LCM Communication Best Practices

Always collecAl Always take a log

Use lcm-logger to record a system's communications. Run lcm-logger continually while performing tests. Give the log file a descriptive name with the date. Save it where your team can share it.

Use lcm-spy / lcm-logger to troubleshoot to an application

Think of lcm-spy as a network debugger. Without type decoding you can use it to confirm a process is transmitting on the expected channel at the expected rate. With type decoding you can inspect transmitted values. In many cases it will allow you to troubleshoot issues to a specific program where you can finish off the bug using normal process debugging techniques.

Debug From Log Files

The best bug is one you can reproduce with a log file. If you collect a log and are able to trigger the bug using lcm-logplayer-gui then you can fix it, and more importantly, confirm the fix.

Make sure your custom types are always decodable

Make sure your custom types are always decodable by lcm-spy by creating a jar file and having it in your shell's CLASSPATH. Automate this by keeping your types in a specific location and having a script that rebuilds your jar file. Put this script in your PATH so you can call it from anywhere.

Make sure your host's clocks are synchronized

Use NTP or some other system to synchronize the clocks of all computers participating in an LCM system. If using NTP, remember that it takes a few minutes to servo your clock to agreement.

Prefer a stateless design

Make all the data your application needs to function correctly be part of an LCM transmission. Include timing information directly in the message – I.E. do not derive it using your host's clock.

This makes your processes more robust to communication dropouts as well as allows you to record communications and replay them at a later time and at a different rate (lcm-logplayer-gui has a message step function) than was originally captured while preserving the same functional behavior.

Initialize your types (C/C++ only)

For performance reasons C/C++ do not initialize the values in your lcm types when an instance is created on the stack (or heap). Make an init_type function in your application and use it to construct instances of your type with every member initialized. Only use this function to obtain an instance of your type. This is especially important with variable length arrays.

Article ID: 66

Source URL: http://localhost:8888/kb2017/lcm-communication-best-practices