VMMC Stream Sockets User's Guide (Windows NT)

Release 2.0

The Shrimp Project

Department of Computer Science
Princeton University

April 1999

About this Document

Welcome to VMMC Sockets! This document describes how to use the VMMC implementation of stream sockets on the Windows NT platform.

We welcome input regarding the VMMC Sockets implementation and this document. Please send your comments, bug reports, etc, to snd@cs.princeton.edu. For more information on The SHRIMP Project (including technical papers) please visit our web site at http://www.cs.princeton.edu/shrimp.

VMMC Sockets

1. Introduction

Vmmc-Sock is a user-level implementation of the stream sockets communication API. Working together with VMMC, it provides low-latency high-bandwidth communication to application programmers.

2. Installing VMMC Sockets

VMMC Sockets is distributed in the file `vmmc_sock_2_0.zip` which contains the following:

- the user-level libraries that implement VMMC Sockets

`wsock32-st.dll`	single-threaded
`wsock32-std.dll`	single-threaded with debug logging
`wsock32-mt.dll`	multi-threaded
`wsock32-mtd.dll`	multi-threaded with debug logging

- this document in two formats
  `vmmc_sock.pdf`
  `vmmc_sock.ps`

- sockets test programs

`sockperf.exe`	latency & bandwidth test
`sockgrind.exe`	exhaustive test
`sockcmd.exe`	communication pattern generator

- directory containing source code for the test programs
  `examples\`

3. Using VMMC Sockets

Using the Windows NT version of VMMC sockets is straightforward thanks to Windows' dynamic link libraries (DLL). It is not necessary to recompile applications. Simply copy one of the four supplied libraries into the same directory as the application (executable). The library must be renamed as `wsock32.dll`. When the application is run, it will automatically use VMMC sockets, as specified by the configuration file (described below).

4. Important Issues and Limitations

Supported Functions

Currently, only a small set of core functions of the Winsock API are implemented:

- `accept (SOCKET, struct sockaddr *, int *)`
- `bind (SOCKET, const struct sockaddr *, int)`
- `close (SOCKET)`

- `connect (SOCKET, const struct sockaddr *, int)`
- `ioctlsocket (SOCKET, long, ulong *)`
  Note: this call the only supported as:
  `ioctlsocket (sd, FIONREAD, &bytes_available)`
- `listen (SOCKET, int)`
- `recv (SOCKET, char *, int, int)`
- `send (SOCKET, const char *, int, int)`
- `shutdown (SOCKET, int)`
- `socket (int, int, int)`
- `select (int, fd_set *, fd_set *, fd_set *, const struct timeval *);`
  Note: select() works but is inefficient and best avoided for performance critical stuff. Faster support for select is under development.

Please note that the following are currently not supported:

- out-of-band data
- scatter/gather
- asynchronous I/O
- peeking into the internal socket buffers
- `setsockopt()`
- `fcntl()`
- `ioctl()`

The current set of supported functions is sufficient for many applications, we will support additional functions on request. Please email us at `snd@cs.princeton.edu` and let us know what you need.

Specifying VMMC Connections

A file named `vmmcsock.cfg` must exist in the same directory as the `wsock32.dll`. This configuration file is used to specify which sockets connections are to use VMMC. The file specifies one host per line followed by an optional list of port numbers. If no port numbers are specified for a particular host, then all connections to that host will use VMMC sockets. A sample configuration file looks like this:

```
# this is a comment
# sample vmmcsock.cfg file

# for the host sade we want only port 6500 and 9200
sade 6500 9200

# all connections to the following machines will use VMMC
u2
led
zeppelin
calvin
hobbes

# the \comment command is used to ignore groups of hosts
\comment start
```

```
alanis
clapton  10020 10021 10022
sting
\comment end
# more hosts…
```

The configuration file gives the user full control over which machines are used for VMMC. And it allows VMMC to coexist with kernel-level (Ethernet) sockets.

WARNING: all cluster nodes must use the same configuration file otherwise applications may run into problems. For example, a connection cannot be established if one node tries to use VMMC to connect to another node that is not using VMMC.

5. Sample Sockets Programs

Generally, all three of the socket examples provided run on two separate nodes, that is a server and a client. Because they do not spawn process on other nodes, these programs do not need to use `cfgvmmc` to run or set up the `VMMCHOSTS` environment variable. All they require is that the VMMC driver and the MCP are loaded properly and that the `wsock32.dll` is in same directory as the executable. These programs will also run without VMMC's `wsock32.dll` in the directory so that they can use normal winsock instead of vmmc-sockets.

- `sockperf:` tests the ping-pong bandwidth between client and server for specified iterations and one or more sizes. The syntax is following:
  server node: `sockperf –s`
  client node: `sockperf <servername> <iterations> <num_bytes>+`

- `sockgrind:` tests the byte alignment of communication between client and server. It should print "LOOKS GOOD" at the end if the connection is good. The syntax is:
  server node: `sockgrind –s`
  client node: `sockgrind <servername> <wraps> <start_nbytes> <end_nbytes> <increment>`

  The test starts with packet size `<start_nbytes>` and increases `<increment>` until it reaches `<end_nbytes>`. For each packet size, `<wraps>*64k` bytes is sent between the two nodes. Setting the increment to one allows for many alignment tests.

- `sockcmd:` takes 2 script file (one for client and one for server) to execute the communication pattern in the script files. The syntax is:
  server node: `sockcmd -s <script_file>`
  client node: `sockcmd <server_name> <script_file>`

  Each line of the script file should be in following format:

  ```
  send|recv <size> <repeat> <char>
  ```

  which means send (or receive) a packet with `<size>` bytes `<repeat>` times. Each packet is filled with the ASCII charater `<char>`. For example:
  "`recv 1024 10 A`" on server side can be matched with "`send 512 20 A`" or two "`send 512 10 A`" lines on client side.