

DispatchQueue.sync するときのスレッド

DispatchQueueでありがちなデッドロック

```
class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()

        importantSet()
    }
}

var value: Int = 0
func importantSet() {
    DispatchQueue.main.sync { Thread 1: EXC_BAD_INSTRUCTION (code=EX)
        value = 1
    }
}
```

軽い気持ちで呼んだ関数が内部で `DispatchQueue.sync` を使っていた

便利関数が用意される

```
func runOnMainThread(closure: () -> ()) {
    if Thread.isMainThread {
        closure()
    } else {
        DispatchQueue.main.sync {
            closure()
        }
    }
}
```

これでクラッシュしない！

```
class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()

        importantSet()
    }
}

var value: Int = 0
func importantSet() {
    runOnMainThread {
        value = 1
    }
}
```

めでたしめでたし

しかしここで疑問が

次のようなコードを実行するとき

```
func runOnMainThread(closure: () -> ()) {
    if Thread.isMainThread {
        closure()
    } else {
        DispatchQueue.main.sync {
            closure()
        }
    }
}

class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()

        let queue = DispatchQueue(label: "my.queue")
        queue.sync {
            runOnMainThread {
                print("Hello")
            }
        }
    }
}
```

```
func runOnMainThread(closure: () -> ()) {
    // 3. myqueueのスレッドにいるはずなのでelseに入るはず
    if Thread.isMainThread {
        closure()
    } else {
        // 4. myqueueはメインスレッドをロックしているのでデッドロック！？
        DispatchQueue.main.sync(execute: closure)
    }
}

class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()

        // 1. ここはメインスレッド
        let myqueue = DispatchQueue(label: "my.queue")
        myqueue.sync {
            // 2. ここはmyqueueのスレッド？
            runOnMainThread {
                print("Hello")
            }
        }
    }
}
```

しかしここで疑問が

- 実際に実行するとクラッシュしない
- この場合 `Thread.currentThread` は `true` を返す
- `myqueue` の中から呼んでいるのに、なぜ？
- (この時点で僕は勘違いをしています)

apple/swift-corelibs-libdispatch

apple / swift-corelibs-libdispatch

Watch 173 ⚡ Star 1,363 Fork 276

Code Pull requests 16 Projects 0 Insights

The libdispatch Project, (a.k.a. Grand Central Dispatch), for concurrency on multicore hardware <http://swift.org>

994 commits 38 branches 952 releases 51 contributors Apache-2.0

Branch: master ▾ New pull request Create new file Upload files Find file Clone or download ▾

File	Description	Time
ktooley-apple	Merge pull request #406 from adierking/printflike	Latest commit f6376cb 10 days ago
cmake	Linux i686 Build Support	14 days ago
dispatch	build: honour `BUILD_SHARED_LIBS`	24 days ago
libdispatch.xcodeproj	merge darwin/libdispatch-913.1.4	a year ago
man	build: remove autotools based build system	8 months ago
os	[gardening] Use https for swift.org links	7 months ago
private	Fix the signature of _dispatch_install_thread_detach_callback()	13 days ago
resolver	Import libdispatch-500.1.5	3 years ago
src	Merge pull request #397 from compnerd/static-stdlib	10 days ago
tests	Merge pull request #406 from adierking/printflike	10 days ago
tools	Merge libdispatch-913.1.4	a year ago
xcodeconfig	Merge libdispatch-913.1.4	a year ago
xcodescripts	Merge libdispatch-743	2 years ago

デッドロックを発生させてエラーからソースをたどる

The screenshot shows the assembly dump for Thread 1. The stack trace on the left lists 16 frames, starting from the current thread and going up to UIApplicationMain(). The assembly code on the right shows instructions from address 0x10babaea31 to 0x10babaea55. A red box highlights the instruction at address 0x10babaea55, which is an `ud2` instruction. To the right of the assembly, a message indicates that Thread 1 has received an EXC_BAD_INSTRUCTION signal.

```
Thread 1 Queue: com.apple.hread (serial) ⚠️
0 _DISPATCH_WAIT_FOR_QUEUE_
1 0x7ffee78af840
2 _dispatch_sync_f_slow
3 ViewController.viewDidLoad()
4 @objc ViewController.viewDidLoad()
5 -[UIViewController loadViewIfRe...
6 -[UIViewController view]
7 -[UIWindow addRootViewController...]
8 -[UIWindow _setHidden:forced:]
9 -[UIWindow makeKeyAndVisible]
10 -[UIApplication _callInitializatio...
11 -[UIApplication _runWithMainSc...
12 __111-[UICanvasLifecycleMo...
13 +[UICanvas _enqueuePostSetti...
14 -[_UICanvasLifecycleMonitor_C...
15 -[_UICanvasLifecycleMonitor_C...
16 __82-[UIApplicationCanvas _tr...

T1/    0x10babaea31 <+408>: popq   %r15
118    0x10babaea33 <+410>: popq   %rbp
119    0x10babaea34 <+411>: retq
120    0x10babaea35 <+412>: movq   %rcx, %rbx
121    0x10babaea38 <+415>: jmp    0x10babe8f5          ; <+92>
122    0x10babaea3d <+420>: leaq    0x274b4(%rip), %rcx      ; "BUG IN
                                                CLIENT OF LIBDISPATCH: dispatch_sync called on queue already
                                                owned by current thread"
123    0x10babaea44 <+427>: movq   %rbx, %rax
124    0x10babaea47 <+430>: movq   %rcx, 0x44c22(%rip)    ;
                                                gCRAnnotations + 8
125    0x10babaea4e <+437>: movq   %rax, 0x44c4b(%rip)    ;
                                                gCRAnnotations + 56
126  ->  0x10babaea55 <+444>: ud2
127

Thread 1: EXC_BAD_INSTRUCTION (code=...
```

- `dispatch_sync called on queue already owned by current thread` というエラー文を swift-corelibs-libdispatch から grep
 - → 出ない

- あった (_dispatch_sync_wait)

```
static void  
_dispatch_sync_wait(dispatch_queue_t top_dq, void *ctxt,  
                   dispatch_function_t func, uintptr_t top_dc_flags,  
                   dispatch_queue_t dq, uintptr_t dc_flags)  
{  
    pthread_priority_t pp = _dispatch_get_priority();  
    dispatch_tid tid = _dispatch_tid_self();  
    dispatch_qos_t qos;  
    uint64_t dq_state;  
  
    dq_state = _dispatch_sync_wait_prepare(dq);  
    if (unlikely(_dq_state_drain_locked_by(dq_state, tid))) {  
        DISPATCH_CLIENT_CRASH((uintptr_t)dq_state,  
                               "dispatch_sync called on queue "  
                               "already owned by current thread");  
    }  
}
```

- C++の文字列リテラルは空白や改行があれば結合されるので、そこで分かれていた

デッドロック判定処理をたどる

- `_dq_state_drain_locked_by` が判定しているっぽい

```
static inline bool  
_dq_state_drain_locked_by(uint64_t dq_state, dispatch_tid tid)  
{  
    return _dispatch_lock_is_locked_by((dispatch_lock)dq_state, tid);  
}
```

```
typedef uint32_t dispatch_tid;  
typedef uint32_t dispatch_lock;
```

```
#define DLOCK_OWNER_MASK ((dispatch_lock)0xfffffffffc)
```

```
static inline bool  
_dispatch_lock_is_locked_by(dispatch_lock lock_value, dispatch_tid tid)  
{  
    // equivalent to _dispatch_lock_owner(lock_value) == tid  
    return ((lock_value ^ tid) & DLOCK_OWNER_MASK) == 0;  
}
```

- Cにありがちなビット演算
- dq_state と tid の下位3~32bitが同一かどうかで比較されてる
- dq_state は DispatchQueue が管理している値っぽいので、tid が何か分かれば良さそう

_dispatch_sync_waitに戻る

```
static void
_dispatch_sync_wait(dispatch_queue_t top_dq, void *ctxt,
                     dispatch_function_t func, uintptr_t top_dc_flags,
                     dispatch_queue_t dq, uintptr_t dc_flags)
{
    pthread_priority_t pp = _dispatch_get_priority();
    dispatch_tid tid = _dispatch_tid_self();
    dispatch_qos_t qos;
    uint64_t dq_state;

    dq_state = _dispatch_sync_wait_prepare(dq);
    if (unlikely(_dq_state_drain_locked_by(dq_state, tid))) {
        DISPATCH_CLIENT_CRASH((uintptr_t)dq_state,
                               "dispatch_sync called on queue "
                               "already owned by current thread");
    }
}
```

- `tid` は `_dispatch_tid_self()` から來てるっぽい

```
#define _dispatch_tid_self() \
    ((dispatch_tid)_dispatch_thread_port())
```

```
#define _dispatch_thread_port() \
    pthread_mach_thread_np(_dispatch_thread_self())
```

```
#if defined(_WIN32)
#define _dispatch_thread_self() ((uintptr_t)GetCurrentThreadId())
#else
#if DISPATCH_USE_DIRECT_TSD
#define _dispatch_thread_self() ((uintptr_t)_dispatch_thread_getspecific( \
    _PTHREAD_TSD_SLOT_PTHREAD_SELF))
#else
#define _dispatch_thread_self() ((uintptr_t)pthread_self())
#endif
#endif
```

- ようやく見慣れた関数が出てくる (`pthread_self()`)
- OSによって分岐しているが、実態はpthreadのスレッドID（あるいは同等にみなせる代物）っぽい

_dispatch_sync_waitに戻る

```
static void
_dispatch_sync_wait(dispatch_queue_t top_dq, void *ctxt,
                     dispatch_function_t func, uintptr_t top_dc_flags,
                     dispatch_queue_t dq, uintptr_t dc_flags)
{
    pthread_priority_t pp = _dispatch_get_priority();
    dispatch_tid tid = _dispatch_tid_self();
    dispatch_qos_t qos;
    uint64_t dq_state;

    dq_state = _dispatch_sync_wait_prepare(dq);
    if (unlikely(_dq_state_drain_locked_by(dq_state, tid))) {
        DISPATCH_CLIENT_CRASH((uintptr_t)dq_state,
                               "dispatch_sync called on queue "
                               "already owned by current thread");
    }
}
```

- sync処理を行う対象のスレッドが現在のスレッドならデッドロックと判定しクラッシュさせている

**最初の疑問に戻り、なぜrunOnMainThreadは問題なさ
そうなのか**

```
func runOnMainThread(closure: () -> ()) {
    // 3. myqueueのスレッドにいるはずなのでelseに入るはず
    // → ではなく、メインスレッドのままなのでelseには入らない
    if Thread.isMainThread {
        closure() // 4. デッドロックしない
    } else {
        DispatchQueue.main.sync(execute: closure)
    }
}

class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()

        // 1. ここはメインスレッド
        let myqueue = DispatchQueue(label: "my.queue")
        myqueue.sync {
            // 2. ここはmyqueueのスレッド？
            // → ではなく、myqueue側がメインスレッドで動作している
            runOnMainThread {
                print("Hello")
            }
        }
    }
}
```

そもそもの勘違い

- DispatchQueue はそれぞれが専用のスレッドを持っていてその中でのみ動作するものだと思いこんでいた
 - これだとconcurrent queueは何者やねん！ってなる

最後に

- 事の発端となった `DispatchQueue.main.sync` だが、そもそもこれをしなきゃいけない場面はほぼないはず
- 適切に `DispatchQueue` (や、`Lock`) を使っていこう