RMR Using Tracing Data

The message which RMR places on the wire consists of a small header of meta-data, and the user application payload. Inside of the header is optionally a tracing data field, the size of which is user application controlled. The trace field is intended to carry information generated by a non-RMR library to measure information about the message as it passes from application to application. The nature of the trace data need not be actual trace data; it would be possible for an application to define it's own data layout such that it might be able to record one-way and round trip latency between the applications.

Summary

Briefly, these functions exist in RMR to manage the trace information:

rmr_init_trace()
Initialise tracing such that each subsequently allocated message buffer will have a trace area of the indicated size allocated by default.

rmr_set_trace()
Copy trace data from a user's buffer to a message, reallocating the message if the current trace area is not large enough.

rmr_get_trace()
Copy trace data from a message to a user supplied buffer.

rmr_trace_ref()
Get a pointer to the trace area in the message buffer for direct access.

rmr_get_trlen()
Return the length of the trace data allocated in a message (doesn't indicate if it was populated).

Trace Data Size

The trace data size is variable in order to allow for any tracing library, or the user application's own data, to fit. There are several ways to specify the length of the trace data reserved when a message header is allocated; these are described below.

Default Allocation

Once RMR has been initialised via a call to rmr_init(), and the user application has an RMR contex, the default trace data length may be set using the rmr_init_trace() function. This function accepts the RMR context and the default trace data length; following this call, all messages allocated will by default be created with the trace data field.

Allocate On Write

The rmr_set_trace() function accepts a message buffer, a pointer to the trace data, and the trace data length, then copies the trace data into the message. If the allocated space in the message is not large enough for the data the message is reallocated, and the trace space for the message is extended to allow the data to fit.

This is a single time allocation, and results in a message clone which can be inefficient as the payload bytes must also be copied. However, this may be the only option available if the user application must use the return to sender function on a message which does not have enough (any) trace data space allocated.

Setting Trace Data

The preferred method of writing trace data into a message is to use the rmr_set_trace() function. This accepts the data and copies it into the message. This is preferred as the RMR function ensures that there isn't a data overrun should more bytes be provided than the message can handle. (The function reallocates the message in this case.)

It is possible for the user application to get a direct reference (pointer) to the trace data and to use that to set the data. The risk is that should the user application cause a "buffer overflow" the results when attempting to send the message, or parse the message on receipt, might not be desired.

Reading Trace Data

The rmr_get_trace() function will copy the trace data from the provided message into a buffer provided by the user application. This is useful if the trace data must out live the message buffer that transported it in.

Trace Data Reference

While more risky, in some situations having a pointer to the trace data in the message header will be the optimal means to read or write the data. The rmr_trace_ref() function accepts a pointer to a message buffer, and returns a pointer to the trace data within that buffer. Optionally, the user application may provide a pointer to an integer which the function will populate with the trace data length. If there is no trace data in the message, a nil pointer is returned.

It is the user application's responsibility when writing directly to the trace data in a message buffer, that it does not write more bytes than the trace data length.

Trace Data Length

The rmr_get_trlen() accepts a pointer to an RMR message buffer and will return the length of the trace data contained.

Example

As an example, consider a pair of cooperating applications which exchange messages in a request/response manner and wish to track both one-way and round trip latencies. The applications might define a structure with four timestamp structs (e.g. struct timespec), and set the default trace length created in each message to be the size of the structure.

After message allocation, the sender would set the first timstamp at the desired point in the code, and send the message normally. When the message arrives, the receiving application would get the reference to the trace data from the message and populate the second, receipt, timestamp. Before replying to the message, the receiver would populate the third timestamp to track both the internal processing time, and to mark the start of the sending process. Finally, when the response arrives at the original sender, it does the same "receipt marking" into the fourth timestamp. The application is then free to examine and record the various latencies.