Greetings!

Leveraging Multi-Threading in C++11:

A Logged Case Study

Simple Logger

Features/Reqs

- Format-string-based printing
- Any number of args*
- Any arg type(s) the formatter supports
- Ability to determine output-verbosity based on configured level at runtime
- Storage to file

Simple Logger : Levels

```
enum class LogLevel
{
        Alert = 1,
        Error,
        Warn,
        Info,
        Trace,
        Verbose
};
```

Simple Logger : Usage

```
FILE* log = fopen("single_threaded_logger.txt", "w");
LogLevel log_level(LogLevel::Info);

// for some integers a and b
LOG(log, "V", "Message=%s;ParamX=%x;DecParam=%d;", "This is a message", a, b);
LOG_ALERT(log, log_level, "Message=%s;", "This is a message");
LOG_ERROR(log, log_level, "ParamX=%x;", a);
LOG_WARN(log, log_level, "DecParam=%d;", b);
LOG_INFO(log, log_level, "ParamX=%x;DecParam=%d;", a, b);
LOG_TRACE(log, log_level, "This won't print");
LOG_VERBOSE(log, log_level, "Neither will this, right? %d", 1);
```

Simple Logger : Implementation

```
#define LOG(log, level, fmt, ...) \
{ fprintf(log, "|" level "|" __FILE__ ":%d|" fmt "\n", __LINE__, __VA_ARGS__); }

#define LOG_ALERT(log, log_level, fmt, ...) \
if (log_level >= LogLevel::Alert) LOG(log, "A", fmt, __VA_ARGS__)

#define LOG_ERROR(log, log_level, fmt, ...) \
if (log_level >= LogLevel::Error) LOG(log, "E", fmt, __VA_ARGS__)

#define LOG_WARN(log, log_level, fmt, ...) \
if (log_level >= LogLevel::Warn) LOG(log, "W", fmt, __VA_ARGS__)

#define LOG_INFO(log, log_level, fmt, ...) \
if (log_level >= LogLevel::Info) LOG(log, "I", fmt, __VA_ARGS__)

#define LOG_TRACE(log, log_level, fmt, ...) \
if (log_level >= LogLevel::Trace) LOG(log, "T", fmt, __VA_ARGS__)

#define LOG_VERBOSE(log, log_level, fmt, ...) \
if (log_level >= LogLevel::Verbose) LOG(log, "V", fmt, __VA_ARGS__)
```

Methodology

Performance Measuring

Machine:

- Proc: 3.2 GHz AMD Phenom II x4 955 Black Edition Deneb
- Mem: 4x 2GB 240-Pin DDR3 SDRAM 1600 (PC3 12800)
- MB: GA-790FXTA-UD5

Windows 7 Home Premium (64-bit)

- HD: WD Black 1TB 7200 Sata internal

Ubuntu:

- HD: WD Caviar Blue 320 GB ?Speed? Sata internal

Methodology

```
// for some values a and b passed in through cmd line:
LOG(log, "V", "Message=%s;ParamX=%x;DecParam=%d;", "This is a message", a, b);
for (int i = 0; i < repeat_count; ++i)
        LOG_ALERT(log, log_level, "Message=%s;ParamX=%x;DecParam=%d;", "This is a
message", a, b);
LOG_ERROR(log, log_level, "Message=%s;ParamX=%x;DecParam=%d;", "This is a message", a,
b);
LOG_WARN(log, log_level, "Message=%s;ParamX=%x;DecParam=%d;", "This is a message", a,
b);
LOG_INFO(log, log_level, "Message=%s;ParamX=%x;DecParam=%d;", "This is a message", a, b);
LOG_TRACE(log, log_level, "Message=%s;ParamX=%x;DecParam=%d;", "This is a message", a,
b);
LOG_VERBOSE(log, log_level, "Message=%s;ParamX=%x;DecParam=%d;", "This is a message",
a, b);
```

Simple Logger : Results

		Min	Avg	Max	Std Dev
Windows	**printf**	1.762	1.856	1.937	0.038
Linux	**printf**	0.453	1.751	3.616	0.541

Naive Threading Logger

Naive Threading

- std::async() launches new thread, simply
- lambda can capture all needed data by value

Naive Threading Logger : Implementation

```
#include <future>

#define LOG(log, level, fmt, ...) { \
        int line = __LINE__; \
        std::async([=](){ \
                fprintf( \
                        log, \
                        "|" level "|" __FILE__ ":%d|" fmt "\n", \
                        line, \
                        __VA_ARGS__); \
        }); \
}
```

Naive Threading Logger : Results

		Min	Avg	Max	Std Dev
Windows	**printf**	1.762	1.856	1.937	0.038
	naive	3.799	4.412	6.259	0.428
Linux	**printf**	0.453	1.751	3.616	0.541
	naive	0.211	0.744	0.909	0.217

Naive Threading Logger : Fallout

Recognizing Changes

- Arguments must be copyable

Naive Threading Logger : Fallout

Recognizing Changes

- Arguments must be copyable
- Printing is possibly out-of-order

Naive Threading Logger : Fallout

Recognizing Changes

- Arguments must be copyable
- Printing is possibly out-of-order
- No guarantee that closing the file is safe

Naive Threading Logger : Fallout

Recognizing Changes

- Arguments must be copyable
- Printing is possibly out-of-order
- No guarantee that closing the file is safe
- Performance benefits are highly OS dependent

Naive Threading Logger : Next Steps

Naive Threading Logger : Next Steps

What Went Wrong?

- One thread per message loses ordering
- Creation of threads can be expensive

Naive Threading Logger : Next Steps

What Went Wrong?

- One thread per message loses ordering
- Creation of threads can be expensive
- Each thread adds potential resource contention

Naive Threading Logger : Next Steps

What Went Wrong?

- One thread per message loses ordering
- Creation of threads can be expensive
- Each thread adds potential resource contention
- Lambda has no knowledge of shared resource (file)

Naive Threading Logger : Next Steps

What Went Wrong?

- One thread per message loses ordering
- Creation of threads can be expensive
- Each thread adds potential resource contention
- Lambda has no knowledge of shared resource (file)
- Relying on OS (in)efficiencies

One Consumer

By using a single consumer thread, we can:

- Regain proper ordering
- Avoid needless contention
- Reduce variability
- Others?

But How?

Message Passing

Did you get the Memo?

- Messages capture all needed info (Still need to copy)
- Separates the Production of the info from Consumption

Message Passing

```
struct Message
{
        Message() {}
        virtual ~Message() {}
        virtual void print(std::ostream& os) const = 0;
};
```

Message Passing

```
template<int fmt_len, typename... ValuesTs>
struct MessageImpl : Message
{
        MessageImpl(char const (&fmt_)[fmt_len], ValuesTs... vals_)
                : vals(vals_...)
        { memcpy(fmt, fmt_, fmt_len); }

        char fmt[fmt_len];
        std::tuple<ValuesTs...> vals;

        void print(std::ostream& os) const { /* print code */ }
};
```

Message Passing

```
template<typename FormatterT, typename MsgT, int TotalIndices, int CurrIndex>
struct TupleFormatter
{
        static FormatterT operate(FormatterT fmt, MsgT const& msg)
        {
                return TupleFormatter<FormatterT, MsgT, TotalIndices, CurrIndex - 1>::
                        operate(fmt % std::get<TotalIndices - CurrIndex>(msg.vals), msg);
        }
};

template<typename FormatterT, typename MsgT, int TotalIndices>
struct TupleFormatter<FormatterT, MsgT, TotalIndices, 0>
{ static FormatterT operate(FormatterT& fmt, MsgT const& msg) { return fmt; } };

// for message impl
void print(std::ostream& os) const
{
        os      << TupleFormatter<
                        decltype(boost::format(fmt)),
                        decltype(*this),
                        std::tuple_size<decltype(vals)>::value,
                        std::tuple_size<decltype(vals)>::value>
                ::operate(boost::format(fmt), *this) << std::endl;
}
```

Queue-Based Logger : Interface

```
class OneQueueLogger
{
public:
        OneQueueLogger(std::string const& fname);
        ~OneQueueLogger();

        void startConsumption();
        bool hasFinished() const;

        template<int fmt_len, typename... ValuesTs>
        void handleProduced(
                char const (&fmt_)[fmt_len],
                ValuesTs... vals);
};
```

Queue-Based Logger : Interface 2

```
class OneQueueLogger
{
private:
        typedef std::vector<std::unique_ptr<Message>> queue_type;
private:
        void consumptionThreadFn();
        void endConsumption();
private:
        std::ofstream _outputFile; // write destination
        bool _finished; // Marks end of production
        std::unique_ptr<std::thread> _consumerThread;

        queue_type _queue;
        mutable std::mutex _queueMutex;
        std::condition_variable _queueHasData;

        bool _consumerFinished; // marks that consumption is done
};
```

Queue-Based Logger : Ctor

```
OneQueueLogger::OneQueueLogger(std::string const& str)
        : _outputFile(str)
        , _finished(false)
        , _consumerFinished(false)
{
}
// Second phase of initialization
void OneQueueLogger::startConsumption()
{
        _consumerThread.reset(new std::thread([=](){this->consumptionThreadFn();}));
}
```

Queue-Based Logger : Produce

```
template<int fmt_len, typename... ValuesTs>
void OneQueueLogger::handleProduced(
        char const (&fmt_)[fmt_len],
        ValuesTs... vals)
{
        {
                std::lock_guard<std::mutex> lock(_queueMutex);
                auto newMessage
                        = new MessageImpl<fmt_len, ValuesTs...>(fmt_, vals...);
                _queue.push_back(std::unique_ptr<Message>());
                _queue.back().reset(std::move(newMessage));
        }
        _queueHasData.notify_one();
}
```

Queue-Based Logger : Consume

```
inline void OneQueueLogger::consumptionThreadFn()
{
        while (true)
        {
                std::unique_lock<std::mutex> l(_queueMutex);
                if (_queue.empty())
                {
                        if (_finished)
                        { _consumerFinished = true; return; }
                        _queueHasData.wait(l);
                        if (_finished && _queue.empty())
                        { _consumerFinished = true; return; }
                }

                _queue | range::apply(
                        [=](const std::unique_ptr<Message>& msg) {
                                msg->print(_outputFile);
                });
                _queue.clear();
        }
        _consumerFinished = true;
}
```

Queue-Based Logger : Dtor

```
OneQueueLogger::~OneQueueLogger()
{
        endConsumption();
}

void OneQueueLogger::endConsumption()
{
        _finished = true;
        if (_consumerThread)
        {
                auto signaler = std::async([this](){
                        while (!this->_consumerFinished)
                        { this->_queueHasData.notify_one(); }
                });
                signaler.wait(); // Race condition in linux if this follows the join
                _consumerThread->join();
        }
}
```

Queue-Based Logger

```
#define LOG(log, level, fmt, ...) { \
        log.handleProduced( \
                "|" level "|" __FILE__ ":%d|" fmt, \
                __LINE__, __VA_ARGS__); \
}
```

Queue-Based Logger : Results

		Min	Avg	Max	Std Dev
Windows	**printf**	1.762	1.856	1.937	0.038
	naive	3.799	4.412	6.259	0.428
	queue	17.334	21.163	24.618	2.409
Linux	**printf**	0.453	1.751	3.616	0.541
	naive	0.211	0.744	0.909	0.217
	queue	0.568	14.663	38.382	9.587

Dual-Queue Logger : Results

		Min	Avg	Max	Std Dev
Windows	**printf**	1.762	1.856	1.937	0.038
	naive	3.799	4.412	6.259	0.428
	queue	17.334	21.163	24.618	2.409
	2queue	0.161	0.267	0.479	0.045
Linux	**printf**	0.453	1.751	3.616	0.541
	naive	0.211	0.744	0.909	0.217
	queue	0.568	14.663	38.382	9.587
	2queue	0.120	0.380	0.710	0.220

Lockless Logger : Results

		Min	Avg	Max	Std Dev
Windows	**printf**	1.762	1.856	1.937	0.038
	naive	3.799	4.412	6.259	0.428
	queue	17.334	21.163	24.618	2.409
	2queue	0.161	0.267	0.479	0.045
	lockless	0.121	0.135	0.168	0.008
Linux	**printf**	0.453	1.751	3.616	0.541
	naive	0.211	0.744	0.909	0.217
	queue	0.568	14.663	38.382	9.587
	2queue	0.120	0.380	0.710	0.220
	lockless	0.068	0.173	0.308	0.095