УДК 004.432

ПАРАЛЛЕЛИЗМ В ЯЗЫКЕ ПРОГРАММИРОВАНИЯ SWIFT

Юрченко А.Н.

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Научный руководитель: Гременок В.Ф. – д-р физ.-мат. наук

Аннотация. В работе рассмотрен механизм выполнения параллельных задач в языке программирования Swift.Фреймворк GCD и его принцип работы.

Ключевые слова: Swift, параллелизм, GCD.

Говоря о параллелизме в Swift, обычно подразумевают очереди. Очередь – это массив задач, каждая из которых начинает выполнение в том порядке, в котором она была добавлена. Основной поток выполнения также представляет собой очередь, которая называется основной. Есть два типа очередей:

1. Последовательные;
2. Параллельные.

В последовательной очереди задачи выполняются одна за другой. Следующая задача не будет запущена, пока не будет завершена предыдущая. В параллельной очереди задачи также выполняются одна за другой. Однако каждая задача выполняется в отдельном потоке, и очередь не ждет, пока предыдущая задача запустит следующую [1]. Следующая задача запускается немедленно, если ресурсов достаточно для создания еще одного потока.

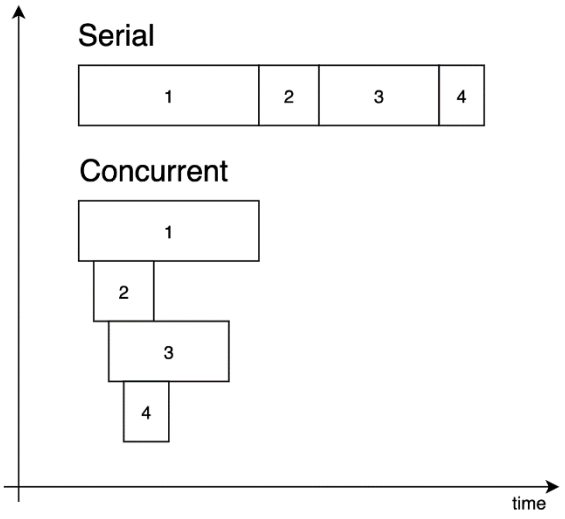

Рисунок 1. – Схема сравнения выполнения задач на параллельной и последовательной очередях [1]

Задачу можно добавить в любую очередь двумя способами:

1. Синхронно;
2. Асинхронно.

При добавлении задачи с помощью метода синхронизации текущий поток блокируется до тех пор, пока задача не будет завершена - независимо от того, является ли очередь последовательной или параллельной.

При использовании метода async текущий поток, наоборот, продолжает выполнение сразу после добавления задачи.

Сам текущий поток может быть основным или еще одной последовательной или параллельной очередью.

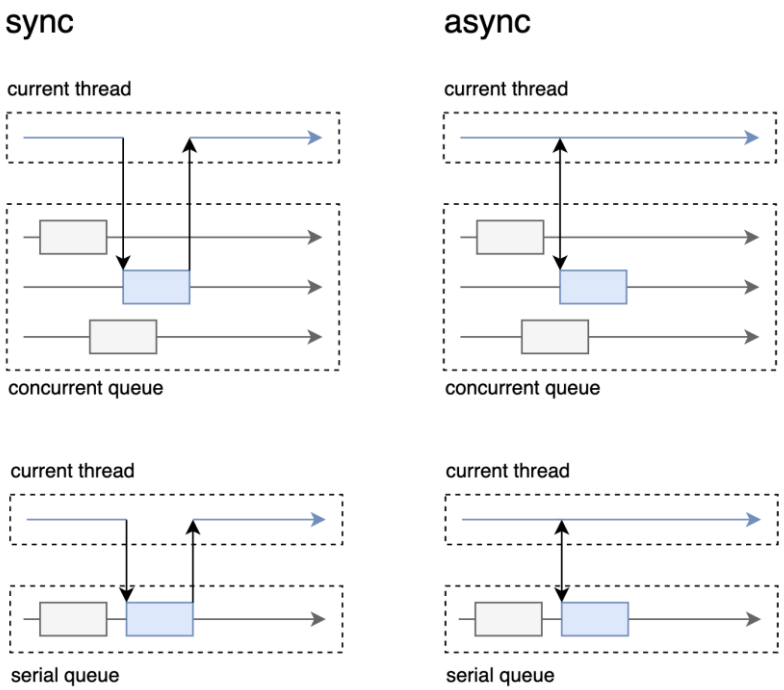

Рисунок 2. – Схема сравнения выполнения задач синхронно и асинхронно [2]

Проблемы параллелизма:

1. Состояние гонки - результат зависит от порядка одновременного выполнения задачи. Порядок может быть разным на каждом прогоне;

2. Инверсия приоритета - неожиданно повышается или понижается приоритет конкретной задачи;

3. Тупик - потоки заблокированы, потому что их задачи ожидают ресурсов, которые ими заблокированы.

Состояние гонки возникает, когда одновременные задачи изменяют общие ресурсы. И в зависимости от порядка этих изменений ресурсы имеют разные значения.

Инверсия приоритета появляется, когда параллельные задачи ожидают общих ресурсов, заблокированных другими задачами. В результате, даже если их приоритеты выше, чем у других, им приходится ждать, пока задачи с низким приоритетом получат доступ к заблокированным ресурсам.

Тупик - самая опасная проблема параллелизма. Это происходит, если есть вложенный параллельный доступ к общим ресурсам. Допустим, есть вложенная параллельная задача. Ожидает ресурсов, которые заблокированы закрывающей задачей. А сама вмещающая задача ожидает завершения вложенной задачи. Итак, оба потока полностью заблокированы друг другом.

Добавление задачи в очередь из основного потока через синхронизацию метода может вызвать взаимоблокировку.

Рассмотрим встроенный фреймворк языка программирования Swfit, Dispatch, также известный как Grand Central Dispatch (GCD), который содержит языковые функции, runtime библиотеки и системные улучшения, которые обеспечивают поддержку параллельного выполнения кода на многоядерном оборудовании в macOS, iOS, watchOS и tvOS. Подсистема BSD, Core Foundation и API-интерфейсы Cocoa были расширены для использования этих улучшений, чтобы помочь системе и приложениям работать быстрее, эффективнее и с лучшей отзывчивостью. GCD, работающий на системном уровне, может лучше удовлетворить потребности всех запущенных приложений, сбалансированно сопоставляя их с доступными системными ресурсами. DispatchQueue имеет ссылку на основную очередь и пять глобаль-

ных параллельных очередей, которые различаются по своему качеству обслуживания (приоритету) или QoS (Quality of service):

1. UserInteractive – самый высокий QoS. Он используется для задач, которые инициируются пользователем и должны быть выполнены в первую очередь. Задачи не должны занимать много времени. Например, пользователь проводит пальцем по экрану. Существует логика для обработки действий пользователя, требующих времени для ответа. Эту логику можно добавить, как асинхронную задачу в очередь с наивысшим приоритетом, чтобы избежать задержек пользовательского интерфейса. Сам основной поток продолжит прослушивание действий других пользователей;

2. UserInitiated – приоритет меньше предыдущего, но все равно относительно высокий. Задачи должны реагировать на действия пользователя как можно скорее, но пользователь может подождать некоторое время (несколько секунд). Например, пользователь нажал кнопку и ждет результата;

3. Utility – это средний QoS. Задачи не запускаются пользователем, и они могут занять более нескольких секунд. Например, пользователь выходит на экран, и он должен загрузить и показать там какое-то изображение. Пока изображение не отображается, пользователь видит индикатор активности;

4. Background – это самое низкое QoS. Задачи никак не связаны с UI и визуализацией. Они могут занять даже часы. Например, синхронизация с iCloud;

5. Default – пытается взять QoS из других источников. В противном случае он устанавливает приоритет между UserInitiated и Utility.

Важно помнить, что глобальные очереди являются системными. И они могут содержать не только пользовательские задачи, но и системные. Основная очередь – это основной поток. Любое обновление пользовательского интерфейса можно сделать только там. Основная очередь – это уникальная последовательная глобальная очередь. В ней не должно быть задач, требующих много ресурсов или много времени.

Список использованных источников:

1. Усов В., Swift Основы разработки приложений под iOS и macOS, Питер, 2018. – 444 с.
2. Грей, Э., Swift Карманный справочник, Москва: Вильямс, 2016. – 288 с.

UDC 004.432

CONCURRENCY IN SWIFT PROGRAMMING LANGUAGE

Yurchenko A.N.

Belarusian State University of Informatics and Radioelectronics
Minsk, Republic of Belarus

Gremenok V.F. - Doctor phys. math. Sciences

Annotation. The paper deals with the mechanism for executing concurrency tasks in the Swift programming language, the GCD framework and its principle of operation.

Key words: Swift, concurrency, GCD.