一个模拟的负载均衡系统的实现

软件设计说明书

项目负责人：XXX

编写：	XXX	日期：	2013 年 7 月 14 日
校对：	XXX	日期：	2013 年 7 月 14 日
审核：		日期：	
批准：		日期：	

文档版本：V2.0

2013 年 8 月 3 日

文件修改记录

修改日期	版本	修改页码、章节	修改描述	作者
2013年7月14日	1.0		撰写	╳╳╳
2013年8月3日	2.0		补充	╳╳╳

目　录

软件设计说明书

关键词：NTP 协议

摘　要：一个模拟的负载均衡系统的实现

缩略语说明：

缩略语	英文全名	中文解释
NTP	Network Time Protocol	网络时间协议

参考资料：

RFC-1305 :　　David L. Mills, March 1992

一、 目的

本文档的目旨在推动软件工程的规范化，使设计人员遵循统一的详细设计书写规范。节省制作文档的时间、降低系统实现的风险，做到系统设计资料的规范性与全面性，以利于系统的实现、测试、维护、版本升级等。

本文档作者在查阅了 RFC 文档后，将 NTP(Network Time Protocol)协议的内容整理为本系统的内部标准，将 t_msg 的内容变为包含 ID 和 NTP 数据的 Packet，简化了开发人员理解 NTP 协议和阅读 RFC 文档的时间。

二、 代码框架描述

目录名称	目录说明	包含源文件列表
Time inquiry	项目总目录	defines.h,struct.h,global.h,global.c
Circular	轮转 DLL 目录	Circular.c
client	客户端目录	client.c
Fast	最快响应 DLL 目录	Fast.c
LB	负载均衡服务器目录	Errorlevel.h,function.h,LB.h,luah.h, Balance.c,downlink.c,errorlevel.c,heartbeat.c,LB.c, Loadfile.c,uplink.c,pool.cpp,Keep.cpp
Prorate	按比例分发 DLL 目录	Prorate.c
server	服务器目录	server.c
Release	Release 版本可执行文件	
Debug	Debug 版本可执行文件	

2.1 目录结构

```
Time inquiry                                    项目总目录
  .........\Circular                            轮转 DLL 目录
  .........\........\Debug                      Debug 目录
  .........\........\.....\Circular.exp
  .........\........\.....\Circular.lib
  .........\........\Release                    Release 目录
  .........\........\.......\Circular.exp
  .........\........\.......\Circular.lib
  .........\........\Circular.c                 Circular.c 源码
  .........\........\Circular.dsp               Circular 项目文件
  .........\........\Circular.plg
  .........\Fast                                最快响应 DLL 目录
  .........\....\Debug                          Debug 目录
  .........\....\.....\Fast.exp
  .........\....\.....\Fast.lib
  .........\....\Release                        Release 目录
  .........\....\.......\Fast.exp
  .........\....\.......\Fast.lib
  .........\....\Fast.c                         Fast.c 源码
  .........\....\Fast.dsp                       Fast 项目文件
  .........\....\Fast.plg
```

```
.........\Prorate                          按比例分发 DLL 目录
.........\.......\Debug                    Debug 目录
.........\.......\.....\Prorate.exp
.........\.......\.....\Prorate.lib
.........\.......\Release                  Release 目录
.........\.......\.......\Prorate.exp
.........\.......\.......\Prorate.lib
.........\.......\Prorate.c                Prorate.c 源码
.........\.......\Prorate.dsp              Prorate 项目文件
.........\.......\Prorate.plg
.........\client                           客户端目录
.........\......\Debug                     Debug 目录
.........\......\Release                   Release 目录
.........\......\client.c                  client.c 源码
.........\......\client.dsp                client 项目文件
.........\......\client.plg
.........\LB                               负载均衡服务器目录
.........\..\Debug                         Debug 目录
.........\..\Release                       Release 目录
.........\..\Lua                           Lua 库
.........\..\...\lauxlib.h
.........\..\...\lua.h
.........\..\...\luaconf.h
.........\..\...\lualib.h
.........\..\...\lua52.lib
.........\..\LB.h
.........\..\errorlevel.h
.........\..\function.h
.........\..\luah.h
.........\..\LB.c
.........\..\loadfile.c
.........\..\uplink.c
.........\..\downlink.c
.........\..\Balance.c
.........\..\errorlevel.c
.........\..\heartbeat.c
.........\..\pool.cpp
.........\..\Keep.cpp
.........\..\LB.dsp                        LB 项目文件
.........\..\LB.plg
.........\server                           服务端目录
.........\......\Debug                     Debug 目录
.........\......\Release                   Release 目录
.........\......\server.c                  server.c 源码
.........\......\server.dsp                server 项目文件
.........\......\server.plg
.........\Debug                            Debug 版本可执行文件
.........\.....\MSVCRTD.DLL                Debug 运行库
.........\.....\Circular.dll
.........\.....\Fast.dll
.........\.....\Prorate.dll
.........\.....\client.exe
.........\.....\LB.exe
.........\.....\server.exe
.........\.....\LB.cfg                     LB 配置文件
.........\.....\LB.log                     LB 日志文件
```

```
.........\Release                                  Release 版本可执行文件
.........\.......\Circular.dll
.........\.......\Fast.dll
.........\.......\Prorate.dll
.........\.......\client.exe
.........\.......\LB.exe
.........\.......\server.exe
.........\.......\LB.cfg                           LB 配置文件
.........\.......\LB.log                           LB 日志文件
.........\Readme.txt                               Readme.txt
.........\defines.h                                公共头文件
.........\struct.h
.........\global.h
.........\global.c
.........\Time inquiry.dsw
.........\Time inquiry.ncb
.........\Time inquiry.opt
.........\题目要求.txt                             题目要求.txt
.........\Clean_up.bat
.........\运行_client(显示窗口).bat
.........\运行_client(最小化窗口).vbs
.........\运行_LB(显示窗口).bat
.........\运行_server(显示窗口).bat
.........\运行_server(最小化窗口).vbs
.........\RFC-1305.pdf                             RFC 文档
.........\软件设计说明书.pdf                       软件设计说明书
.........\用户使用说明书.pdf                       用户使用说明书
```

2.2 源文件说明

源文件名称	文件描述
defines.h	公共宏定义
struct.h	公共结构体定义
global.h	
global.c	包含 Client、LB、Server 共享的函数
	数据包显示函数
	网络初始化函数
	显示初始化函数
	窗口事件处理函数
	键盘监听线程函数
Circular.c	轮转算法
Fast.c	最快响应算法
Prorate.c	按比例分发算法
client.c	包含 Client 端发送以及接收数据包的函数
	构造数据包的函数
	发送、接收数据包的函数
server.c	包含 Server 端发送以及接收数据包的函数，和发送心跳包的函数
	构造数据包的函数
	发送、接收数据包的函数
	发送心跳包的函数
luah.h	Lua 库引入
errorlevel.h	
errorlevel.c	出错信息统一显示函数

loadfile.c	配置文件读取模块
	打印配置信息
	加载负载均衡 DLL
Balance.c	内置负载均衡模块
	轮转算法
	最快响应算法
	按比例分发算法
heartbeat.c	服务端健康检测模块
uplink.c	上行(Client to Server)数据处理模块
downlink.c	下行(Server to Client)数据处理模块
function.h	
LB.h	
LB.c	LB 端的主文件
	初始化模块(信号量、互斥锁)
pool.cpp	包含 STL 链表的定义，及其相关的操作函数
	会话信息维护模块(插入、查找、删除)
Keep.cpp	包含 STL 集合的定义，及其相关的操作函数
	会话保持模块(插入、查找、删除)

2.3 配置文件说明

```
-- LB [config file]

-- LB 的 ID
LB_id = 1

-- 开放给 Client 的端口（0-65535）
client_udp_port = 123

-- 与 Server 通信的端口（0-65535）
server_udp_port = 1230

-- 负载均衡 DLL 文件路径
-- 三种方式："Circular.dll"、"Prorate.dll"、"Fast.dll"
-- DLL 加载失败时使用内置轮转算法
Balance = "Circular.dll"

-- 会话保持方式
-- 两种方式："src_id"、"usr_id"
-- 设置为其他字符串可关闭此功能
Keep = "off"

-- 以下重复指明每一个 Server 的 ID、IP、通信端口、权重
-- 增减服务器时添加/删除一项或多项服务器信息即可
-- 服务器信息是由大括号括起的一组 ID、IP、通信端口、权重
servers = {
    {
        id     = 5000,
        ip     = "127.0.0.1",
        port   = 5000,
        weight = 3,
    },
```

```
    {
        id   = 5001,
        ip   = "127.0.0.1",
        port = 5001,
        weight = 4,
    },
    {
        id   = 5002,
        ip   = "127.0.0.1",
        port = 5002,
        weight = 5,
    },
}
```

client.exe 的参数说明：
依次为：[client_id] [usr_id] [LB 的 ID] [发送数量] [LB 的 IP] [LB 的端口号]

server.exe 的参数说明：
依次为：[server_id] [LB 的 ID] [LB 与服务端通信的端口号]

三、 数据结构

3.1 数据结构定义

```
struct ID {
    unsigned int src_id;    /* 消息发送方的id */
    unsigned int dst_id;    /* 消息接收方的id */
    unsigned int usr_id;    /* 发送“时间请求”时填写，回复时其值要与请求值一致
*/
    unsigned int msg_tp;    /* 消息类型： 时间请求/应答，心跳请求/应答 */
};
```

```
struct ntptime {                          /* NTP 64位UTC时间*/
    unsigned int coarse;                  /* 精确到秒*/
    unsigned int fine;                    /* 精确到秒以下*/
};
struct NTP_DATA{                        /* 定义NTP数据结构*/
    unsigned char  Flags;              /* LI(2bit),VERSION(3bit),MODE(3bit)
*/
    unsigned char  Peer_Clock_Stratum;     /* 距离UTC源的远近*/
             char  Peer_Polling_Interval;  /* */
             char  Peer_Clock_Precision;   /* */
             long  Root_Delay;             /* */
    unsigned long  Root_Dispersion;        /* */
    unsigned long  Reference_Identifier;   /* 参考源*/
    struct ntptime Reference_Timestamp;    /* 本地上一次被校准的时间*/
    struct ntptime Origin_Timestamp;       /* request离开Client的时间*/
    struct ntptime Receive_Timestamp;      /* request到达Server的时间*/
    struct ntptime Transmit_Timestamp;     /*   reply离开Server的时间*/
};
```

```
struct Packet {                           /* 完整数据包*/
    struct ID      id;                    /* ID相关数据 (16 bytes) */
    struct NTP_DATA ntpd;                 /* NTP协议数据(48 bytes) */
};
```

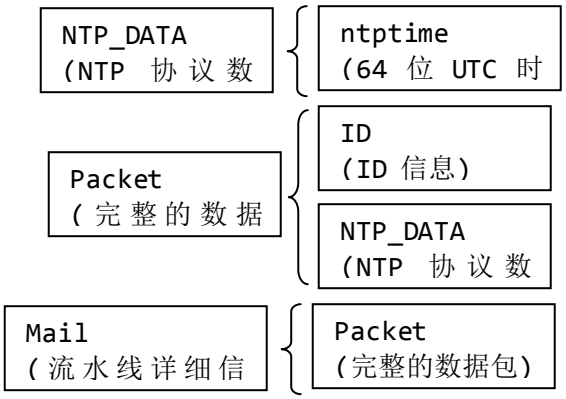
```
struct Server {                           /* Server Info */
    unsigned int         ser_id;          /* Server ID */
    HANDLE              sThread;          /* 暂时没用*/
    unsigned short       busy;            /* Server繁忙程度 */
    struct   sockaddr_in ser;             /* Server 地址*/
};
```

```
struct Stat {                             /* 统计信息 */
    unsigned int sent;
    unsigned int correct;
    unsigned int wrong;
};
```

```
struct Mail{                            /* 数据包在处理流水线上的完整信息 */
    struct   Packet      pkt;           /* 待处理的数据包 */
    struct   sockaddr_in src;           /* 从哪来 */
    struct   sockaddr_in dst;           /* 到哪去 */
    volatile int         stat;          /* 进行到哪了 */
};
```

```
struct Chat {                          /* 会话信息*/
    unsigned int        usr_id;
    unsigned int        cli_id;
    unsigned int        ser_id;
    unsigned int        Timeout;   /* 会话超时 */
    struct   sockaddr_in client;
    struct   sockaddr_in server;
};
```

3.2 数据结构关系图



3.3 公共头文件定义

```
global.h 主要函数列表:
void displayPacket(struct Packet *pkt, struct sockaddr_in* ser, int rs)
功能: 显示 Packet 的详细信息
int NetworkInit(SOCKET *s)
功能: 初始化 SOCKET
int BindPort(SOCKET *s, struct sockaddr_in *saddr)
功能: 绑定端口
int StartInit()
功能: 初始化屏幕显示
int cls(HANDLE hOut, CONSOLE_SCREEN_BUFFER_INFO *bInfo, unsigned int r)
功能: 清屏
DWORD WINAPI MoreInfo(void *display_stat)
功能: 按下 Alt+S 显示统计信息, 按下 Alt+D 显示每个数据包的详细信息
BOOL WINAPI ConsoleHandler(DWORD Event)
功能: 控制台事件处理回调函数
```

3.3.1 displayPacket

【功　能】	显示数据包
【参　数】	struct Packet *pkt : 指向要显示的Packet struct sockaddr_in* ser : 与Packet相关的地址信息 int rs : Sendto(1)、Receivefrom(0)
【返回值】	void
【算　法】	无
【使用说明】	无

3.3.2 NetworkInit

【功　能】	初始化 SOCKET
【参　数】	SOCKET *s
【返回值】	0 ： 正常 ErrorCode
【使用说明】	无

3.3.3 BindPort

【功　能】	绑定端口
【参　数】	SOCKET *s　　: SOCKET struct sockaddr_in *saddr : 地址信息
【返回值】	0 ： 正常 1 ： 绑定失败
【使用说明】	无

3.3.4 StartInit

【功　能】	初始化屏幕显示
【参　数】	无
【返回值】	0 ： 成功 1 ： 显示临界区初始化失败
【使用说明】	无

3.3.5 cls

【功　能】	清屏
【参　数】	HANDLE hOut　: 标准输出句柄 CONSOLE_SCREEN_BUFFER_INFO *bInfo : 显示缓冲区信息 unsigned int r : 从第几行开始清屏
【返回值】	大于 0 的整数
【使用说明】	无

3.3.6 MoreInfo

【功　能】	按下 Alt+S 显示统计信息，按下 Alt+D 显示每个数据包的详细信息
【参　数】	void *display_stat : 显示统计信息的函数指针
【返回值】	无
【算　法】	循环接收键盘输入
【使用说明】	无

3.3.7 ConsoleHandler

【功　能】	控制台事件处理回调函数
【参　数】	DWORD Event : 控制台窗口事件
【返回值】	FALSE
【使用说明】	无

四、 客户端设计说明

4.1 数据结构

```
struct Packet {                                 /* 完整数据包*/
    struct ID       id;                         /* ID相关数据 (16 bytes) */
    struct NTP_DATA ntpd;                        /* NTP协议数据(48 bytes) */
};
struct Stat {                                   /* 统计信息*/
    unsigned int sent;
    unsigned int correct;
    unsigned int wrong;
};
```

4.2 处理流程详细说明

在 main 函数中初始化后开始构造并发送数据包，在另一个线程中接收数据包。

4.3 编码设计

主要函数列表如下：

主要函数名称	函数简要说明
resource.h	
int StartInit	初始化屏幕显示
int NetworkInit	初始化 SOCKET
void displayPacket	显示 Packet 的详细信息
client.c	
void buildPacket	按照 NTP 协议的标准，构造标准的时间查询数据包，并在头部填写 ID 相关信息
DWORD WINAPI Recvpacket	接收数据包
int main	初始化、发送数据包、显示统计信息

4.3.1 buildPacket

【功　能】	按照 NTP 协议的标准，构造标准的时间查询数据包，并在头部填写 ID 信息
【参　数】	struct Packet *pkt　: 被填写的数据包指针 unsigned int msg_tp : 请求类型,其值只能为0(时间请求)
【返回值】	void
【算　法】	
【使用说明】	无

4.3.2 Recvpacket

【功　能】	接收数据包
【参　数】	void *s : SOCKET*类型的指针
【返回值】	无
【算　法】	循环接收数据包
【使用说明】	无

4.3.3 main

【功　能】	初始化、发送数据包、显示统计信息
【参　数】	argc　: 7 argv[1] : Client ID argv[2] : usr_id argv[3] : LB ID argv[4] : 发送数量 argv[5] : 目标IP argv[6] : 目标端口
【返回值】	0　: 正常退出 -1　: 网络初始化失败 -255: 参数错误
【算　法】	
【使用说明】	参数顺序不能错

五、 服务端设计说明

5.1 数据结构

```
struct Packet {                                 /* 完整数据包*/
    struct ID       id;                         /* ID相关数据 (16 bytes) */
    struct NTP_DATA ntpd;                        /* NTP协议数据(48 bytes) */
};
struct Stat {                                   /* 统计信息*/
    unsigned int sent;
    unsigned int correct;
    unsigned int wrong;
};
```

心跳包的数据结构与 Packet 相同。

5.2 处理流程详细说明

在 main 函数中初始化后开始接收并构造和返回数据包。
在 MoreInfo 线程中侦听键盘的 Alt+S 和 Alt+D 事件，分别对应显示统计信息和详细信息。
在 Heartbeat 线程中，发送心跳包

5.3 编码设计

主要函数列表如下：

主要函数名称	函数简要说明
resource.h	
int StartInit	初始化屏幕显示
int NetworkInit	初始化 SOCKET
int BindPort	绑定端口
void displayPacket	显示 Packet 的详细信息
int cls	清屏
DWORD WINAPI MoreInfo	按下 Alt+S 显示统计信息，按下 Alt+D 显示详细信息
server.c	
void buildPacket	按照 NTP 协议的标准，构造标准的时间应答数据包，并在头部填写 ID 相关信息
void buildheart	构造心跳应答数据包
DWORD WINAPI Heartbeat	周期发送心跳应答数据包
int main	初始化、接收、发送数据包

5.3.1 buildPacket

【功　能】	按照 NTP 协议的标准，构造标准的时间应答数据包，并在头部填写 ID 信息
【参　数】	struct Packet *pkt　：被填写的数据包指针 unsigned int msg_tp ：请求类型,其值只能为1(时间应答)
【返回值】	void
【算　法】	
【使用说明】	无

5.3.2 buildheart

【功　能】	构造心跳应答数据包
【参　数】	struct Packet *hpk　: 被填写的数据包指针 unsigned int msg_tp : 请求类型,其值只能 3(心跳应答)
【返回值】	void
【算　法】	循环接收数据包
【使用说明】	无

5.3.3 Heartbeat

【功　能】	周期发送心跳应答数据包
【参　数】	void *s : SOCKET*类型的指针
【返回值】	无
【算　法】	循环发送心跳包
【使用说明】	无

5.3.4 main

【功　能】	初始化、接收、发送数据包
【参　数】	argc　: 4 argv[1] : Server ID argv[2] : LB ID argv[3] : 绑定的本地端口
【返回值】	0　: 正常退出 -1　: 网络初始化失败 -2　: 绑定端口失败 -255: 参数错误
【算　法】	
【使用说明】	参数顺序不能错

六、 LB 端设计说明

6.1 数据结构

```
struct Packet {                                /* 完整数据包*/
    struct ID       id;                        /* ID相关数据 (16 bytes) */
    struct NTP_DATA ntpd;                       /* NTP协议数据(48 bytes) */
};
struct Server {                                /* Server Info */
    unsigned int         ser_id;               /* Server ID */
    HANDLE               sThread;              /* 暂时没用*/
    unsigned short       busy;                 /* Server繁忙程度*/
    struct   sockaddr_in ser;                  /* Server 地址*/
};
struct Stat {                                  /* 统计信息*/
    unsigned int sent;
    unsigned int correct;
    unsigned int wrong;
};
struct Mail{                                   /* 数据包在处理流水线上的完整信息*/
    struct   Packet      pkt;                  /* 待处理的数据包*/
    struct   sockaddr_in src;                  /* 从哪来*/
    struct   sockaddr_in dst;                  /* 到哪去*/
    volatile int         stat                  /* 进行到哪了*/
};
struct Chat {                             /* 会话信息*/
    unsigned int         usr_id;
    unsigned int         cli_id;
    unsigned int         ser_id;
    unsigned int         Timeout;    /* 会话超时*/
    struct   sockaddr_in client;
    struct   sockaddr_in server;
};
```

6.2 处理流程详细说明

在 main 函数中初始化后开始监听子线程状态，直到进程结束。
在 MoreInfo 线程中侦听键盘的 Alt+S 和 Alt+D 事件，分别对应显示统计信息和详细信息。通过 ConsoleHandler 回调函数监视窗口事件。
在 ClientRecever 线程中接收来自 Client 端的数据包，只做简单的验证就将数据包写入上行处理流水线。
在 C2S_rebuilder 线程中处理 ClientRecever 放入缓冲区的数据包，并通过负载均衡模块与会话保持模块选择要发送到的 Server 端。
在 ServerSender 线程中负责将上行缓冲区中被 C2S_rebuilder 处理好的数据包发送给 Server。
在 ServerRecever 线程中接收来自 Server 端的数据包，只做简单的验证就将数据包写入下行处理流水线。
在 S2C_rebuilder 线程中处理 ServerRecever 放入缓冲区的数据包，并定位要发送给的 Client 端。

在 ClientSender 线程中负责将下行缓冲区中被 S2C_rebuilder 处理好的数据包发送给 Client 端。
会话缓冲区中保存着每一个数据包所对应的 Client 和 Server 等会话信息和超时信息，以供下行数据定向发送所用。
Heartbeat 线程每隔 0.5 秒向每一个服务端发送心跳请求数据包。
如果连续 4 次收不到某个服务端的心跳响应，就认为服务端出现了故障。后续对时间请求消息做负载均衡时，就不再分发给此服务端处理。
在 CKCleaner 线程中周期性清理会话缓冲区中保存的会话信息以及会话保持信息。

6.3 编码设计

主要函数列表如下：

主要函数名称	函数简要说明
global.c	
int StartInit	初始化屏幕显示
int NetworkInit	初始化 SOCKET
int BindPort	绑定端口
void displayPacket	显示 Packet 的详细信息
int cls	清屏
DWORD WINAPI MoreInfo	按下 Alt+S 显示统计信息，按下 Alt+D 显示详细信息
BOOL WINAPI ConsoleHandler	控制台事件处理回调函数
LB.c	
int syncInit	初始化信号量、互斥锁，初始化上、下行数据缓冲区
int main	初始化、监视子线程工作状况
Balance.c	
void Circular	轮转算法
void Fast	最快响应算法
void Prorate	按比例分发算法
errorlevel.c	
void PRINT	出错信息统一显示函数
heartbeat.c	
int PickPosition	根据 ser_id 查找在服务器缓冲区中的位置
void HealthMon	从会话缓冲区中定位与之匹配的 Client 地址
DWORD WINAPI Heartbeat	每隔 0.5 秒向每个服务端发送心跳请求，并监视延迟
loadfile.c	
int loadconfig	根读配置文件
void showconfig	打印配置信息
int LoadDllFunc	载入 DLL 文件并获取负载均衡函数地址
uplink.c	
DWORD WINAPI ClientRecever	只负责从 Client 接收数据包(时间请求)的线程函数
DWORD WINAPI C2S_rebuilder	C2S_rebuilder 线程函数，负责将 ClientRecever 线程处理过的数据包修改 dst_id，选择目标 Server，并保存 Client 的地址信息
DWORD WINAPI ServerSender	负责向 Server 发送数据包的线程函数
downlink.c	
DWORD WINAPI ServerRecever	只负责从 Server 接收数据包(时间应答、心跳包)的线程函数
DWORD WINAPI S2C_rebuilder	S2C_rebuilder 线程函数，负责将 ServerRecever 线程处理过的数据包修改 src_id，并定位与之匹配的 Client 地址
DWORD WINAPI ClientSender	只负责向 Client 发送数据包的线程函数

Keep.cpp	
int Select	查找会话保持信息
int Insert	插入会话保持信息
void Delete	删除会话保持信息中的超时项
pool.cpp	
int Alloc_Conversation	会话缓冲区初始化
int find_c	从会话缓冲区中定位与之匹配的 Client 地址
DWORD WINAPI CKCleaner	负责清除会话缓冲区中的超时信息

6.3.1 synclnit

【功　能】	初始化信号量、互斥锁，初始化上、下行数据缓冲区
【参　数】	无
【返回值】	0　：初始化成功 -1　：初始化失败
【算　法】	无
【使用说明】	无

6.3.2 loadconfig

【功　能】	读取配置文件 LB.cfg，初始化启动变量
【参　数】	struct Server **psInfo：服务器列表
【返回值】	读取成功:返回值为0 读取失败:返回非零值
【算　法】	无
【使用说明】	配置文件内容的顺序不能改变

6.3.3 LoadDllFunc

【功　能】	载入 DLL 文件并获取负载均衡函数地址
【参　数】	const char * dll ：DLL文件
【返回值】	读取成功:返回值为0 读取失败:返回非零值
【算　法】	无
【使用说明】	无

6.3.4 Alloc_Conversation

【功　能】	会话缓冲区初始化
【参　数】	unsigned int MAX_Ser：服务器数量
【返回值】	初始化成功返回 0
【算　法】	无
【使用说明】	无

6.3.5 main

【功　能】	初始化、监视子线程工作状况
【参　数】	无
【返回值】	0　：正常退出 1　：创建日志文件失败

	2 ：安装事件回调函数失败 3 ：读取日志文件失败 4 ：载入DLL失败 5 ：信号量初始化失败或缓冲区内存分配失败 6 ：客户端通信初始化失败 7 ：客户端通信端口绑定失败 8 ：服务端通信初始化失败 9 ：服务端通信端口绑定失败 <0 ：线程异常退出
【算　法】	
【使用说明】	无

七、 负载均衡模块

7.1 数据接口

主程序向负载均衡模块传递的信息有：服务端数量、服务端信息缓冲区地址、数据处理缓冲区地址、服务端编号指针。

负载均衡模块将服务端编号指针所指的 DWORD 数据修改为算法返回的处理结果。

7.2 函数原型

```
__declspec(dllexport) void __stdcall Balance(unsigned int *pre)
__declspec(dllexport) void __stdcall setsInfo(unsigned int max,
                                              struct Server *p,
                                              unsigned int *m)
```

7.3 编码设计

7.3.1 Balance

【功　能】	初始化信号量、互斥锁，初始化上、下行数据缓冲区
【参　数】	unsigned int *pre ：选择的服务器位置
【返回值】	无
【算　法】	无
【使用说明】	函数名及函数声明不能改变

7.3.2 setsInfo

【功　能】	读取配置文件 LB.cfg，初始化启动变量
【参　数】	unsigned int max ：服务器数量 struct Server *p ：服务器列表 unsigned int *m ：数据处理缓冲区
【返回值】	无
【算　法】	无
【使用说明】	函数名及函数声明不能改变

八、 附录

8.1 RFC-1305

Network Time Protocol (Version 3)

Specification, Implementation and Analysis

Abstract

This document describes the Network Time Protocol (NTP), specifies its formal structure and summarizes information useful for its implementation. NTP provides the mechanisms to synchronize time and coordinate time distribution in a large, diverse internet operating at rates from mundane to lightwave. It uses a returnable-time design in which a distributed subnet of time servers operating in a self-organizing, hierarchical-master-slave configuration synchronizes local clocks within the subnet and to national time standards via wire or radio. The servers can also redistribute reference time via local routing algorithms and time daemons.

Network Time Protocol

This section consists of a formal definition of the Network Time Protocol, including its data formats, entities, state variables, events and event-processing procedures. The specification is based on the implementation model illustrated in Figure 1, but it is not intended that this model is the only one upon which a specification can be based. In particular, the specification is intended to illustrate and clarify the intrinsic operations of NTP, as well as to serve as a foundation for a more rigorous, comprehensive and verifiable specification

Data Formats

All mathematical operations expressed or implied herein are in two's-complement, fixed-point arithmetic. Data are specified as integer or fixed-point quantities, with bits numbered in big-endian fashion from zero starting at the left, or high-order, position. Since various implementations may scale externally derived quantities for internal use, neither the precision nor decimal-point placement for fixed-point quantities is specified. Unless specified otherwise, all quantities are unsigned and may occupy the full field width with an implied zero preceding bit zero. Hardware and software packages designed to work with signed quantities will thus yield surprising results when the most significant (sign) bit is set. It is suggested that externally derived, unsigned fixed-point quantities such as timestamps be shifted right one bit for internal use, since the precision represented by the full field width is seldom justified.

Since NTP timestamps are cherished data and, in fact, represent the main product of the protocol, a special timestamp format has been established. NTP timestamps are represented as a 64-bit unsigned fixed-point number, in seconds relative to 0h on 1 January 1900. The integer part is in the first 32 bits and the fraction part in the last 32 bits. This format allows convenient multiple-precision arithmetic and conversion to Time Protocol representation (seconds), but does complicate the conversion to ICMP Timestamp message representation (milliseconds). The precision of this representation is about 200 picoseconds, which should be adequate for even the most exotic requirements.

Timestamps are determined by copying the current value of the local clock to a timestamp when some significant event, such as the arrival of a message, occurs. In order to maintain the highest accuracy, it is important that this be done as close to the hardware or software driver associated with the event as possible. In particular, departure timestamps should be redetermined for each link-level retransmission. In some cases a particular timestamp may not be available, such as when the host is rebooted or the protocol first starts up. In these cases the 64-bit field is set to zero, indicating the value is invalid or undefined.

Note that since some time in 1968 the most significant bit (bit 0 of the integer part) has been set and that the 64-bit field will overflow some time in 2036. Should NTP be in use in 2036, some external means will be necessary to qualify time relative to 1900 and time relative to 2036 (and other multiples of 136 years). Timestamped data requiring such qualification will be so precious that appropriate means should be readily available. There will exist an 200-picosecond interval, henceforth ignored, every 136 years when the 64-bit field will be zero and thus considered invalid.

Common Variables

The following variables are common to two or more of the system, peer and packet classes. Additional variables are specific to the optional authentication mechanism as described in Appendix C. When necessary to distinguish between common variables of the same name, the variable identifier will be used.

Peer Address (peer.peeraddr, pkt.peeraddr), Peer Port (peer.peerport, pkt.peerport): These are the 32-bit Internet address and 16-bit port number of the peer.

Host Address (peer.hostaddr, pkt.hostaddr), Host Port (peer.hostport, pkt.hostport): These are the 32-bit Internet address and 16-bit port number of the host. They are included among the state variables to support multi-homing.

Leap Indicator (sys.leap, peer.leap, pkt.leap): This is a two-bit code warning of an impending leap second to be inserted in the NTP timescale. The bits are set before 23:59 on the day of insertion and reset after 00:00 on the following day. This causes the number of seconds (rollover interval) in the day of insertion to be increased or decreased by one. In the case of primary servers the bits are set by operator intervention, while in the case of secondary servers the bits are set by the protocol. The two bits, bit 0 and bit 1, respectively, are coded as follows:

00 no warning
01 last minute has 61 seconds
10 last minute has 59 seconds
11 alarm condition (clock not synchronized)

In all except the alarm condition (112), NTP itself does nothing with these bits, except pass them on to the time-conversion routines that are not part of NTP. The alarm condition occurs when, for whatever reason, the local clock is not synchronized, such as when first coming up or after an extended period when no primary reference source is available.

Mode (peer.mode, pkt.mode): This is an integer indicating the association mode, with values coded as follows:

0 unspecified
1 symmetric active
2 symmetric passive
3 client
4 server
5 broadcast
6 reserved for NTP control messages
7 reserved for private use

Stratum (sys.stratum, peer.stratum, pkt.stratum): This is an integer indicating the stratum of the local clock, with values defined as follows:

0 unspecified
1 primary reference (e.g., calibrated atomic clock, radio clock)
2-255 secondary reference (via NTP)

For comparison purposes a value of zero is considered greater than any other value. Note that the maximum value of the integer encoded as a packet variable is limited by the parameter NTP.MAXSTRATUM.

Poll Interval (sys.poll, peer.hostpoll, peer.peerpoll, pkt.poll): This is a signed integer indicating the minimum interval between transmitted messages, in seconds as a power of two. For instance, a value of six indicates a minimum interval of 64 seconds.

Precision (sys.precision, peer.precision, pkt.precision): This is a signed integer indicating the precision of the various clocks, in seconds to the nearest power of two. The value must be

rounded to the next larger power of two; for instance, a 50-Hz (20 ms) or 60-Hz (16.67 ms) power-frequency clock would be assigned the value -5 (31.25 ms), while a 1000-Hz (1 ms) crystal-controlled clock would be assigned the value -9 (1.95 ms).

Root Delay (sys.rootdelay, peer.rootdelay, pkt.rootdelay): This is a signed fixed-point number indicating the total roundtrip delay to the primary reference source at the root of the synchronization subnet, in seconds. Note that this variable can take on both positive and negative values, depending on clock precision and skew.

Root Dispersion (sys.rootdispersion, peer.rootdispersion, pkt.rootdispersion): This is a signed fixed-point number indicating the maximum error relative to the primary reference source at the root of the synchronization subnet, in seconds. Only positive values greater than zero are possible.

Reference Clock Identifier (sys.refid, peer.refid, pkt.refid): This is a 32-bit code identifying the particular reference clock. In the case of stratum 0 (unspecified) or stratum 1 (primary reference source), this is a four-octet, left-justified, zero-padded ASCII string, for example (see Appendix A for comprehensive list):

Stratum	Code	Meaning
0	DCN	DCN routing protocol
0	TSP	TSP time protocol
1	ATOM	Atomic clock (calibrated)
1	WWVB	WWVB LF (band 5) radio
1	GEOS	GOES UHF (band 9) satellite
1	WWV	WWV HF (band 7) radio

In the case of stratum 2 and greater (secondary reference) this is the four-octet Internet address of the peer selected for synchronization.

Reference Timestamp (sys.reftime, peer.reftime, pkt.reftime): This is the local time, in timestamp format, when the local clock was last updated. If the local clock has never been synchronized, the value is zero.

Originate Timestamp (peer.org, pkt.org): This is the local time, in timestamp format, at the peer when its latest NTP message was sent. If the peer becomes unreachable the value is set to zero.

Receive Timestamp (peer.rec, pkt.rec): This is the local time, in timestamp format, when the latest NTP message from the peer arrived. If the peer becomes unreachable the value is set to zero.

Transmit Timestamp (peer.xmt, pkt.xmt): This is the local time, in timestamp format, at which the NTP message departed the sender.