;

err = ftp\_checkresp('1');

if(err)return -1;

FILE\* pf = fopen(localfile,"r");

if(NULL==pf)return -1;

char sendbuf[256];

size\_t len = 0;

while((len = fread(sendbuf,1,255,pf))>0)

{

err = send(m\_sockdata,sendbuf,len,0);

if(err<0)return -1;

}

close(m\_sockdata);

fclose(pf);

err = ftp\_checkresp('2');

if(err)return -1;

return 0;

}

int CFTP::ftp\_setpasv()

{

sprintf(m\_cmd,"PASV\r\n");

int err = ftp\_sendcmd();

if(err)return -1;

err = ftp\_checkresp('2');

if(err)return -1;

m\_sockdata = socket(AF\_INET,SOCK\_STREAM,0);

unsigned int v[6];

union {

struct sockaddr sa;

struct sockaddr\_in in;

} sin;

sscanf(m\_resp,"%\*[^(](%u,%u,%u,%u,%u,%u",&v[2],&v[3],&v[4],&v[5],&v[0],&v[1]);

sin.sa.sa\_family = AF\_INET;

sin.sa.sa\_data[2] = v[2];

sin.sa.sa\_data[3] = v[3];

sin.sa.sa\_data[4] = v[4];

sin.sa.sa\_data[5] = v[5];

sin.sa.sa\_data[0] = v[0];

sin.sa.sa\_data[1] = v[1];

int on =1;

if (setsockopt(m\_sockdata,SOL\_SOCKET,SO\_REUSEADDR,

(const char\*) &on,sizeof(on)) == -1)

{

perror("setsockopt");

close(m\_sockdata);

return -1;

}

struct linger lng = { 0, 0 };

if (setsockopt(m\_sockdata,SOL\_SOCKET,SO\_LINGER,

(const char\*) &lng,sizeof(lng)) == -1)

{

perror("setsockopt");

close(m\_sockdata);

return -1;

}

err = connect(m\_sockdata,(sockaddr\*)&sin,sizeof(sin));

if(err)return -1;

return 0;

}

int CFTP::ftp\_download(char\* localfile,char\* remotefile)

{

/\* sprintf(m\_cmd,"TYPE A\r\n",remotefile);

int err = ftp\_sendcmd();

if(err)return -1;

err = ftp\_checkresp('2');

if(err)return -1;\*/

ftp\_setpasv();

sprintf(m\_cmd,"RETR %s\r\n",remotefile);

int err = ftp\_sendcmd();

if(err)return -1;

err = ftp\_checkresp('1');

if(err)return -1;

FILE\* pf = fopen(localfile,"w");

if(NULL==pf)return -1;

char recvbuf[256];

int len = 0;

while((len = recv(m\_sockdata,recvbuf,256,0))>0)

{

err = fwrite(recvbuf,len,1,pf);

if(len<0)return -1;

}

close(m\_sockdata);

fclose(pf);

err = ftp\_checkresp('2');

if(err)return -1;

return 0;

}

### SSL

### 相关知识。

SSL：使用命令:加密openssl enc -des3 -salt -a -in 需要加密文件 -out 加密后文件.des3。解密:openssl enc -d -des3 -salt -a -in 加密文件.des3 -out 解密后文件。

单向加密的特征码：openssl dgst -sha 需要加密的文件, 优势:文件稍微改动特征码完全不一样，可看出传输过程中是否被修改。

证书的生成：

首先用户要有一对公钥和密钥，然后向CA机构发起证书发出请求：但是作为CA机构自己也要有自己的证书，所以需要先为自己生成一个证书：

进入CA目录(find / -name CA)

(umask 66; openssl genrsa 2048 > private/cakey.pem) //为自己生成一个证书

openssl req -new -x509 -key private/cakey.pem -out cacert.pem -days 3650(执行后有交互式回答,完成之后就能生成一个证书)

然后用户申请证书：

生成私钥:(umask 66; openssl genrsa 1024 >my.key)。

生成公钥:openssl rsa -in my.key（私钥） -pubout -out my.pub。公钥是从私钥中提取出来的。

CA(证书申请): openssl req -new -key my.key -out my.csr(或者openssl req -newkey rsa:1204 -out req1.pem(证书申请) -keyout sslclientkey.pem(私钥)  
)

接下来就是CA机构进行审核然后颁发证书:

之前要编辑openssl.cnf文件的dir路径。(也可执行CA脚本文件，./CA -newca 可生成所需目录，和cakey.pem、cacert.pem需要输入密码为了审核)

在CA目录下mkdir certs newcerts crl //建立证书、新证书、证书撤销目录

#touch index.txt serial

#echo “01”>serial //初始化序列号

最后审核颁发openssl ca -in 证书申请文件名.pem -out 签发的证书文件名.pem -days 3650

### 相关例子。

SSL编程:需要<openssl/err.h>、<openssl/ssl.h>编译用加-lssl。

1. OpenSSL初始化  
   int SSL\_library\_int(void);  
   （2）选择会话协议(C/S要相互兼容的协议，TLSv1.0、SSLv2、SSLv3、SSLv2/v3
2. 创建会话环境

申请会话环境:SSL\_CTX \*SSL\_CTX\_new(SSL\_METHOD \* method);  
SSL握手阶段的证书验证和加载自己的证书:

制定证书验证方式的函数是：int SSL\_CTX\_set\_verify(SSL\_CTX \*ctx,int mode(SSL\_VERIFY\_NONE),int(\*verify\_callback),int(X509\_STORE\_CTX \*));  
为SSL会话环境加载CA证书的函数是：  
SSL\_CTX\_load\_verify\_location(SSL\_CTX \*ctx,const char \*Cafile,const char \*Capath);  
为SSL会话加载用户证书的函数是：  
SSL\_CTX\_use\_certificate\_file(SSL\_CTX \*ctx, const char \*file,int type);  
为SSL会话加载用户私钥的函数是：  
SSL\_CTX\_use\_PrivateKey\_file(SSL\_CTX \*ctx,const char\* file,int type);  
在将证书和私钥加载到SSL会话环境之后，就可以调用下面的函数来验证私钥和证书是否相符：  
int SSL\_CTX\_check\_private\_key(SSL\_CTX \*ctx);

(4)建立SSL套接字

SSL \*SSL\_new(SSL\_CTX \*ctx) //申请一个SSL套接字

int SSL\_set\_fd(SSL \*ssl, int fd) //绑定读写套接字

int SSL\_set\_rfd(SSL \*ssl, int fd)//绑定只读套接字

int SSL\_set\_wfd(SSL \*ssl, int fd)//绑定只写套接字

1. 完成SSL握手

在成功创建SSL套接字后，客户端应使用函数SSL\_connect( )替代传统的函数connect( )来完成握手过程:  
int SSL\_connect(SSL \*ssl);

服务端：int SSL\_accept(SSL \*ssl);  
握手过程完成之后，通常需要询问通信双方的证书信息，以便进行相应的验证:

X509 \*SSL\_get\_peer\_certificate(SSL \*ssl);//从SSL套接字中提取对方的证书信息，这些信息已经被SSL验证过了。

X509\_NAME \*X509\_get\_subject\_name(X509 \*a);//该函数得到证书所用者的名字。

1. 进行数据传输

int SSL\_read(SSL \*ssl,void \*buf,int num);  
int SSL\_write(SSL \*ssl,const void \*buf,int num);  
(7)结束SSL通信

int SSL\_shutdown(SSL \*ssl);//关闭SSL套接字

void SSL\_free(SSL \*ssl); //释放套接字

void SSL\_CTX\_free(SSL\_CTX \*ctx); //释放SSL会话环境  
  
服务端框架:

SSL\_library\_init();

OpenSSL\_add\_all\_algorithms();

SSL\_load\_error\_strings();

cxt = SSL\_CTX\_new(SSLv23\_server\_method());

if(ctx == NULL)

{

ERR\_print\_errors\_fp(stdout);

exit(1);

}

{//加载用户证书

SSL\_CTX\_use\_certificate\_file(ctx, “用户签发后的证书名”,SSL\_FILETYPE\_PEM);

//加载用户私钥

SSL\_CTX\_check\_private\_key(ctx,”用户私钥”,SSL\_FILETYPE\_PEM)

//检查用户私钥是否正确

SSL\_CTX\_check\_private\_key(ctx)}

fd =socket();

Bind();

Listen();

accept();

SSL \*ssl = SSL\_new(ctx);

SSL\_set\_fd(ssl,fd);

SSL\_accept(ssl);

SSL\_read(ssl, buf, sizeof(buf));

客户端框架:

SSL\_library\_init();

OpenSSL\_add\_all\_algorithms();

SSL\_load\_error\_strings();

cxt = SSL\_CTX\_new(SSLv23\_server\_method());

fd =socket();

connect();

SSL \*ssl = SSL\_new(ctx);

SSL\_set\_fd(ssl, fd);

SSL\_connect(ssl)

{

X509 \*cert = SSL\_get\_peer\_certificate(ssl);//获取服务端发来的证书

char \*line = X509\_NAME\_oneline(X509\_get\_subject\_name(cert),0,0);//服务器证书line

line = X509\_NAME\_oneline(X509\_get\_issuer\_name(cert),0,0); //证书签发者:line

}

SSL\_write(ssl,”hello”,strlen(“hello”))