**Многопоточность. Конспект статьи Habr.**

В iOS программировании многопоточность предоставляется в виде нескольких инструментов: Thread, Grand Central Dispatch (сGCD) и Operation — и используется с целью увеличения производительности и отзывчивости пользовательского интерфейса.

Thread (поток) - низкоуровневый механизм.

GCD – центральный диспетчер задач.

Operation - объектно-ориентированное API, построенное поверх GCD

**Очередь**

Основным понятием, которым оперируют инструменты многопоточности в ios является очередь (queue).

В очередь на выполнение выстраиваются замыкания (closure).

Очереди (queues) следуют FIFO паттерну (First In, First Out), кто первым поставлен в очередь, будет первым направлен на выполнение.

Суть в том, что собой представляют очереди в смысле выполнения заданий по отношению друг к другу:

- последовательное

- параллельное

и с помощью какого метода:

- синхронного

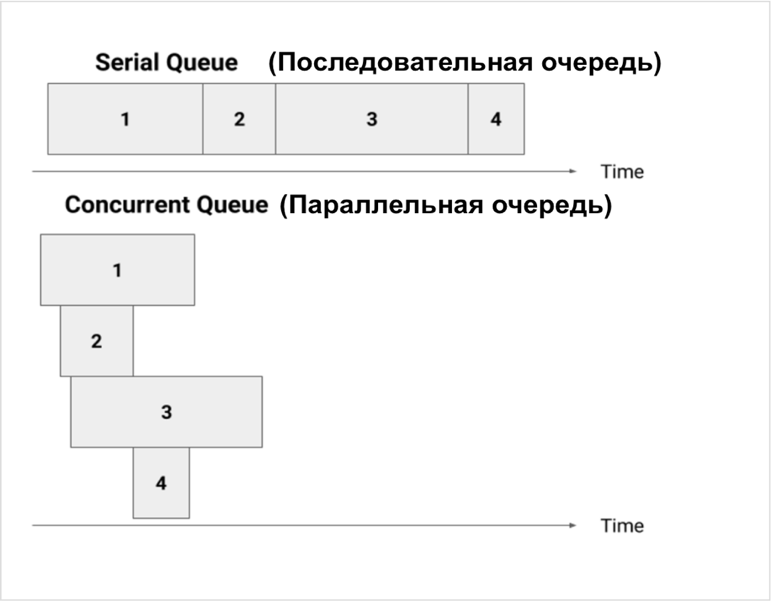
- асинхронного

задания помещаются в очередь, тем самым блокируя или не блокируя текущую очередь.

**Последовательные (serial) и параллельные (concurrent) очереди**

*Последовательная очередь (serial queues)* - когда задача (замыкание), которая находится на вершине очереди, запускается iOS и работает до тех пор, пока не закончится, затем вытягивается следующий элемент из очереди и т.д. последовательно.

*Параллельная очередь (concurrent queues)* – когда iOS запускает замыкание, находящееся на вершине очереди, на выполнение в определенном потоке. Если у системы еще есть ресурсы, то она берет следующий элемент из очереди и запускает его на выполнение в другом потоке в то время, пока первая функция еще работает, т.е. параллельно ей.



На serial очереди в любой текущий момент времени выполняется только одно задание, а на concurrent число заданий в любой текущий момент времени может меняться.

**Синхронное и асинхронное выполнение заданий**

Как только очередь (queue) создана, задание на ней можно разместить с помощью двух методов: sync — синхронное выполнение по отношению к текущей очереди и async — асинхронное выполнение по отношению к текущей очереди.

*Синхронный метод sync* возвращает управление на текущую очередь только после полного завершения задания, тем самым блокируя текущую очередь.

*Асинхронный метод async*, в противоположность функции sync, возвращает управление на текущую очередь немедленно после запуска задания на выполнение в другой очереди (параллельной или последовательной), не ожидая его завершения. Таким образом, асинхронная функция async не блокирует выполнение заданий на текущей очереди.

**Глобальные очереди iOS**

1. *Последовательная очередь Main queue*, в которой происходят все операции с пользовательским интерфейсом (UI). Эта очередь имеет наивысший приоритет среди глобальных очередей.
2. *4 фоновых (concurrent) параллельных глобальных очереди* с разным качеством обслуживания qos и разными приоритетами:
3. *userInteractive - наивысший приоритет let userInteractiveQueue* - для заданий, которые взаимодействуют с пользователем в данный момент и занимают очень мало времени, анимация, выполняются мгновенно. Пользователь не хочет этого делать на Main queue, однако это должно быть сделано по возможности быстро, так как пользователь взаимодействует со view прямо сейчас. Эта очередь имеет очень высокий приоритет, но ниже, чем у Main queue.
4. *userInitiated —* для заданий, которые инициируются пользователем и требуют обратной связи, но это не внутри интерактивного события, пользователь ждет обратной связи, чтобы продолжить взаимодействие; может занять несколько секунд. Имеет высокий приоритет, но ниже, чем у предыдущей очереди;
5. *utility —* для заданий, которые требуют некоторого времени для выполнения и не требуют немедленной обратной связи, например, загрузка данных или очистка некоторой базы данных. Делается что-то, о чем пользователь не просит, но это необходимо для данного приложения. Задание может занять от несколько секунд до нескольких минут; приоритет ниже, чем у предыдущей очереди;
6. *background* — для заданий, не связанных с визуализацией и не критичных ко времени исполнения; например, backups или синхронизация с web — сервисом. Это то, что обычно запускается в фоновом режиме, происходит только тогда, когда никто не хочет никакого обслуживания. Просто фоновая задача, которая занимает значительное время от минут до часов; имеет наиболее низкий приоритет среди всех глобальных очередей.

**Проблемы многопоточности**

*Race Condition (гонка данных)* – когда несколько процессов пытаются получить доступ/изменить общие ресурсы. В результаты, если процессы не упорядочены, изменение ресурса может произойти после того, как он уже был отображен пользователю.

*Инверсия приоритетов (priority inversion)* - когда низкоприоритетная задача блокирует выполнение высокоприоритетной задачи. Пример:

Задача A начинает выполнение, но в какой-то момент ей требуется доступ к ресурсу, который заблокирован задачей B.

Задача B, поскольку имеет низкий приоритет, не прерывается для освобождения ресурса, и, следовательно, задерживает выполнение задачи A.

Таким образом, задача A испытывает инверсию приоритетов, когда ее выполнение замедляется из-за блокировки ресурса низкоприоритетной задачей B.