编　　制	沈胜文
审　　核	
批　　准	
实施责任人	

Amendment history 修改历史记录

版本号	修改说明	修改批准人	修改人	日期	签收人
101	创建文档		沈胜文	2008-3-31	
102	修改		沈胜文	2008-4-11	

编　　制	沈胜文
审　　核	
批　　准	
实施责任人	

Table of Content 目录

1. Introduction 简介

1.1. Objective 编写目的

在 Mu_FTAD 项目中，需要实现 HTTP 与 HTTPS 二种连接或是下载，但是 HTTP 与 HTTPS 在具体的实现时，存在差异。HTTPS 使用 Openssl 库实现，使用独立的，与原 socket 不一致的 I/O 操作函数；

出于方便应用，并且也方便维护的目的，我们考虑将各 I/O 操作函数使用数据结构关联所有的操作函数，使所有 I/O 函数独立于协议，实现协议无关。向用户提供独立的操作函数接口；

同时，对不同网络，不同协议实现过程中将会使用的 I/O 操作方法进行封装，对外提供统一的接口，也是为了使所有网络相关项目均可使用该 I/O 函数，避免重复开发，提高开发效率；

本文档将尽可能详尽地说明相关数据结构的设计与开发，但是设计与开发不相符之处，需讨论决定，并且修改本文档。

最终设计以代码为准。

1.2. Background 背景

本程序是法电自动下载（MuFTAD）软件项目中的一部分，负责网络通信时的 I/O 操作函数，包括：connect, write, read, close, poll, select……

设计的出发　点：使用 MuFTAD 项目中涉及的网络通信函数独立于 MuFTAD 项目，也独立于具体协议，方便相关操作的更新与完善；
本软件的提出者：沈胜文
本软件的开发者：沈胜文
本软件的用　户：MuFTAD 项目和其他所有使用网络的项目；

1.3. Terms & Abberviation 术语&缩写解释

Terms&Abbreviation 术语&缩写	Description 解释

Openssl	The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade, full-featured, and Open Source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols as well as a full-strength general purpose cryptography library.
MuFTAD	软件名称 Microunit France-Telecom Auto-Download
HTTPS	HTTPS 实际上应用了 Netscape 的安全套接字层（SSL）作为 HTTP 应用层的子层。（HTTPS 使用端口 443，而不是象 HTTP 那样使用端口 80 来和 TCP/IP 进行通信。）SSL 使用 40 位关键字作为 RC4 流加密算法，这对于商业信息的加密是合适的。HTTPS 和 SSL 支持使用 X.509 数字认证，如果需要的话用户可以确认发送者是谁。
SSL	SSL 是一个缩写，代表的是 Secure Sockets Layer。它是支持在 Internet 上进行安全通信的 标准，并且将数据密码术集成到了协议之中。数据在离开您的计算机之前就已经被加密，然后只有 到达它预定的目标后才被解密。
HTTP	HTTP（Hyper Text Transfer Protocol），即超文本传输协议是一种 Internet 上最常见的协议，用于传输超文本标记语言（HTML--Hyper Text Markup Language）写的文件。

1.4. Reference Material 参考资料

OpenSSL 程序设计.pdf
O'Reilly.Network.Security.with.OpenSSL.pdf
用 OpenSSL 进行 TLS_SSL 编程.pdf
Unix 网络编程

2. Rules 规则

2.1. Name Rules 命名规则

相关数据结构的操作服务仅于 MuFTAD 项目。该程序中，所有的函数均以 Mu_的形式开头，而不是 MuFTAD_。因此在该程序内，所有的函数形如：Mu_XXXX()；

2.2. Illuminate 说明

针对每个程序，都必须注明其开发目的，开发者，开发时间，等等。以下字段必须被包含于程序的开头部分。

```
/**********************************************************************
*                        ====Microunit Techonogy Co,.LTD.====
* File Name:
*
*          XMLParse.c
*
* Description:
*
*          This file get the file name, which store the XML Contents.
*          The Functions Open the file, parse it, then return the Information we need.
*
* Revision History:
*
*          10-3-2008 ver1.0
*
*Author:
*
*          ssw    (fzqing@gmail.com)
*
*                        ***PROTECTED BY COPYRIGHT***
**********************************************************************/
```

2.3. Note Rules 注释规则

程序中的各个函数均需要明确注释其功能，并能简要描述其实现，及注意点。特别应该注意的是：在描述时，应该详细包括对锁，输入和输出进行详细说明；

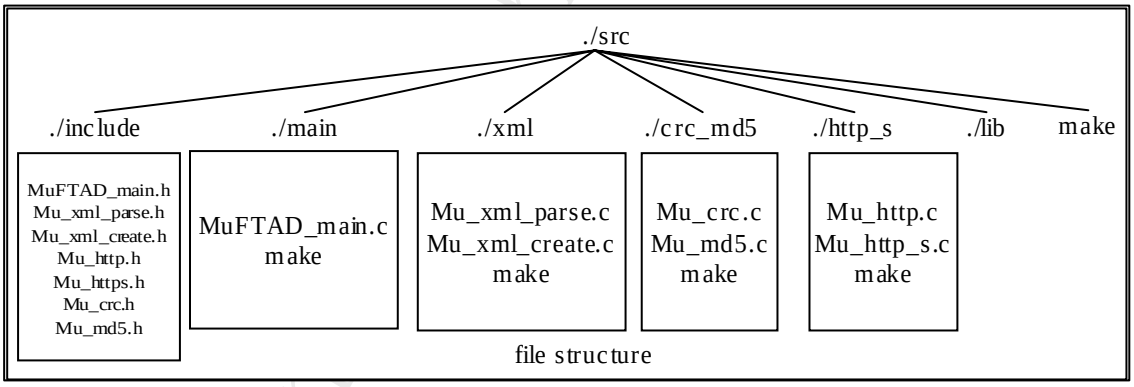
可参考模板

```
/*****************************************************************
*Description:
*       This Function is Parse the XML, return the Informations to caller use the
*       Value pointer;
*Input:
*       filename: the file name , which stored the XML contents
*Output:
*       Pointer: which is a pointer, point to the buffer stored the Informations\
*LOCK:
*       NONE
*Modify:
*       ssw (fzqing@gmail.com    10-3-2008)
*****************************************************************/
```

2.4. File Structure 文件结构

[注意]:

1、此为整个项目的文件组织方式;



file structure

./src/include：文件夹，包含该项目中的所有头文件；

./src/main：文件夹，包含所有按法电《do3c00_SoftProtocol_0.1.0_RC1》流程所开发的程序；

./src/xml：文件夹，包含项目中所需要的 xml 处理库函数源代码；

./src/crc_md5：文件夹，包含项目中所需要的校验函数源代码，包括 CRC 和 MD5校验代码；

./src/http_s：文件夹，包含项目中所需要的与服务器交互的方式，包括 HTTP(s) GET、POST 方式；

./src/lib：文件夹，用于存储编译所生成的 xml、http 和 https、crc/md5库。软件编译连接时使用该文件夹下的库；

./src/make：文件，总的编译入口；

[注意]:

1、各对应文件夹下的源文件按需要添加，但是所作修改必须对 makefile 文件作相应的修改，以正确编译;

3. Structure Of Routines 程序结构

3.1. Overview 总述

本文档中所涉及的待开发部分都是相互独立的，除了一个用户可调用的初始化函数接口；

现阶段，本文档仅仅包含网络通信的二个部分：普通 Socket I/O 操作和 Openssl 所提供的 I/O 操作；

初始化对各个函数组织到网络 I/O 结构体中；

3.2. Routines List 函数列表

```
int Mu_NetIOInit(MuNetIOPtr *Ptr);

int Mu_Selecter(int fd, int wr, double timeout);
int Mu_SocketSelect(int fd, int wr, double timout);
int Mu_OpensslSelect(int fd , int wr, double timeout);

int Mu_Reader(int fd, char *buf, int len);
int Mu_SocketRead(int fd, char *buf, int len);
int Mu_OpensslRead(int fd, char *buf, int len);

int Mu_Writer(int fd, char *buf, int len);
int Mu_SocketWrite(int fd, char *buf, int len);
int Mu_OpensslWrite(int fd, char *buf, int len);

int Mu_Connecter(int *socket, const char *host, int port, double timeout);
int Mu_SocketConnect(int socket, const char *host, int port, double timeout);
int Mu_OpensslConnect(int socket);

int Mu_Peeker(int fd, char *buffer, int len);
int Mu_SocketPeek(int fd, char *buffer, int len);
```

```
int Mu_OpensslPeek(int fd, char *buffer, int len);

int Mu_Closer(int fd);
int Mu_SocketClose(int fd);
int Mu_OpensslClose(int fd);
```

4. Global Description 全局描述

4.1. Global Type 全局类型

```
#define MAX_THD 10

typedef int (*mu_reader)( int, char *, int);
typedef int (*mu_writer)( int, char *, int);
typedef int (*mu_reader)( int, char *, int);
typedef int (*mu_peeker)( int, char *, int);
typedef int (*mu_closer)( int);
typedef int (*mu_connecter)( int *, const char *, int, double);
typedef int (*mu_selecter)( int, int, double);
```

4.2.Structure Definition 结构体定义

4.2.1. Mu_NetType

```
typedef enum _MuNetType{
    MU_NONE=0,
    MU_HTTP,
    MU_HTTPS,
}MuNetType
```

[注意]:

1、该联合体用于指明在下载过程中所使用的协议类型;

4.2.2. Mu_WR

```
typedef enum _MuWR{
    MU_WRITE = 0,
    MU_READ
```

}MuWR

4.2.3. Mu_NetIO

```
typedef struct _MuNetIO{
    volitale int type;          //MuNetType
    mu_reader reader;           //for read
    mu_writer writer;           //for writer
    mu_poller poller;       //for poll
    mu_selecter selecter; //for select
    mu_peeker peeker;           //for read with options peek
    mu_closer closer;           //close socket
    mu_connecter connect;     //connect to server
}MuNetIO, *MuNetIOPtr;
```

[注意]:
1、该 IO 结构在操作前必须被初始化，并且被设置为全局变量;
2、对 URL 进行解析后，必须修改 type 值;
3、该结构在本项目中初始化为 MuIO;

4.2.4. Mu_SslTab

```
typedef Struct _MuSslTab{
    int fd;
    SSL *ssl;
}MuSslTab, *MuSslTab;
```

MuSslTab[MAX_THD]

[注意]:
1、多线程 HHTPS 下载时，必须用 fd 在 MuSslTab 中查找 SSL;
2、MuSslTab 必须被置为全局变量;

4.3. Macro 宏

4.3.1. Mu_FindSsl

```
#define Mu_FindSsl(ssl, fd) do{\
    int i = 0;\
```

```
        while(i < MAX_THD){\
            if(MuSslTab[i] == fd)\
                break;\
            i ++;\
        }\

        if(i == MAX_THD)\
            ssl = NULL;\
        else\
            ssl = MuSslTab[i].ssl;\
}while(0)
```

5. Routines Detail 函数细节

5.1. Mu_InitNetIO

5.1.1. Name 函数名称

int Mu_InitNetIO(MuNetIOPtr * Ptr)

5.1.2. Description 函数描述

该函数用于申请空间存储 MuNetIO 结构体成员，并初始化该结构体，用我们自己的 I/O 操作函数来填充该结构体；

调用者必须要提供一个 MuNetIO 类型的二级指针参数，用于返回内存空间；

函数在初始化正确后，需要返回状态值，以表明初始化成功与否；

[注意]:

*1、Ptr 是一个二级指针，*Ptr 所指空间必须由用户显示释放；否则它会一直占用动态内存空间;*

5.1.3. Function 功能

初始化结构体 MuNetIO，向调用者提供包含所有 I/O 操作函数指针的结构体；

[注意]:
1、注册的操作函数不一定都支持所有方式，但是我们应该明白如何去调用;
2、调用无相应操作的函数，程序会返回错;

5.1.4. Capability 性能

略

5.1.5. Input 输入

Ptr：MuNetIOPtr 指针类型，以二级指针的形式传入函数，同时，它也作为返回值，向调用者返回结构者的空间地址；

5.1.6. Output 输出

操作状态码：
NO_MUERROR：无错误；
ERROR_BUFF_EMPTY：无内存空间保存该结构；

5.1.7. Arithmetic 算法

略

5.1.8. Process 处理流程

略

5.1.9.Pseudocode 伪代码

```
int Mu_InitNetIO(MuNetIOPtr *Ptr)
{
    MuNetIOPtr ret = NULL;

    *Ptr = (MuNetIOPtr)malloc(sizeof MuNetIO));
    if(NULL == *Ptr){
```

```
        do error;
        return MUNBUF;
    }

    ret = *Ptr;
    ret->type = MU_NONE;
    ret->reader = (mu_reader) Mu_Reader;
    ret->writer = (mu_writer)Mu_Writer;
    ret->connecter = (mu_connecter)Mu_Connecter;
    ret->selecter = (mu_selecter)Mu_Seclecter;
    ret->closer = (mu_closer)Mu_Closer;
    ret->peeker = (mu_peeker)Mu_Peeker;

    return MUOK;
}
```

5.1.10. Interface 接口

本函数在完成初始化时，需要利用其他函数指针来填充结构体成员，函数包括：

Mu_Reader：读取数据函数；
Mu_Writer：写数据函数；
Mu_Connecter：连接函数；
Mu_Selecter：复用函数；
Mu_Closer：关闭连接函数；
Mu_Peeker：取值函数，不同于 read，Network Buffer 中仍保留原有数据；

5.1.11. Malloc 存储分配

函数利用动态分配函数来分配内存空间存储结构体 MuNetIO，申请的内存空间，必须由调用者自己释放该内存空间；

5.1.12. Restrict 限制

略

5.1.13. Test 测试

略

5.1.14. Unsolve 未解决情况

略

5.2. Mu_Selecter

5.2.1. Name 函数名称

int Mu_Selecter(int fd, int wr, double timeout)

5.2.2. Description 函数描述

该函数复用普通套接口和 SSL 类型的套接口，避免在读或写一个套接口时，因套接口不可用而长期阻塞；

该函数将被用于 Mu_Writer、Mu_Reader、Mu_Peeker 中，以检测一个套接口（对于 HTTPS 则是一个 SSL 对像）是否可读或可写；

该函数按照类型值 type，选用不同的 I/O 复用函数；

[注意]:
1、Openss1 提供的库函数中，I/O 复用函数有别于 Linux 库函数中函数;

5.2.3. Function 功能

按照所使用的协议，选择不同的描述符检测函数；

5.2.4. Capability 性能

略

5.2.5. Input 输入

fd：int 类型，套接字接口；
wr：int 类型，用于标记复用的 I/O 是用于读的还是用于写的；
timeout：double 类型，复用 I/O 时的超时时间；

5.2.6. Output 输出

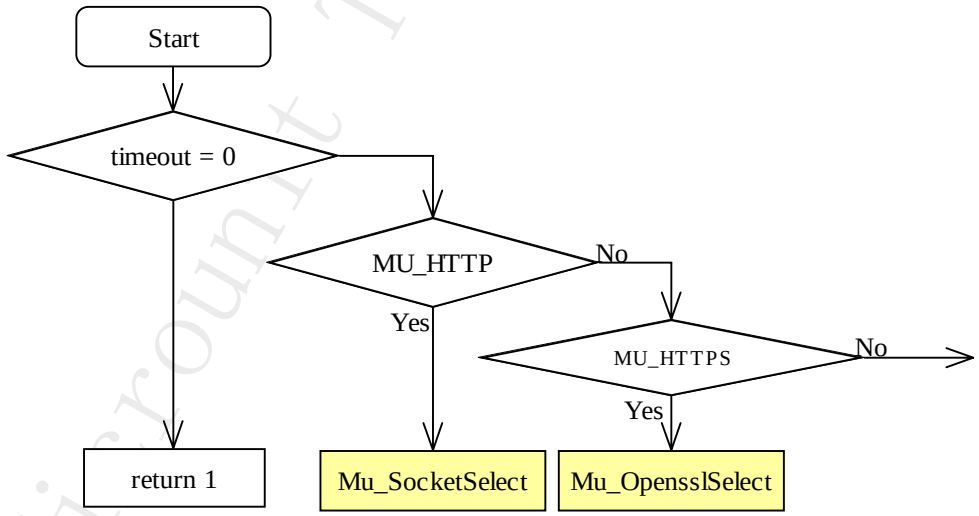
函数运行状态码；

0：表示不可读写；
非0：表示可读写；

5.2.7. Arithmetic 算法

略

5.2.8. Process 处理流程



5.2.9. Pseudocode 伪代码

int Mu_Selecter(int fd, int wr, double timeout)

```
{
    int ret = -1;

    if(timeout == 0)
        return 1;

    switch(MuIO->type){
        case MU_HTTP:
            ret = Mu_SocketSelect(int fd, int wr, double timeout);
            break;
        case MU_HTTPS:
            ret = Mu_OpensslSelect(int fd, int wr, double timeout)
            break;
        default:
            break;
    }

    if(!ret)
        errno = ERROR_TIMEOUT;

    if(ret < 0)
        return 0;

    return ret;
}
```

5.2.10. Interface 接口

根据复用套接口类型，选用不同的复用函数，因此在该函数中，需要调用其他接口：

Mu_SocketSelect;
Mu_OpensslSelect;

5.2.11. Malloc 存储分配

略

5.2.12. Restrict 限制

略

5.2.13. Test 测试

略

5.2.14. Unsolve 未解决情况

略

5.3. Mu_SocketSelect

5.3.1. Name 函数名字

int Mu_SocketSelect(int fd, int wr, double timout)

5.3.2. Description 函数描述

类似于 Linux 系统提供的 Select 函数，该函数实际上也是调用 select 函数对普通套接口 I/O 复用；

返回值类似于 select 函数；
< 0：出错，无可读写 I/O；
=0：超时，并且此时无可用 I/O；
>0：有可读写 I/O，返回值即可读写 I/O 的句柄号；

5.3.3. Function 功能

复用普通套接口 I/O；

5.3.4. Capability 性能

略

5.3.5. Input 输入

同5.2.5.中说明

5.3.6. Output 输出

同5.2.6.中说明

5.3.7. Arithmetic 算法

略

5.3.8.Process 处理流程

略

5.3.9.Pseudocode 伪代码

```
int Mu_SocketSelect(int fd, int wr, double timout)
{
    fd_set fdset;
    fd_set *rd = NULL, *wr = NULL;
    struct timeval tmout;
    int ret = 0;

    FD_ZERO (&fdset);
    FD_SET (fd, &fdset);
    if (wr & WAIT_FOR_READ)
        rd = &fdset;
    if (wr & WAIT_FOR_WRITE)
        wr = &fdset;

    tmout.tv_sec = (long)timeout;
    tmout.tv_usec = 1000000 * (timeout - (long) timeout);

    do
        ret = select (fd + 1, rd, wr, NULL, &tmout);
    while (ret< 0 && errno == EINTR);
```

```
    return ret;
}
```

5.3.10. Interface 接口

略

5.3.11. Malloc 内存分配

略

5.3.12. Restrict 限制

略

5.3.13. Test 测试

略

5.3.14. Unsolve 未解决情况

略

5.4. Mu_OpensslSelect

5.4.1. Name 函数名称

int Mu_OpensslSelect(int fd , int wr, double timeout)

5.4.2. Descriptor 函数描述

对 Openssl 库提供的 SSL 对像进行复用；

在 HTTPS 中，使用 Openssl 库进行开发，该库将套接口使用 SSL 对像进行封装，提供的读写函数均是针对于该对像的，因此，在复用 I/O 时，应该同时考虑到对该对像的复用；

针对于读或写，SSL 对像的复用也是不一样的；
[注意]:
1、具体的细节参考文献文档;

5.4.3. Function 功能

复用 SSL 对像，检测出对像或是套接口的可读写性；

5.4.4. Capability 性能

略

5.4.5. Input 输入

同5.3.5. 中说明

5.4.6. Output 输出

< 0：出错，无可读写 I/O；

=0：超时，并且此时无可用 I/O；
也表明 SSL Buffer 中有数据，不可写；

>0：同5.3.6.中说明；
也表明 SSL Buffer 中有数据可读；

5.4.7. Arithmetic 算法

在实现对 openssl 提供的库或是套接口 I/O 进行复用时，应该分清楚其提供的操作函数及其提供的内存空间在整个流程中所处的位置；

编　　制	沈胜文
审　　核	
批　　准	
实施责任人	

在进行设计前，必须弄清楚 Openssl 使用的内存 SSL buffer 同 Application Buff、Network Buffer 之间的区别；

同时，也必须搞清楚 select、SSL_pending 所处的位置，所操作的对像；

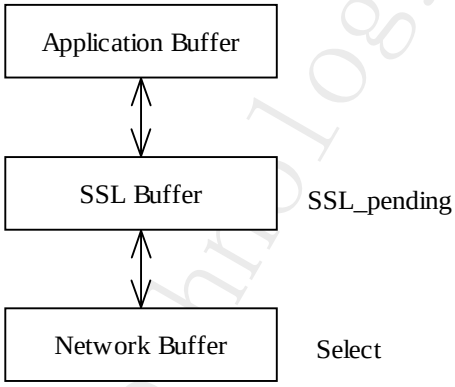
[注意]:

1、SSL_pending 函数在 Openssl 库中，只提供对 SSL Buffer 中是否有数据可读进行检测，并不对可写进行相应操作;

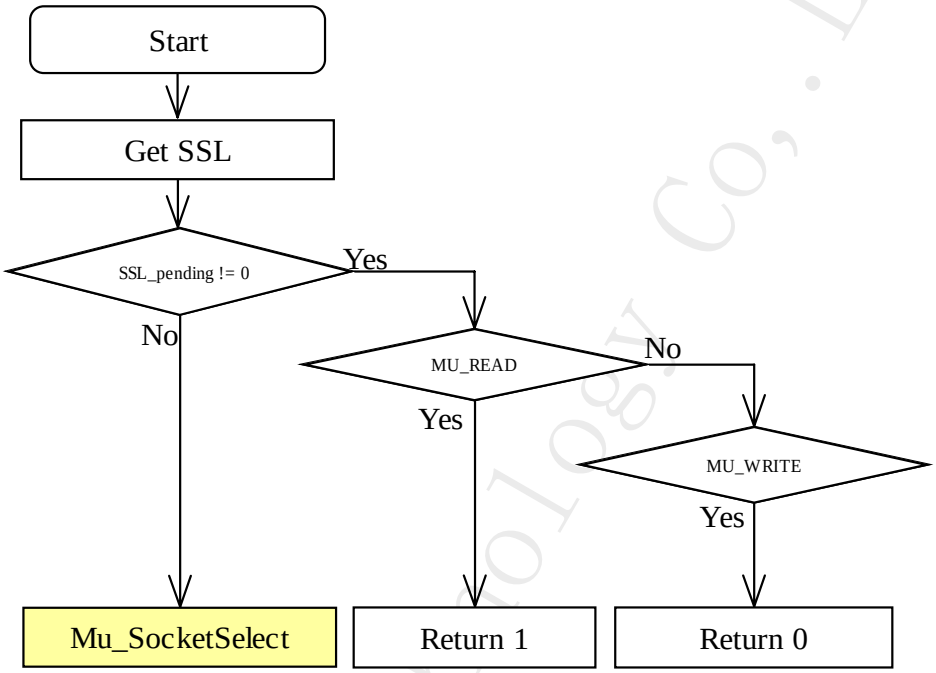
2、SSL_pending 返回 SSL Buffer 空间中可读取的字节数;

3、为了保证写时 SSL Buffer 中没有可读取数据，在复用可写 SSL 对像时，先保证 SSL_pending 函数返回0;

下图简要说明其关系：



5.4.8. Process 处理流程



5.4.9. Pseudocode 伪代码

```
int Mu_OpensslSelect(int fd , int wr, double timeout)
{
    SSL *ssl;
    Mu_FindSsl(ssl, fd);

    if(NULL == ssl)
        return 0;

    if(SSL_pending(ssl)){
        switch(wr){
            case READ:
                return 1;
            case WRITE:
                return 0;
            default:
                return 0;
        }
    }
    else
```

```
        return Mu_SocketSelect(fd, wr, timeout);
}
```

5.4.10. Interface 接口

当 SSL Buffer 中没有数据时，我们使用 Mu_SocketSelect 函数来复用底层的套接口 I/O，以检测是否有数据可写或可读；

Mu_SocketSelect 函数实际上也是调用 Linux 系统函数 Select；

5.4.11. Malloc 存储分配

略

5.4.12. Restrict 限制

略

5.4.13. Test 测试

略

5.4.14. Unsolve 未解决情况

略

5.5. Mu_Reader

5.5.1. Name 函数名称

int Mu_Reader(int fd, char *buf, int len)

5.5.2. Description 函数描述

函数接受用户的输入，按类型 type 选择合适的应用函数来读取数据，但是它本身并不从实际的 fd 中读取网络数据；

5.5.3. Function 功能

选择合适的函数读取 Fd 或是 SSL Buffer 中的数据；

5.5.4. Capability 性能

略

5.5.5. Input 输入

fd：套接口描述字；
buf：用于存储网络数据的内存区域；
len：可读取的数据长度；
[注意]:
1、上述参数均被传递到真正的读取函数;

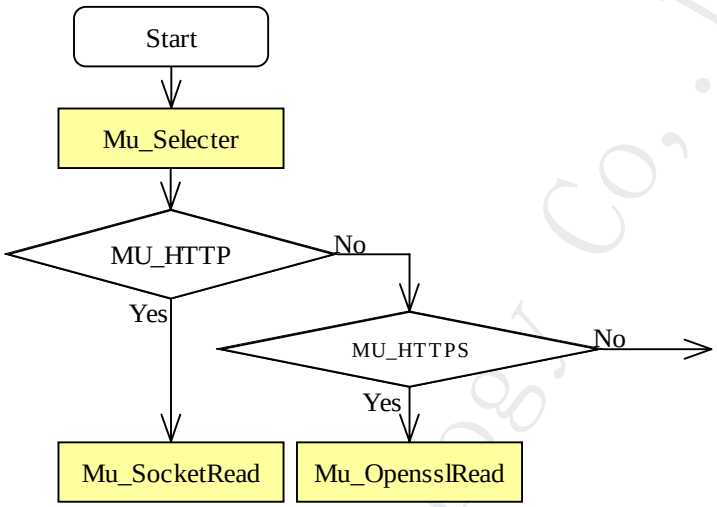
5.5.6. Output 输出

同 Linux 库函数 read 返回值；
>0 表示读取的字节数；
=0 表示到达尾部，无字节可读；
<0 发生错误；
[注意]:
1、返回字节数并不等于传递的参数 len;

5.5.7. Arithmetic 算法

略

5.5.8. Process 处理流程



5.5.9. Pseudocode 伪代码

```
int Mu_Reader(int fd, char *buf, int len)
{
    if(!Mu_Selecter(type, fd, READ, 30))
        return -1;

    switch(MuIO->type){
    case MU_HTTP:
        return Mu_SocketRead(int fd, char *buf, int len);
    case MU_HTTPS:
        return Mu_OpensslRead(int fd, char *buf, int len);
    default:
        return MUEERO;
    }
}
```

5.5.10. Interface 接口

函数功能中已说明该函数仅仅用于按照 type 值分解所提交的处理，所以函数中也用到了其他函数：

Mu_SocketRead;
Mu_OpensslRead;

5.5.11. Malloc 存储分配

略

5.5.12. Restrict 限制

略

5.5.13. Test 测试

略

5.5.14. Unsolve 未解决情况

略

5.6. Mu_SocketRead

5.6.1. Name 函数名称

int Mu_SocketRead(int fd, char *buf, int len)

5.6.2. Description 函数描述

从普通套接口描述字中读取数据；

5.6.3. Function 功能

从 fd 中读取数据，不保证一定能读取到长度为 len 的数据。函数被调用一次能从中读取的数据最多为 len；

5.6.4. Capability 性能

略

5.6.5. Input 输入

同5.2.5.中说明

5.6.6. Output 输出

同5.2.6.中说明

5.6.7. Arithmetic 算法

从套接口中读取数据时，应该考虑到中断对读操作的影响；

5.6.8. Process 处理流程

略

5.6.9.Pseudocode 伪代码

```
int Mu_SocketRead(int fd, char *buf, int len)
{
    int res;
    do
        res = read(fd, buf, len)
    while(res == -1 && errno == EINTR)

    return res;
}
```

5.6.10. Interface 接口

函数的实现利用了 Linux 网络 I/O 接口函数 read;

5.6.11. Malloc 内存分配

略

5.6.12. Restrict 限制

略

5.6.13. Test 测试

略

5.6.14. Unsolve 未解决情况

略

5.7. Mu_OpensslRead

5.7.1. Name 函数名称

int Mu_OpensslRead(int fd, char *buf, int len)

5.7.2. Description 函数描述

从 SSL Buffer 中读取数据，返回读取结果；

5.7.3. Function 功能

从 SSL Buffer 中读取数据，不保证一定能读取到长度为 len 的数据。函数被调用一次能从中读取的数据最多为 len；

5.7.4. Capability 性能

略

5.7.5. Input 输入

同5.2.5.中说明

5.7.6. Output 输出

同 SSL_read 函数相同，具体说明可以查阅 openssl 说明文档；

5.7.7. Arithmetic 算法

使用 SSL_read 函数从 SSL Buffer 中读取数据时，应该考虑到多种出错可能，在 Openssl 中，对错误的情况的查询使用 SSL_get_error 函数，具体使用可以查阅 openssl 说明文档；

针对读/写的 I/O 操作时，需要处理的错误往往有：
SSL_ERROR_NONE：无错误，一般不作处理；
SSL_ERROR_WANT_READ：有数据需要读或写，需要再次调用 I/O 操作函数；
SSL_ERROR_WANT_WRITE：同上；
SSL_ERROR_ZERO_RETURN：连接错误；
SSL_ERROR_SSL：SSL 库函数调用错误；
SSL_ERROR_SYSCALL：I/O 错误；

本程序中，只处理由于中断导致的 I/O 操作错误，对其他错误，返回错误码；

5.7.8. Process 处理流程

略

5.7.9.Pseudocode 伪代码

```
int Mu_SocketRead(int fd, char *buf, int len)
{
    int res;
    SSL *ssl = NULL;

    Mu_FindSsl(ssl, fd);
    if(NULL == ssl)
        return error;

    do
        res = SSL_read(ssl, buf, len)
    while(res == -1
        && SSL_get_error(ssl, res) == SSL_ERROR_SYSCALL
        &&errno == EINTR);

    return res;
}
```

5.7.10. Interface 接口

函数的实现利用了 Openssl 库提供的网络 I/O 接口函数 SSL_read;

5.7.11. Malloc 内存分配

略

5.7.12. Restrict 限制

略

5.7.13. Test 测试

略

5.7.14. Unsolve 未解决情况

略

5.8. Mu_Writer

5.8.1. Name 函数名称

int Mu_Writer(int fd, char *buf, int len)

5.8.2. Description 函数描述

函数接受用户的输入，按类型 type 选择合适的应用函数来读取数据，但是它本身并不往实际的套接口或是 SSL Buffer 中写入数据；

5.8.3. Function 功能

选择合适的函数往套接口或是 SSL Buffer 中写入数据；

5.8.4. Capability 性能

略

5.8.5. Input 输入

fd：可写的套接口
buf：用于存储待写入套接口 fd 的数据；
len：待写入的数据长度；
[注意]:
1、上述参数均被传递到真正的写函数;

5.8.6. Output 输出

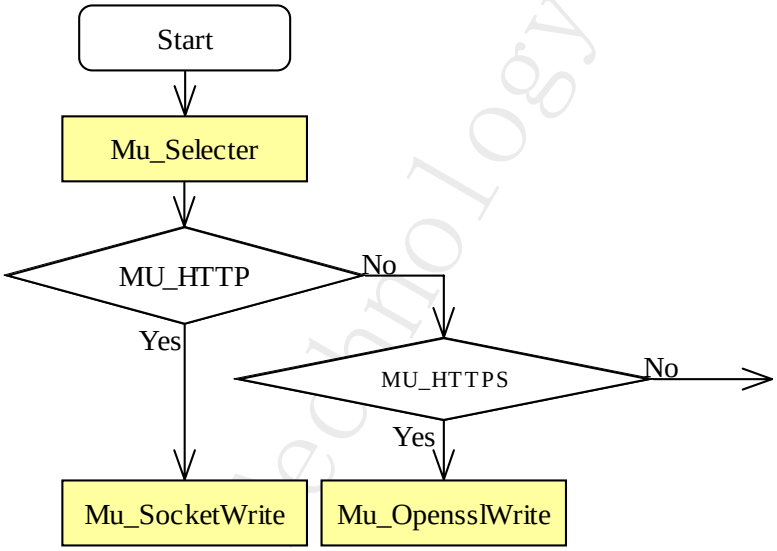
同 Linux 库函数 write 返回值；

<0 写未完成，出错返回；
=0 写完成

5.8.7. Arithmetic 算法

略

5.8.8. Process 处理流程



5.8.9. Pseudocode 伪代码

```
int Mu_Writer(int fd, char *buf, int len)
{
    if(!Mu_Selecter(type, fd, WRITE, 30))
        return -1;

    switch(MuIO->type){
    case MU_HTTP:
        return Mu_SocketWrite(int fd, char *buf, int len);
        break;
    case MU_HTTPS:
        return Mu_OpensslWrite(int fd, char *buf, int len);
        break;
    default:
```

return **MUEERO**；
}

5.8.10. Interface 接口

函数功能中已说明该函数仅仅用于按照 type 值分解所提交的处理，所以函数中也用到了其他函数：
Mu_SocketWrite;
Mu_OpensslWrite;

5.8.11. Malloc 存储分配

略

5.8.12. Restrict 限制

略

5.8.13. Test 测试

略

5.8.14. Unsolve 未解决情况

略

5.9. Mu_SocketWrite

5.9.1. Name 函数名称

int Mu_SocketWrite(int fd, char *buf, int len)

5.5.2. Description 函数描述

将待发送数据从普通套接口 fd 发送出去；

5.9.3. Function 功能

将数据从套接口 fd 中发送出去，发送时保证发送长度为 len，或是出错；

5.9.4. Capability 性能

略

5.9.5. Input 输入

同5.2.5.中说明

5.9.6. Output 输出

同5.2.6.中说明

5.9.7. Arithmetic 算法

往套接字里写数据时，必须保证写入长度与实际长度相符，否则返回错；

5.9.8. Process 处理流程

略

5.9.9.Pseudocode 伪代码

```
int Mu_SocketWrite(int fd, char *buf, int len)
{
    int res;
```

```
    while(len){

        do
            res = write(fd, buf, len)
        while(res == -1 && errno == EINTR)

        if(res <= 0)
            return res;

        buf += res;
        len -= res;
    }

    return res;
}
```

5.9.10. Interface 接口

函数的实现利用了 Linux 网络 I/O 接口函数 read;

5.9.11. Malloc 内存分配

略

5.9.12. Restrict 限制

略

5.9.13. Test 测试

略

5.9.14. Unsolve 未解决情况

略

5.10. Mu_OpensslRead

5.10.1. Name 函数名称

int Mu_OpensslWrite(int fd, char *buf, int len)

5.10.2. Description 函数描述

往 SSL Buffer 中写数据；

5.10.3. Function 功能

将数据从 SSL Buffer 中发送出去，发送时保证发送长度为 len，或是出错；

5.10.4. Capability 性能

略

5.10.5. Input 输入

同5.2.5.中说明

5.10.6. Output 输出

同5.2.6.中说明

5.10.7. Arithmetic 算法

略

5.10.8. Process 处理流程

略

5.10.9.Pseudocode 伪代码

```
int Mu_OpensslWrite(int fd, char *buf, int len)
{
    int res;

    Mu_FindSsl(ssl, fd);
    if(NULL == ssl)
        return MUESSL;

    while(len){

        do
            res = SSL_write(ssl, buf, len)
        while(res == -1
            && SSL_get_error(ssl, res) == SSL_ERROR_SYSCALL
            &&errno == EINTR);

        if(res < = 0)
            return res;

        buf += res;
        len -=res;
    }

    return res;
}
```

5.10.10. Interface 接口

函数的实现利用了 Openssl 库提供的网络 I/O 接口函数 SSL_write;

5.10.11. Malloc 内存分配

略

5.10.12. Restrict 限制

略

5.10.13. Test 测试

略

5.10.14. Unsolve 未解决情况

略

5.11. Mu_Connecter

5.11.1. Name 函数名称

int Mu_Connecter(int *socket, const char *host, int port, double timeout)

5.11.2. Description 函数描述

连接服务器，调用者必须提供 host 的地址字段，超时时间及连接端口，该函数并不连接服务器，仅仅依据类型参数 type，选择合适的函数来处理连接；

要处理的连接包括 HTTP 和 HTTPS；

5.11.3. Function 函数功能

调用合适的函数连接服务器；

5.11.4. Capability 性能

略

5.11.5. Input 输入

socket：整型指针，它用于返回所创建的套接字描述符；
port：用于连接服务器的端口，如何该端口值为0，连接函数将以默认的端口连接服务器；

HTTP 协议的连接将会采用80端口；
HTTPS 协议的连接将会采用344端口；

timeout：超时时间，本项目中的连接使用非阻塞式连接；

5.11.6. Output 输出

函数的运行状态码；

5.11.7. Arithmetic 算法

略

5.11.8.Process 处理流程

略

5.11.9.Pseudocode 伪代码

```
int Mu_Connecter(int *socket, const char *host, int port, double timeout)
{
    int ret = 0;
    struct sockaddr_in server;
    struct hostent *hp, hostbuf;
    int herr;

    char *temphost;
    int temphostlen = 2048;

    //check the parameters
    if(!port){
        switch(MuIO->type){
            case MU_HTTP:
                port = 80;
                break;
            case MU_HTTPS:
                port = 344;
                break;
```

```
            default:
                break;
}

if(NULL == host)
    return MUNHST;

bzero((void *)&server, sizeof(server));

//get the host name
if(1 != inet_pton(AF_INET, host, &(server.sin_addr))){
    if(NULL ==(temphost = malloc(temphostlen))
        return MUEDNS;

    memset(temphost, 0, temphostlen);
    while(gethostbyname_r(host, &hostbuf, temphost, 2048, &hp, &err)){
        temphostlen *= 2;
        if(NULL == (temphost = realloc(temphost, temphostlen)){
            free(temphost);
            return MUEDNS;
        }
        memset(temphost, 0, temphostlen);
    }
    memcpy((void *)&server.sin_addr, hp->h_addr, hp->h_length);
}

server.sin_family = AF_INET;
server.sin_port = htons(port);

//free the temp buffer
free(temphost);

//creat socket
if((*socket = socket(AF_INET, SOCK_STREAM, 0)) < 1){
    do error;
    return MUNSKT;
}

//connect to server
ret = Mu_SocketConnect(*socket, host, port, timeout);
if(!ret)
    return ret;

//connect USE openssl
if(MuIO->type == MU_HTTPS)
```

```
        ret = Mu_OpensslConnect(*socket);

    return ret;
}
```

5.11.10. Interface 接口

本函数仅仅是执行了参数检查，调用具体的连接函数执行连接请求，并且若连接是 HTTPS，还需要调用 Openssl 库来对套接口执行操作；

涉及的函数有：
Mu_SocketConnect;
Mu_OpensslConnect；

5.11.11. Malloc 内存分配

略

5.11.12. Restrict 限制

略

5.11.13. Test 测试

略

5.11.14. Unsolve 未解决情况

略

5.12. Mu_SocketConnect

5.12.1. Name 函数名称

int Mu_SocketConnect(int socket, const char *host, int port, double timeout)

5.12.2. Description 函数描述

本函数使用 port 端口，连接服务器 host，本函数使用非阻塞式的连接，超时连接时间为 timeout；

5.12.3.Function 功能

连接服务器；

5.12.4. Capability 性能

略

5.12.5.Input 输入

同5.11.5.中说明

5.12.6.Output 输出

同5.11.6.中说明

5.12.7.Arithmetic 算法

实现时使用了非阻塞式连接，在具体实现时，需要了解非阻塞式连接的实现方式，及出错处理；

5.12.8.Process 处理流程

略

5.12.9. Pseudocode 伪代码

int Mu_SocketConnect(int socket, const char *host, int port, double timeout)

```
{

    int flags, block;
    int statuse;

    //change to noblock IO
    flags = fcntl(socket ,F_GETFL, 0);
    if(flags != -1)
        block = fcntl(socket, F_SETFL, flags|O_NONBLOCK);
    else
        block = -1;

    //connect to socket, when connect to,will return a error,
    //reference to the book 《UNIX Network Program》
    statuse = connect(socket, (struct sockaddr *)&server, sizeof(server));

    if(statuse == -1 && block != -1 && errno == EINPROGRESS){
        fd_set fdset;
        struct timeval tv;

        FD_ZERO(&fdset);
        FD_SET(socket, &fdset);

        tv.tv_sec = (long)timeout;
        tv.tv_usec = 1000000 * (timeout - (long) timeout);

        status = select((socket + 1), NULL, &fdset, NULL, &tv);

        if(status > 0){
            socklen_t arglen = sizeof(int);

            //may be a BUG to USE,
            //look into the book <UNIX Network Program>
            if(getsocketopt(socket, SOL_SOCKET, SO_ERROR, &status,
&arglen)<0)
                status = errno;

            if(status != 0){
                errno = status;
                status = -1;
            }

            if(error == EINPROGRESS)
                errno = ETIMEOUT;
```

```
        }
        else if(status == 0){
            errno = ETIMEOUT;
            status = -1;
        }
    }

    if(status < 0){
        close (socket)

        if(errno == ECONNREFUSED)
            return ERROR_MUCONNREFUSED;
        else
            return ERROR_MUCONNECT;

    }
    else{
        flags = fcntl(socket, F_GETFL, 0);

        if(flags != -1){
            fcntl(socket, F_SETFL, flags& ~O_NONBLOCK);
        }
    }

    return MUOK;
}
```

5.12.10. Interface 接口

函数中主要使用了套接口函数 socket,connect 还使用了一些 I/O 操作函数，例如：select, fcntl 等；

5.12.11.Malloc 内存分配

函数中为了解析域名，使用了全局变量，在使用结束后，由函数本身释放该空间；

5.12.12.Restrcit 限制

略

5.12.13.Test 测试

略

5.12.14.Unsolve 未解决情况

略

5.13.Mu_OpensslConnect

5.13.1.Name 函数名称

int Mu_OpensslConnect(int socket)

5.13.2. Description 函数描述

使用 Openssl 库提供的加密方法，在套接口 socket 上连接服务器；

5.13.3.Function 功能

连接服务器

5.13.4.Capability 性能

略

5.13.5. Input 输入

套接口描述符

5.13.6.Output 输出

同5.12.6.中描述

5.13.7.Arithmetic 算法

略

5.13.8.Process 处理流程

略

5.13.9.Pseudocode 伪代码

```
int Mu_OpensslConnect(int socket)
{
    SSL *conn;
    int i = 0;

    if(ssl_ctx == NULL)
        goto error;

    conn = SSL_new(ssl_ctx);
    if(NULL == conn)
        goto error;

    if(!SSL_set_fd(conn, socket))
        goto error;

    SSL_set_connect_state(conn);
    if(SSL_connect(conn) < =0
        ||conn->state != SSL_ST_OK)
        goto error;


    //record to MuSslTab
    while(i<MAX_THD && MuSslTab[i])
        i ++;
    if(i == MAX_THD)
        goto error;

    MuSslTab[i].fd = socket;
    MuSslTab[i].ssl = conn;

    return MUOK;
```

```
error:
    do error;
    return MUESSL;
}
```

5.13.10.Interface 接口

使用了 Openssl 提供的库函数实现安全边接；

5.13.11.Malloc 内存分配

函数 SSL_new 在获得 SSL 对像时，会分配内存空间以存储该结构，我们在断开该安全连接时，我们必须手动释放该内存空间；

也即释放 MuSslTab 数组中的成员的 SSL 字段；

5.13.12.Restrict 限制

略

5.13.13.Test

略

5.13.14.Unsolve 未解决情况

略

5.14.Mu_Peeker

同 Mu_Reader；
提供函数
int Mu_Peeker(int fd, char *buffer, int len);

5.15. Mu_SocketPeek

同 Mu_SocketRead
提供函数
int Mu_SocketPeek(int fd, char *buffer, int len)

5.16. Mu_OpensslPeek

同 Mu_OpensslRead
提供函数
int Mu_OpensslPeek(int fd, char *buffer, int len)

5.17.Mu_Closer

同 Mu_Reader；
提供函数
int Mu_Closer(int fd);

5.18. Mu_SocketClose

同 Mu_SocketRead
提供函数
int Mu_SocketClose(int fd)

5.19. Mu_OpensslClose

同 Mu_OpensslRead
提供函数

```
int Mu_OpensslClose(int fd)
{
    SSL *ssl = NULL;
    int i = 0;

    Mu_FindSsl(ssl, fd)
    if(NULL == ssl)
        goto error;

    SSL_shutdown(ssl);
```

```
    SSL_free(ssl);

    while(i < MAX_THD && MuSslTab[i] == fd)
        i ++;

    if(i == MAX_THD)
        goto error;

    MuSslTab[i].fd = -1;
    MuSslTab[i].ssl = NULL;

    return MUOK;

error:
    return MUESSL;
}
```