대본

안녕하세요 오늘 CS교육 진행을 맡은 최준영이라고 합니다.
오늘은 동시성 프로그래밍에 대해 발표를 해보려구 합니다.
개발파트 분들은 동시성 프로그래밍을 구현하기위해 화면에 보이시는 개념을 접해보신적이 있을 겁니다.
오늘은 이개념들(동기, 비동기, 블로킹, 논 블로킹)에 대해 다같이 알아보는 시간을 가져보려고 합니다.
여러분들은 카페알바를 해본적이 있으신가요?
예를들어 2명에서 일하는 카페가 있습니다. 아이스아메라카노 주문이 들어왔을 때
가장빨리 커피를 내보내는 방법은 무엇일까요? 바로 한명은 얼음을 푸고 한명은 커피를 내리는 것입니다.
이 작업들은 각자 '동시' 에 발생하게되고 마지막에 '결과물' 로 동기화 됩니다.
한명이 모든작업을 순차적으로 처리하는 것보다 훨씬 효율적인 운영방법입니다.
설로다른 작업이 동시에 처리되는 개념은 컴퓨터에도 적용시킬 수 있는댕 이를 동시성 프로그래밍이라고 합니다.
동시성 프로그래밍을 이해하기 위해서 화면에 보이는 4가지 개념을 알 필요가 있습니다.
-동기/비동기
먼저 동기와 비동기 부분부터 살펴보겠습니다.
앞서 설명한 카페예제를 다시한번 보겠습니다. 얼음을 푸는 작업과 커피를 내리는 작업이 완료된후 서로 동기화되어
아이스 아메리카노라는 결과물을 만들어 내게 되지요.
즉, 이때 사용되는 동기화가 동기성입니다.
동기성의 특징은 특정 작업의 완료 여부를 확인한다는 점입니다.
얼음을 푸고 커피를 내리는 작업이 끝난 것을 확인한 후 아이스아메리카노를 만들어 내는 것처럼요
반면 비동기성은 특정 작업의 완료시점을 신경쓰지 않는 개념입니다.
알바생1은 커피를 내리면서 알바생2가 얼음을 잘 내리고 있는지 상관하지 않는 다는 것이죠. 이 관점에서 두작업은 비동기적으로 수행된다고 합니다.
코드를 간략하게 해당 동작을 살펴보겠습니다.
(코드 & 코드출결과)
일단 두개의 작업이 동시에 발생되는 것을 확인할 수 있습니다.
(알바생들을 쓰레드로 표현한 그림)
하지만 알바생1은 알바생2의 작업의 종료여부를 확인하지 않음으로 두작업은 비동기적으로 실행됩니다.
그렇다면 비동기방식은 실제 프로그래밍 환경에서 작업의 종료여부를 어떻게 알 수 있을까요?
(메인쓰레드가 아닌 다른쓰레드에서 호출되는 콜백함수)
(코드)
보통 해당작업이 끝나면 호출되는 코드블록을 전달하는 방식으로 구현됩니다. 이때 전달되는 코드블록을 콜백함수라고 부릅니다.
-블로킹/논블로킹

블로킹, 단어 그 자체로 뭔가를 막는다라는 의미가 있습니다.

현재 실행되는 작업에서 다른작업이 실행될때 실행중이던 작업이 중지되는 것을 블로킹이이라고 합니다.
(중지되는 그림)
논블로킹은 앞서 알바생1과 2가 동시에 작업하는 상황을 뜻합니다.
알바생2가 얼음을 푼다고 해서 알바생1이 커피를 내리지 못하는 건 아니죠 이렇게 두작업은 서로의 작업실행을
막지 않습니다. 즉 블로킹 하지 않습니다.
하지만 알바생이 1명이라면 어떡할까요?
두작업을 동시에 실행하지 못할겁니다. 얼음을 푼다면 커피를 내리는 작업을 뭠처야 하는 것이지요
(블로킹 방식 그림)
코드로 이 동작을 한번 살펴보겠습니다.
(코드 및 실행결과)
얼음작업이 끝나기 전까지 커피를 내리는 작업이 시작되지 못했던 결과를 얻을 수 있습니다.
작업이 블로킹 된것이지요
-블로킹과 동기의 차이
(작업완료확인 그림과 블로킹 그림)
두 그림을 보면 두 방식은 유사해보입니다.
하지만 동기성방식은 실행한 작업의 완료여부를 확인한다고 했습니다.
작업완료의 여부를 그림과 같이 블로킹 방식으로 확인할 필요는 없습니다.
제일 처음에 든 예시를 기억하시나요
(아이스 아메리카노로 두작업이 동기화)
두 작업은 서로 논블로킹으로 실행되지만 마지막에 동기화 됩니다.
즉, 마지막에 동기화됨으로 동기성을 띕니다.
해당 작업을 코드로 한번 살펴보겠습니다.
(코드및 실행결과)
마지막에 while문이 반복해서 두작업의 종료여부를 확인하는 것을 볼 수 있습니다.
동기와 블로킹은 이런 차이가 있습니다.
-사용
동기 비동기 논블로킹 블로킹에 대한 개념적인 부분을 살펴보았습니다.
여기까지 보면 논블로킹 비동기가 좋은 것이다 라고 생각하실 수도있을 것 같습니다.
하지만 앞서 블로킹과 동기성의 차이에서 살펴보았듯이 동기적으로 작동한다고 멀티쓰레딩을 사용하지 못하는 것이 아닙니다.
각각의 컨셉들의 특성에 맞춰 작업처리를 설계하는 것이 중요합니다.
예를들어 멀트쓰레딩은 말그대로 여러개의 쓰레드에서 작업이 동시에 수행되는 것을 가능하게 합니다.
하지만 쓰레드를 생성하는 행위는 추가적인 리소스를 필요로함으로 상황에 맞춰 적절한 수의 쓰레드를 생성하는 것이 중요합니다.
-Race Condition
마지막으로 멀티쓰레드 환경에서 발생할 수 있는 대표적인 문제인 Race condition에 대해 알아보겠습니다.
자원을 공유하지 않는 멀트 프로세스 방식과 달리 멀트쓰레드는 각 쓰레드 간에 자원을 공유한다는 특징이 있습니다.

(여러 쓰레드의 자원 공유사진)
이 때 복수의 쓰레드에서 공유자원에 동시에 접근하는 경우 예상치 못한 동작이 발생할 수 있습니다.
보이시는 코드는 하나의 배열을 3개의 쓰레드에서 동시에 접근하는 코드입니다.
(코드)
결과를 보시면 해당 배열에는 중복된 값이 없음에도 같은 숫자들이 서로다른 쓰레드에서 각각 추출된 것을 확인할 있습니다.
이런 상태를 Race Condition이라고 합니다. 다중쓰레드 환경에서 이문제를 해결하는 것은 정말 중요한 문제라고 할 수 있습니다.