LoadRunner Winsocket 协议知识总结

测试中心 田渊文

2007 年 11 月 9 日

目 录

序

初次接触 LoadRunner Windows Sockets 协议是在做 Tuxedo 客户端软加密性能测试的时候，由于从来没有接触过 socket 协议，因此当时碰到了很多的问题，不过过程虽然有点曲折，但在大家的帮助下还是问题都得到了很好的解决。当时就想对一些问题总结一下，不过想想自己了解到的东西还是太少了，等再多了解一些再说吧，免得班门弄斧，贻笑大方。幸运的是因项目需要，很快就又有机会接触到 LR 的 Socket 协议。在后来的测试过程中，确实对 socket 协议的脚本编写有了更深的了解，接触到了更多的函数，当然也遇到了一些新问题，而解决问题的过程本身又是一个深入学习的过程。

最近也有一些时间来学习一些自己想学的东西，便想着能整理一下 LR Socket 的相关内容，一是为自己理清一下思路以备忘（本人记性很不好，呵呵），二是在整理过程中也得到了大家的认可。因此最后决定将原本随便整理一下的想法修正一下，就是希望能系统一点，把所有的函数和曾经遇到过的问题都写出来，在共享的基础上能够共同探讨。但是由于本人在技术上本就是一只菜鸟，再加上 English 也不怎么样，因此花了比预期更多的时间，甚感惭愧。

其实这些内容只能算是铺个底吧，很多细节性的问题还需要大家一块探讨，在实践中进行验证。由于本人水平有限，文中纰漏甚至错误之处在所难免，还望大家不吝指正。

在此向所有在工作中帮助过我的同事们表示衷心的感谢！

修正版说明

由于时间关系，1.0 版中存在很多不足，很多地方解释的不够清楚，本次修正及补充的内容主要包括以下几个方面：

1、对文档中的一些错字、别字进行了校正；
2、对文档中的绝大多数函数进行了较为详细的补充说明及用法；
3、对文档中涉及到的所有脚本实例进行了调试（LoadRunner8.0 下）并全部通过；
4、“常见问题”中增加了一个从文件中读取数据的方法介绍；
5、增加了“知识链接”部分，旨在加深对相关函数的理解。

由于本人水平有限，编写这个文档花了较多的时间，到现在总算可以正式发布了。虽然是修正版，但文档中也难免一些纰漏或错误，特别是对 lrs_length_receive 和 lrs_length_send 两个函数还是没有彻底搞清楚，这也是我最为遗憾之处，对于文档的完整性也是一个缺憾，真诚地希望这方面的高手能给予指点。

其实我编写此文档的初衷是很简单的，就是希望在整理、总结的过程中能够加深对这方面知识的理解和掌握，后来此想法得到了项目组同事们的大力支持，于是就希望能写得完整一点，在此也对他们再次表示衷心地感谢！现在这个文档可以说是基本完成了，如果说要给自己定位的话，我觉得更多的程度上自己只是一个资源整合者，因为很多知识点前辈们都已经总结得很好了，只是没有人去整合、去归纳（也可能是我们没有找到而已），而本文档的编写就是完成了这样一个任务，只是在其中融入了我自己的实践而已。

一、函数

1．基本函数

lrs_accept_connection 接受侦听套接字连接

格式：

int lrs_accept_connection (char *old_socket, char *new_socket);

参数说明：

old_socket: 被侦听的套接字标识符，如 socket0。

new_socket: 侦听到请求时建立的新套接字标识符，如 socket1。

返回值：

成功返回 0。

用法：

lrs_accept_connection(“socket0”,”socket1”);

函数说明：

函数从 old_socket 上的挂起连接队列中取出第一个连接，使用相同属性创建新的套接字。原来的套接字对于其他连接仍然可用。

实际的侦听机制是：当 accept 函数监视的 old_socket 收到连接请求时，old_socket 执行体将建立一个新的 socket 连接（标识符为 new_socket），收到服务请求的初始 socket（即 old_socket）仍可以继续在以前的 socket 上监听，同时可以在新的 socket 描述符上进行数据传输操作。

lrs_close_socket 关闭打开的套接字

格式：

int lrs_close_socket (char *s_desc);

参数说明：

s_desc: 已打开的套接字标识符。

返回值：

成功返回 0，否则返回失败错误码。

用法：

lrs_close_socket(“socket0”);

lrs_create_socket 初始化套接字

格式：

int **lrs_create_socket** (char *s_desc, char *type, [char* *LocalHost*,] [char* *peer,*] [char *backlog,] LrsLastArg);

参数说明：

S_desc: 要初始化的套接字标识符，如 socket0。

type: 套接字类型，有 TCP 和 UDP 两种。

LocalHost: 绑定套接字的本地地址、端口，如 LocalHost=4002，也可以在端口前面加上本机名称或 IP 地址，如 LocalHost=overn:4002，或 LocalHost=168.3.4.127:4002。可选。

peer: 处理套接字请求的远程机端口，如 RemoteHost=168.3.1.230:6666。可选。

backlog: 请求连接队列的最大长度，如 backlog=20，可选。

LrsLastArg: 参数结束标志

返回值：

初始化成功返回 0；如果是 VuG 错误则返回 VuG 错误码，如果是套接字错误则返回 Windows 套接字错误码。错误码详见第三部分。

用法：

lrs_create_socket("socket0", "UDP", "LocalHost=4002", LrsLastArg);

lrs_create_socket("socket0", "TCP", "RemoteHost=168.3.1.230:6666", LrsLastArg);

函数说明：

lrs_create_socket 函数用来初始化一个套接字。该函数通过执行 socket 命令来打开一个新的套接字连接。如果提供 LocalHost 参数它将执行 bind 命令来绑定该套接字；如果指定 peer 参数它将执行 connect 命令和远端主机建立一个连接；如果提供 backlog 参数将会执行 listen 命令来侦听该套接字。Backlog 对队列中等待服务的请求的数目进行了限制，大多数系统缺省值为 20。如果一个服务请求到来时，请求队列已满，该 socket 将拒绝连接请求，客户端将会收到一个出错信息

指定 LocalHost 时可以在端口前加上主机名（如"LocalHost=overn:80"）,如果主机的 IP 地址是绑定的也可以指定为它的 IP 地址，如"LocalHost=168.3.4.127:6666"。

该函数在客户端或服务器端的 Windows Socket 会话期间都会被自动记录，如果是客户端会话则 peer 和 port 两个参数会被记录，如果是服务器端则会记录 port 和 backlog。

lrs_disable_socket 禁用套接字操作

格式：

int lrs_disable_socket (char **s_desc*, int *operation*);

参数说明：

s_desc: 要禁用的套接字标识符。

operation: 禁用的操作类型，有 SEND,RECEIVE,SEND-RECEIVE 三种。

返回值：

成功返回 0，否则返回失败错误码。

用法：

lrs_diable_socket(“socket0”,DISABLE_SEND);

lrs_exclude_socket 重播期间排除套接字

格式：

int lrs_exclude_socket (char **s_desc*);

参数说明：

s_desc: 要排除的套接字标识符

返回值：

用法：

```
lrs_exclude_socket("socket1");
lrs_create_socket("socket0", "TCP", "RemoteHost=168.3.1.230:6666",    LrsLastArg);
lrs_set_recv_timeout2(1,0);
lrs_receive("socket0", "buf0", LrsLastArg);
lrs_send("socket0", "buf1", LrsLastArg);
lrs_receive("socket0", "buf2", LrsLastArg);
/****以下所有 socket1 的操作都将被忽略****/
lrs_create_socket("socket1", "TCP", "RemoteHost=168.3.1.230:6666",    LrsLastArg);
lrs_set_recv_timeout2(1,0);
lrs_receive("socket1", "buf49", LrsLastArg);
lrs_send("socket1", "buf50", LrsLastArg);
lrs_receive("socket1", "buf51", LrsLastArg);
lrs_send("socket1", "buf52", LrsLastArg);
lrs_receive("socket1", "buf53", LrsLastArg);
/****下面 socket2 的操作将正常执行****/
lrs_create_socket("socket2", "TCP", "RemoteHost=168.3.1.230:6666",    LrsLastArg);
lrs_set_recv_timeout2(1,0);
```

```
lrs_receive("socket2", "buf98", LrsLastArg);
lrs_send("socket2", "buf99", LrsLastArg);
lrs_receive("socket2", "buf100", LrsLastArg);
```

lrs_get_socket_attrib 获取套接字属性

格式：

char *lrs_get_socket_attrib (char **s_desc* , int *attribute*);

参数说明：

s_desc: 套接字标识符。

attribute: 指定要获取的属性名，以下为可用的属性列表：

LOCAL_ADDRESS	本地 IP 地址
LOCAL_HOSTNAME	本地主机名
LOCAL_PORT	本地套接字连接端口
REMOTE_ADDRESS	对方的 IP 地址（仅 TCP 连接可用）
REMOTE_HOSTNAME	对方主机名（仅 TCP 连接可用）
REMOTE_PORT	对方套接字端口（仅 TCP 连接可用）

返回值：

该函数以字符串的形式返回获得的属性值；如果没有可用的返回值，则返回 NULL。

用法：

```
char *ReturnValue;
lrs_startup(257);
lrs_create_socket("socket0","TCP","RemoteHost=168.3.1.230:6666",LrsLastArg);
ReturnValue = lrs_get_socket_attrib("socket0",REMOTE_ADDRESS); //取得属性值
lr_output_message("the return value is %s",ReturnValue);
```

输出结果：

```
the return value is 168.3.1.230
```

lrs_get_socket_handler 获取指定套接字的套接字句柄

格式：

int lrs_get_socket_handler (char **s_desc*);

参数说明：

s_desc: 要获取句柄的套接字标识符。

返回值：

成功返回整数形式的套接字句柄，否则返回错误码。

用法：

```
int handler;
lrs_startup(257);
lrs_create_socket("socket0","TCP","RemoteHost=168.3.1.230:6666",LrsLastArg);
/*获取 socket0 的句柄*/
handler = lrs_get_socket_handler("socket0");
/*输出获取到的句柄*/
lr_output_message("the socket0's handler is %d",handler);
```

输出结果：

```
the socket0's handler is 252
```

lrs_length_receive 接收来自指定长度的缓冲区的数据

格式：

int lrs_length_receive(char **socket_descriptor*, char **buffer*, int *location_option*, [char* *locators*], [char* *additional_params*], LrsLastArg);

返回值：

成功返回 0，否则返回错误码。

lrs_length_send 向流套接字发送指定长度的缓冲区数据

格式：

int lrs_length_send(char **socket_descriptor*, char **buffer*, int *location_option*, [char* *locators*], [char* *additional_params*], LrsLastArg);

[以上两个函数很多细节尚不明白，未敢落笔，还望各位不吝赐教]

lrs_receive 接收来自套接字的数据

格式：

int lrs_receive (char **s_desc*, char *bufindex, [char **flags*], LrsLastArg);

参数说明：

s_desc: 套接字标识符。

bufindex: 缓冲区标识符。

flags: 接收/发送数据标记，参看 **Receive and Send Flags**。可选。

返回值：

成功接收返回 0。

用法：

lrs_send("socket0", "buf3", LrsLastArg);

lrs_receive("socket0", "buf4", LrsLastArg);

函数说明：

该函数从数据包或流套接字接收数据。如果没有接收到数据 lrs_receive 会一直等待，除非 socket 状态为 non-blocking。

VuGen 会给出预期的接收长度，如果实际接收的数据长度不等于预期长度，lrs_receive 会重新读取 socket 传入的数据，直到超时为止。lrs_receive 默认的超时时间为 10 秒，你可以通过 lrs_set_recv_timeout 或 lrs_set_recv_timeout2 来指定接收超时时间。

注意：lrs_receive 成功执行不意味返回数据被成功接收。如果接收到的数据和预期值不匹配，VuGen 会给出输出一个 mismatch 的消息，然后继续执行脚本。

若 socket 连接类型为 TCP，如果接收方向的数据传输被关闭，则 lrs_receive 不会返回任何数据。如果连接异常中断则会返回错误代码。

lrs_receive_ex 接收指定长度的数据报或流套接字

格式：

int lrs_receive_ex (char *s_desc, char *bufindex, [char *flags,][char *size,][char *terminator,][char *mismatch,] [char *RecordingSize,]LrsLastArg);

参数说明：

s_desc: 套接字标识符。

bufindex: 缓冲区标识符。

flags: 接收/发送数据标记，参看 **Receive and Send Flags**。可选。

size: 要接收的长度（字节数），格式为" NumberOfBytesToRecv=*xx*"，仅 TCP 连接可用。

terminator: 指定接收结束标记，格式为 "StringTerminator=*value*"(字符串) 或 "BinaryStringTerminator= *value* "（二进制）。仅 TCP 连接可用。

mismatch: 指定匹配标准 (长度或内容)，格式为 "Mismatch= *value*",value 值为 MISMATCH_SIZE (缺省)或 MISMATCH_CONTENT。

RecordingSize: 接收的长度等于预期长度，格式为"RecordingSize"。仅 TCP 连接可用。

返回值：

成功返回 0。

用法：

```
char *RecvBuf;
int Num;
lrs_receive_ex("socket0","buf2","NumberOfBytesToRecv=12",LrsLastArg);
lrs_get_last_received_buffer("socket0",&RecvBuf,&Num);
lrs_save_param_ex("socket0","user",RecvBuf,1,Num,"ascii","NewParam");
lr_output_message("NewParam = %s",lr_eval_string("<NewParam>"));
```

执行结果：

```
vuser_init.c(23): lrs_receive_ex(socket0, buf2)
vuser_init.c(25): lrs_get_last_received_buffer(socket0, buf_p, size_p)
vuser_init.c(26): lrs_save_param_ex(socket0, user, buf_p, 1, 12, ascii, NewParam)
vuser_init.c(27): Param = \x00\x00M\x00\x00\x00\x00\x00\x00\x00\x00\x03
```

函数说明：

函数 lrs_receive_ex 从接收到的数据报或流套接字中读取指定的长度。该函数扩展了 lrs_receive，可以指定接收的数据长度。

VuGen 在录制会话时已经确定了期望的接收长度，如果实际接收到的缓冲区数据长度和期望值不匹配（小于或大于），lrs_receive_ex 将重新读取返回值直到 receive_timeout 超时。lrs_receive_timeout 缺省的超时时间为 10 秒，可以通过函数 lrs_set_recv_timeout 来设定超时时间。

lrs_receive_ex 的成功执行不代表数据全部被成功接收。如果接收的数据和期望值不匹配，VuGen 会报出 Mismatch 的警告信息，然后继续执行后面的脚本。

当 socket 连接类型为 TCP 时，如果接收方向的数据传输被关闭，则 lrs_receive 不会返回任何数据。如果连接被中断则会返回错误代码。

lrs_send 将数据发送到数据报上或流套接字中

格式：

```
int lrs_send ( char *s_desc, char *buf_desc, [char *target], [char *flags,] LrsLastArg );
```

参数说明：

s_desc: 套接字连接标识符。

buf_desc: 发送缓冲区标识符。

target: 发送目标地址，格式为点分十进制 IP 地址或主机名。可选。

flags: 接收/发送数据标记，参看 Receive and Send Flags。可选。

返回值：

成功返回 0，失败返回错误码。

用法一：

```
lrs_create_socket("socket0", "TCP", "RemoteHost=168.3.1.230:6666",  LrsLastArg);
lrs_send("socket0", "buf3", LrsLastArg);
```

用法二：

```
lrs_create_socket("socket0", "UDP", "LocalHost=4002",  LrsLastArg);
lrs_send("socket0", "buf0", "TargetSocket=219.133.41.75:8000", LrsLastArg);
```

函数说明：

该函数用来向已连接的数据报或流套接字发送数据。如果不能成功地将缓冲区中的数据全部发送，lrs_send 将尝试重新发送直到超时为止。如果没有可用连接，lrs_send 也将尝试寻找连接直至超时。lrs_send 缺省超时时间为 10 秒，可以使用 lrs_set_send_timeout 函数来修改超时时间。lrs_send 成功执行不代表数据被成功传送。如果用于传输的缓冲空间不足，lrs_send 将阻断传输除非 I/O 模式被置为 non-blocking。关于”blocking”和”non-blocking”请参阅“知识链接”部分相关内容。

lrs_set_receive_option 设置套接字接收选项

格式：

int lrs_set_receive_option (int *option*, int *value,* [char * *terminator*]);

参数说明：

option: 接收选项，表示何时停止接收数据。可用选项有 Mismatch 和 EndMarker。

value: 选项的值，详见列表。

terminator: 用于 value 中的结束接收标记。该选项仅在 EndMarker 选项被指定为 StringTerminator 时需要。

option	可用的值
Mismatch	1. MISMATCH_SIZE (缺省) 2. MISMATCH_CONTENT
EndMarker	1. EndMarker_None (缺省) – 接收全部数据 2. StringTerminator – 中断接收的字符串. (仅 TCP 连接可用) 3. BinaryStringTerminator – 中断接收的二进制标识串. (仅 TCP 连接可用) 4. RecordingSize – 指定接收长度等于预期长度. (仅 TCP 连接可用)

返回值：

成功返回 0。

用法一：

/*该例指定对接收到的数据进行内容匹配性验证*/

假如预期接收数据中包含一个"!"字符，而回放时该位置接收到的是"@"字符，则该例会报出 Mismatch 信息。

```
lrs_set_receive_option(Mismatch, MISMATCH_CONTENT);
lrs_send("socket1", "buf2", 1, 0);
lrs_receive("socket1", "buf2", LrsLastArg);
lrs_close_socket("socket1");
```

如果选项值为 MISMATCH_SIZE 则不会报出 Mismatch 信息。

用法二：

该例中 lrs_set_receive_option 函数指定接收的套接字长度等于预期长度。

```
lrs_create_socket("socket1", "TCP", "RemoteHost=199.203.77.246:21", LrsLastArg);
lrs_set_receive_option(EndMarker, RecordingSize);
lrs_receive("socket1", "buf2", LrsLastArg);
```

假如录制时得到的 buf2 的长度为 20 个字节，而回放时从服务器接收到的长度为 30 个字节，以上脚本将只接收前 20 个字节，剩余的 10 个字节将被下一个 lrs_receive 接收。

如果该例的 option 值为 EndMarker_None，那么 lrs_receive 会接收完整的返回值并报一个 mismatch 消息。

函数说明：

函数 lrs_set_receive_option 为 lrs_receive 设置一个接收选项。该选项表示如何中止接收套接字信息（是出现一个不匹配时终止还是检测到结束字符串时终止）。该函数对 lrs_receive_ex 无效。该设置将应用于之后出现的所有 lrs_receive 操作，除非遇到另一个 lrs_set_receive_option。

lrs_set_socket_handler 为指定的套接字设置处理句柄

格式：

```
int lrs_set_socket_handler ( char *s_desc, int handler );
```

参数说明：

s_desc: 套接字标识符。

handler: 指定的套接字句柄。

返回值：

成功返回 0。

用法：

```
/****下面的脚本中将 socket0 和 socket1 的套接字句柄互换。****/
int socket0_handler=0, socket1_handler=0;
lrs_create_socket("socket0", "UDP", LrsLastArg);
socket0_handler=lrs_get_socket_handler("socket0");
lrs_create_socket("socket1","UDP","LocalHost=0","RemoteHost=localhost:1496",
LrsLastArg);
socket1_handler=lrs_get_socket_handler("socket1");
lrs_set_socket_handler("socket1", socket0_handler);
lrs_set_socket_handler("socket0", socket1_handler);
return 0;
```

函数说明：

该函数为指定的套接字关联一个套接字句柄。

lrs_set_socket_options 设置套接字选项

格式：

```
int lrs_set_socket_options ( char *s_desc, int option, char *option_value );
```

参数说明：

s_desc: 套接字标识符。

option: 套接字选项。目前只有一个可用选项：LRS_NO_DELAY。

option_value: 选项值或类型。目前可用的值有 TRUE 和 FALSE，设置为 TRUE 使 LRS_NO_DELAY 可用。

返回值：

成功返回 0。

用法：

```
lrs_create_socket("socket1", "TCP", "RemoteHost=168.3.1.230:6666", LrsLastArg);
lrs_set_socket_options("socket1", LRS_NO_DELAY, TRUE);
lrs_send("socket1", "buf2", LrsLastArg);
```

函数说明:

正常情况下，TCP 会将有报头的包立即传输。某些情况下发送的包被对方收到后需要请求对方确认，如果没有收到确认信息，后续的传输数据就会被延迟。而对某些客户端事件，像 windows 系统的鼠标事件就会产生大量的数据而导致严重的网络延迟问题。

因此，TCP 提供了一个布尔选项 TCP_NODELAY 用以优化上述传输算法和预防网络延迟。

LRS_NO_DELAY 就是 LRS 提供的一个具有相同功能的函数。

该函数只用于 TCP 协议的套接字。

更多 TCP_NODELAY 的内容请参见“知识链接”部分。

2．缓冲区函数

lrs_free_buffer 释放分配给缓冲区的内存

格式：

int lrs_free_buffer (char **buffer*);

参数说明：

buffer: 要释放的缓冲区标识符。

返回值：

成功返回 0。

用法：

```
char *ActualBuffer;
int NumberOfBytes;
lrs_receive("socket2", "buf20", LrsLastArg);
/* 获得最后接收的缓冲区的内容和长度，此时发生内存分配*/
lrs_get_last_received_buffer("socket2", &ActualBuffer, &NumberOfBytes);
lr_output_message("The last buffer's size is:%d\n", NumberOfBytes);
lrs_free_buffer(ActualBuffer); //释放缓冲区
```

函数说明：

函数 lrs_free_buffer 为释放已分配给指定缓冲区的内存区块。当调用 lrs_get_buffer_by_name 或 lrs_get_last_received_buffer 时就会发生内存分配。此函数只能对 lrs_get_buffer_by_name 或 lrs_get_last_received_buffer 这两个函数所产生的内存分配进行释放，而不能释放由其它函数调用所发生的内存分配。

lrs_get_buffer_by_name 从数据文件中获取缓冲区及其大小

格式：

int lrs_get_buffer_by_name (char **buf_desc*, char ***data*, int **size*);

参数说明：

buf_desc: 缓冲区标识符。

data: 指向缓冲区内容的指针。

size: 指向缓冲区内容长度的指针。

返回值：

成功返回 0。

用法：

下面的脚本用来获取 buf1 的内容和长度，其中 buf1 的内容为：

```
send buf1 3
    "\x00\xff\x59"
/****脚本内容：****/
char *ActualBuffer;
int NumberOfBytes;
lrs_create_socket("socket0","TCP","RemoteHost=168.3.1.230:6666",LrsLastArg);
//注意下面函数对 ActualBuffer 和 NumberOfBytes 两个变量的引用
lrs_get_buffer_by_name("buf1",&ActualBuffer,&NumberOfBytes);
lr_output_message("the size of buf1 is %d",NumberOfBytes);
/*由于指针 ActualBuffer 指向的是二进制的值，因此不能用字符串的形式直接获取。
下面的脚本将 ActualBuffer 的值保存到变量 param 中,然后利用函数 lr_eval_string 获取*/
lrs_save_param_ex("socket0","user",ActualBuffer,0,NumberOfBytes,"ascii","param");
lr_output_message("the content of buf1 is %s",lr_eval_string("<param>"));
```

脚本执行结果为：

```
vuser_init.c(11): lrs_get_buffer_by_name (buf1, buf_p, size_p)
vuser_init.c(12): the size of buf1 is 3
vuser_init.c(15): lrs_save_param_ex(socket0, user, buf_p, 0, 3, ascii, param)
vuser_init.c(16): the content of buf1 is \x00\xffY
```

函数说明：

函数 lrs_get_buffer_by_name 取得指定缓冲区的数据内容（二进制）和长度。在使用函数前必须先声明一个接收内容的字符型指针变量和接收长度的整形变量。函数被调用后将自动分配缓冲区，但是你必须用函数 lrs_free_buffer 手工将其释放。

lrs_get_last_received_buffer 获取套接字上最后接收到的缓冲区数据及其大小

格式：

int lrs_get_last_received_buffer (char **s_desc*, char ***data*, int **size*);

参数说明：

s_desc: 套接字标识符。

data: 指向缓冲区数据内容的指针。

size: 指向缓冲区数据长度的指针。

返回值：

成功返回 0。

用法：

该例中 buf4 在 data.ws 中的内容：

```
recv    buf4 27
"\x00\x00\x00\x17\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
"\x1f\x00\x00\x00\x00"
"PID"
```

```
/*下面是脚本部分*/
char *ActualBuffer;
int NumberOfBytes;
lrs_send("socket0", "buf3", LrsLastArg);
lrs_receive("socket0", "buf4", LrsLastArg);
lrs_get_last_received_buffer("socket0",&ActualBuffer,&NumberOfBytes);
lr_output_message("the size of last received buffer is %d",NumberOfBytes);
/*由于指针 ActualBuffer 指向的是二进制的值，因此不能用字符串的形式直接获取。
下面的脚本将 ActualBuffer 的值保存到变量 param 中，然后利用函数 lr_eval_string 获取.*/
lrs_save_param_ex("socket0","user",ActualBuffer,0,NumberOfBytes,"ascii","param");
lr_output_message("the content of last received buffer is %s",lr_eval_string("<param>"));
```

脚本执行结果：

```
vuser_init.c(28): lrs_get_last_received_buffer(socket0, buf_p, size_p)
vuser_init.c(29): the size of last received buffer is 27
vuser_init.c(32): lrs_save_param_ex(socket0, user, buf_p, 0, 27, ascii, param)
vuser_init.c(33): the content of last received buffer is
\x00\x00\x00\x17\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x1f\x00\x
00\x00\x00PID
```

函数说明：

lrs_get_last_received_buffer 函数可以获取使用该函数之前最后接收到的数据内容及其长度。data 和 size 所需要的缓冲区会自动分配，但是需要通过 lrs_free_buffer 手工释放。在调用函数之前必须声明一个字符型指针变量和一个整型变量用以存放缓冲区内容和长度。

lrs_get_last_received_buffer_size 获取套接字上接收到的最后一个缓冲区的大小

格式：

int lrs_get_last_received_buffer_size (char **s_desc*);

参数说明：

s_desc: 套接字标识符。

返回值：

返回实际接收到的字节数。

用法：

```
lrs_receive("socket1", "buf2", LrsLastArg);
if (lrs_get_last_received_buffer_size("socket1") < 1) {
    lr_error_message("No data received from buf2 on socket1");
    return -1;
}
```

函数说明：

该函数获取指定套接字上最后接收到的缓冲区数据长度，函数返回的是以二进制表示的字节长度。当不需要关心数据内容是，可以用该函数来取代 lrs_get_last_received_buffer 以确认已接收的数据长度。

函数 lrs_get_last_received_buffer_size 和 lrs_get_last_received_buffer 总是紧跟在 receive 函数之后，receive 函数包括 lrs_receive, lrs_receive_ex 和 lrs_length_receive。

lrs_get_received_buffer 获取指定长度的最后接收到的缓冲区数据

格式：

char *lrs_get_received_buffer (char **s_desc*, int *offset*, int *length*, char **encoding*);

参数说明：

s_desc: 套接字标识符。

offset: 缓冲区偏移量。

length: 要获取的字节数。

encoding: 接收数据的编码方式，有”ascii”,”ebcdic”或者 NULL。

返回值：

成功返回指向用户缓冲区的一个指针。

用法：

```
char *ReceivedBuffer;
lrs_receive("socket0", "buf4", LrsLastArg);
ReceivedBuffer = lrs_get_received_buffer("socket0",24,3,NULL);
lrs_save_param_ex("socket0","user",ReceivedBuffer,0,3,"ascii","param");
lr_output_message("the content of received buffer is %s",lr_eval_string("<param>"));
```

脚本执行结果：

```
vuser_init.c(27): lrs_receive(socket0, buf4)
vuser_init.c(29): lrs_get_received_buffer(socket0, buf_p, size_p)
vuser_init.c(32): lrs_save_param_ex(socket0, user, buf_p, 0, 3, ascii, param)
vuser_init.c(33): the content of received buffer is PID
```

函数说明：

该函数返回的是最后一个接收缓冲区的数据，内容可以通过 offset 和 length 两个参数指定。该函数必须跟在 lrs_receive 之后，函数获取的值是二进制值，因此其内容必须通过转换才能读取。如果函数中 encoding 方式设置为 NULL，则返回的编码方式为客户端默认的编码方式。若要获取接收缓冲区的实际长度，可以使用 lrs_get_last_received_buffer_size。

lrs_get_static_buffer 获取静态缓冲区或其一部分

格式：

char *lrs_get_static_buffer (char **s_desc*, char **buffer*, int *offset*, int *length*, char **encoding*);

参数说明：

s_desc: 套接字标识符。

buffer: 静态缓冲区标识符。

offset: 开始获取的偏移量。

length: 要获取的字节数。

encoding: 指定返回数据的编码方式，可以是”ascii”,”ebcdic”或者 NULL。

返回值：

成功返回指向用户缓冲区的一个指针。

用法：

```
char *StaticBuffer;
```

StaticBuffer = lrs_get_static_buffer("socket0",”buf4”,24,3,NULL);

lrs_save_param_ex("socket0","user",StaticBuffer,0,3,"ascii","param");

lr_output_message("the content of static buffer is %s",lr_eval_string("<param>"));

脚本执行结果：

vuser_init.c(29): lrs_get_static_buffer(socket0, buf4, 24, 3, null)

vuser_init.c(30): lrs_save_param_ex(socket0, user, buf_p, 0, 3, ascii, param)

vuser_init.c(31): the content of static buffer is PID

函数说明：

函数 lrs_get_static_buffer 从数据文件（data.ws）中读取一个静态缓冲区的部分或全部数据，可以通过设定 offset 和 length 来指定要获取的缓冲区内容。

函数执行后读取的数据会以指定的编码格式存放在一个用户缓冲区中。返回值是一个指向该用户缓冲区的指针。如果参数 encoding 被置为 NULL 则用户缓冲区将以原始的客户端编码格式存放数据（ebcdic 或 ascii）。

可以用 lrs_get_user_buffer 来读取用户缓冲区的内容，而用 lrs_get_user_buffer_size 来获得其长度。

lrs_get_user_buffer 获取套接字的用户数据的内容

格式：

char *lrs_get_user_buffer (char *s_desc);

参数说明：

s_desc: 套接字标识符。

返回值：

返回指向用户缓冲区的指针。

用法：

char *UserBuffer;

lrs_get_static_buffer("socket0","buf4",24,3,NULL);

UserBuffer = lrs_get_user_buffer("socket0");

lr_output_message("the content of UserBuffer is %s",UserBuffer);

脚本执行结果：

vuser_init.c(28): lrs_get_static_buffer(socket0, buf4, 24, 3, null)

vuser_init.c(29): lrs_get_user_buffer(socket0)

vuser_init.c(30): the content of UserBuffer is PID

函数说明：

该函数获取的是指定套接字上用户缓冲区的数据，如果在该函数使用之前有多次分配用户缓冲区，则获取的是最后一次分配的缓冲区的内容。如果没有分配用户缓冲区则返回 0。

lrs_get_user_buffer_size 获取套接字的用户缓冲区的长度

格式：

long lrs_get_user_buffer_size (char *s_desc);

参数说明：

s_desc: 套接字标识符。

返回值：

用户缓冲区的长度。

用法：

```
int LenOfUserBuffer;
lrs_get_static_buffer("socket0","buf4",24,3,NULL);
LenOfUserBuffer = lrs_get_user_buffer_size("socket0");
lr_output_message("the size of UserBuffer is %d",LenOfUserBuffer);
```

脚本执行结果：

```
vuser_init.c(28): lrs_get_static_buffer(socket0, buf4, 24, 3, null)
vuser_init.c(29): lrs_get_user_buffer_size(socket0)
vuser_init.c(30): the size of UserBuffer is 3
```

lrs_set_send_buffer 指定要在套接字上发送的缓冲区

格式：

int lrs_set_send_buffer (char *s_desc, char *buffer, int size);

参数说明：

s_desc: 套接字标识符。

buffer: 指定要发送的缓冲区。

size: 要发送的字节数。

返回值：

发送成功返回 0，失败返回错误码。

用法：

```
char *Buffer;
int Size;
```

```
lrs_receive("socket2", "buf20", LrsLastArg);
/*将接收缓冲区 buf20 中的值保存到用户缓冲区 Buffer 中*/
lrs_get_last_received_buffer("socket2", &Buffer, &Size);
/*指定要发送的数据从 Buffer 中读取，长度为 10 个字节*/
lrs_set_send_buffer("socket2", Buffer, 10 );
/*此处的 buf21 将被 Buffer 所取代，实际发送的数据是从 Buffer 中读取的 10 个字节*/
lrs_send("socket2","buf21", LrsLastArg );
lrs_free_buffer(Buffer);
```

函数说明：

函数 lrs_set_send_buffer 用来指定下一次调用 lrs_send 时要发送的缓冲区内容。该缓冲区在 lrs_set_send_buffer 中被指定后，其后的一个 lrs_send 中指定的缓冲区内容将不会被发送。

3．环境函数

lrs_cleanup 终止 Windows 套接字 DLL 的使用

格式：

```
int lrs_cleanup ( );
```

返回值：

成功返回 0，失败返回错误码。

lrs_startup 初始化 Windows 套接字 DLL

格式：

```
int lrs_startup ( int version );
```

参数说明：

version: Windows 套接字版本。

用法：

```
lrs_startup(257);
```

函数说明：

该函数指定应用程序可用的 windows 套接字的最高版本。该函数必须放在所有 lrs 函数的前面，一般都出现在 vuser_init 部分的开始处，如果初始化失败，脚本将会马上中止执行。

4．关联语句函数

lrs_save_param 将静态或接收到的缓冲区（或缓冲区部分）保存到参数中

格式：

int lrs_save_param (char *s_desc, char *buf_desc, char *param_name, int offset, int param_len);

参数说明：

s_desc: 套接字标识符。

buf_desc: 缓冲区标识符。

param_nanme: 存放缓存数据的参数名称。

offset: 被保存到参数中的缓存区偏移量。

param_len: 要保存到参数中的字节数。

返回值：

成功返回 0 ，失败返回错误码。

用法一：

```
/*保存静态缓冲区数据到参数中*/
lrs_save_param("socket0","buf4","param1",24,3);
lr_output_message("the content of param1 is %s",lr_eval_string("<param1>"));
```

脚本执行结果：

```
vuser_init.c(28): lrs_save_param(socket0, buf4, param1, 24, 3)
vuser_init.c(29): the content of param1 is PID
```

用法二：

```
/*将最后接收到的缓冲区数据保存到参数中*/
lrs_receive("socket0", "buf4", LrsLastArg);
lrs_save_param("socket0",NULL,"param1",24,3);
lr_output_message("the content of param1 is %s",lr_eval_string("<param1>"));
```

函数说明：

利用该函数将指定的缓冲区数据成功保存到参数中以后，指定的参数名和普通参数一样，可以在脚本的其它地方自由引用。

lrs_save_param_ex 将用户、静态或接收到的缓冲区（或缓冲区部分）保存到参数中

格式：

int lrs_save_param_ex (char *s_desc, char *type, char *buff, int offset, int length, char *encoding, char *param);

参数说明：

s_desc: 套接字标识符。

type: 要将数据保存到参数中的缓冲区类型，有”user”(用户缓冲区)、”static”(data.ws 中的静态缓冲区)和”received”(最后接收的缓冲区数据)三种。

buff: 和 type 的值有关，如果 type 的值是”user”则 buff 的值为指定的用户缓冲区，如果 type 的值是”static”则 buff 的值为指定的静态缓冲区，如果 type 的值是”received”则 buff 参数可以设为 NULL。

offset: 缓冲区偏移量。

length: 保存到参数中的字节数。

encoding:编码方式可以指定为”ascii”或”ebcdic”, 如果是用户缓冲区则 NULL 默认为”ascii”，如果 type 为”static”或”received”则 NULL 默认为客户端编码方式。

param: 参数名称。

返回值：

成功返回 0，失败返回错误码。

用法一：

```
/*保存用户缓冲区数据到参数中*/
char *UserBuffer = "Overn";
lrs_save_param_ex("socket0","user",UserBuffer,0,5,"ascii","param1");
lr_output_message("the content of param1 is %s",lr_eval_string("<param1>"));
```

脚本执行结果：

```
vuser_init.c(28): lrs_save_param_ex(socket0, user, buf_p, 0, 5, ascii, param1)
vuser_init.c(29): the content of param1 is Overn
```

用法二：

```
/*保存静态缓冲区数据到参数中*/
lrs_save_param_ex("socket0","static","buf4",24,3,"ascii","param1");
lr_output_message("the content of param1 is %s",lr_eval_string("<param1>"));
```

脚本执行结果：

```
vuser_init.c(28): lrs_save_param_ex(socket0, static, buf_p, 24, 3, ascii, param1)
vuser_init.c(29): the content of param1 is PID
```

用法三：

```
/*保存最后接收的缓冲区数据到参数中*/
lrs_receive("socket0", "buf4", LrsLastArg);
lrs_save_param_ex("socket0","received",NULL,24,3,NULL,"param1");
lr_output_message("the content of param1 is %s",lr_eval_string("<param1>"));
```

脚本执行结果：

```
vuser_init.c(28): lrs_save_param_ex(socket0, received, buf_p, 24, 3, null, param1)
vuser_init.c(29): the content of param1 is PID
```

函数说明：

函数 lrs_save_param_ex 不识别用户缓冲区的参数化，比如如果将上例用法一中的”Overn”换为”Ov<a>rn”(其中 a 为参数名，值为字符 e)，那么执行结果将会是 vuser_init.c(29): the content of param1 is Ov<a>。

lrs_save_searched_string 在静态或接收到的缓冲区中搜索出现的字符串，将出现字符串的缓冲区部分保存到参数中

格式：

int **lrs_save_searched_string** (char* *s_desc*, char* *buf_desc*, char* *param_name*, char* *left_boundary*, char* *right_boundary*, int *ordinal*, int *offset*, int *param_len*);

参数说明：

s_desc: 套接字标识符。

buf_desc: 缓冲区标识符。

param_name: 保存缓冲区数据的参数名称。

left_boundary: 标识要搜索缓冲区部分的左边界的字符串，格式为”LB=XXX”。

right_boundary: 标识要搜索缓冲区部分的右边界的字符串，格式为”RB=XXX”。

ordinal: 表示从第几次出现的左边界字符串开始搜索，如果指定了左边界则 ordinal 的值一定大于 0，如果没有指定左边界则将 ordinal 设为-1。

offset: 要开始搜索的偏移量。如果指定了左边界则此偏移量相对于左边界计算，否则就从缓冲区的开始计算偏移量。

param_len: 要保存到参数中的缓冲区数据字节数。适用于没有指定右边界的情况，如果指定了右边界则设 param_len 为-1。

返回值：

成功返回 0，否则返回错误码。

用法一：

```
/******数据文件 data.ws 中的内容：******/
recv    buf2 81
      "\x00\x00\x00"
      "M"
     "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x1a\x00\x00\x00"
      "\x00"
      "旸 OSGMBHDEFAULTLOGIN"    //此行开头的字符为 ascii 的字符”D”
      "\x95"
      "  晻  旸 EFAULT"
      "\x95"
      " "
      "\x95"
      "210286"
      "\x95"
      " "
      "\x95"
      "1"
      "\x95"
      "  盼 UCCESS"
      "\x95"
/*如果将 left_boundary 和 right_boundary 均设为 NULL，并指定 offset 和 param_len,
  则该函数等同于 lrs_save_param */
  lrs_save_searched_string("socket0","buf2","NewParam",NULL,NULL,-1,26,12);
  //等同于 lrs_save_param("socket0","buf2","NewParam",26,12);
  lr_output_message("Param = %s",lr_eval_string("<NewParam>"));
  执行结果：
  vuser_init.c(22): lrs_save_searched_string(socket0, buf2, NewParam, null, null, -1, 26, 12)
  vuser_init.c(23): Param = OSGMBHDEFAUL
```

用法二：

```
/**指定 left_boundary（字符”D”）和 right_boundary（字符”T”），
并且从第二次出现字符”D”（字符”D”在 buf2 中一共出现 3 次），
再向右偏移 2 个字节开始保存**/
lrs_save_searched_string("socket0","buf2","NewParam","LB=D","RB=T",2,2,-1);
lr_output_message("Param = %s",lr_eval_string("<NewParam>"));
```

执行结果：

vuser_init.c(21): lrs_save_searched_string(socket0, buf2, NewParam, LB=D, RB=T, 2, 2, -1)

vuser_init.c(22): Param = AUL

函数说明：

函数 lrs_save_searched_string 将缓冲区的一部分数据保存到参数中。如果左右边界指定的是二进制字符串，则使用”LB/BIN”或”RB/BIN”来指定，如”LB/BIN=\\x95”。缓冲区标识符表明了从哪个缓冲区中获取数据保存到参数中，如果是从数据文件 data.ws 中读取则指定为”bufxxx”（如”buf2”），如果设为 NULL 则表示从最后一个接收到的缓冲区（lrs_last_received）中读取数据。

该函数的参数列表有六种组合方式：

a. 左右边界均设为 NULL，并指定偏移量和长度（等同于 lrs_save_param）。
b. 指定左右边界和左边界出现的次数。
c. 指定左边界、左边界出现的次数及读取的数据长度（字节数）。
d. 左边界、左边界出现的次数、从左边界算起的偏移量及右边界。
e. 左边界、左边界出现的次数、从左边界算起的偏移量、读取的字节数。
f. 只限定偏移量和右边界。

5．转换函数

lrs_ascii_to_ebcdic 将缓冲区数据从 ASCII 格式转换成 EBCDIC 格式

格式：

char *lrs_ascii_to_ebcdic (char **s_desc*, char **buff*, long *length*);

参数说明：

s_desc: 套接字标识符。

buff: 缓冲区标识符。

length: 缓冲区长度。

返回值：

成功返回一个指向缓冲区的指针。

用法：

```
char *UserBuffer;
/* 以 ascii 格式接收返回的缓冲区数据 */
UserBuffer = lrs_get_received_buffer("socket0",0,-1,"ascii");
```

```
/* 将缓冲区数据格式从 ascii 转换为 ebcdic 格式 */
UserBuffer = lrs_ascii_to_ebcdic("socket0",UserBuffer,lrs_get_user_buffer_size("socket0"));
```

lrs_decimal_to_hex_string 将十进制整数转换为十六进制字符串

格式：

```
char* lrs_decimal_to_hex_string( char* s_desc, char* buf, long length);
```

参数说明：

s_desc: 套接字标识符。

buf: 缓冲区标识符。

length: 转换的长度。

返回值：

返回转换后的字符串指针。

用法：

```
char *UserBuffer, *UserBufferHex;
/* 取得被转换的整数并打印 */
UserBuffer = lrs_get_received_buffer("socket0",65,1,NULL);
lrs_save_param_ex("socket0","user",UserBuffer,0,1,"ascii","NewParam");
lr_output_message("UserBuffer = %s",lr_eval_string("<NewParam>"));
 /* 把整数转换成十六进制并打印出转换后的结果 */
UserBufferHex = lrs_decimal_to_hex_string("socket0",UserBuffer,1);
lr_output_message("UserBufferHex = %s",UserBufferHex);
```

执行结果：

```
vuser_init.c(24): lrs_save_param_ex(socket0, user, buf_p, 0, 1, ascii, NewParam)
vuser_init.c(25): UserBuffer = 6
vuser_init.c(27): lrs_decimal_to_hex_string(socket0, ....)
vuser_init.c(28): UserBufferHex = \x06
```

lrs_ebcdic_to_ascii 将缓冲区数据从 EBCDIC 格式转换成 ASCII 格式

格式：

```
char *lrs_ebcdic_to_ascii ( char *s_desc, char *buff, long length );
```

参数说明：

s_desc: 套接字标识符。

buff: 缓冲区标识符。

length: 字符长度

返回值：

返回字符串指针。

用法：

```
char *UserBuffer;
/* 以 ebcdic 格式接收返回的缓冲区数据 */
UserBuffer = lrs_get_received_buffer("socket0",0,-1,"ebcdic");
/* 将缓冲区数据格式从 ebcdic 转换为 ascii 格式 */
UserBuffer = lrs_ebcdic_to_ascii("socket0",UserBuffer,lrs_get_user_buffer_size("socket0"));
```

lrs_hex_string_to_int 将十六进制字符串转换为整数

格式：

int lrs_hex_string_to_int (char* *buff*, long *length*, int* *mpiOutput*);

参数说明：

buff: 缓冲区标识符。

mpiOutput: 指向整型结果的指针。

返回值：

返回转换后的整数。

用法：

```
char *UserBuffer;
int i = 0;
UserBuffer = "\x1b";
lrs_hex_string_to_int(UserBuffer,1,&i);
lr_output_message("UserBufferInt = %i",i);
```

脚本执行结果：

```
vuser_init.c(28): lrs_hex_string_to_int(....)
vuser_init.c(29): UserBufferInt = 27
```

6．超时函数

lrs_set_accept_timeout 为接受套接字设置超时

格式：

void lrs_set_accept_timeout (long *seconds*, long *u_sec*);

参数说明：

seconds: 超时的秒数。

u_sec: 超时的毫秒数。

返回值：

该函数无返回值。

用法：

```
lrs_set_accept_timeout(30,0);
lrs_create_socket("socket0", "TCP", "RemoteHost=168.3.1.230:6666",  LrsLastArg);
lrs_accept_connection(“socket0”,”socket1”);
lrs_send(“socket1”,”buf1”,LrsLastArg);
```

函数说明：

lrs_set_accept_timeout 函数设置服务器在检测到可读套接字 (select)并完成关联(accept) 之前等待的时间。如果套接字当前正在侦听，并且收到了传入的连接请求，它将被标记为可读。一旦套接字被标记为可读，就可以保证完成 关联，不会阻塞。可以为超时值指定秒数和毫秒数。如果套接字在超时时间间隔内未被接受，则脚本执行将终止。默认的超时值为 10 秒。如果您发现默认时间不够，请做相应修改。

lrs_set_connect_timeout 为连接到套接字设置超时

格式：

void lrs_set_connect_timeout (long *seconds*, long *u_sec*);

参数说明：

seconds: 超时的秒数。

u_sec: 超时的毫秒数。

返回值：

该函数无返回值。

用法：

```
lrs_set_connect_timeout(100,0);
lrs_create_socket("socket0", "TCP", "RemoteHost=168.3.1.230:6666",  LrsLastArg);
```

函数说明：

该函数设置套接字连接的超时时间，该函数属全局函数，一旦设定对脚本中所有的套接字连接有效，直至遇到下一个 lrs_set_connect_timeout 为止。

lrs_set_recv_timeout 为接收套接字上的初始预期数据设置超时

格式：

```
void lrs_set_recv_timeout ( long sec, long u_sec );
```

参数说明：

sec: 超时的秒数，缺省为 10 秒。

u_sec: 超时的毫秒数。

返回值：

该函数无返回值。

用法：

```
lrs_set_recv_timeout(15,0);
lrs_receive("socket0", "buf0", LrsLastArg);
```

函数说明：

该函数设置 VuGen 从套接字连接上接受预期数据的超时时间。当缓冲区长度与预期的数据长度不匹配（小于或大于）时，lrs_receive 将会重新从套接字上读取接收数据，直到超时。该函数是全局函数，对 VuGen 中所有的套接字连接有效，除非遇到下一个 lrs_set_recv_timeout。

lrs_set_recv_timeout2 为建立连接后接收套接字上的预期数据设置超时

格式：

```
void lrs_set_recv_timeout2( long sec, long u_sec );
```

参数说明：

sec: 超时的秒数。

u_sec: 超时的毫秒数。

返回值：

该函数无返回值。

用法：

lrs_set_recv_timeout2(15,0);

lrs_receive("socket0", "buf0", LrsLastArg);

函数说明：

函数 lrs_set_recv_timeout2 设置了从套接字接收数据的超时时间。当 lrs_receive 接收到数据后它将和预期的数据长度进行比较，如果长度不匹配，它将重新从套接字上读取返回值，直到超时为止。你可以通过日志查看这些迭代过程。

你可以通过本函数为应用程序的不同行为定义超时时间。

比如一个应用程序需要一个较长的时间处理请求，但是一旦处理完毕，服务器将会用很快的速度返回数据。这种情况下，你可以用 lrs_set_recv_timeout 来设定一个较大的超时时间，而用 lrs_set_rcv_timeout2 来设定一个较小的超时时间来接收数据。例如当你执行一个对数据库的查询操作，服务器可能需要一些时间来处理请求以返回数据，可是一旦开始发送返回数据则基本没有延迟，这样接收数据时 timeout2 的超时时间就可以设定得小一些。

而其他一些应用程序可能连接服务器很快，但是可能由于网络拥堵数据传输会花费比较长的时间，这种情况下则可以将 lrs_set_recv_timeout 设定得小一些，而为 lrs_set_recv_timeout2 设定一个较大的值。

lrs_set_send_timeout 为发送套接字数据设置超时

格式：

void lrs_set_send_timeout (long sec, long u_sec);

参数说明：

sec: 超时的秒数。

u_sec: 超时的毫秒数。

返回值：

该函数无返回值。

用法：

lrs_set_send_timeout(90,0);

lrs_send("socket0", "buf1", LrsLastArg);

函数说明：

函数 lrs_send 向一个已连接的数据报或流套接字发送数据，如果不能成功将缓冲区中的所有数据全部发送，它将尝试重新发送直到超时为止。缺省的超时时间是 10 秒，你可以通过 lrs_set_send_timeout 来指定。通过该函数指定的超时时间对所有的套接字均有效，直到遇到下一个 lrs_set_send_timeout。

7．Receive and Send Flags

可以用下表的值来指定 lrs_receive 和 lrs_send 的 flag 选项。

Flag 值	含义
MSG_PEEK	lrs_receive 使用的标志。将接收的返回值复制到缓冲区但不会从接收队列中删除该数据。下次读取返回值将从缓冲区中读取，因此读取到的返回值内容将不会改变。
MSG_DONTROUTE	是 lrs_send 函数使用的标志。这个标志告诉 IP 协议，目的主机在本地网络内，没必要查找路由表，该标志一般用在网络诊断和路由程序里。
MSG_OOB	发送或接收带外数据（out-of-band）。关于 OOB 详见“知识链接”部分。

二、LRS 错误码

由于脚本本身的错误在脚本回放时会返回以下错误：

LRS 错误信息	错误代码	错误原因
Error		以下为错误
LRS_GENERIC_ERROR	9001	General system function fault. 系统函数错误
LRS_SOCKET_DOESNT_EXIST	9002	An invalid socket descriptor was transferred to the function. 无效的 socket 标识符
LRS_PARAM_FAILED	9003	Parameterization failed. 参数化失败
LRS_CREATE_SOCK_FAILED	9004	Creation of a new socket failed. 创建套接字连接失败
LRS_INVALID_FUNC_PARAM	9005	An invalid parameter was transferred to the function. 无效的参数
LRS_READ_DATA_FILE_ERR	9006	Unable to read from the data file (data.ws)不能从数据文件(data.ws)中读取数据
LRS_DATA_BUFFER_ERR	9007	Error reading data buffer (either from data file or last received buffer). 读

		取数据缓冲区错误（可能是数据文件或最后接收的缓冲区）
LRS_UNKNOWN_HOST	9008	Specified host is unknown. 找不到指定的主机
LRS_UNKNOWN_FUNC_PARAM	9009	The specified parameter is unknown. 无法识别的参数
LRS_DELETE_SOCK_FAILED	9010	An attempt to delete the socket, failed. 试图删除套接字失败
LRS_BUFFER_DOESN'T_EXIST	9011	The specified buffer does not exist. 指定的缓冲区不存在
LRS_BUF_ALLOC_ERR	9012	Buffer allocation error occurred. 分配缓冲区发生错误
LRS_TRANSALTION_ERR	9013	Error in buffer translation. 缓冲区数据转换发生错误
LRS_BUFFER_TYPE_ERR	9014	Wrong buffer type specified. 错误的缓冲区类型
LRS_SAVE_PARAM_ERR	9015	Error in saving parameter to a variable. 向变量中保存参数时发生错误
LRS_HOST_FIND_ERR	9016	Error in locating specified host. 定位指定的主机时出错
LRS_EXIT_BY_TIMEOUT	9017	Exit test after failing to complete operation due to timeout. 由于超时执行操作失败
LRS_SELECT_ERR	9018	A SELECT related error occurred. 与 SELECT 相关的错误
Warning		以下为警告信息
LRS_RECV_MISMATCH	9101	A mismatch occurred between the expected and received buffers. 预期数据与接收到的缓冲数据不匹配
LRS_STRING_NOT_FOUND	9102	The specified string was not found. 找不到指定的字符串

标准的 Windows Socket 错误码:

错误码	英文提示	中文说明
10004 - WSAEINTR	Interrupted system call	函数调用中断。该错误表明由于对 WSACancelBlockingCall 的调用，造成了一次调用被强行中断。
10009 - WSAEBADF	Bad file number	文件句柄错误。该错误表明提供的文件句柄无效。在 Microsoft Windows CE 下，socket 函数可能返回这个错误，表明共享串口处于“忙”状态。
10013 - WSEACCES	Access denied	权限被拒。尝试对套接字进行操作，但被禁止。若试图在 sendto 或 WSASendTo 中使用一个广播地址，但是尚未用 setsockopt 和 so_broadcast 这两个选项设置广播权限，便会产生这类错误。
10014 - WSAEFAULT	Bad address	地址无效。传给 Winsock 函数的指针地址无效。若指定的缓冲区太小，也会产生这个错误。
10022 - WSAEINVAL	Invalid argument	参数无效。指定了一个无效参数。例如，假如为 WSAIoctl 调用指定了一个无效控制代码，便会产生这个错误。另外，它也可能表明套接字当前的状态有错，例如在一个目前没有监听的套接字上调用 accept 或 WSAAccept。
10024 - WSAEMFILE	Too many open files	打开文件过多。提示打开的套接字太多了。通常，Microsoft 提供者只受到系统内可用资源数量的限制。
10035 - WSAEWOULDBLOCK	Operation would block	资源暂时不可用。对非锁定套接字来说，如果请求操作不能立即执行的话，通常会返回这个错误。比如说，在一个非暂停套接字上调用 connect，就会返回这个错误。因为连接请求不能立即执行。
10036 - WSAEINPROGRESS	Operation now in progress	操作正在进行中。当前正在执行非锁定操作。一般来说不会出现这个错误，除非正在开发 16 位 Winsock 应用程序。

10037 - WSAEALREADY	Operation already in progress	操作已完成。一般来说，在非锁定套接字上尝试已处于进程中的操作时，会产生这个错误。比如，在一个已处于连接进程的非锁定套接字上，再一次调用 connect 或 WSAConnect。另外，服务提供者处于执行回调函数（针对支持回调例程的 Winsock 函数）的进程中时，也会出现这个错误。
10038 - WSAENOTSOCK	Socket operation on non-socket Socket	无效套接字上的套接字操作。任何一个把 SOCKET 句柄当作参数的 Winsock 函数都会返回这个错误。它表明提供的套接字句柄无效。
10039 - WSAEDESTADDRREQ	Destination address required	需要目标地址。这个错误表明没有提供具体地址。比方说，假如在调用 sendto 时，将目标地址设为 INADDR_ANY（任意地址），便会返回这个错误。
10040 - WSAEMSGSIZE	Message too long	消息过长。这个错误的含义很多。如果在一个数据报套接字上发送一条消息，这条消息对内部缓冲区而言太大的话，就会产生这个错误。再比如，由于网络本身的限制，使一条消 息过长，也会产生这个错误。最后，如果收到数据报之后，缓冲区太小，不能接收消息时，也会产生这个错误。
10041 - WSAEPROTOTYPE	Protocol is wrong type for socket	套接字协议类型有误。在 socket 或 WSASocket 调用中指定的协议不支持指定的套接字类型。比如，要求建立 SOCK_STREAM 类型的一个 IP 套接字，同时指定协议为 IPPROTO_UDP，便 会产生这样的错误。
10042 - WSAENOPROTOOPT	Bad protocol option	协议选项错误。表明在 getsockopt 或 setsockopt 调用中，指定的套接字选项或级别不明、未获支持或者无效。
10043 - WSAEPROTONOSUPPORT	Protocol not supported	不支持的协议。系统中没有安装请求的协议或没有相应的实施方案。比如，如果系统中没有安装 TCP/IP，而试着建立 TCP 或 UDP 套

		接字时，就会产生这个错误。
10044 - WSAESOCKTNOSUPPORT	Socket type not supported	不支持的套接字类型。对指定的地址家族来说，没有相应的具体套接字类型支持。比如，在向一个不支持原始套接字的协议请求建立一个 SOCK_RAW 套接字类型时，就会产生这个错 误。
10045 - WSAEOPNOTSUPP	Operation not supported on socket	不支持的操作。表明针对指定的对象，试图采取的操作未获支持。通常，如果试着在一个不支持调用 Winsock 函数的套接字上调用了 Winsock 时，就会产生这个错误。比如，在一个数据报套接字上调用 accept 或 WSAAccept 函数时，就会产生这样的错误。
10046 - WSAEPFNOSUPPORT	Protocol family not supported	不支持的协议家族。请求的协议家族不存在，或系统内尚未安装。多数情况下，这个错误可与 WSAEAFNOSUPPORT 互换（两者等价）；后者出现得更为频繁。
10047 - WSAEAFNOSUPPORT	Address family unsupported by protocol	地址家族不支持请求的操作。对套接字类型不支持的操作来说，在试着执行它时，就会出现这个错误。比如，在类型为 SOCK_STREAM 的一个套接字上调用 sendto 或 WSASendTo 函数时，就会产生这个错误。另外，在调用 socket 或 WSASocket 函数的时候，若同时请求了一个无效的地址家族、套接字类型及协议组合，也会产生这个错误。
10048 - WSAEADDRINUSE	Address already in use	地址正在使用。正常情况下，每个套接字只允许使用一个套接字地址（例如，一个 IP 套接字地址由本地 IP 地址及端口号组成）。这个错误一般和 bind、connect 和 WSAConnect 这三个 函数有关。可在 setsockopt 函数中设置套接字选项 SO_REUSEADDR，允许多个套接字访问同一个本地 IP 地址及端口号
10049 -	Cannot assign	不能分配请求的地址。API 调用中指定的地址

WSAEADDRNOTAVAIL	requested address	对那个函数来说无效时，就会产生这样的错误。例如，若在 bind 调用中指定一个 IP 地址，但却没有对应的本地 IP 接口，便会产生这样的错误。另外，通过 connect、WSAConnect、sendto、WSASendTo 和 WSAJoinLeaf 这四个函数为准备连接的远程计算机指定端口 0 时，也会产生这样的错误。
10050 - WSAENETDOWN	Network is down	网络断开。试图采取一项操作时，却发现网络连接中断。这可能是由于网络堆栈的错误，网络接口的故障，或者本地网络的问题造成的。
10051 - WSAENETUNREACH	Host is unreachable	网络不可抵达。试图采取一项操作时，却发现目标网络不可抵达（不可访问）。这意味着本地主机不知道如何抵达一个远程主机。换言之，目前没有已知的路由可抵达那个目标主机。
10052 - WSAENETRESET	Network was reset	网络重设时断开了连接。由于“保持活动”操作检测到一个错误，造成网络连接的中断。若在一个已经无效的连接之上，通过 setsockopt 函数设置 SO_KEEPALIVE 选项，也会出现这样的错误。
10053 - WSAECONNABORTED	Software caused connection abort	软件造成连接取消。由于软件错误，造成一个已经建立的连接被取消。典型情况下，这意味着连接是由于协议或超时错误而被取消的。
10054 - WSAECONNRESET	Connection reset by peer	连接被对方重设。一个已经建立的连接被远程主机强行关闭。若远程主机上的进程异常中止运行（由于内存冲突或硬件故障），或者针对套接字执行了一次强行关闭，便会产生这样的错误。针对强行关闭的情况，可用 SO_LINGER 套接字选项和 setsockopt 来配置一个套接字。
10055 - WSAENOBUFS	No buffer space is supported	没有缓冲区空间。由于系统缺少足够的缓冲区空间，请求的操作不能执行。

10056 - WSAEISCONN	Socket is already connected Socket	套接字已经连接。表明在一个已建立连接的套接字上，试图再建立一个连接。要注意的是，数据报和数据流套接字均有可能出现这样的错误。使用数据报套接字时，假如事先已通过 connect 或 WSAConnect 调用，为数据报通信关联了一个端点的地址，那么以后试图再次调用 sendto 或 WSASendTo，便会产生这样的错误。
10057 - WSAENOTCONN	Socket is not connected Socket	套接字尚未连接。若在一个尚未建立连接的“面向连接”套接字上发出数据收发请求，便会产生这样的错误。
10058 - WSAESHUTDOWN	Can't send after socket shutdown	套接字关闭后不能发送。表明已通过对 shutdown 的一次调用，部分关闭了套接字，但事后又请求进行数据的收发操作。要注意的是，这种错误只会在已经关闭的那个数据流动方向 上才会发生。举个例子来说，完成数据发送后，若调用 shutdown，那么以后任何数据发送调用都会产生这样的错误。
10059 - WSAETOOMANYREFS	Too many references	参照太多，资源耗尽。在 UNIX 作业系统中的解释是指系统核心资源消耗殆尽。不过在 WinSock1.1 版的规格书中并没有任何一个函数会发生这种错误。
10060 - WSAETIMEDOUT	Connection timed out	连接超时。若发出了一个连接请求，但经过规定的时间，远程计算机仍未作出正确的响应（或根本没有任何响应），便会发生这样的错误。要想收到这样的错误，通常需要先在套接字上设置好 SO_SNDTIMEO 和 SO_RCVTIMEO 选项，然后调用 connect 及 WSAConnect 函数。
10061 - WSAECONNREFUSED	Connection refused	连接被拒。由于被目标机器拒绝，连接无法建立。这通常是由于在远程机器上，没有任何应用程序可在那个地址之上，为连接提供服务。

10062 - WSAELOOP	Too many levels of symbolic links	符号链接的层次太多。在 UNIX 作业系统中，这个错误的意思是指路径参考过多的符号链接。
10063 - WSAENAMETOOLONG	Name too long	名字太长。
10064 - WSAEHOSTDOWN	Host is down	主机关闭。这个错误指出由于目标主机关闭，造成操作失败。然而，应用程序此时更有可能收到的是一条 WSAETIMEDOUT（连接超时）错误，因为对方关机的情况通常是在试图建立一个连接的时候发生的。
10065 - WSAEHOSTUNREACH	Host in unreachable	没有到主机的路由。应用程序试图访问一个不可抵达的主机。该错误类似于 WSAENETUNREACH。
10066 - WSAENOTEMPTY	Socket not empty.	
10067 - WSAEPROCLIM	Too many processes.	进程过多。有些 Winsock 服务提供者对能够同时访问它们的进程数量进行了限制。
10068 - WSAEUSERS	Too many users	
10069 - WSAEDQUOT	DQ out error.	
10070 - WSAESTALE	Data is stale	用尽磁盘片配额。
10071 - WSAEREMOTE	Host in remote	
10091 - WSASYSNOTREADY	System is not ready	网络子系统不可用。调用 WSAStartup 时，若提供者不能正常工作（由于提供服务的基层系统不可用），便会返回这种错误。
10092 - WSAVERNOTSUPPORTED	Version is not supported	Winsock.dll 版本有误。表明不支持请求的 Winsock 提供者版本。
10093 - WSANOTINITIALISED	Not initialized	Winsock 尚未初始化。尚未成功完成对 WSAStartup 的一次调用。
10094 - WSADISCON	Graceful shutdown in progress.	
11001 - WSAHOST_NOT_FOUND	Host not found	主机没有找到。这个错误是在调用 gethostbyname 和 gethostbyaddr 时产生的，表明没有找到一个授权应答主机（Authoritative Answer Host）。
11002 - WSATRY_AGAIN	Try again	非授权主机没有找到。这个错误也是在调用 gethostbyname 和 gethostbyaddr 时产生的，表

		明没有找到一个非授权主机，或者遇到了服务器故障。
11003 - WSANO_RECOVERY	Non-recoverable error	遇到一个不可恢复的错误。这个错误也是在调用 gethostbyname 和 gethostbyaddr 时产生的，指出遇到一个不可恢复的错误，应再次尝试操作。
11004 -WSANO_DATA	No data record available	没有找到请求类型的数据记录。这个错误也是在调用 gethostbyname 和 gethostbyaddr 时产生的，指出尽管提供的名字有效，但却没有找到与请求类型对应的数据记录。

三、常见问题

1．一个完整的 VuGen 脚本（Winsocket TCP）

以下是用 Windows Sockets 协议 TCP 连接录制的完整登陆脚本。

```
/************   vuser_init 部分：*************/
#include "lrs.h"
vuser_init()
{
    lrs_startup(257);

    lrs_create_socket("socket0", "TCP", "RemoteHost=168.3.1.230:6666",   LrsLastArg);

    lrs_receive("socket0", "buf0", LrsLastArg);

    return 0;
}
/************   Action 部分：*************/
#include "lrs.h"
Action()
{

    lrs_send("socket0", "buf1", LrsLastArg);
```

```
	lrs_receive("socket0", "buf2", LrsLastArg);

	lrs_send("socket0", "buf3", LrsLastArg);

	lrs_receive("socket0", "buf4", LrsLastArg);

	lrs_send("socket0", "buf5", LrsLastArg);

	lrs_receive("socket0", "buf6", LrsLastArg);

	lrs_send("socket0", "buf7", LrsLastArg);

	lrs_receive("socket0", "buf8", LrsLastArg);

	lrs_send("socket0", "buf9", LrsLastArg);

	lrs_receive("socket0", "buf10", LrsLastArg);

	lrs_send("socket0", "buf11", LrsLastArg);

	return 0;
}
/************   vuser_end 部分：*************/
#include "lrs.h"
vuser_end()
{
	lrs_receive("socket0", "buf12", LrsLastArg);

	lrs_disable_socket("socket0", DISABLE_SEND);

	lrs_close_socket("socket0");

	lrs_cleanup();
```

```
	return 0;
}
/************   data.ws 部分：************/

;WSRData 2 1

recv    buf0 6348
	"\n"
	"This is cib230 running the TradeDesign 2 server with\n"
	"\n"
	"DOKA 5\n"
	"\n"
	"Tue Nov 6 09:04:46 BEIST 2007\n"
	"\n"
	"\xff\x00\x00\x18"
	"i"
	"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x11"
	"l"
	"\x00\x00\x00\x17\x00\x00"
	"Kvx 隉溝 o"
	"\x1b"
	"Kr 秋"
	"\xfe"
	"+"
	"\x18\x11"
	"販澴骸繾!\a 娟 l"
	"\xd9\x14\xa9"
	"%)蹅"
	/******由于篇幅限制，这里省略了 buf0 的大部分内容，实际长度为 6348Bytes******/
send    buf1 71
	"\x00\x00\x00"
	"C"
	"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x18\x00\x00\x00"
```

"\x00"

"旸 OSGMBHDEFAULTLOGIN"

"\x95"

" 暔 旸 EFAULT"

"\x95"

" "

"\x95"

"210286"

"\x95"

" "

"\x95"

"1"

"\x95"

recv buf2 81

"\x00\x00\x00"

"M"

"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x1a\x00\x00\x00"

"\x00"

"旸 OSGMBHDEFAULTLOGIN"

"\x95"

" 暔 旸 EFAULT"

"\x95"

" "

"\x95"

"210286"

"\x95"

" "

"\x95"

"1"

"\x95"

" 盼 UCCESS"

"\x95"

```
send    buf3 24
        "\x00\x00\x00\x14\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00"
        "\x1b\x00\x00\x00\x00"

recv    buf4 27
        "\x00\x00\x00\x17\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
        "\x1f\x00\x00\x00\x00"
        "PID"

send    buf5 32
        "\x00\x00\x00\x1c\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
        " "
        "\x00\x00\x00\x00"
        "PID 4060"

recv    buf6 751
        "\x00\x00\x00\x19\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
        "\x1e\x00\x00\x00\x00"
        "SWC 1"
        "\x00\x00\x02\xce\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
        "\x1f\x00\x00"
        "\n"
        "Qx 跁謁 n"
        "\xa3"
        "0"
        "\x10"
        "\a 皤"
        "\xe4\x04"
        "塭"
        "\xc6"
        "6"
        "\xf6\x01"
        "鯽"
        "\x1f\xb6"
```

```
"'═"
"\xd1\x00"
"堁|-"
"\x10"
"┻{靴 w 拐"
"\x8c"
"\tHI^  赕 d 其礴  亏拚類数 W 镛"
"\xfb\x1d"
"hkw("
"\x04"
"\t-\b"
"\xb4"
"\"%辦 o>]YN.祺"
"\xa7\x10\x01"
"伶\n"
/******由于篇幅限制，这里省略了 buf6 的大部分内容，实际长度为 751Bytes******/

send  buf7 29
"\x00\x00\x00\x19\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
" "
"\x00\x00\x00\x00"
"NIM "
"\x95"

recv  buf8 2283
"\x00\x00"
"\b]"
"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x1d\x00\x00\x1a"
"\xfc"
"x 跐櫹 R 闶"
"\x15"
"圇~姵 c 懤春[-橹璠"
"\xc5\x05"
"<C\r\f.p 扽 PE 蔗"
```

```
    "\xdb"
    "(W 朵"
    "\x92"
    "\fC 轈"
    "\x0f"
    "慦"
    /******由于篇幅限制，这里省略了 buf8 的大部分内容，实际长度为 2283Bytes******/
    "\x86\x1d"
    "\\2"
    "\x00\x00\x00\x86\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
    "\x1f\x00\x00\x00\x00"
    "MBOX POK 瞰僮髡敝骰 ┑ 讲怀晒Γ 砦笪"
    "\xaa"
    ": 0286(0286)柜员未签退,不能再签到                                                    "
    //上面一行为录制下来的出错信息。
    "  "

send  buf9 30
    "\x00\x00\x00\x1a\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
    " "
    "\x00\x00\x00\x00"
    "MBOX 1"

recv  buf10 27
    "\x00\x00\x00\x17\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
    "\x1f\x00\x00\x00\x00"
    "PID"

send  buf11 32
    "\x00\x00\x00\x1c\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
    " "
    "\x00\x00\x00\x00"
    "PID 4060"
```

```
recv    buf12 27
    "\x00\x00\x00\x17\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
    "\x1e\x00\x00\x00\x00"
    "EXI"
-1
```

2．VuGen 脚本格式说明

1) 在录制的脚本中可以发现，Windows Sockets 的 VuGen 脚本包括 vuser_init、Action、vuser_end 和 data.ws 四个部分，其中前三部分为动作部分，均包含头文件 lrs.h,格式为：

```
#include "lrs.h"
```

最后的 data.ws 为数据文件，会话期间所有的传输数据均被保存在该文件中。

2) 在典型的会话期间，将录制下列函数顺序：

lrs_startup	初始化 WinSock DLL
lrs_create_socket	初始化套接字
lrs_send	在数据报上或者向流套接字发送数据
lrs_receive	接收来自数据报或流套接字的数据
lrs_disable_socket	禁用套接字操作
lrs_close_socket	关闭打开的套接字
lrs_cleanup	终止 WinSock DLL 的使用

注：VuGen 在 Windows 上使用 Windows 套接字协议支持应用程序的录制和重播；而在 UNIX 平台上仅支持重播。

3) 数据文件 data.ws 的正确格式是：

```
;WSRData 2 1
/*****中间部分为静态缓冲区数据******/
-1
```

静态缓冲区数据包括 send 和 recv 两种，具体格式为：

数据类型 缓冲区标识符 缓冲区数据长度

数据内容

如：

send buf1 3	recv buf2 2
“abc”	“OK”

3. 脚本的参数化

由于会话期间所有的传输数据均保存在数据文件 data.ws 中，因此一般情况下我们对脚本的参数化主要是对该文件中相关数据的参数化。

在参数化的时候，我们要先找到需要进行参数化的数据，比如我们的登陆柜员（上例中为 210086），但是我们同时也发现数据文件中的内容大部分是不可识别的字符，这时该怎么办呢？其实我们仔细查找就可以发现，如果是数字或英文字符则一般是不会变成乱码的，这样就很简单了，我们只需要利用 Ctrl+F 来查找刚刚录制的时候输入的内容，比如在上例中我们用 Ctrl+F 来查找 210086 就可以在 buf1 中找到，利用同样的方法其它输入的内容也可以找到，这样就可以对其进行参数化了。

3．data.ws 中的 send 和 recv

以上述脚本（“一个完整的 VuGen 脚本”）为例在 buf1 中找到 210086 后，仔细的你就会发现在 buf2 中也有同样的内容，再仔细看一下，buf1 前的关键字是 send，而 buf2 的关键字是 recv，也就是说 buf1 的内容是我们要传送给远程服务器的内容，而 recv 的内容是我们录制时远程服务器返回的数据，也是我们在回放时的期望返回数据。因此一般情况下我们只需要参数化 send 中的相关内容就行了。

4．Mismatch 的问题

我们在回放脚本的时候经常会发现 Mismatch（不匹配）的警告信息：

Action.c(15): Mismatch (expected 80 bytes, 81 bytes actually received)

要讨论 Mismatch 的问题首先要明白 Mismatch 的匹配机制。通过本文档函数部分的介绍，我们知道 Mismatch 有两种匹配方式：长度匹配和内容匹配。所谓长度匹配就是当 lrs_receive 在接收到数据之后就会和我们期望的缓冲区数据进行长度（字节数）对比，如果实际接收到的数据长度不等于我们的期望值（recv buf2 81 中的最后一个数字），就会报出 Mismatch 警告信息。长度匹配是 Mismatch 缺省的匹配方式。

如果我们用 lrs_set_receive_option(Mismatch, MISMATCH_CONTENT); 来指定匹配方式为内容匹配，那么 lrs_receive 在接收到返回数据后将与期望的数据内容进行匹配对比，这时即使长度相同，如果内容不一样（比如实际接收到的是”a”,而期望数据是”A”）同样会报出 Mismatch 信息。

而在上例中 buf1 和 buf2 中都有柜员信息”210086”，如果是缺省的长度匹配，那么我们就只需要参数化 buf1 中的柜员信息就可以了；而如果我们将匹配方式改为内容匹配，那么就需要考虑对 buf2 中的柜员信息也要进行参数化了，否则在回放时就会报出 Mismatch 的警告信息。

这里要提醒的是，回放时出现 Mismatch 的警告信息不一定意味着回放失败，这是因为有些返回数据可能含有随机性的数据（比如受环境影响），这就需要对业务的熟悉和较多的经验来判断了。

5．lrs_receive 等待时间太长的问题

在回放脚本的时候经常会发现运行至 lrs_receive 处出现长时间等待的问题，这是由于 LoadRunner Winsocket 中对返回数据的接收超时机制引起的，有时候缺省的超时时间不是我们真正需要的，这时就可以利用 lrs_set_recv_timeout 和 lrs_set_recv_timeout2 两个函数对脚本进行优化。对这两个函数的使用方法参见前面的函数部分。

6．一个对上传数据处理的例子

[测试背景] XXX 银行 Tuxedo 客户端软加密性能测试，由于没有客户端脚本为手工编写。

[问题描述] 报文是开发方提供的业务测试报文。但在脚本执行过程中总是报出 Mismatch 信息，返回数据为 0。报文内容如下：

```
/****data.ws 的部分内容*****/
;WSRData 2 1
send buf0 144
    "                    /home/wbgs/gswsz/tuxedo_fileupload/test/uploadTest.wsz
                                                                                "
recv buf1 1
    "\0x00"
-1
```

[问题分析] 同样的报文在开发环境下发送上去是没问题的，可是通过脚本发送返回数据总是 0。最后让开发人员在后台查看交易日志，发现通过脚本送上去的报文中最后的空格全是 20（十六进制），而正确的报文应该是 00（十六进制）。在十六进制编辑器里面查看到以上报文尾部空字符确实是 20。

[解决思路] 在脚本执行 lrs_send 之前将所有的尾部空字符全部替换成十六进制的 00，确保报文正确。

[实现]

```
char *SrcBuf;
SrcBuf = lrs_get_static_buffer("socket0","buf1",0,144,NULL);
for(i=0;i<74;i++){
    SrcBuf[70+i] = 0x00;    //此处实现替换，0x 表示十六进制
}
```

[思考] 此例虽然实现了报文的正确传送，但是由于字符的替换是在 Action 中实现的，那么在脚本执行的时候此处必然会花费一定的时间，这样就降低了脚本的执行效率。我们知道 data.ws 中是可以处理十六进制字符的，那么我们就可以通过十六进制转换器将上述报文转换为十六进制，然后将尾部的 20 全部改为 00，再编辑成 data.ws 中可以识别的格式（如”\0x00\0x2d”）直接替换 buf0 的内容即可。通过这样处理可以大大提高脚本的执行效率。

7．从文件中读取数据到用户缓冲区

[问题描述] 我们在编写脚本的时候可能会遇到这样一种情况：就是我们要发送数据的数据源不在数据文件 data.ws 中，而是在一个已知的文件中，这样的话我们能不能直接从文件中读取数据然后发送到远端主机上呢？答案是肯定的，下面就为大家介绍一下实现方法。

[解决思路] 由于 LRS 中没有提供直接读取文件的方法，但是我们知道 LoadRunner 支持 C 语言语法，这样我们就可以使用 fopen 函数来读取文件，然后将文件内容保存到用户缓冲区，这样就可以利用函数 lrs_set_send_buffer 将读取的文件内容发送到远端主机。

[实现脚本]

本例要读取的文件是 file.tud，其内容如下：

```
F00144                                                    overn
D85939B811A510335D4CFE8935A5E71B
```

以上内容将通过替换 lrs_send 中的 buf0 被发送出去，data.ws 中 buf0 的内容为：

```
send buf0 102
    "F00144
D85939B811A510335D4CFE8935A5E71B"
```

脚本编写如下：

```
char *filename = "file.tud";
long fileStream;
char userBuffer[500];//必须大于要发送的内容长度
int count; //存放读取的文件内容长度
```

```
if((fileStream = fopen(filename,"r")) == NULL) {    //打开文件
    lr_output_message("Error:can't open the file: %s",filename);
    return -1;
}

while(!feof(fileStream)) {
    count = fread(userBuffer, sizeof(char), 500, fileStream);    //读取文件内容
    lr_output_message("%d read", count);    //打印读取的内容长度
    if (ferror(fileStream)) {    //检查是否有 I/O 错误
        lr_output_message("error while reading file %s",filename);
        break;
    }
}

if (fclose(fileStream))
    lr_error_message("Error while closing file %s",filename);

lr_output_message("content of the userBuffer is ***%s***",userBuffer);
//以上语句将保存到用户缓冲区的内容打印出来
lrs_create_socket("socket0","TCP","RemoteHost=168.3.1.169:7900",LrsLastArg);
//以上语句建立 socket 连接
lrs_set_send_buffer("socket0",userBuffer,count);
//以上语句指定下一个 lrs_send 将从 userBuffer 中读取内容并发送
lrs_send("socket0","buf0",LrsLastArg); //此处 buf0 将被 userBuffer 所替换
```

脚本执行结果：

```
Action.c(19): 97 read
Action.c(29): content of the userBuffer is ***F00144                    overn
D85939B811A510335D4CFE8935A5E71B***
Action.c(31): lrs_create_socket(socket0, TCP, ...)
Action.c(33): lrs_set_send_buffer(socket0, buf, 97)
Action.c(35): lrs_send(socket0, buf0)
```

8．10053 错误说明

我在脚本调试和执行过程中发现出现最多且不好定位的就是 10053 错误（错误信息为 Software caused connection abort），因此单独对此问题加以说明，仅供参考。调试中发现错误原因主要有以下几种：

1、 由于连接的远程端口状态异常可能导致该问题。这种情况只能等端口状态恢复正常即可。

2、 在并发的时候出现 10053 错误。原因为 Param List 中的 Select next row 未设置为 Unique，也可能由于柜员（参数）重复而导致该错误。解决办法为将 Param List 的 Select next row 设置为 Unique，并确保参数不重复。

3、 第三种原因比较特殊，问题出现是我在录制客户端时采用的密码是 6 位，开始回放时是正常的，后来由于密码重置全部设为了 1 位，结果再回放就出现了 10053 的错误。由于经验不足，因此花了很大的力气和时间调试，发现是这个问题后便把 data.ws 中相关 buf 的长度和密码全改了，结果还是报错。解决办法：重新录制。

9．编码方式

LoadRunner 编码方式分为：ASCII 码、EBCDIC 码。如果选择 Translation Tables（Tools\Recording Options）中“None”方式（缺省），就是 ASCII 编码；其他都是选择 EBCDIC 编码方式，比如 00250352。其实 Server 是用 0025 方式编码，Client 是用 0352 方式。

LoadRunenr 有自己的 ebcdic 字典，路径是“\ebcdic”。

EBCDIC 码：8 位编码，可表示 256 个字符。EBCDIC 是 Extended Binary Coded Decimal Interchange Code 之缩写，成为扩展式 2 进制 10 进数交换码或称扩展式 BCD 码。它是以左边 4 个区域位元(Zone bit)及右边 4 个数位位元合计 8 个位元组的资料码，一共可组合 2 的 8 次方=256 种组合方式。

四、本文档遗留问题

在本文档的整理过程中，对以下两个函数的使用方法不太明白，现列出如下但不限于此，有望大家共同探讨：

lrs_length_receive

lrs_length_send

五、知识链接

1．带外数据(out-of-band，OOB)

传输层协议使用带外数据(out-of-band，OOB)来发送一些重要的数据，如果通信一方有重要的数据需要通知对方时，协议能够将这些数据快速地发送到对方。为了发送这些数据，协议一般不使用与普通数据相同的通道，而是使用另外的通道。linux 系统的套接字机制支持低层协议发送和接受带外数据。但是 TCP 协议没有真正意义上的带外数据。为了发送重要协议，TCP 提供了一种称为紧急模式(urgent mode)的机制。TCP 协议在数据段中设置 URG 位，表示进入紧急模式，接收方可以对紧急模式采取特殊的处理。很容易看出来，这种方式数据不容易被阻塞，可以通过在我们的服务器端程序里面捕捉 SIGURG 信号来及时接受数据或者使用带 OOB 标志的 recv 函数来接受。

2．阻塞和非阻塞（blocking 和 non-blocking）

阻塞函数在完成其指定的任务以前不允许程序调用另一个函数。例如，程序执行一个读数据的函数调用时，在此函数完成读操作以前将不会执行下一程序语句。当服务器运行到 accept 语句时，而没有客户连接服务请求到来，服务器就会停止在 accept 语句上等待连接服务请求的到来。这种情况称为阻塞（blocking）。而非阻塞操作则可以立即完成。比如，如果你希望服务器仅仅注意检查是否有客户在等待连接，有就接受连接， 否则就继续做其他事情，则可以通过将 Socket 设置为非阻塞方式来实现。非阻塞 socket 在没有客户在等待时就使 accept 调用立即返回。

```
#include
#include
……
sockfd = socket(AF_INET,SOCK_STREAM,0);
fcntl(sockfd,F_SETFL,O_NONBLOCK)；
……
```

通过设置 socket 为非阻塞方式，可以实现"轮询"若干 Socket。当企图从一个没有数据等待处理的非阻塞 Socket 读入数据时，函数将立即返回，返回值为-1，并置 errno 值为 EWOULDBLOCK。但是这种"轮询"会使 CPU 处于忙等待方式，从而降低性能，浪费系统资源。而调用 select()会有效地解决这个问题，它允许你把进程本身挂起来，而同时使系统内核监听所要求的一组文件描述符的任何活动，只要确认在任何被监控的文件 描述符上出现活动，select()调用将返回指示该文件描述符已准备好的信息，从而实现了为进程选出随机的变化，而不必由进程本身对输入进行测试而浪费 CPU 开销。Select 函数原型为:

int select(int numfds,fd_set *readfds,fd_set *writefds，

fd_set *exceptfds,struct timeval *timeout);

其中 readfds、writefds、exceptfds 分别是被 select()监视的读、写和异常处理的文件描述符集合。如果你希望确定是否可以 从标准输入和某个 socket 描述符读取数据，你只需要将标准输入的文件描述符 0 和相应的 sockdtfd 加入到 readfds 集合中；numfds 的值 是需要检查的号码最高的文件描述符加 1，这个例子中 numfds 的值应为 sockfd+1；当 select 返回时，readfds 将被修改，指示某个文件 描述符已经准备被读取，你可以通过 FD_ISSSET()来测试。为了实现 fd_set 中对应的文件描述符的设置、复位和测试，它提供了一组宏：

FD_ZERO(fd_set *set)----清除一个文件描述符集；

FD_SET(int fd,fd_set *set)----将一个文件描述符加入文件描述符集中；

FD_CLR(int fd,fd_set *set)----将一个文件描述符从文件描述符集中清除；

FD_ISSET(int fd,fd_set *set)----试判断是否文件描述符被置位。

Timeout 参数是一个指向 struct timeval 类型的指针，它可以使 select()在等待 timeout 长时间后没有文件描述符准备好即返回。struct timeval 数据结构为：

```
struct timeval {
    int tv_sec; /* seconds */
    int tv_usec; /* microseconds */
};
```

3．TCP_NODELAY 和 TCP_CORK

这两个选项都对网络连接的行为具有重要的作用。许多 UNIX 系统都实现了 TCP_NODELAY 选项，但是，TCP_CORK 则是 Linux 系统所独有的而且相对较新；它首先在内核版本 2.4 上得以实现。此外，其他 UNIX 系统版本也有功能类似的选项，值得注意的是，在某种由 BSD 派生的系统上的 TCP_NOPUSH 选项其实就是 TCP_CORK 的一部分具体实现。

TCP_NODELAY 和 TCP_CORK 基本上控制了包的“Nagle 化”，Nagle 化在这里的含义是采用 Nagle 算法把较小的包组装为更大的帧。John Nagle 是 Nagle 算法的发明人，后者就是用他的名字来命名的，他在 1984 年首次用这种方法来尝试解决福特汽车公司的网络拥塞问题（欲了解详情请参看 IETF RFC 896）。他解决的问题就是所谓的 silly window syndrome ，中文称“愚蠢窗口症候群”，具体含义是，因为普遍终端应用程序每产生一次击键操作就会发送一个包，而典型情况下一个包会拥有一个字节的数据载荷以及 40 个字节长的包头，于是产生 4000%的过载，很轻易地就能令网络发生拥塞,。 Nagle 化后来成了一种标准并且立即在因特网上得以实现。它现在已经成为缺省配置了，但在我们看来，有些场合下把这一选项关掉也是合乎需要的。

现在让我们假设某个应用程序发出了一个请求，希望发送小块数据。我们可以选择立即发送数据或者等待产生更多的数据然后再一次发送两种策略。如果我们马上发送数据，那么交互性的以及客户/服务器型的应用程序将极大地受益。例如，当我们正在发送一个较短的请求并且等候较大的响应时，相关过载与传输的数据总量相比就会比较低，而且，如果请求立即发出那么响应时间也会快一些。以上操作可以通过设置套接字的 TCP_NODELAY 选项来完成，这样就禁用了 Nagle 算法。

另外一种情况则需要我们等到数据量达到最大时才通过网络一次发送全部数据，这种数据传输方式有益于大量数据的通信性能，典型的应用就是文件服务器。应用 Nagle 算法在这种情况下就会产生问题。但是，如果你正在发送大量数据，你可以设置 TCP_CORK 选项禁用 Nagle 化，其方式正好同 TCP_NODELAY 相反（TCP_CORK 和 TCP_NODELAY 是互相排斥的）。下面就让我们仔细分析下其工作原理。

假设应用程序使用 sendfile()函数来转移大量数据。应用协议通常要求发送某些信息来预先解释数据，这些信息其实就是报头内容。典型情况下报头很小，而且套接字上设置了 TCP_NODELAY。有报头的包将被立即传输，在某些情况下（取决于内部的包计数器），因为这个包成功地被对方收到后需要请求对方确认。这样，大量数据的传输就会被推迟而且产生了不必要的网络流量交换。

但是，如果我们在套接字上设置了 TCP_CORK（可以比喻为在管道上插入“塞子”）选项，具有报头的包就会填补大量的数据，所有的数据都根据大小自动地通过包传输出去。当数据传输完成时，最好取消 TCP_CORK 选项设置给连接“拔去塞子”以便任一部分的帧都能发送出去。这同“塞住”网络连接同等重要。

总而言之，如果你肯定能一起发送多个数据集合（例如 HTTP 响应的头和正文），那么我们建议你设置 TCP_CORK 选项，这样在这些数据之间不存在延迟。能极大地有益于 WWW、FTP 以及文件服务器的性能，同时也简化了你的工作。

4．什么是句柄

句柄是 WONDOWS 用来标识被应用程序所建立或使用的对象的唯一整数，WINDOWS 使用各种各样的句柄标识诸如应用程序实例，窗口，控制，位图，GDI 对象等等。WINDOWS 句柄有点象 C 语言中的文件句柄。

从上面的定义中的我们可以看到，句柄是一个标识符，是拿来标识对象或者项目的，它就象我们的姓名一样，每个人都会有一个，不同的人的姓名不一样，但是，也可能有一个名字和你一样的人。从数据类型上来看它只是一个 16 位的无符号整数。应用程序几乎总是通过调用一个 WINDOWS 函数来获得一个句柄，之后其他的 WINDOWS 函数就可以使用该句柄，以引用相应的对象。

如果想更透彻一点地认识句柄，我可以告诉大家，句柄是一种指向指针的指针。我们知道，所谓指针是一种内存地址。应用程序启动后，组成这个程序的各对象是住留在内存的。如果简单地理解，似乎我们只要获知这个内存的首地址，那么就可以随时用这个地址访问对象。但是，如果您真的这样认为，那么您就大错特错了。我们知道，Windows 是一个以虚拟内存为基础的操作系统。在这种系统环境下，Windows 内存管理器经常在内存中来回移动对象，依此来满足各种应用程序的内存需要。对象被移动意味着它的地址变化了。如果地址总是如此变化，我们该到哪里去找该对象呢?

为了解决这个问题，Windows 操作系统为各应用程序腾出一些内存储地址，用来专门登记各应用对象在内存中的地址变化，而这个地址(存储单元的位置)本身是不变的。Windows 内存管理器在移动对象在内存中的位置后，把对象新的地址告知这个句柄地址来保存。这样我们只需记住这个句柄地址就可以间接地知道对象具体在内存中的哪个位置。这个地址是在对象装载(Load)时由系统分配给的，当系统卸载时(Unload)又释放给系统。

句柄地址(稳定)→记载着对象在内存中的地址→对象在内存中的地址(不稳定)→实际对象

本质：WINDOWS 程序中并不是用物理地址来标识一个内存块，文件，任务或动态装入模块的，相反的，WINDOWS API 给这些项目分配确定的句柄，并将句柄返回给应用程序，然后通过句柄来进行操作。

但是必须注意的是程序每次从新启动，系统不能保证分配给这个程序的句柄还是原来的那个句柄，而且绝大多数情况的确不一样的。假如我们把进入电影院看电影看成是一个应用程序的启动运行，那么系统给应用程序分配的句柄总是不一样，这和每次电影院售给我们的门票总是不同的一个座位是一样的道理。

六、结束语

本文档第一、二两部分参考 LR 英文帮助文档，如有出入请以帮助文档为准。

对文档中没有说明的函数、纰漏或错误，我再次表示深深的歉意，也同时也真诚地希望各位能加以斧正，我会在以后加以补充修正，谢谢！

Email：tian.yuanwen@dcfs.digitalchina.com