Interaction with the MME#

If you have enabled a log-level of three or higher (see the Table of Debug Commands below) then the MME will log most of the control interactions from its clients. The MME will log to slogger with major code 27 so you can extract it by dumping the whole slog, or by filtering based on this major code:

```
sloginfo [-m 27]
```

The MME will log incoming commands from clients as well as outgoing events. It will not necessarily log return codes, though usually errors will be logged an a lack of error logged can be taken to suggest success. Often the MME will be used in an "asynchronous" mode whereby it will return the function call before completing the requested action, therefor the event(s) that follow a call should be used to determine the state of the MME.

Here is an example of a log:

```
Jan 01 00:59:12.879.5 00027 00 MME:io_handle_msg(182): RCV -> MME_IOMSG_TYPE_TRKVIEW_UPDATE
Jan 01 00:59:12.981.5 00027 00 MME:ntfy_log_event(670): MME_EVENT_TRKSESSIONVIEW_INVALID(48)
Jan 01 00:59:13.067.5 00027 00 MME:ntfy_log_event(665): MME_EVENT_TRKSESSIONVIEW_UPDATE(48)
Jan 01 00:59:13.067.5 00027 00 MME:ntfy_log_event(660): MME_EVENT_TRKSESSIONVIEW_COMPLETE(48)
```

If you are familiar with the MME API it is usually fairly easy to determine which call was made based on the message received. In the above example mme_trksessionview_update() was called. The time when io_handle_msg() logs the "`RCV -> MME_IOMSG_TYPE_TRKVIEW_UPDATE`" is the time when the MME received this request. We can see that the MME reacted by delivering three events: an `MME_EVENT_TRKSESSIONVIEW_INVALID` event, followed by an `MME_EVENT_TRKSESSIONVIEW_UPDATE` and `MME_EVENT_TRKSESSIONVIEW_COMPLETE` event.

Many of the "unexpected" behaviours of the MME can be explained by studying the interaction of the MME and its client(s). Often it is as simple as an expected call was not made, or an unhandled event was delivered. Even if this isn't the case, seeing how the MME is interacting with the clients is an extremely valuable tool in diagnosing problems.

Examining a Core File#

This is general, but is useful in debugging an MME crash.

First, you must have unstripped, raw binaries on your host. These should not be binaries you have taken from the target, but rather from your build machine or directly from your binary repository. In addition to the binary you are debugging, you should be prepared with any shared objects the binary loads as well.

It is critical that the application binary and the libraries all match the version on the target, including and especially libc. You can use the output of "`use -i`" of all binaries to compare the versions on the target to the versions on your host. You may need all of the libraries that are loaded at the time of the crash, but gdb will let you know if you are missing any.

To see what libraries are loaded by a process use "`pidin mem`".

If you have any doubt about which version of a library is loaded, you will have to check the environment variables of the process. To see what the active environment variables are for a process use "`pidin env`".

Typically the library will be in the `LD_LIBRARY_PATH`. You should also look for `LD_PRELOAD`. If the library is loaded from an absolute path you can find out what that path is by running the binary again with the `DL_DEBUG` environment variable set. For example:

```
DL_DEBUG=1 /sbin/mme-generic -c /etc/mme.conf
```

Now, with the binary and the core start gdb. I'm examining a core file from an SH4 board so I'm using ntosh-gdb.

```
# ntosh-gdb ./mme-generic ./mme-generic.core
GNU gdb 5.2.1qnx-nto
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=ntosh"...

warning: exec file is newer than core file.
Error while mapping shared library sections:
libthirdparty_qdb_icu_3_5.so: No such file or directory.
Error while mapping shared library sections:
libthirdparty_icu_3_5.so: No such file or directory.
Reading symbols from /opt/qnx632/target/qnx6/shle/lib/libm.so.2...done.
Loaded symbols for /opt/qnx632/target/qnx6/shle/lib/libm.so.2
Reading symbols from /opt/qnx632/target/qnx6/shle/lib/libc.so.2...done.
Loaded symbols for /opt/qnx632/target/qnx6/shle/lib/libc.so.2
Error while reading shared library symbols:
libthirdparty_qdb_icu_3_5.so: No such file or directory.
Error while reading shared library symbols:
libthirdparty_icu_3_5.so: No such file or directory.
Reading symbols from /opt/qnx632/target/qnx6/shle/lib/libecpp-ne.so.4...done.
Loaded symbols for /opt/qnx632/target/qnx6/shle/lib/libecpp-ne.so.4
#0  0x70334bb8 in MsgReceive () from /opt/qnx632/target/qnx6/shle/lib/libc.so.2
(gdb)
```

As an example you'll notice that libthirdparty_qdb_icu_3_5.so is not found above. I've put a copy in /home/ryallen/src and will tell gdb to look there:

```
(gdb) set solib-search-path /home/ryallen/src:/home/ryallen/share
Error while mapping shared library sections:
libthirdparty_icu_3_5.so: No such file or directory.
Reading symbols from /opt/qnx632/target/qnx6/shle/lib/libm.so.2...done.
Loaded symbols for /opt/qnx632/target/qnx6/shle/lib/libm.so.2
Reading symbols from /opt/qnx632/target/qnx6/shle/lib/libc.so.2...done.
Loaded symbols for /opt/qnx632/target/qnx6/shle/lib/libc.so.2
Reading symbols from /home/ryallen/src/libthirdparty_qdb_icu_3_5.so...done.
Loaded symbols for /home/ryallen/src/libthirdparty_qdb_icu_3_5.so
Error while reading shared library symbols:
libthirdparty_icu_3_5.so: No such file or directory.
Reading symbols from /opt/qnx632/target/qnx6/shle/lib/libecpp-ne.so.4...done.
Loaded symbols for /opt/qnx632/target/qnx6/shle/lib/libecpp-ne.so.4
```

Now, if you were debugging a crash you could hopefully see the call stack using "bt". The core I've loaded isn't for a crash. But you can see that thread 1 was active at the time the core was created:

```
(gdb) info threads
  12 process 12  0x70334bb8 in MsgReceive () from /opt/qnx632/target/qnx6/shle/lib/libc.so.2
  11 process 11  0x70334e52 in MsgSend () from /opt/qnx632/target/qnx6/shle/lib/libc.so.2
  10 process 10  0x70334bb8 in MsgReceive () from /opt/qnx632/target/qnx6/shle/lib/libc.so.2
  9 process 9  0x70334bb8 in MsgReceive () from /opt/qnx632/target/qnx6/shle/lib/libc.so.2
  8 process 8  0x70334bb8 in MsgReceive () from /opt/qnx632/target/qnx6/shle/lib/libc.so.2
  7 process 7  0x703358e6 in SyncCondvarWait_r () from /opt/qnx632/target/qnx6/shle/lib/libc.so.2
  6 process 6  0x703358e6 in SyncCondvarWait_r () from /opt/qnx632/target/qnx6/shle/lib/libc.so.2
  5 process 5  0x70334c5c in MsgReceivePulse () from /opt/qnx632/target/qnx6/shle/lib/libc.so.2
```

```
  4 process 4  0x703358e6 in SyncCondvarWait_r () from /opt/qnx632/target/qnx6/shle/lib/libc.so.2
  3 process 3  0x70335096 in MsgSendv_r () from /opt/qnx632/target/qnx6/shle/lib/libc.so.2
  2 process 2  0x70334c5c in MsgReceivePulse () from /opt/qnx632/target/qnx6/shle/lib/libc.so.2
* 1 process 1  0x70334bb8 in MsgReceive () from /opt/qnx632/target/qnx6/shle/lib/libc.so.2
```

And if I switch to thread 3 I'll see the call stack:

```
(gdb) thread 3
[Switching to thread 3 (process 3)]#0  0x70335096 in MsgSendv_r ()
   from /opt/qnx632/target/qnx6/shle/lib/libc.so.2
(gdb) bt
#0  0x70335096 in MsgSendv_r () from /opt/qnx632/target/qnx6/shle/lib/libc.so.2
#1  0x7033a960 in devctl () from /opt/qnx632/target/qnx6/shle/lib/libc.so.2
#2  0x081c873c in sendint32 ()
#3  0x081c8ed0 in iom_send_command ()
#4  0x081b3d3e in msgcmd ()
#5  0x081b2390 in cc_main ()
```

Now, if this were a real crash you would what function the MME crashed in; you would also know how we got there. You can poke around with disassembling, looking at memory, and seeing what awaits you in the registers. Of course, now that you have the MME source you can also make life a little easier and build a debug variant of the MME.

Appendices#

Table of Debug Commands#

Component	Arguments
mme-generic	-v[vvvvvvvvv] -D[D]] (additional debug, output to console) at runtime: * mme_set_debug() * mme_get_logging() * mme_set_logging()
io-media-generic	-D[DDDDD] -S (stderr instead of slogger) at runtime: * echo setresource verbosity i32 4 ¦ testapp-cmdline -cdebug
io-fs-media	-v[vvvv]
iofs-ipod.so	-dipod,verbose=[1-9],[logfile=/path/to/filename] at runtime: * DCMD_MEDIA_CONFIG "verbose=[0-9]" * DCMD_MEDIA_CONFIG "logfile=/path/to/filename"
iofs-pfs.so	-dpfs,debug=[1-5]
qdb	-v[vvvvvv] -V (output to console)
mcd	-v[vvvvvv] -V (output to console)