How to use SocketPriority

Overview

The using of SocketPriority FIXEdge.properties session parameter is based on variety of ways of doing network I/O (input/output):

- synchronous non-blocking I/O with readiness notifications
- synchronous non-blocking I/O with busy polling
- synchronous blocking I/O

As well as various ways to perform an I/O operation:

- a thread from a pool
- a dedicated per session thread
- the current thread (the one calling "Session::put")

Dedicated threads allow lower response time, while using a pool provides higher throughput. So the optimal combination of "how to do I/O" and "what performs I/O (current thread/thread pool/dedicated thread)" choices depends on the application.

FIX engine allows to pick such a combination using non-blocking I/O on a per session basis and SocketPriority is the corresponding session parameter.

Valid values of SocketPriority parameter:

- EVEN (default) - share worker thread among all session in the Engine;
- AGGRESSIVE_SEND - use dedicated per session thread to send outgoing messages;
- AGGRESSIVE_RECEIVE - use dedicated per session thread to receive incoming messages;
- AGGRESSIVE_SEND_AND_RECEIVE - use dedicated per session threads to send and receive messages;
- DIRECT_SEND - use the current thread for sending, if this would block, performs as "EVEN".

Values

EVEN

Use thread pool for performing I/O for send and receive operation. The operations are distributed between the worker threads by a dispatcher thread which polls sockets and notifies worker threads when some socket is ready. In this mode FIX engine batches incoming/outgoing messages, i.e. several FIX messages can be processed in one send() system call. This is the default value.

Business scenario

It's recommended to use EVEN when there are many concurrent sessions ("many" means much more that the count of cores used), i.e. application demands **a high throughput rather than a low response time**. Allocating dedicated per session threads is not an option in such cases, so using a pool makes sense.

AGGRESSIVE

AGGRESSIVE_SEND

Tries to send the message from the current thread, if that fails (would block) delegates sending to a dedicated per session thread. Uses a combination of busy polling and readiness notifications. Intended for applications which need low response time (e.g. to send an order as soon as possible).

AGGRESSIVE_RECEIVE

Use a dedicated per session thread handles receiving. It uses I/O with busy polling and switches to readiness notifications if no data has been received during the certain configurable time interval. Intended for applications which need low response time (e.g. to receive and react as soon as possible).

AGGRESSIVE_SEND_RECEIVE

Use dedicated per session threads to send and receive messages;

Business scenario

Aggressive strategies of I/O are recommended to use when **minimum latency** is needed. But it can reduce the throughput.

DIRECT_SEND

Try sending from the current thread, if this would block, does the same thing as "EVEN", i.e. queues the message and performs the operation in a worker thread when the corresponding socket becomes ready. The dispatcher thread handles readiness notifications and notifies workers whenever some socket has data to send/receive.

Business scenario

It's recommended to use DIRECT_SEND when there are many concurrent sessions ("many" means much more that the count of cores used), but on the other hand application needs **a low(er) response time** rather than a high throughput.