QDebug
~~vs~~ and
other tracing facilities

ANR-15-CE25-0010

Arnaud.Clere @ MinMaxMedical.com

http://modmed.minmaxmedical.com/

Benchmark

Disclaimer: based on Qt 5.9.5, few HW tested, only Win10 and Linux

Linux:

xDebug to File	**qDebug+Ctxt**	**LTTng+Cbor**	**Tsv+Json**	**Json**
<< literal	16	3	16	60
<< atoms+map	23	58	88	152
<< float[]	26	25	48	104
xDebug to Console	**qDebug**	**~qDefaultMessa**	**Summary**	
<< literal	7	10	9	
<< atoms+map	15	54	67	
<< float[]	18	35	40	

Benchmark

Windows:

xDebug to File	**qDebug+Ctxt**	**ETL+Cbor**	**Tsv+Json**	**Json**
<< literal	6	4	19	49
<< atoms+map	13	52	90	162
<< float[]	17	21	46	97
xDebug to Console	**qDebug**	**~qDefaultMessa**	**Summary**	
<< literal	21	29	29	
<< atoms+map	31	67	85	
<< float[]	30	49	56	

Code: https://gricad-gitlab.univ-grenoble-alpes.fr/modmed/modmedLog

QDebug

Decent performance:

Win:

- **mDebug** to file of **metadata + format**: **6µsec**
- **ETL** tracepoint of **metadata + format**: **4µsec** (could be less…)

Linux:

- **mDebug** to file of **metadata + format**: **16µsec**
- **LTTng** tracepoint of **metadata + format**: **3µsec** (could be less…)

Only weird thing is the cost of 1st tracepoint:

- calls codecForName() > qt_create_qhash_seed() > loads a DLL
- ~50% of the time for 100 iterations
- But it will be amortized…
  (cost is presumably shared with any other Qhash, qrand, etc.)

QDebug

Manipulating QString seems Ok for console output

- no noticeable sampling of toLocal8Bit with our small benchmark

Sidenote:
Win Console is 2-3 times slower than file while Linux console is slightly faster

QString not so good for files

- Most file formats prefer utf8 (JSON, CBOR)
- Literals runtime encoding is more and more utf8
- QString in the middle is a little bit of a waste
- In a sampling of the previous benchmark:
  - qFormatLogMessage: ~50% (appeared twice out of 4 samples)
  - fromUtf8: ~50% (appeared twice out of 4 samples)

Statically Defined Tracepoints (LTTng, ETW)

Max performance... and **Qt Tracegen helps a lot!**

Before:

…

```
TP_FIELDS(
ctf_string(_source_path, logData->_source_path)
ctf_integer(int, _source_line, logData->_source_line)
ctf_string(_function, logData->_function)
ctf_string(_category, logData->_category)
ctf_string(_format, logData->_format)
…
```

After:

```
qlibraryprivate_load_entry(const QString &fileName)
qlibraryprivate_load_exit(bool success)
```

Statically Defined Tracepoints (LTTng, ETW)

But we are so lazy… or ignorant… or both ☺

And there are so many existing tracepoints !

+ usability limitations:

- More adapted to built-in types (complex types must be decomposed)
- System tools may complicate the management of traces

Structured Traces fill a gap

- Compatible with existing qDebug tracepoints
- Trace files easy to read using tools: Python, etc.

But one format does not fit all:

- XML, JSON, TSV, CBOR, Sqlite, your format…

… and flexibility comes with a performance cost …

Structured traces performance cost

	QDebug	BindableCopy	Json	Cbor
<< literal	0	0	2	1
<< atoms+map	5	3	41	35
<< float[]	8	0	24	15

We take a BindableCopy of each arg to later format it

Async formatting is good to disturb tracepoint as least as possible:

- Our preferred choice
- May not fit all needs (e.g. ETW, LTTng events out of sync)

Sidenote: BindableCopy copies args but with most tracepoint argument types, it is Ok (either small or COW)

Structured traces performance cost

First implementation: Stack-based => capture problems

Current implementation still aimed at Read+Write:

- check parse points, report errors

Full static implementation seems possible

- Quick prototype not fully static already 2.5 times faster
- Probably never faster than QDebug though…
- Except with: binary formats, better use of utf8 literals

In Qt’s context

Requires 1 new handler for messages

(*QtMessageHandler**Ex**)(..., const **QUtf8String**& format, const QVector<**BindableCopy**>& args);

=> Switch with #define ?! or chain at runtime ?!

Is our Bind interface ok for other general tasks ?!

```
CborWriter(&output).write().bind(JsonReader(&input).read());
```

=> Steps on a few feet: QVariant, std::any…