MUTEX를 이용한 쓰레드 동기화

기초 원리 학습

OneKick

- 멀티쓰레드 프로그램을 구현할때 쓰레드 간의 동기화 문제가 발생할 수 있음
- 동기화 문제
  - 메시지 전송 혹은 같은 변수에 접근을 하게 되는 경우 쓰레드간의 충돌이 발생하는 경우
  - 2개 이상의 쓰레드가 1개의 변수를 공유해서 사용하는 경우 읽고, 계산하고, 저장하는 단계로 진행이 되는데 같이 읽고 같이 쓰는 경우 올바르지 않은 값이 저장 될 수 있기 때문

기초 원리 학습

OneKick

- 충돌이 발생할 수 있는 코드 영역을 Critical Section이라 함
  - 임계영역은 쓰레드가 한번에 하나씩만 들어갈 수 있게 순서를 정해 주어야 하는데 이것을 상호배제라고 함
  - 영어로는 Mutual Exclusion이라고 부르며 이것을 줄여서 Mutex
  - Mutex라는 클래스가 존재 하며, 비슷한 용도로 Semaphore라고 하는 클래스 제공

기초 원리 학습

OneKick

- 세마포어
  - Semaphore 클래스를 이용
  - 임계영역의 데이터를 여러 쓰레드가 접근해서 충돌이 발생하는 것을 방지
  - 키를 여러개 발급 할 수 있기 때문에 단순히 상호배제 뿐만 아니라, 쓰레드 간의 순서를 정하는 기능도 가능

기초 원리 학습

OneKick

- 뮤텍스
  - Mutex 클래스를 이용
  - 임계영역의 데이터를 여러 쓰레드가 접근했을 때 충돌을 막음

- Thread 동기화에서 중요한 요소가 Critical Section
  - 멀티태스크킹 환경에서 여러 프로세스가 데이터를 공유하면서 수행될 때 각 프로세스에서 공유 자원을 접근하는 프로그램 코드 부분을 가리키는 것

- 공유 자원을 여러 쓰레드가 동시에 접근하면 동기화 문제로 인해 논리적 오류
  - 한 쓰레드가 위험 부분을 수행하고 있을 때, 즉 공유 데이터를 액세스하고 있을 때는 다른 쓰레드는 대기할 수 있도록 해 주어야 함

기초 실습

OneKick

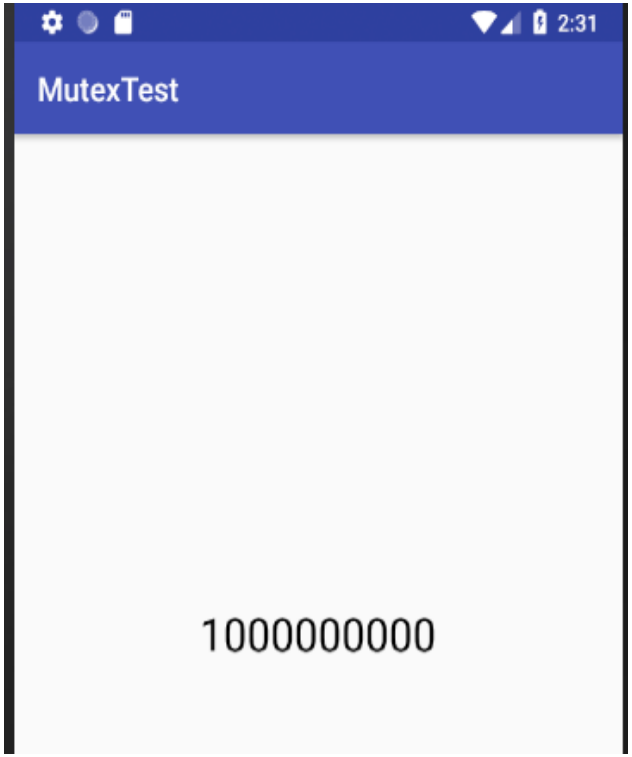
- 새 프로젝트 만들기
  - 화면에는 텍스트뷰 1개만 추가하여 숫자를 표시하도록 구현

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="안드로이드 APK 파일의 스키마 경로는 생략합니다."

    xmlns:app="안드로이드 APK 파일의 스키마 경로는 생략"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent" // 부모를 안벗어나게 최대화
    android:layout_height="match_parent" // 부모를 안벗어나게 최대화
    tools:context=".MainActivity">
        <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content" // 컨텐츠에 맞게 최소화
        android:layout_height="wrap_content" // 컨텐츠에 맞게 최소화
        android:text="Hello World!"
        android:textSize="28dp"
        android:textColor="#000000"
        app:layout_constraintBottom_toBottomOf="parent" // 부모를 기준으로 맞춤
        app:layout_constraintLeft_toLeftOf="parent" // 부모를 기준으로 맞춤
        app:layout_constraintRight_toRightOf="parent" // 부모를 기준으로 맞춤
        app:layout_constraintTop_toTopOf="parent" // 부모를 기준으로 맞춤 />
</android.support.constraint.ConstraintLayout>
```

기초 실습

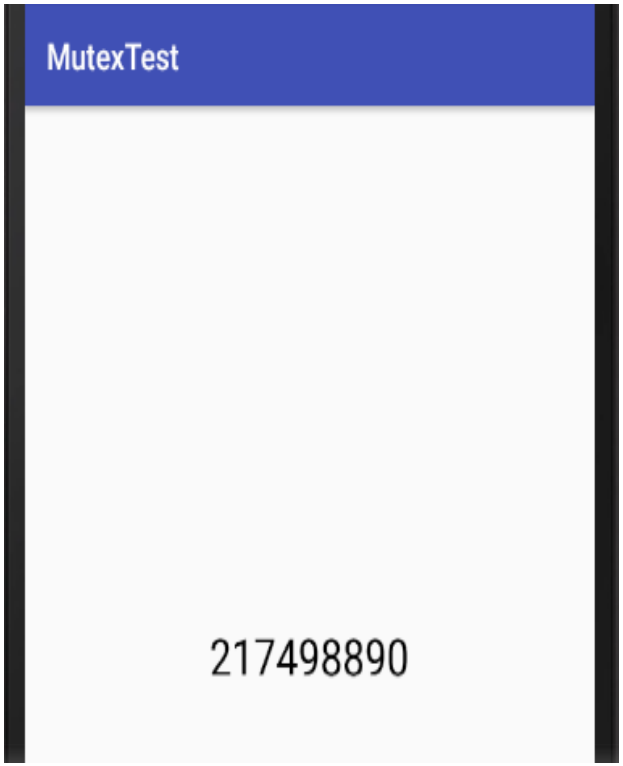
- 아래와 같이 쓰레드 1개를 생성해서 1,000,000,000까지 number를 1씩 증가시키는 앱을 구현하시오.



응용 실습

OneKick

- 앞의 예제에서 쓰레드를 1개더 추가해서 number를 1,000,000,000회 1씩 감소시키도록 만드시오.
  - 한 쓰레드는 1,000,000,000 회 1씩 증가
  - 한 쓰레드는 1,000,000,000 회 1씩 감소
    - 결과는 0이 나오지 않음



동기화 문제로 0 이 나오지 않음

응용 실습

OneKick

- 임계영역을 상호배제 시켜서 해결하기
  - Semaphore 사용해보기

```
package com.iot.mutextest;

import android.support.v7.app.AppCompatActivity; // 액티비티 부모 클래스
import android.os.Bundle; // 액티비티 생성 번들
import android.widget.TextView;
import java.util.concurrent.Semaphore;

public class MainActivity extends AppCompatActivity { // 메인 화면
        private int number =0; // critical section
        private TextView textView;
        private Semaphore semaphore = new Semaphore(1);

        @Override // 부모 메소드 재정의
        protected void onCreate(Bundle savedInstanceState) { // 화면생성 이벤트
                super.onCreate(savedInstanceState); // 부모 생성자 호출
                setContentView(R.layout.activity_main); // 메인 화면 표시
                textView = (TextView)findViewById(R.id.textView);
```

응용 실습

OneKick

```
new Thread() {
        @Override // 부모 메소드 재정의
        public void run() {
                super.run();

                try {
                        semaphore.acquire();
                        for(int i=0; i<1000000000;i++) number ++;
                        semaphore.release();
                } catch (InterruptedException e) {
                                e.printStackTrace();
                }
                runOnUiThread(new Runnable() {
                        @Override // 부모 메소드 재정의
                        public void run() {
                                textView.setText(""+number);
                }
                });
        }
}.start();
```

Semaphore 클래스는 가상의 Permit 을 만들어 내부 상태를 관리하며, 클래스를 생성할 때 인자로 Permit 의 수를 넘겨주게 되어 있다.
외부 스레드는 Permit 을 요청해 확보(acquire)하거나, 이전에 확보한 Permit 을 반납(release)할 수 있다.
현재 사용할 수 있는 남은 permit이 없는 경우 (permit = 0), acquire 메소드는 남는 permit이 생기거나, 인터럽트가 걸리거나, 지정한 시간을 넘겨 타임아웃이 걸리기 전까지 대기한다.
release 메소드는 확보했던 permit 을 다시 세마포어에 반납하는 기능을 한다.

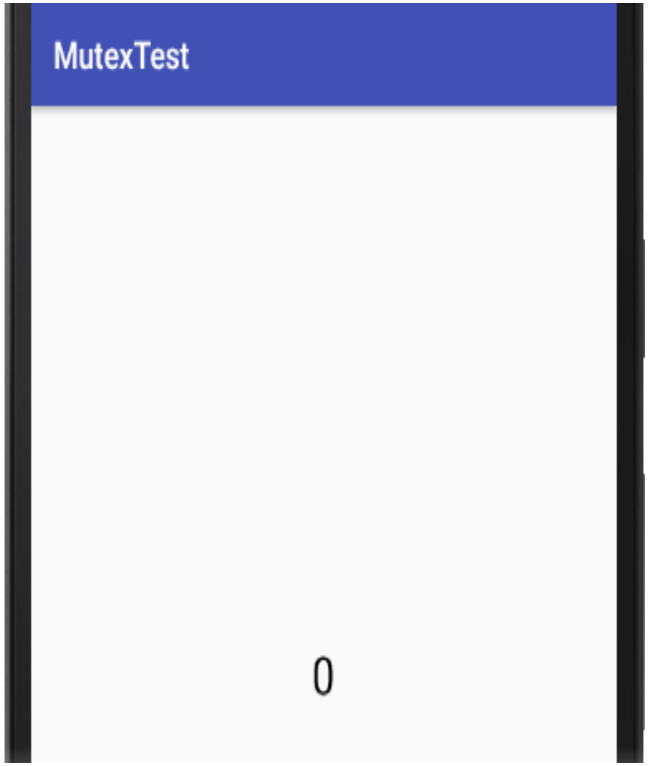
응용 실습

OneKick

```
            new Thread() {
            @Override // 부모 메소드 재정의
                  public void run() {
                        super.run();

                        try {
                              semaphore.acquire();
                              for(int i=0; i<1000000000;i++) number --;
                              semaphore.release();
                        } catch (InterruptedException e) {
                              e.printStackTrace();
                        }
                        runOnUiThread(new Runnable() {
                              @Override // 부모 메소드 재정의
                              public void run() {
                                    textView.setText(""+number);
                              }
                        });
                  }
            }.start();
      }
}
```

- 상호배제가 구현되면 아래와 같이 0으로 표시되는 것을 볼 수 있다.
  - 같은 방식으로 Mutex를 이용하여 상호배제를 구현해 보도록 한다.



상호배제 구현으로 0 출력 성공