# Taming real-time logging

## Lessons learned from the trenches

Chris Apple

# About me

## Chris Apple - `cjappl`

- Lead Audio Software Engineer - Spatial Inc.
- 8 years experience in the audio industry
  - Dolby
  - Roblox
- Specialist in immersive audio.
  - Playback and content creation.


Why do we need a real-time logger?

# Motivation

## 1. Diagnostics

```
LOG_CRIT("Error occurred! Send help! %s", someErrorString);
```

## 2. Metrics

```
LogRenderStatistics(mAverageRenderTime, mPeakRenderTime);
```

## 3. User generated Lua code

```
RenderUserLua();
```


Version 0
The problem with simple printf logging

# Version 0: The problem with simple printf logging

```
1   void RealtimeLog(const char* format, ...)
2   {
3      va_list args;
4
5      va_start(args, format);
6      vprintf(format, args);
7      va_end(args);
8   }
9
10  int RealtimeCallback()
11  {
12     RealtimeLog("Hello %s. My Lucky number is %d", "World", 777);
13  }
```

# Version 0: The problem with simple printf logging

```
1   void RealtimeLog(const char* format, ...)
2   {
3      va_list args;
4
5      va_start(args, format);
6      vprintf(format, args);
7      va_end(args);
8   }
9
10  int RealtimeCallback()
11  {
12     RealtimeLog("Hello %s. My Lucky number is %d", "World", 777);
13  }
```

# Version 0: The problem with simple printf logging

```
1   void RealtimeLog(const char* format, ...)
2   {
3      va_list args;
4
5      va_start(args, format);
6      vprintf(format, args);
7      va_end(args);
8   }
9
10  int RealtimeCallback()
11  {
12     RealtimeLog("Hello %s. My Lucky number is %d", "World", 777);
13  }
```

# Real-time safety recap

No system calls

No allocations

No mutexes

# Real-time safety recap

No system calls

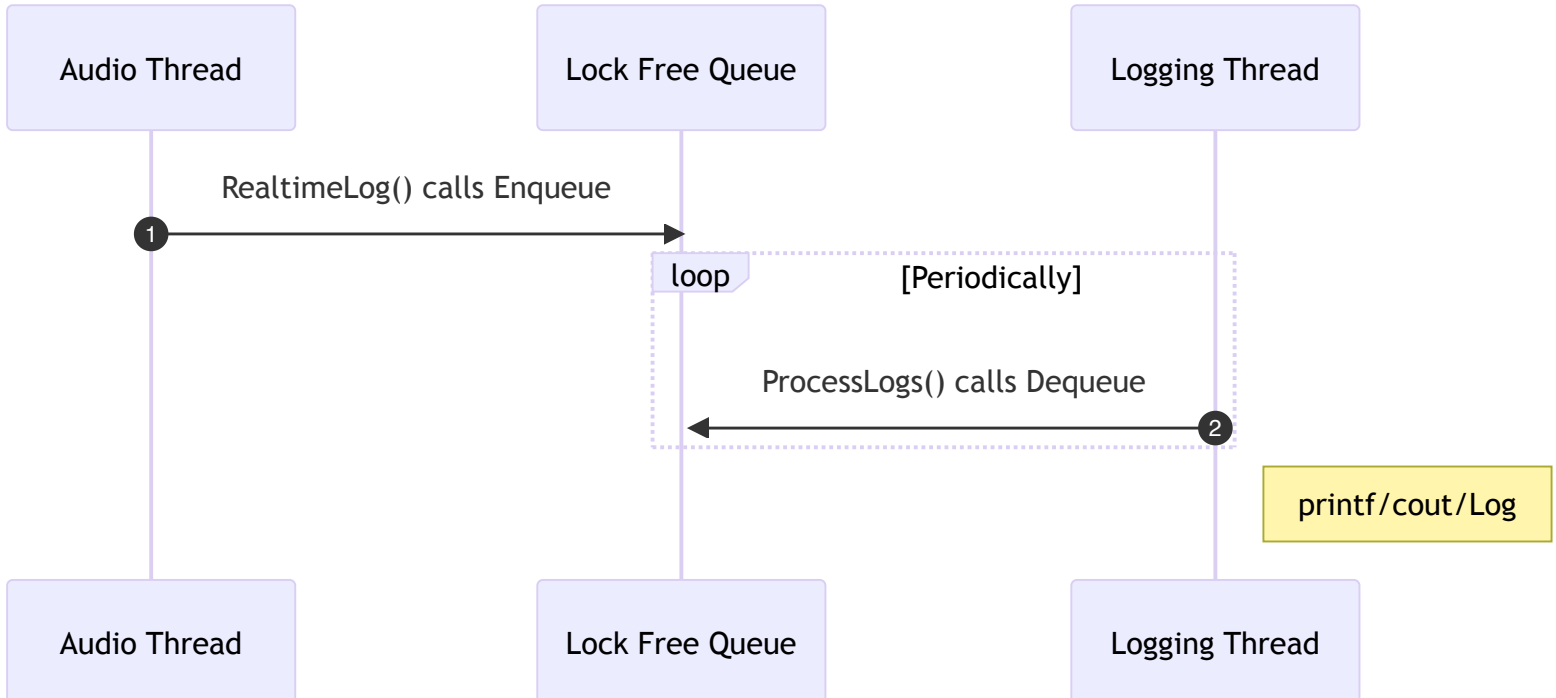
No allocations

No mutexes

```
void RealtimeLog(const char* format, ...)
{
   va_list args;

   va_start(args, format);
   vprintf(format, args);
   va_end(args);
}
```


Version 1
Using a logging thread

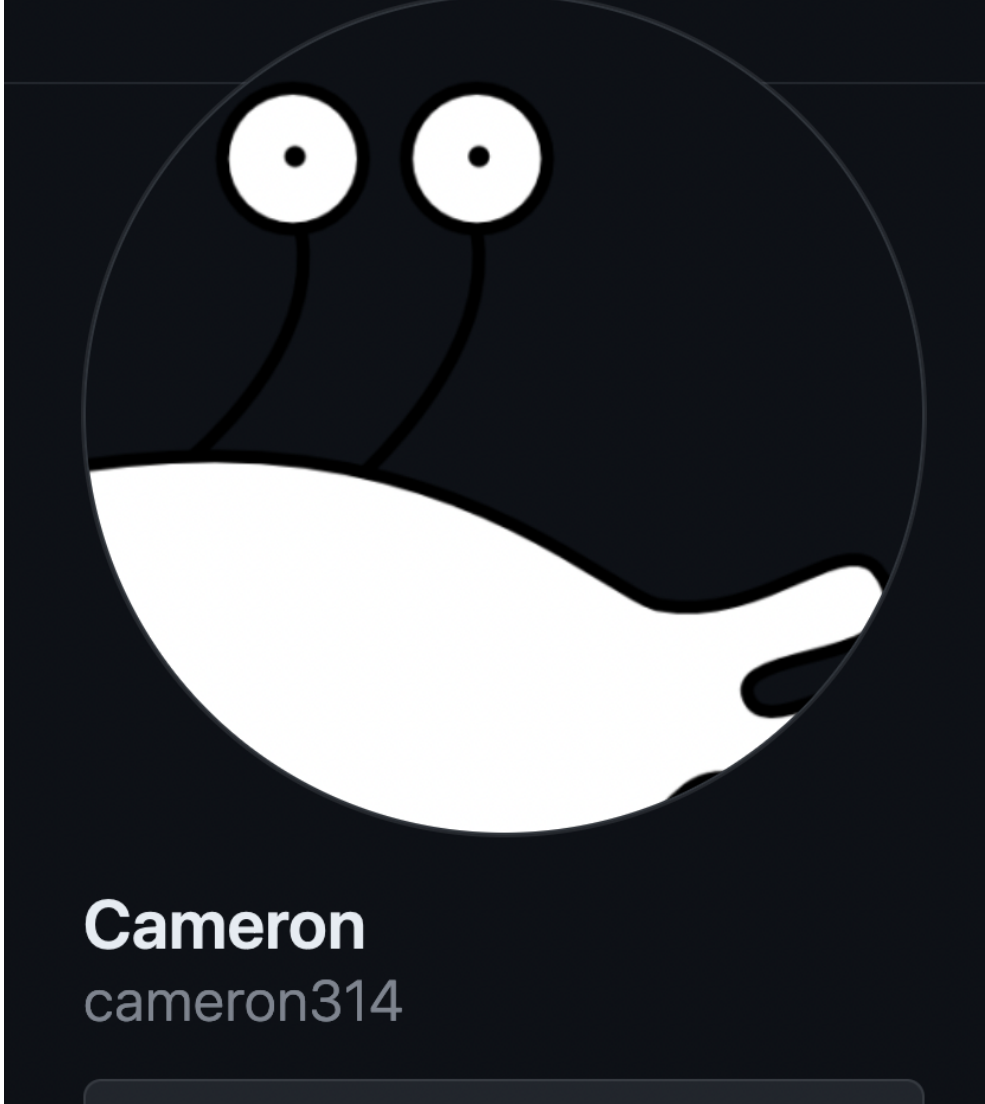
# Version 1: Logging thread with lock free queue



# A lock free queue

```cpp
using namespace moodycamel;

// Reserve space for 100 elements
ReaderWriterQueue<int> q{100};

// Try to enqueue (never allocates)
bool succeeded = q.try_enqueue(18);
assert(succeeded);
```

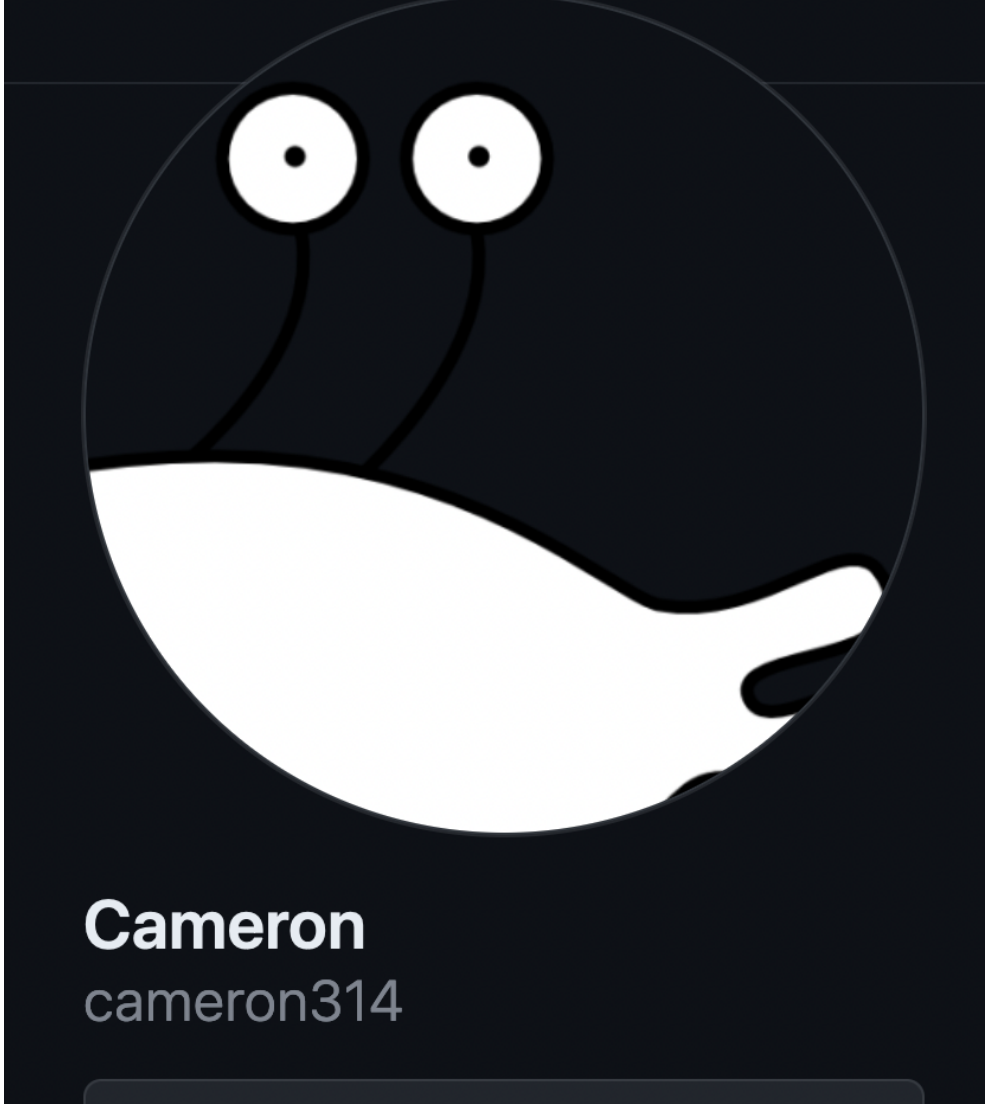
A Fast Lock-Free Queue for C++



# A lock free queue

```cpp
using namespace moodycamel;

// Reserve space for 100 elements
ReaderWriterQueue<int> q{100};

// Try to enqueue (never allocates)
bool succeeded = q.try_enqueue(18);
assert(succeeded);
```

A Fast Lock-Free Queue for C++



# Version 1: Logging thread with lock free queue

```
1   struct LoggingData
2   {
3      LogRegion region;
4      LogLevel  level;
5      char      message[MAX_MESSAGE_SIZE];
6   };
```

# Version 1: Logging thread with lock free queue

```
struct LoggingData
{
   LogRegion region;
   LogLevel  level;
   char      message[MAX_MESSAGE_SIZE];
};
```

```
void RealtimeLog(LogRegion region, LogLevel level, const char* format, ...)
{
   LoggingData data;
   data.region = region;
   data.level = level;

   .. va_args_nonsense ..
   vsnprintf(data.message, MAX_MESSAGE_SIZE, format, args);

   mLoggingQueue.try_enqueue(data);
}
```

# Version 1: Logging thread with lock free queue

```
1   struct LoggingData
2   {
3      LogRegion region;
4      LogLevel  level;
5      char      message[MAX_MESSAGE_SIZE];
6   };
```

```
1   void RealtimeLog(LogRegion region, LogLevel level, const char* format, ...)
2   {
3      LoggingData data;
4      data.region = region;
5      data.level = level;
6
7      .. va_args_nonsense ..
8      vsnprintf(data.message, MAX_MESSAGE_SIZE, format, args);
9
10     mLoggingQueue.try_enqueue(data);
11  }
```

# Version 1: Logging thread with lock free queue

```
struct LoggingData
{
   LogRegion region;
   LogLevel  level;
   char      message[MAX_MESSAGE_SIZE];
};
```

```
void RealtimeLog(LogRegion region, LogLevel level, const char* format, ...)
{
   LoggingData data;
   data.region = region;
   data.level = level;

   .. va_args_nonsense ..
   vsnprintf(data.message, MAX_MESSAGE_SIZE, format, args);

   mLoggingQueue.try_enqueue(data);
}
```

# Version 1: Logging thread with lock free queue

```
struct LoggingData
{
   LogRegion region;
   LogLevel  level;
   char      message[MAX_MESSAGE_SIZE];
};
```

```
void RealtimeLog(LogRegion region, LogLevel level, const char* format, ...)
{
   LoggingData data;
   data.region = region;
   data.level = level;

   .. va_args_nonsense ..
   vsnprintf(data.message, MAX_MESSAGE_SIZE, format, args);

   mLoggingQueue.try_enqueue(data);
}
```

# Version 1: Logging thread with lock free queue

```
void RealtimeLog(LogRegion region, LogLevel level, const char* format, ...)
{
   ...

   mLoggingQueue.try_enqueue(data);
}
```

```
void LoggingThread()
{
   while(true) {

       LoggingData data;
       while (mLoggingQueue.try_dequeue(data)) {
           std::cout << "[" << data.level << "] ";
           std::cout << "(" << data.region << ") ";
           std::cout <<       data.message;
           std::cout << '\n';
       }

   }
}
```

# Version 1: Logging thread with lock free queue

```
void RealtimeLog(LogRegion region, LogLevel level, const char* format, ...)
{
   ...

   mLoggingQueue.try_enqueue(data);
}
```

```
void LoggingThread()
{
   while(true) {

       LoggingData data;
       while (mLoggingQueue.try_dequeue(data)) {
           std::cout << "[" << data.level << "] ";
           std::cout << "(" << data.region << ") ";
           std::cout <<       data.message;
           std::cout << '\n';
       }

   }
}
```

# Version 1: Logging thread with lock free queue

```
void RealtimeLog(LogRegion region, LogLevel level, const char* format, ...)
{
   ...

   mLoggingQueue.try_enqueue(data);
}
```

```
void LoggingThread()
{
   while(true) {

       LoggingData data;
       while (mLoggingQueue.try_dequeue(data)) {
           std::cout << "[" << data.level << "] ";
           std::cout << "(" << data.region << ") ";
           std::cout <<       data.message;
           std::cout << '\n';
       }

   }
}
```

# Truncation and data loss

```
1   constexpr auto MAX_MESSAGE_SIZE = 512;   // WILL TRUNCATE ANOTHING MORE!
2   constexpr auto LOG_QUEUE_MAX_SIZE = 100; // WILL DROP ANY MESSAGES IF QUEUE IS FULL!
3
4   struct LoggingData
5   {
6      LogRegion region;
7      LogLevel  level;
8      char      message[MAX_MESSAGE_SIZE];
9   };
10
11  using LockFreeLoggingQueue = moodycamel::ReaderWriterQueue<LoggingData>;
12
13  LockFreeLoggingQueue mLoggingQueue { LOG_QUEUE_MAX_SIZE };
```

# Truncation and data loss

```
1   constexpr auto MAX_MESSAGE_SIZE = 512;   // WILL TRUNCATE ANOTHING MORE!
2   constexpr auto LOG_QUEUE_MAX_SIZE = 100; // WILL DROP ANY MESSAGES IF QUEUE IS FULL!
3
4   struct LoggingData
5   {
6      LogRegion region;
7      LogLevel  level;
8      char      message[MAX_MESSAGE_SIZE];
9   };
10
11  using LockFreeLoggingQueue = moodycamel::ReaderWriterQueue<LoggingData>;
12
13  LockFreeLoggingQueue mLoggingQueue { LOG_QUEUE_MAX_SIZE };
```

# Truncation and data loss

```
1   constexpr auto MAX_MESSAGE_SIZE = 512;   // WILL TRUNCATE ANOTHING MORE!
2   constexpr auto LOG_QUEUE_MAX_SIZE = 100; // WILL DROP ANY MESSAGES IF QUEUE IS FULL!
3
4   struct LoggingData
5   {
6      LogRegion region;
7      LogLevel  level;
8      char      message[MAX_MESSAGE_SIZE];
9   };
10
11  using LockFreeLoggingQueue = moodycamel::ReaderWriterQueue<LoggingData>;
12
13  LockFreeLoggingQueue mLoggingQueue { LOG_QUEUE_MAX_SIZE };
```

# Truncation and data loss

```
1   constexpr auto MAX_MESSAGE_SIZE = 512;   // WILL TRUNCATE ANOTHING MORE!
2   constexpr auto LOG_QUEUE_MAX_SIZE = 100; // WILL DROP ANY MESSAGES IF QUEUE IS FULL!
3
4   struct LoggingData
5   {
6      LogRegion region;
7      LogLevel  level;
8      char      message[MAX_MESSAGE_SIZE];
9   };
10
11  using LockFreeLoggingQueue = moodycamel::ReaderWriterQueue<LoggingData>;
12
13  LockFreeLoggingQueue mLoggingQueue { LOG_QUEUE_MAX_SIZE };
```

# All done??

No system calls

No allocations

No mutexes

# What’s wrong here?

```
1   void RealtimeLog(/* */)
2   {
3      LoggingData data;
4      data.region = region;
5      data.level = level;
6
7      .. va_args_nonsense ..
8      vsnprintf(data.message, MAX_MESSAGE_SIZE, format, args);
9
10     mLoggingQueue.try_enqueue(data);
11  }
```

# What’s wrong here?

```
1   void RealtimeLog(/* */)
2   {
3      LoggingData data;
4      data.region = region;
5      data.level = level;
6
7      .. va_args_nonsense ..
8      vsnprintf(data.message, MAX_MESSAGE_SIZE, format, args);
9
10     mLoggingQueue.try_enqueue(data);
11  }
```

# What's wrong here?

```
1   void RealtimeLog(/* */)
2   {
3      LoggingData data;
4      data.region = region;
5      data.level = level;
6
7      .. va_args_nonsense ..
8      vsnprintf(data.message, MAX_MESSAGE_SIZE, format, args);
9
10     mLoggingQueue.try_enqueue(data);
11  }
```

```
1   man 3 vsnprintf
```

> … **thousands and decimal separator returned by localeconv(3)**. …

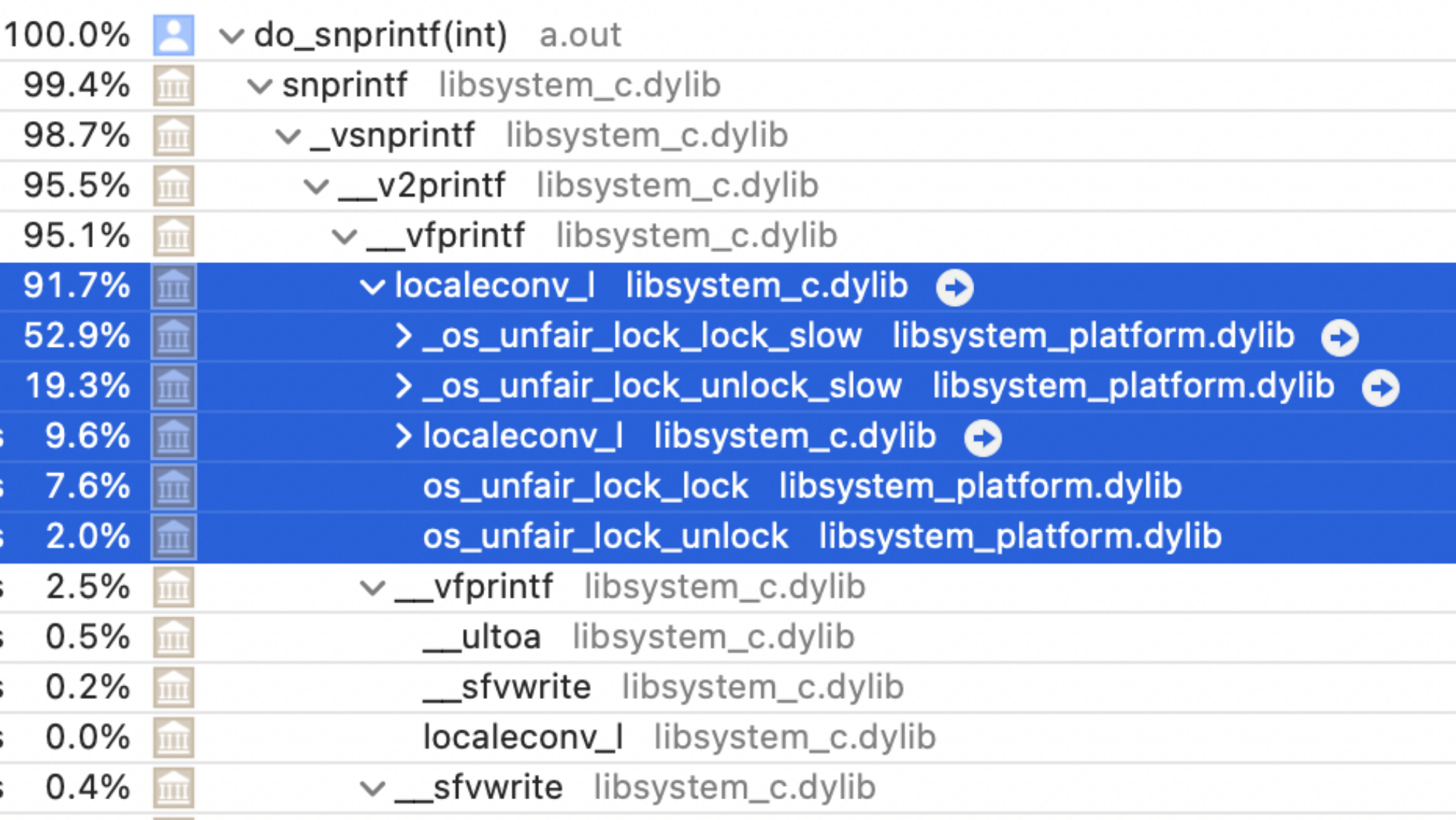
# What's wrong here?

```
1   void RealtimeLog(/* */)
2   {
3      LoggingData data;
4      data.region = region;
5      data.level = level;
6
7      .. va_args_nonsense ..
8      vsnprintf(data.message, MAX_MESSAGE_SIZE, format, args);
9
10     mLoggingQueue.try_enqueue(data);
11  }
```

```
1   man 3 vsnprintf
```

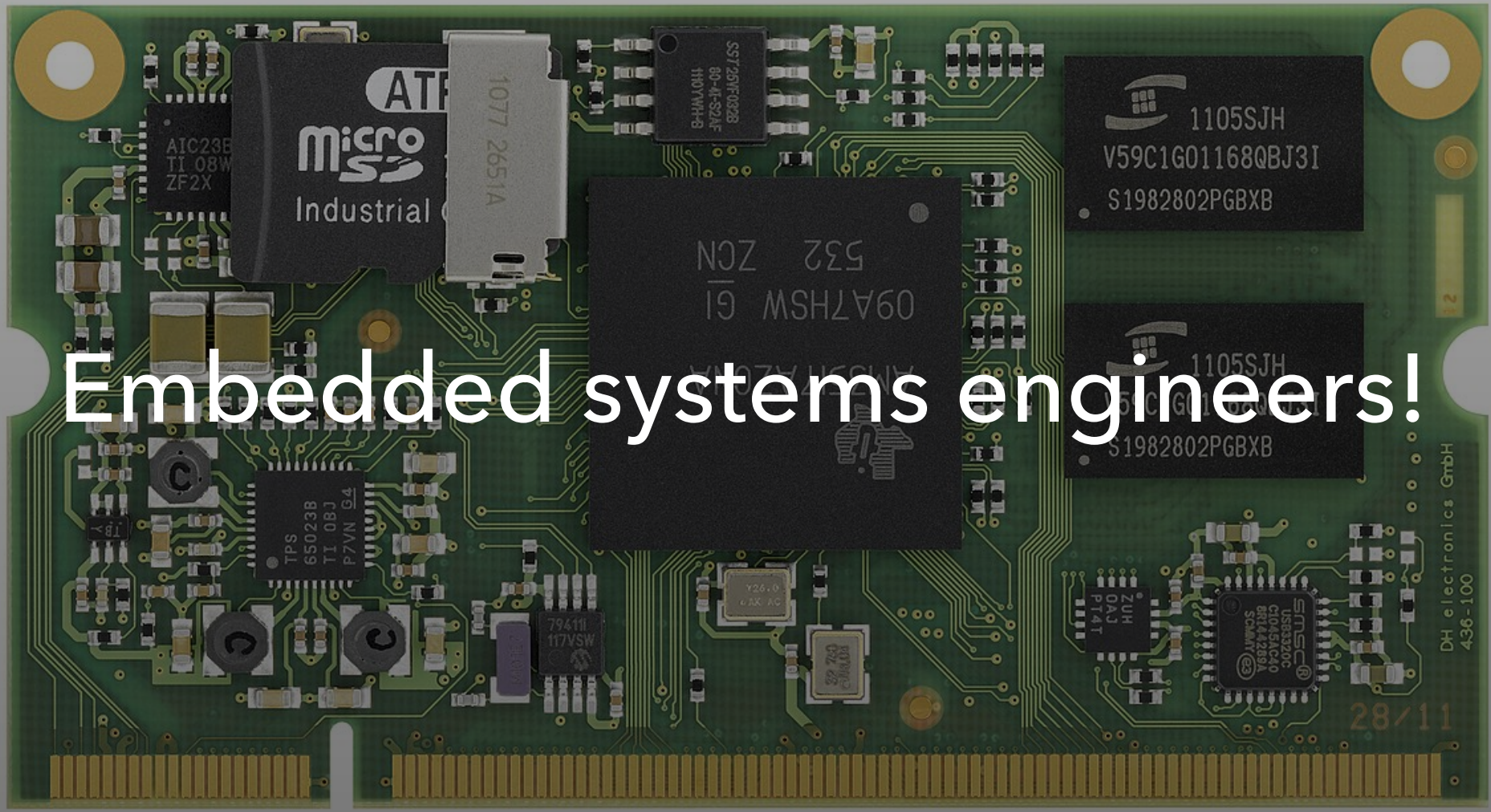
> … thousands and decimal separator returned by localeconv(3). …

```
1   1,234.56 -> 1.234,567 -> 1 234,567
```

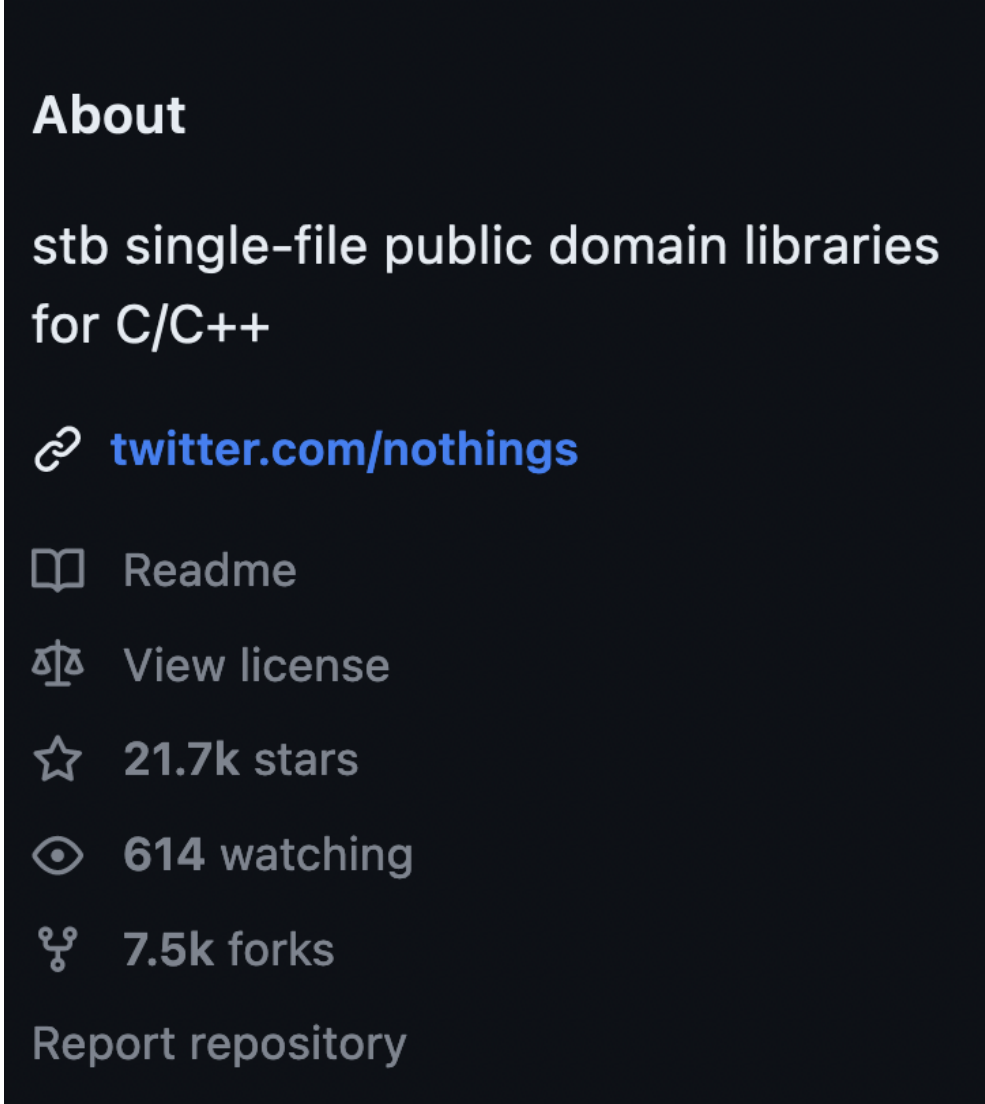

100.0% do_snprintf(int) a.out
99.4% snprintf libsystem_c.dylib
98.7% _vsnprintf libsystem_c.dylib
95.5% __v2printf libsystem_c.dylib
95.1% __vfprintf libsystem_c.dylib
91.7% localeconv_l libsystem_c.dylib
52.9% _os_unfair_lock_lock_slow libsystem_platform.dylib
19.3% _os_unfair_lock_unlock_slow libsystem_platform.dylib
9.6% localeconv_l libsystem_c.dylib
7.6% os_unfair_lock_lock libsystem_platform.dylib
2.0% os_unfair_lock_unlock libsystem_platform.dylib
2.5% __vfprintf libsystem_c.dylib
0.5% __ultoa libsystem_c.dylib
0.2% __sfvwrite libsystem_c.dylib
0.0% localeconv_l libsystem_c.dylib
0.4% __sfvwrite libsystem_c.dylib

# Log thread using std library snprintf functions


A group more paranoid than
audio software engineers?


Embedded systems engineers!

# `stb` - single file libraries



# `stb` - single file libraries

```
1  // in stb_sprintf.h
2  // for va_arg(), va_list()
3  #include <stdarg.h>
4
5   // size_t, ptrdiff_t
6  #include <stddef.h>
```

## About

stb single-file public domain libraries for C/C++

twitter.com/nothings

Readme

View license

**21.7k** stars

**614** watching

**7.5k** forks

Report repository

# Version 2: Using a third party vsnprintf

```
1
2   #define STB_SPRINTF_IMPLEMENTATION
3   #include "stb_sprintf.h"
4
5   void RealtimeLog(/* */)
6   {
7      ...
8
9      va_list args;
10
11     va_start(args, format);
12
13     //  vsnprintf(data.message, MAX_MESSAGE_SIZE, format, args);
14     stb_vsnprintf(data.message, MAX_MESSAGE_SIZE, format, args);
15
16     va_end(args);
17
18     mLoggingQueue.try_enqueue(data);
19  }
```


Is using va_args real-time safe?

# Yes!

```
1   void func (int a, ...)
2   {
3      // va_start
4      char *p = (char *) &a + sizeof a;
5
6      // va_arg
7      int i1 = *((int *)p);
8      p += sizeof (int);
9
10     // va_arg
11     long i2 = *((long *)p);
12     p += sizeof (long);
13  }
```

# Yes!

```
void func (int a, ...)
{
   // va_start
   char *p = (char *) &a + sizeof a;

   // va_arg
   int i1 = *((int *)p);
   p += sizeof (int);

   // va_arg
   long i2 = *((long *)p);
   p += sizeof (long);
}
```

# ...ish (?)

```
> man 3 va_args # https://linux.die.net/man/3/va_arg
...
Finally, on systems where arguments are passed in registers,
it may be necessary for va_start() to allocate memory
...
```

# Variadic Templates as an alternative to va_args

```
template<typename ...T>
void RealtimeLogFmt(/* */, T&&... args)
{
    ...
};
```

# Variadic Templates as an alternative to va_args

```
1   template<typename ...T>
2   void RealtimeLogFmt(/* */, fmt::format_string<T...> fmtString, T&&... args)
3   {
4       ...
5       fmt::format_to_n(data.message, MAX_MESSAGE_SIZE, fmtString, args...);
6
7       mLoggingQueue.try_enqueue(data);
8   };
9
10  RealtimeLogFmt(/* */, "Hello {}. My lucky number is {}", "world", 777);
```

Variadic Templates are Funadic - CppCon 2012

# Version 2: Logging thread + `stb_vsnprintf`

No system calls

No allocations

No mutexes

```
1   void RealtimeLog(LogRegion region, LogLevel level, const char* format, ...) {
2       LoggingData data;
3       data.region = region;
4       data.level = level;
5
6       va_list args;
7       va_start(args, format);
8       stb_vsnprintf(data.message, MAX_MESSAGE_SIZE, format, args);
9       va_end()
10
11      mLoggingQueue.try_enqueue(data);
12  }
```

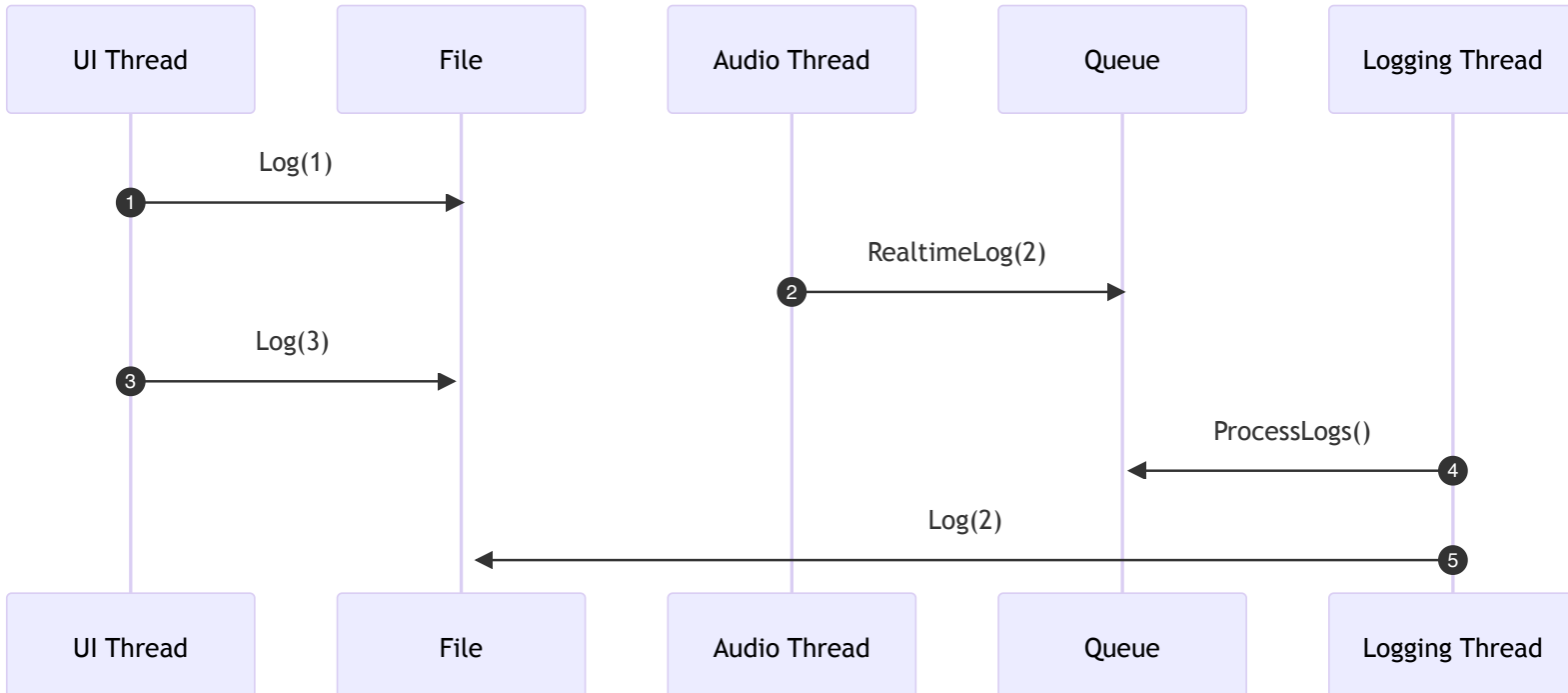

A note on ordering

# A note on ordering

```
void UserInterfaceThread()
{
    Log(1);

    Log(3);
}
```

```
void RealtimeCallback()
{

    RealtimeLog(2);

}
```

# A note on ordering



# A note on ordering

```
> ./testPrinter.out
UI   : 1
UI   : 3
AUDIO: 2
```

# Version 3: An atomic "sequence number"

```
1   #include <atomic>
2
3   static std::atomic<int> gSequenceNumber { 0 };
```

# Version 3: An atomic "sequence number"

```
#include <atomic>

static std::atomic<int> gSequenceNumber { 0 };
```

```
void Log(/* */) {
   PrintToFile(++gSequenceNumber, ...);
}
```

# Version 3: An atomic "sequence number"

```
#include <atomic>

static std::atomic<int> gSequenceNumber { 0 };
```

```
void Log(/* */) {
   PrintToFile(++gSequenceNumber, ...);
}
```

```
void RealtimeLog(/* */) {
   ...
   data.sequenceNumber = ++gSequenceNumber;
   mLoggingQueue.try_enqueue(data);
}
```

# Logging thread + `stb_vsnprintf` + seq number

No system calls

No allocations

No mutexes

## Relative ordering preserved!

```
{"region": "LOG  ", "severity": "INFO", "message": "...", "seq": 1}
{"region": "SPATL", "severity": "DEBG", "message": "...", "seq": 3}
{"region": "DEMON", "severity": "INFO", "message": "...", "seq": 4}
{"region": "DEMON", "severity": "INFO", "message": "...", "seq": 5}
{"region": "AUDIO", "severity": "INFO", "message": "...", "seq": 2}
```

# Implementation summary

## On initialization:

1. Create a lock-free queue containing type `LogData` - contains any custom type plus a `char` buffer.
2. Create a thread to periodically call `ProcessLog`.

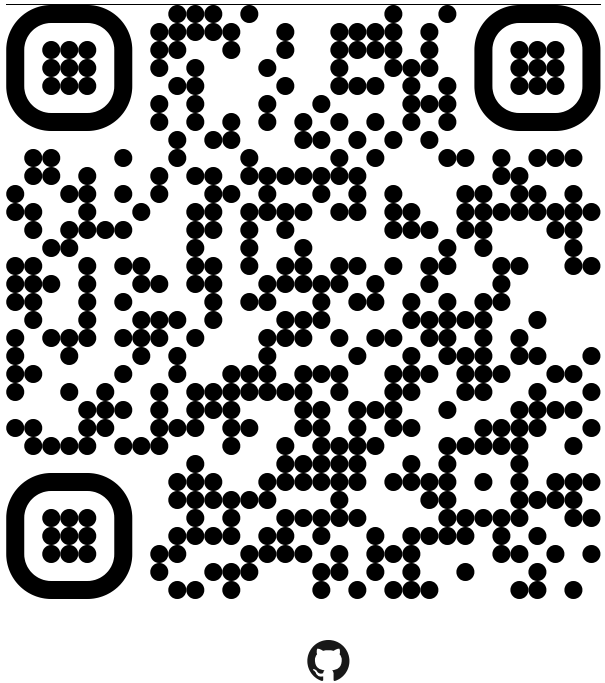
## In the `RealtimeLog` function:

1. Create a stack variable of type `LogData`.
2. Using a real-time safe `printf` family method, print your variable arguments into the buffer.
3. Fill in `LogData`'s sequence number with the next atomic sequence number to preserve ordering.
4. Try to enqueue the data.

## In the `ProcessLog` function:

1. Periodically dequeue the messages in the `LogData` queue, and Log them!

`cjappl/rtlog-cpp`



# What did we learn?

# What did we learn?

- Don’t use normal logging in your real-time thread.

# What did we learn?

- Don’t use normal logging in your real-time thread.
- A real-time logger is a lock-free queue that you can print messages into.

# What did we learn?

- Don’t use normal logging in your real-time thread.
- A real-time logger is a lock-free queue that you can print messages into.
- Beware the sneaky system calls to `localeconv` in standard `printf` family code.

# What did we learn?

- Don’t use normal logging in your real-time thread.
- A real-time logger is a lock-free queue that you can print messages into.
- Beware the sneaky system calls to `localeconv` in standard `printf` family code.
- Use sequence numbers to ensure your loggers have proper ordering.

# What did we learn?

- Don’t use normal logging in your real-time thread.
- A real-time logger is a lock-free queue that you can print messages into.
- Beware the sneaky system calls to `localeconv` in standard `printf` family code.
- Use sequence numbers to ensure your loggers have proper ordering.
- Beware the possibility of data loss using the real-time logger.

# What did we learn?

- Don’t use normal logging in your real-time thread.
- A real-time logger is a lock-free queue that you can print messages into.
- Beware the sneaky system calls to `localeconv` in standard `printf` family code.
- Use sequence numbers to ensure your loggers have proper ordering.
- Beware the possibility of data loss using the real-time logger.
- `va_args` is real-time safe on many platforms.
  - Variadic templates are probably the best general solution for all systems.
  - Also unlocks the use of `libfmt` - which is type-safe as well!

# Special thanks

## Reviewers

- Ryan Avery
- Palmer Hogen
- Eric Odland
- David O'Neal
- Matt Oshry

## Open source libraries

- `moodycamel/readerwriterqueue`
- `nothings/stb`

## Slides created in

- `slidevjs/slidev`


Appendix

# Variadic templates and `libfmt`

According to the author of `libfmt` you can use `format_to_n` safely with no allocations!

```
template<typename ...T>
void RealtimeLogFmt(LogRegion region, LogLevel level, fmt::format_string<T...> fmtString, T&&... args)
{
    NewLoggingData data;
    auto& buffer = data.message;

    ...
    fmt::format_to_n(buffer, MAX_MESSAGE_SIZE, fmtString, args...);

    mLoggingQueue.try_enqueue(data);
};

int main()
{
    RealtimeLogFmt(LogRegion::Network, LogLevel::Info, FMT_STRING("{} - {:.2f} - {}"), 42, 3.14, "hello");
    ProcessAndPrintLogs();
    return 0;
}
```

# Variadic templates and `libfmt`

According to the author of `libfmt` you can use `format_to_n` safely with no allocations!

```
template<typename ...T>
void RealtimeLogFmt(LogRegion region, LogLevel level, fmt::format_string<T...> fmtString, T&&... args)
{
    NewLoggingData data;
    auto& buffer = data.message;

    ...
    fmt::format_to_n(buffer, MAX_MESSAGE_SIZE, fmtString, args...);

    mLoggingQueue.try_enqueue(data);
};

int main()
{
    RealtimeLogFmt(LogRegion::Network, LogLevel::Info, FMT_STRING("{} - {:.2f} - {}"), 42, 3.14, "hello");
    ProcessAndPrintLogs();
    return 0;
}
```

# Variadic templates and `libfmt`

According to the author of `libfmt` you can use `format_to_n` safely with no allocations!

```
1   template<typename ...T>
2   void RealtimeLogFmt(LogRegion region, LogLevel level, fmt::format_string<T...> fmtString, T&&... args)
3   {
4       NewLoggingData data;
5       auto& buffer = data.message;
6
7       ...
8       fmt::format_to_n(buffer, MAX_MESSAGE_SIZE, fmtString, args...);
9
10      mLoggingQueue.try_enqueue(data);
11  };
12
13  int main()
14  {
15      RealtimeLogFmt(LogRegion::Network, LogLevel::Info, FMT_STRING("{} - {:.2f} - {}"), 42, 3.14, "hello");
16      ProcessAndPrintLogs();
17      return 0;
18  }
```

# Variadic templates and `libfmt`

According to the author of `libfmt` you can use `format_to_n` safely with no allocations!

```
template<typename ...T>
void RealtimeLogFmt(LogRegion region, LogLevel level, fmt::format_string<T...> fmtString, T&&... args)
{
    NewLoggingData data;
    auto& buffer = data.message;

    ...
    fmt::format_to_n(buffer, MAX_MESSAGE_SIZE, fmtString, args...);

    mLoggingQueue.try_enqueue(data);
};

int main()
{
    RealtimeLogFmt(LogRegion::Network, LogLevel::Info, FMT_STRING("{} - {:.2f} - {}"), 42, 3.14, "hello");
    ProcessAndPrintLogs();
    return 0;
}
```


CAUTION:
C++20 libfmt not guaranteed to be real-time safe!

# rtlog-cpp

```cpp
struct LogData
{
    LogLevel level;
    LogRegion region;
};

static auto PrintMessage = [](const LogData& data, size_t sequenceNumber, const char* fstring, ...)
                           __attribute__ ((format (printf, 4, 5)))
{
...
};

rtlog::Logger<LogData, MAX_NUM_LOG_MESSAGES, MAX_LOG_MESSAGE_LENGTH, gSequenceNumber> gRealtimeLogger;
rtlog::LogProcessingThread thread{gRealtimeLogger, PrintMessage, std::chrono::milliseconds(10)};

void SomeFunction()
{
    gRealtimeLogger.Log({LogLevel::Debug, LogRegion::Engine}, "Hello, %lu!", 123l);
}
```

# rtlog-cpp

```
Status Log(const LogData& inputData, const char* format, ...) __attribute__ ((format (printf, 3, 4))) {
    auto retVal = Status::Success;

    InternalLogData dataToQueue;
    dataToQueue.mLogData = inputData;
    dataToQueue.mSequenceNumber = ++SequenceNumber;

    va_list args;
    va_start(args, format);
    auto result = stbsp_vsnprintf(dataToQueue.mMessage.data(), dataToQueue.mMessage.size(), format, args);
    va_end(args);

    if (result < 0 || result >= dataToQueue.mMessage.size())
        retVal = Status::Error_MessageTruncated;

    // Even if the message was truncated, we still try to enqueue it to minimize data loss
    const bool dataWasEnqueued = mQueue.try_enqueue(dataToQueue);

    if (!dataWasEnqueued)
        retVal = Status::Error_QueueFull;

    return retVal;
}
```