부스트캠프 3주차 모둠과제 - 스레드

D조, 쌍쌍바

---

1. 스레드에 대해 알아보고, 스레드가 필요한 이유에 대해 논의하고, 주된 논의 내용과 결론을 요약해주세요.

(1) 스레드란?

- 하나의 프로세스가 동시에 여러가지 작업을 수행할 수 있게 하는 방식으로 CPU 시간을 할애하는 최소 동작 단위를 뜻한다.

- 프로세스란 현재 실행되고 있는 프로그램을 말한다.

- 스레드를 왜 사용하는지 알기 위해 먼저 스레드와 프로세스의 차이점. 그리고 멀티프로세스 프로그래밍의 단점을 이해해야한다.

(2) 스레드와 프로세스의 차이

- 프로세스
  - 실행중인 프로그램
  - Heap, Text, Data 공간과 여러 스레드로 구성
  - 모든 프로세스는 하나 이상의 스레드를 가짐
- 스레드
  - Heap, Text, Data 공간을 공유하고 개별적으로 Stack을 소유
  - 실제로 CPU에 할당되는 단위
  - 한 프로세스 내에서 동작되는 여러 실행 흐름

(3) 멀티 프로세스 프로그래밍의 단점

- 프로세스의 수가 많아진다. 때문에 Context Switching (프로세스의 상태 정보를 저장하고 복원하는 과정)으로 인한 성능 저하가 더 자주 발생한다.

- 프로세스가 생성될 때마다 Heap, Text, Data 공간을 각각 만들어 주어야 하므로 메모리의 주소 공간이 낭비 된다.

- 서로 다른 메모리 주소 공간에 위치하므로 Cache hit rate가 감소하여 속도가 저하된다.

(4) 멀티 스레드 프로그래밍의 장점

- 하나의 프로세스에서 스레드를 여러개 생성하여 각각 실행 시키는 방식으로, 멀티 프로세스 방식의 단점들을 해결해준다.

- 같은 프로세스 내의 스레드들은 Heap, Text, Data 공간을 공유하므로, 메모리 낭비가 적다. 또한 Fork와 비교하였을 때 스레드를 새로 생성하는 데 overhead가 훨씬 작고 스레드를 폐기하는 것 역시 프로세스 kill 보다 overhead가 작다.

- 프로세스에 비해 Context Switching 오버헤드가 작다.

- 스레드는 같은 프로세스 위에서 동작하므로 Cache hit rate가 높아져 성능 저하가 적고, IPC를 사용하지 않아도 된다.

- 위의 장점들을 통해 시스템의 throughput이 향상되고, 시스템의 자원 소모가 줄어들며, 프로그램의 응답 시간이 단축된다.

- **멀티 프로세서 환경**에서는, 하나의 프로세스에서 여러 스레드가 여러 Core에서 물리적으로 동시에 실행될 수 있는데, 이러한 경우에 큰 속도 향상을 이뤄낼 수 있다.

(5) 멀티 스레드 프로그래밍의 단점

- 멀티 스레드 프로그래밍을 할 때는 Race Condition으로 인한 Deadlock이 발생하지 않도록 여러 경우의 수를 고려해야만 한다.

- 미묘한 시간차나 잘못된 변수를 공유함으로써 오류가 발생할 수 있다.

- 프로그램 실행의 흐름이 일정하지 않아, 디버깅이 어렵다.

- 단일 프로세서 환경에서는 큰 성능 상승 효과를 기대하기 어렵다.

2. iOS에서 실제로 스레드는 어떻게 사용되고 있는지, Cocoa Touch Framework에는 어떤 경우에 자동으로 새로운 스레드를 만들어주는지 예를 찾아보세요

- 만약 특정 API가 동작하는데 작업 시간이 오래걸린다면 API가 동작하는 동안 UI가 멈추는 (Freezing)현상이 발생한다. 이는 하나의 스레드에서 UI와 API를 동시에 처리하기 때문인데, 이 문제를 해결하려면 둘 이상의 스레드를 이용해 UI와 API를 분리하여 작업을 처리해야한다. (비동기 방식)

- **적용 사례**

- 모든 앱은 실행 시 하나의 main 스레드인 싱글스레드로 시작된다. Cocoa Touch에서 UIApplication이 main스레드로 실행된다.

- iOS에서 그래픽 렌더링은 전적으로 동기방식이다 만약 비동기 드로잉을 허용할 경우 전체 화면이 렌더링 될 때 깜빡임 또는 화면 누락이 발생할 수 있다.

- Animated. 동적 그래픽을 생성한 후 비동기 방식으로 동시에 처리된다.

- 그래프, 이미지 생성 등 비동기 방식으로 처리 됨.

- UI에 반영되어야 할 작업이라면 꼭 main스레드에서 작업을 해야만 한다. 네트워크 통신, 리소스 및 유저 데이터와 같은 작업을 다른 스레드에서 한 뒤 UI 업데이트를 위해 main 스레드로 View를 업데이트 한다.

Cocoa Touch Framework 사례
- NSOperation 기술
- GCD
- Timer
- NSNotification

3. Foundation Framework에서는 어떤 방식으로 스레드를 다룰 수 있는지 찾아보고 각 예를 실질적인 코드로 표현해보세요

Foundation에서는 크게 NSThread, NSOperation / NSOperationQueue, GCD의 세 가지 방법을 통해 스레드를 다룰 수 있다. 아래에서 각 항목에 대해 알아보도록 하자.

3.1. NSThread

Cocoa (Touch)에서는 NSThread 클래스에서 스레드를 구현한다. 그리고 NSObject에서 새로운 스레드의 생성과 이미 동작하고 있는 스레드들에 대해 코드 실행에 대한 메소드를 제공한다. Swift3에서는 NSThread가 아닌 Thread으로 사용 가능하다.

아래 그림은 NSThread에 대한 예제 코드로, 프로그램이 실행되면 쓰레드가 5개 생성되어 각각의 쓰레드가 공유 자원인 'count'가 100이 될 때까지 1초에 1씩 더하는 코드이다.

먼저, MyThreadClass를 보자. 해당 클래스는 NSObject를 상속받아 Thread를 사용할 수 있다. Thread의 생성은 Thread(target: Any, selector: Selector, object: Any?)를 통해 이루어진다. MyThreadClass의 인스턴스를 생성하면, 실제로 이를 통해 생성자에서 'runLoop'를 selector로 하는 5개의 스레드를 생성시킨 후 start를

하도록 한다.

runLoop는 thread가 수행할 작업에 해당하며 코드를 보면 count를 1 증가시킨 후 print하는 것을 확인할 수 있다. 여기서 lock(: NSLock)이라는 변수를 사용하였는데, 이는 Critical section의 Global data를 보호하는 역할을 수행한다. 만약, 한 쓰레드가 count를 0에서 1로 증가시켜서 이를 count변수에 저장하기 직전에 다른 쓰레드가 count를 참조하는 경우, 해당 쓰레드는 count를 1이 아닌 0으로 알고 다시 0에서 1으로 증가시킨 후 count에 1을 저장하게 된다. 이 경우 우리가 기대한 결과는 count가 2가 되는 것인데 결과적으로 1이 되고 이는 후에 예상치 못한 결과를 초래시킬 수 있다. 때문에 critical section의 바로 위에는 lock.lock()을, 아래에 lock.unlock()을 통해 global data를 보호할 수 있게 된다.

마지막 줄에는 Thread.sleep을 통해 1초간 아무 일도 하지 않도록 하는 코드를 추가하였다. 이를 통해서 쓰레드가 1초마다 count에 1씩 더하는 행위를 구현할 수 있다.

아래의 코드는 간단한 스레드 사용법에 대해 알아보기 위해 작성한 코드이며, 이는 여기에서 받아볼 수 있다. (Apple NSThread문서 참고하였음)

```
class MyThreadingClass: NSObject {
    var count = 0
    let lock = NSLock()

    func runLoop() {
        while Thread.current.isCancelled == false {
            // lock
            lock.lock()

            // critical section
            if count > 100 {
                print("end!")
                return
            }

            count = count + 1
            print("count: \(count) current: \(Thread.current)")

            // unlock
            lock.unlock()

            Thread.sleep(forTimeInterval: 1)
        }
    }

    override init() {
        super.init()
        for _ in 0..<5 {
            Thread(target: self, selector: #selector(runLoop), object: nil).start()
        }
    }
}
```

3.2. NSOperation, NSOperationQueue

- Swift3 부터는 Operation, OperationQueue으로 불러와 사용할 수 있다

- OperationQueue는 maxConcurrentOperationCount 프로퍼티 값을 이용해 주어진 특정 시간동안 동시에 실행 가능한 작업의 숫자를 제한하는것이 가능합니다.

- Operation을 시작하기 위해서는 start 메소드를 호출하거나, Operation을 OperationQueue에 추가하고 큐의 가장 처음에 도달하면 한 번 실행되도록 할 수 있습니다.

- OperationQueue는 작업의 동시 실행을 조절을 간편한 방식으로 제공해줍니다, OperationQueue는 기본적으로 FIFO(First In First Out) 방식으로 동작합니다. 다만, 여기서 나아가 우선순위 큐로써 높은 우선 순위의 작업이 들어오게 되면 더 낮은 우선 순위를 가진 작업들은 건너뛰고 실행되게 됩니다.

1. veryLow
2. low
3. normal
4. high
5. veryHigh

- Operation을 사용하여 얻을 수 있는 많은 장점들이 OperationQueue를 사용하는 것에서 오기 때문에, Operation에서 start를 직접적으로 호출하는 것보다는 OperationQueue에 추가하여 사용하는것이 바람직합니다.

3.3 GCD

- GCD (Grand Central Dispatch) 는 iOS에서 유용한 멀티스레드의 효과를 얻기 위해 사용하는 큐나 세마포어와 같은 저수준의 프레임워크를 제공한다.
- 큐의 속성은 생성시점에 지정될 수 있고, 이는 Swift의 OptionSet 프로토콜을 따르게끔 되어 있어서 순서대로 처리할지 동시에 처리할지, 메모리 관리 옵션은 어떻게 되는지, 서비스 퀄리티는 어떤지 등을 지정할 수 있다.

GCD 큐의 종류

Main(Serial) Queue

iOS에서 UI가 동작하는 큐. UI와 관련된 작업은 항상 main스레드에서 이루어져야한다.

큐에 추가된 순서 안에서 선입선출(FIFO) 방식으로 작업을 실행한다. 또한, 큐에 있는 작업 하나를 실행시킨 후에 실행이 끝날 때까지 큐에 있는 다른 작업들은 기다리고 있다. 즉, 스레드 하나로 순차 처리를 하고 있다. 이는 직렬로 실행하는 별도의 Serial Queue라고 볼 수 있다. 작업이 하나 생성될 때마다 스레드가 하나 더 생기기 때문에 과도하게 많은 숫자를 생성하면 성능에 문제가 있을 수 있다.

Global(Concurrent) Queue

UI와는 별개로 동작하는 큐. 들어오는 작업 만큼 무한대로 멀티스레딩을 한다.

시리얼 큐와는 다르게 실행 중인 작업을 기다리지 않고 큐에 있는 다른 작업들을 동시에 별도의 스레드에 병렬적으로 수행시킨다.

순서가 보장되지 않으므로 순서가 중요치 않은 비동기 작업에 적합하다.

Quality of Service

- GCD에서 제공하는 QoS 클래스가 있다. 이 클래스는 GCD가 동시적 큐에서 과업의 우선순위를 정할 때 고려되어지는 클래스로써 과업의 의도를 추상화하여 표현하여 네이밍을 하였다. QoS를 사용하여 GCD를 사용할 때의 장점은 각 디바이스 아키텍쳐에 맞는 환경을 시스템이 고려하여 멀티 스레딩을 효율적으로 해준다는 것에 있다. 서비스 퀄리티는 예전의 상수가 아닌 enum 케이스로 정의되었다.

DISPATCH_QUEUE_PRIORITY_HIGH >> .userInitiated

- 해당 클래스는 UI 이벤트를 통해 초기화 작업을 해야 할 때를 위해서 제공된다. 관련된 과업은 비동기식으로 실행된다. 예를 들어 사진 필터를 제공하는 기능이 있다고 가정을 하자, 원하는 필터 효과를 적용하기 위해서 버튼을 탭하고 해당 이미지를 필터에 맞는 효과를 적용할 때 필요한 과업을 해당 클래스를 지정하여 실행하면 효과적이다.

DISPATCH_QUEUE_PRIORITY_DEFAULT >> .default

- 시스템에서 제공하는 Qos class 타입을 따르기 위해 사용하는 것으로 다른 qos와 구분을 위해 정의된 것은 아니다.

DISPATCH_QUEUE_PRIORITY_LOW >> .utility

- 파일 I/O 작업이나 네트워킹 작업 같이 긴 시간을 필요로 하는 작업에 적당한 클래스이다.

DISPATCH_QUEUE_PRIORITY_BACKGROUND >> .background

- 이 클래스는 시간에 구애 받지 않고 사용자가 신경쓰지 않아도 되는 작업들을 백그라운드에서 실행하고자 할때 사용 되어지도록 디자인된 클래스이다. 예를 들어 로그아웃 시 로컬에 저장되어 있는 파일을 지워야 한다든지 혹은 이미지 파일을 다운로드 받은 후 캐싱을 한다든지 와 같은 작업을 할 때 적당할 것이다.

GCD 사용 예시 : https://www.appcoda.com/grand-central-dispatch/