

< 인터넷 모바일 프로그래밍 >

[Project 2: NDK application with Android Studio]



국제학부 모바일시스템공학과

32143153 이대훈

32143155 이도경

32143605 이종원

Contents

I.	Introduction-----	03pg
II.	Motivation -----	04pg
III.	Concept -----	04pg
IV.	Program Structure-----	13pg
V.	Build environment -----	25pg
VI.	Problem & Solution -----	25pg
VII.	Personal Feelings -----	31pg

I. Introduction

하드웨어(Hardware)는 컴퓨터나 컴퓨터에 붙어 있는 주변 장치들을 말합니다. 즉, 하드웨어는 각각의 기능을 가진 장치를 의미합니다. 이러한 장치들을 사용하기 위해서는 소프트웨어의 지원이 필요합니다. 이러한 소프트웨어를 장치 드라이버 또는 디바이스 드라이버라고 합니다. 디바이스 드라이버는 시스템이 지원하는 하드웨어를 응용 프로그램에서 사용할 수 있도록 커널에서 제공하는 라이브러리입니다. 따라서 응용 프로그램이 하드웨어를 제어하려면 커널에 자원을 요청하고, 커널은 이런 요청에 따라 시스템을 관리합니다. 리눅스 환경에서는 모든 자원을 파일 형식으로 제공합니다. 이런 디바이스 드라이버 파일들이 /dev/에 저장되어 있습니다.

기본적으로 커널에 이미 들어있는 디바이스 외에 새로운 디바이스를 추가하는 경우 모듈과 같은 방법을 사용할 수 있습니다. 커널 모듈은 리눅스 커널이 부팅되어 동작중인 상태에서 디바이스 드라이버를 동적으로 추가하거나 제거할 수 있게 하는 방법입니다. 이런 모듈 방식은 리눅스에 포함된 디바이스 드라이버를 개발할 때 개발 시간을 단축시킬 뿐만 아니라 필요에 따라 커널에 포함하거나 하지 않을 수 있음으로 효율적으로 다루게 합니다.

일반적으로 대부분의 디바이스 드라이버들은 C 또는 C++로 작성되어있는 경우가 많습니다. 안드로이드에서 사용되는 JAVA 와 함께 호환하기위해 NDK 를 사용합니다.

II. Motivation

다음의 과제를 진행하면서 OS 수업 때 다루지 못한 하드웨어 처리에 대해 배울 수 있습니다. 과거 OS가 없던 시절에는 하드웨어를 관리하기 위해서 사용자가 직접 interface 작업을 했습니다. 하지만 OS가 생기고 나서는 사용자가 직접 하드웨어 관리를 하지않고 하드웨어를 생산한 회사에서 하드웨어를 제어할 수 있는 디바이스 드라이버를 함께 제공합니다. OS를 통해 하드웨어를 사용하기 위한 디바이스 드라이버들을 사용하는 방법 및 kernel에 모듈을 올리는 작업을 직접 실행해봅니다. Menuconfig를 통해 직접 하드웨어 드라이버에 대해 설정을 진행합니다. 또한 NDK를 사용하여 native language를 사용할 수 있는 방법을 배웁니다. 한백전자 킷을 사용하여 다양한 하드웨어를 사용자가 이용해보고 개발한 어플리케이션과 연동하는 작업을 합니다.

III. Concept

Native Development Kit / Java Native Interface

안드로이드 스튜디오에서 주로 사용되는 개발 언어는 대부분 자바 (그리고 최근 개발된 코틀린) 입니다. 그래서 자바나 코틀린 외의 언어로 쓰인 프로그램들은 호환이 어렵습니다. 그래서 C나 C++과같은 native language로 써진 프로그램이나 library를 안드로이드에서도 사용할 수 있게 도와주는 개발 도구가 바로 Native Development Kit (NDK) 입니다. NDK를 사용하는 3가지 이유가 있습니다. 첫 번째로는 기존에 C로 만들어진 대규모 코드를 자바로 다시 쓸 필요 없이 재사용이 가능합니다. 안드로이드 OS를 기반으로 한 기기의 디바이스 드라이버들은 대부분 C/C++과 같은 즉, low level language로 만들어져 있습니다. NDK를 사용하여 디바이스 드라이버들을 Java code로 변경할 필요없이 그대로 사용할 수 있습니다. 두 번째로 Java로는 어려운 시스템 요소들에 접근하는 작업을 native language를 사용하여 대체합니다. 세 번째로 Java보다 lower level language인 C/C++을 사용함으로써 프로그램의 속도 및 성능을 향상시킬 수 있습니다.

안드로이드 스튜디오에서 NDK 를 설치하고 native language 코드를 사용하려면 해당 프로젝트에 자바 method 와 C/C++ 함수를 서로 호출하고 반환할 수 있게 연동시키는 Java Native Interface (JNI)를 만들어야 합니다. 안드로이드 스튜디오에서 JNI 를 구축하는 방법은 다음과 같습니다. 첫 번째로 native 함수를 선언하고 선언된 함수를 호출하여 반환 데이터를 받는 Java Class 를 구현합니다. 두 번째로 그 Java Class 를 컴파일하고 javah 를 사용하여 native language 에서 사용할 수 있는 헤더 파일을 생성합니다. 세 번째로 Java Class code 에서 선언된 native 함수를 C/C++로 정의합니다. 마지막으로 C++파일에 javah 로 생성한 헤더 파일을 포함시킵니다.

Hanback Hardware Drivers

이 프로젝트 어플리케이션에서 사용하는 Hanback 하드웨어들은 Text LCD, LED, 7-Segment, Dipswitch, Piezo, Dot Matrix, 그리고 Keypad 입니다.

- Text LCD
 - Text LCD 는 16 비트로된 CLCD_Ctrl_Reg 에 의해 제어됩니다. 각 비트의 값에 따라 Clear Display, Return Home, Entry Mode Set, Display On/Off Control, Cursor or Display Shift 와 같은 기능을 수행할 수 있습니다. Display Data RAM(DD-RAM)의 번지수는 LCD 화면상의 출력 위치를 의미합니다. 첫 번째 출력 줄의 번지는 0x0 부터 0x10, 그리고 두 번째 줄의 번지는 0x40 부터 0x50 입니다. 이 DD-RAM 의 번지수에는 그 자리에 출력될 ASCII 문자의 데이터가 저장됩니다. ASCII 문자는 0x20 부터 0x7F 로 표현됩니다.
 - Text LCD JNI 에는 출력하고 싶은 jstring 을 /dev/fpga_textlcd 파일에 write 하면 Text LCD 화면에 출력됩니다.

- LED

- LED 디바이스 드라이버는 8 비트 제어 레지스터를 사용합니다. 각 비트는 각 8 개의 LED 를 의미하고 각 비트에 0 또는 1 을 넣음으로써 해당 자리의 LED 를 키고 끌 수 있습니다.
- LED JNI 에서 jint 데이터를 /dev/fpga_led 파일에 write 하면 해당 jint 데이터의 하위 8 비트에 따라 LED 가 켜지고 꺼집니다.

- 7-Segment

- 7-Segment 디바이스 드라이버에서는 6 비트 SEG_Sel_Reg 와 8 비트 SEG_Data_Reg 를 사용합니다. SEG_Sel_Reg 는 6 자릿수 중에 어느 자리에 출력을 할지 결정합니다. SEG_Data_Reg 는 결정된 자릿수에 어떠한 segment 를 켤지를 정합니다. 7-Segment 는 계속해서 오른쪽부터 왼쪽 자릿수들을 출력하기 때문에 남은 잔상에 의해 출력이 지속되어 있는 것처럼 보입니다.
- 7-Segment JNI 에서 /dev/fpga_segment 파일에 jint 데이터를 write 하면 해당 jint 값을 7-Segment 하드웨어에 출력하게 됩니다.

- Dipswitch

- Dipswitch 디바이스 드라이버는 16 비트 DIP Switch Data Register 가 있고 16 비트 DIP Switch Control Register 가 있습니다. Dip Switch Control Register 에서 선택된 DIP Switch 로부터 값을 읽어와 DIP Switch Data Register 에 저장합니다.
- DIP Switch JNI 에서 /dev/fpga_dipsw 파일을 read 하면 당시의 Dip Switch 의 상태를 jint 형 데이터로 반환합니다.

- Piezo

- Piezo 디바이스 드라이버는 8 비트 Piezo_Ctl_Reg 레지스터를 사용하여 제어합니다. Piezo_Ctl_Reg 는 0x01(도), 0x02(#레), 0x03(레), , 0x26(#시), 0x27(시)로 음계를 표현합니다.

- Piezo JNI 에서 /dev/fpga_piezo 파일에 jint 데이터를 write 하면, Piezo 하드웨어가 jint 데이터의 하위 8 비트에 해당하는 음계를 출력합니다.

- Dot Matrix

- Dot Matrix 디바이스 드라이버는 10 비트 Dot_Scan_Reg 와 7 비트 Dot_Data_Reg 를 사용합니다. Dot_Scan_Reg 는 Dot Matrix 의 열을 정의하고, Dot_Data_Reg 는 행을 정의합니다. Dot_Scan_Reg 와 Dot_Data_Reg 의 최하위 비트가 둘다 1 이라면 Dot Matrix 의 좌측 상단 Dot 이 켜지게 됩니다.
- Dot Matrix JNI 에서는 jstring 을 parameter 로 받습니다. 그리고 받은 jstring 을 각 character마다 지정 되어있는 font로 변환합니다. 그 다음 각 character 사이에 빈 Dot Matrix 수직선 2 개를 삽입합니다. String 끝에는 빈 Dot Matrix 수직선 20 개를 추가합니다. 이 변환된 String 을 하나의 Dot Matrix 수직선씩 옮겨지며 출력하여 String 이 우측에서 좌측으로 옮겨지며 출력됩니다.

- Keypad

- Keypad 디바이스 드라이버는 Keypad 버튼을 누를 때 해당 버튼에 정의 되어있는 안드로이드 Key Code Constant 를 입력하게 합니다. 하나의 예를 들어 설명하자면 좌측 상단 Keypad 버튼이 안드로이드 KEYCODE_CAMERA 로 Key Code Constant 가 정의 되어있다면, 버튼을 눌렀을 때 안드로이드에서 카메라가 실행됩니다.

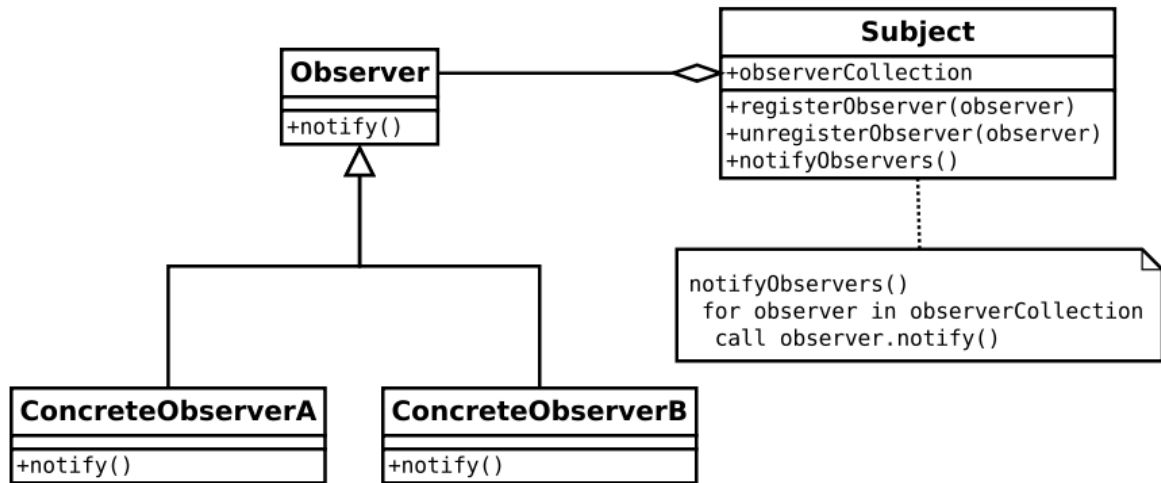
Shared Library

라이브러리는 다른 프로그램들과 링크 되기 위하여 존재하는, 하나 이상의 Subroutine 이나 Function 들의 집합을 나타내며 함께 링크 될 수 있도록 미리 컴파일 된 형태인 Object code 형태로 존재합니다.

정적 라이브러리를 사용하여 컴파일을 하면 linker 가 프로그램이 필요로 하는 부분을 라이브러리에서 찾아 실행 파일에 바로 복사합니다. 따라서 실행할 때 별도의 라이브러리가 필요하지 않습니다. 하지만 실행파일의 크기가 복사된 라이브러리만큼 커지고, 같은 코드를 가진 여러 프로그램을 실행할 경우 코드가 중복이 되니 그만큼 메모리를 낭비합니다.

공유 라이브러리를 사용하여 컴파일을 하면 linker 가 실행파일에 실행될 때 우선적으로 공유 라이브러리를 로딩 시킨다는 표시를 해 둡니다. 그러면 실행할 때 라이브러리에 있는 컴파일 된 코드를 가져와 사용합니다. 이렇게 하는 경우 정적 라이브러리를 사용하는 것보다 파일 크기가 작아지고, 커널이 메모리에 복사한 공유 메모리를 보유하면서 다종의 프로그램들과 공유하기 때문에 메모리를 적게 차지하고, 사용 후 메모리에서 삭제되기 때문에 메모리 사용에 효율적입니다. 리눅스는 기본적으로 공유 라이브러리가 있으면 그것과 링크를 시키고, 그렇지 않으면 정적 라이브러리를 가지고 링크 작업을 합니다. 배포할 때는 공유 라이브러리를 함께 배포해야 합니다. 그렇지 않으면 실행 시 공유 라이브러리를 찾을 수 없다는 에러메시지가 발생합니다.

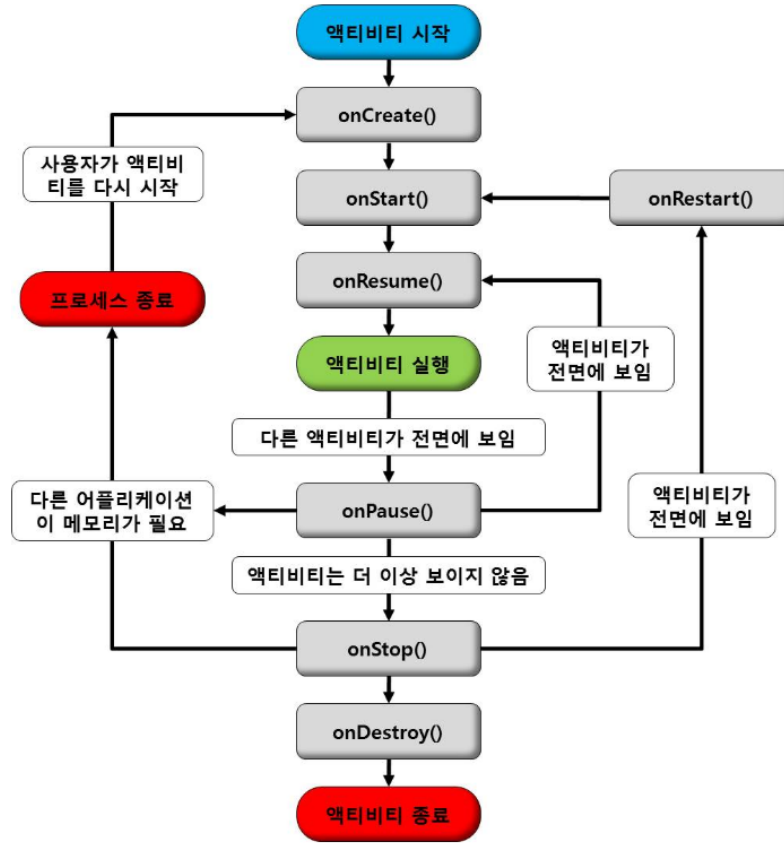
Observer Pattern



옵저버 패턴이란 객체지향 프로그래밍의 디자인 패턴 중 하나입니다. 옵저버 패턴에는 옵저버 또는 리스너라 불리는 하나 이상의 객체들이 한 관찰 대상을 가질 수 있습니다. 이 옵저버들은 subject(관찰대상)의 `observerCollection`에 명시되어 있고, 만약 subject 객체에 변화가 있어 옵저버들을 notify 하면 각 옵저버들은 해당 notification에 대한 메소드를 호출합니다. 옵저버 패턴이 쓰이는 곳의 예시 중 하나는 안드로이드의 `OnClickListener`입니다. 이 경우 `OnClickListener`가 옵저버이고 대상 버튼이 subject입니다. 만약 버튼을 클릭할 시 옵저버에 `notify`가 되고 `OnClickListener`에 정의되어 있는 메소드가 호출되게 됩니다.

Android Activity Lifecycle

안드로이드 activity 는 다음 그림과 같은 lifecycle 을 가지고 있습니다.



메소드	설명	다음 메소드
<code>onCreate()</code>	액티비티가 생성될 때 호출되며 사용자 인터페이스 초기화에 사용됨.	<code>onStart()</code>
<code>onRestart()</code>	액티비티가 멈췄다가 다시 시작되기 바로 전에 호출됨.	<code>onStart()</code>
<code>onStart()</code>	액티비티가 사용자에게 보여지기 바로 직전에 호출됨.	<code>onResume()</code> 또는 <code>onStop()</code>
<code>onResume()</code>	액티비티가 사용자와 상호작용하기 바로 전에 호출됨.	<code>onPause()</code>
<code>onPause()</code>	다른 액티비티가 보여질 때 호출됨. 데이터 저장, 스레드 중지 등의 처리를 하기에 적당한 메소드.	<code>onResume()</code> 또는 <code>onStop()</code>
<code>onStop()</code>	액티비티가 더이상 사용자에게 보여지지 않을 때 호출됨. 메모리가 부족할 경우에는 <code>onStop()</code> 메소드가 호출되지 않을 수도 있음.	<code>onRestart()</code> 또는 <code>onDestroy()</code>
<code>onDestroy()</code>	액티비티가 소멸될 때 호출됨. <code>finish()</code> 메소드가 호출되거나 시스템이 메모리 확보를 위해 액티비티를 제거할 때 호출됨.	없음

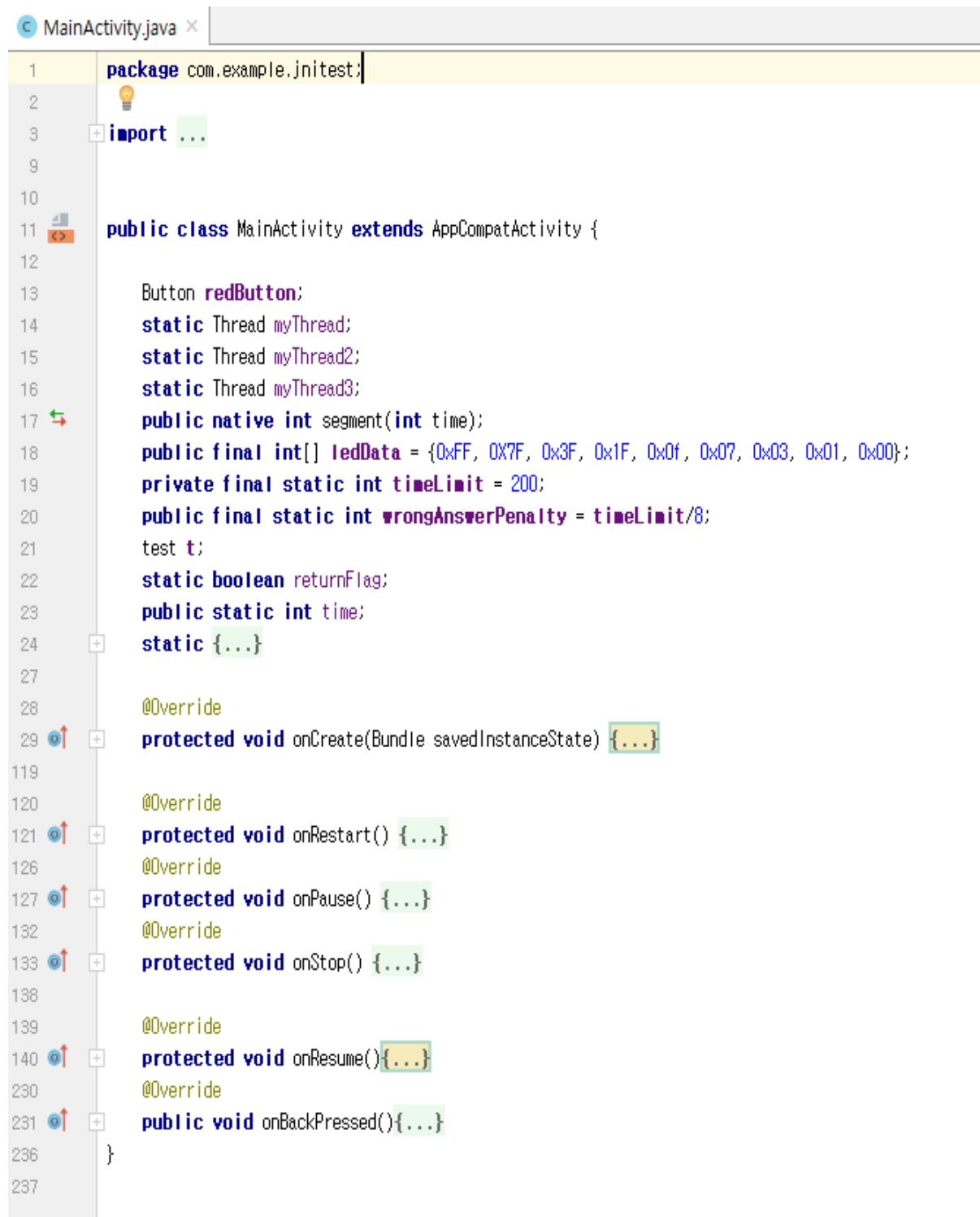
Thread

실행 중인 thread 상태를 변경하는 것을 thread 상태 제어라고 합니다.

- 주어진 시간 동안 일시 정지(sleep())
 - Thread 클래스의 sleep() 메소드는 실행 중인 thread 를 일정 시간 동안 멈추게 합니다. Thread.sleep() 메소드를 호출한 thread 는 주어진 시간 동안 일시 정지 상태가 되고, 다시 실행 대기 상태로 돌아갑니다. Sleep() 메소드는 밀리세컨드 단위로 시간을 입력합니다.
- 실행 중인 thread 에 예외처리 전달(interrupt())
 - Thread 클래스의 interrupt() 메소드는 일시 정지 상태의 thread 에서 InterruptedException 예외를 발생시켜, 예외 처리 코드에서 실행 대기 상태로 가거나 종료 상태로 갈 수 있도록 하는 메소드입니다.
- 다른 thread 에게 실행 양보(yield())
 - Thread 클래스의 yield() 메소드는 동작을 하는 thread 를 실행 시키고, 동작을 하지 않는 thread 는 실행 대기 상태로 만듭니다. Yield() 메소드를 호출한 thread 는 실행 대기 상태로 돌아가고 동일한 우선순위 또는 높은 우선순위를 갖는 다른 thread 가 실행 기회를 가질 수 있도록 합니다.
- 다른 thread 의 종료를 기다림(join())
 - Thread 는 기본적으로 다른 thread 와 독립적으로 실행하는 경우가 대다수입니다. 하지만 다른 thread 가 종료될 때까지 기다렸다가 실행해야 하는 경우가 발생할 수도 있습니다. 이런 경우를 위해 Thread.join() 메소드를 사용합니다. Thread A 가 Thread B 의 join() 메소드를 호출하면 Thread A 는 Thread B 가 종료할 때까지 일시 정지 상태가 됩니다. Thread B 의 run() 메소드가 종료되면 비로소 thread A 는 일시 정지에서 풀려 실행하게 됩니다.

- Thread 를 시작(start())
 - Thread 클래스의 start() 메소드는 thread 를 시작하고 run() 메소드를 호출합니다. 이때 run() 메소드에는 thread 가 실행할 동작이 구현되어 있습니다.

IV. Program Structure

A screenshot of a code editor showing the MainActivity.java file. The code is in Java and defines a MainActivity class that extends AppCompatActivity. It includes various static variables, a native method segment, and several lifecycle methods. The code is color-coded, and the IDE shows line numbers on the left and a gutter with icons for breakpoints and other features.

```
1 package com.example.jnittest;
2
3 import ...
4
5
6
7
8
9
10
11 public class MainActivity extends AppCompatActivity {
12
13     Button redButton;
14     static Thread myThread;
15     static Thread myThread2;
16     static Thread myThread3;
17     public native int segment(int time);
18     public final int[] ledData = {0xFF, 0x7F, 0x3F, 0x1F, 0x0f, 0x07, 0x03, 0x01, 0x00};
19     private final static int timeLimit = 200;
20     public final static int wrongAnswerPenalty = timeLimit/8;
21     test t;
22     static boolean returnFlag;
23     public static int time;
24     static {...}
25
26
27
28     @Override
29     protected void onCreate(Bundle savedInstanceState) {...}
30
31
32
33
34
35
36
37
38
39
40     @Override
41     protected void onRestart() {...}
42
43     @Override
44     protected void onPause() {...}
45
46     @Override
47     protected void onStop() {...}
48
49
50
51     @Override
52     protected void onResume() {...}
53
54     @Override
55     public void onBackPressed() {...}
56 }
57
```

```

myThread = new Thread((Runnable) () -> {
    while(true) {
        try {
            if (time <= 0) {
                //Log.d("thread1", "time over!");
                t.led( data: 0);
                Intent intent = new Intent( packageContext: MainActivity.this, timeover.class);
                startActivity(intent);
                break;
            } else {
            }
            Thread.sleep( millis: 1000);
            if(time > 0)
                time--;
            int numberOfLed = 0;
            numberOfLed = (time+wrongAnswerPenalty-1)/wrongAnswerPenalty;
            t.led(ledData[numberOfLed]);
            //Log.d("thread1", String.format("%d sec", time));
        } catch (InterruptedException e) {
            //Log.d("test", "thread1 killed");
            break;
        }
    }
});

```

Time이 0 이하이면(시간 초과) led를 모두 끄고 timeover activity로 이동합니다. 0 보다 크다면 1초 동안 thread를 sleep하고 나서 time을 1 감소 시킵니다. 그리고 남은 시간에 따라 켜져 있을 led 수를 구한 후 led를 켜줍니다. 인터럽트가 발생 한다면 무한 루프를 나와 thread를 종료합니다.

```

myThread2 = new Thread((Runnable) () -> {
    while(true) {
        try {
            segment(time);
            if (time <= 0) {
                //Log.d("thread2", "time over!");
                //break;
            } else {
            }
            //Log.d("thread2", String.format("%d sec", time));
            myThread2.sleep( millis: 0);
        } catch (InterruptedException e) {
            Log.d( tag: "test", msg: "thread2 killed");
            break;
        }
    }
});

```

Time을 segment에 지속적으로 출력해주는 thread입니다. 인터럽트가 발생하면 종료됩니다.

```

myThread3 = new Thread((Runnable) () → {
    while(true) {
        //time = JNIString();
        try {
            if(time > 0)
                t.dotmatrix( input: "Warning!!!");
            else {
                t.dotmatrix( input: "TIME OVER!!!");
                break;
            }
            myThread3.sleep( millis: 0);
        } catch(InterruptedException e){
            t.dotmatrix( input: "");
            Log.d( tag: "test", msg: "thread3 killed");
            break;
        }
    }
});

```

Dot matrix에 문자열을 출력하는 thread입니다. Time이 0보다 크면 "Warning!!!"을 출력하고 그렇지 않다면 "TIME OVER!!!"를 출력합니다. 인터럽트가 발생하면 thread를 종료합니다.

```

JNIEXPORT jint JNICALL Java_com_example_jnittest_MainActivity_segment (JNIEnv *env, jobject obj, jint time) {
    int dev, ret;
    dev = open("/dev/fpga_segment", O_RDWR | O_SYNC);
    if(time > 0){
        if (dev != -1) {
            ret = write(dev, &time, sizeof(time));
        } else {
            exit(1);
        }
    }
    close(dev);
    return time;
}

```

Jint를 입력 받아 segment에 출력해주는 JNI입니다.

```
stage1.java x
1 package com.example.jnittest;
2
3 import android.content.Intent;
4 import android.support.v7.app.AppCompatActivity;
5 import android.os.Bundle;
6 import android.util.TypedValue;
7 import android.view.View;
8 import android.widget.Button;
9 import android.widget.TextView;
10
11 import java.util.Random;
12
13 public class stage1 extends AppCompatActivity {
14     TextView byteQuiz;
15     Button clear;
16     int byteAnswer;
17     public native int dipswitch();
18     test t;
19     @Override
20     protected void onCreate(Bundle savedInstanceState) {...}
21
22     @Override
23     public void onBackPressed(){}
24 }
25
26
27
```

2byte 크기의 랜덤 값이 16진수 형식으로 출력이 됩니다. 사용자는 16개의 dipswitch를 이용하여 화면에 나온 2byte 수와 같게 만들면 됩니다. 가장 왼쪽 switch가 최상위 비트이며 switch가 on이면 1 off면 0을 의미합니다.


```

JNIEXPORT jint JNICALL Java_com_example_jintest_stage1_dipswitch (JNIEnv *env, jobject obj) {
    int data=0;
    int temp;
    int temp2;
    int dipswInput[2];
    int fd = open("/dev/fpga_dipsw", O_RDWR);
    int ret = read(fd, &data, 2);
    if(ret == 2){
        dipswInput[0] = (data & 0xFF0000) >> 16;
        dipswInput[1] = (data & 0xFF);
        for(int i = 0; i < 2; i++){
            temp = dipswInput[i] & 0xf;
            temp2 = (temp >> 3) | ((temp & 0x1) << 3) | ((temp & 0x2) << 1) | ((temp & 0x4) >> 1);

            temp = dipswInput[i] >> 4;
            temp2 |= ((temp >> 3) | ((temp & 0x1) << 3) | ((temp & 0x02) << 1) | ((temp & 0x4) >> 1)) << 4;
            dipswInput[i] = temp2;
        }
    }
    close(fd);
    data = dipswInput[0] | (dipswInput[1] << 8);
    return data;
}

```

Dipswtich의 bit순서를 재배열하는 JNI입니다.

```

1  package com.example.jnittest;
2
3  import ...
4
11
12  public class stage2 extends AppCompatActivity {
13      Random r = new Random();
14      TextView piezoInput;
15      Button[] piezoButton;
16      Button clear;
17      Button play;
18      String piezoPrintString = "";
19      String piezoQuestionString = "";
20      Thread piezoThread;
21      int[] piezoQuestion;
22      test t;
23      public static native void piezo(int data);
24      final static String[] note = {"도", "레", "미", "파", "솔", "라", "시"};
25      final static int[] piezoData = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07};
26      final int backspace = 7;
27
28      @Override
29      protected void onCreate(Bundle savedInstanceState) {...}
30
103
104      public void piezoPrint(View v){...}
120
121      @Override
122      public void onBackPressed(){}
124  }
125

```

Play 버튼을 누르면 도부터 시까지 무작위로 4개의 음이 재생됩니다. 사용자는 해당하는 음의 게이름이 적힌 버튼을 눌러 재생된 4개의 음을 맞추면 됩니다.

```

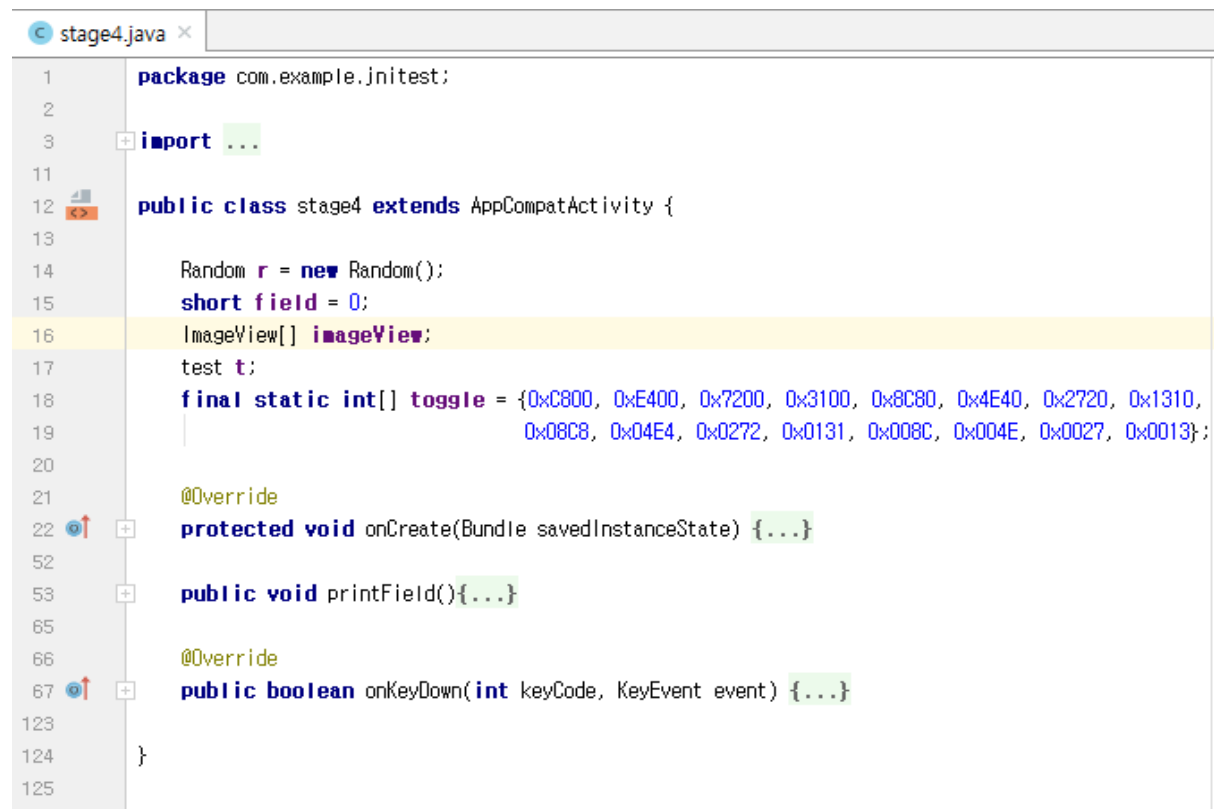
JNIEXPORT void JNICALL Java_com_example_jnittest_stage2_piezo (JNIEnv *env, jobject job, jint data) {
    int fd, ret;

    fd = open("/dev/fpga_piezo", O_WRONLY);

    if (fd < 0) return;
    ret = write(fd, &data, 1);
    usleep(250000);
    close(fd);
    if (ret == 1) return;
    return;
}

```

Jint를 입력 받아 해당 음계를 재생하는 Piezo JNI입니다.



```

1 package com.example.jnittest;
2
3 import ...
4
11
12 public class stage4 extends AppCompatActivity {
13
14     Random r = new Random();
15     short field = 0;
16     ImageView[] imageView;
17     test t;
18     final static int[] toggle = {0xC800, 0xE400, 0x7200, 0x3100, 0x8C80, 0x4E40, 0x2720, 0x1310,
19                                     0x08C8, 0x04E4, 0x0272, 0x0131, 0x008C, 0x004E, 0x0027, 0x0013};
20
21     @Override
22     protected void onCreate(Bundle savedInstanceState) {...}
23
24
25     public void printField(){...}
26
27
28     @Override
29     public boolean onKeyDown(int keyCode, KeyEvent event) {...}
30
31 }

```

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1번부터 16번의 칸이 있습니다. 이 중 한 칸이 눌리면 그 칸을 포함한 상하좌우의 칸들의 색(또는 이미지)가 반전됩니다. 예를 들어, 1번 칸이 눌리면 1, 2, 5번칸이 반전 되고 6번이 눌리면 2, 5, 6, 7번 칸이 반전 되는 것입니다.

전체를 16비트 data type(short)이라 가정합니다. 0과 1에 색 또는 이미지를 부여하고 1번 칸을 최상위 비트, 16번 칸을 최하위 바트로 설정합니다. 각 칸이 눌렸을 때, 반전 되는 칸은 1과 XOR연산을 하고 반전되지 않는 칸은 0과 XOR연산을 하도록 toggle값을 각 칸에 대하여 구합니다.

각 칸의 색은 자신을 포함한 주위의 칸이 몇 번 입력 되었는지에 따라 결정됩니다. 같은 칸이 같은 횟수 눌렀다면 눌린 순서와 상관 없이 같은 결과가 나옵니다. 각 칸이 짝수 번 눌린 것은 그 칸을 누르지 않은 것과 같고 홀수 번 눌린 것은 그 칸을 한 번 누른 것과 같은 결과가 나옵니다. 이와 같은 성질을 이용하여 1번 칸부터 16번 칸까지 0또는 1의 난수를 생성하여 각 칸의 toggle값과 곱하여 전체 16비트 data에 XOR연산을 해줍니다.

이런 방식으로 생성된 퍼즐을 사용자가 key를 이용하여 모두 같은 색으로 만들면 됩니다.

@Override

public boolean onKeyDown(int keyCode, KeyEvent event) {

```
    switch (keyCode) {  
        case KeyEvent.KEYCODE_Q:  
            field ^= toggle[0];  
            break;  
        case KeyEvent.KEYCODE_W:  
            field ^= toggle[1];  
            break;  
        case KeyEvent.KEYCODE_E:  
            field ^= toggle[2];  
            break;  
        case KeyEvent.KEYCODE_R:  
            field ^= toggle[3];  
            break;  
        case KeyEvent.KEYCODE_T:  
            field ^= toggle[4];  
            break;  
        case KeyEvent.KEYCODE_Y:  
            field ^= toggle[5];  
            break;  
        case KeyEvent.KEYCODE_U:  
            field ^= toggle[6];  
            break;  
        case KeyEvent.KEYCODE_I:  
            field ^= toggle[7];  
            break;  
        case KeyEvent.KEYCODE_O:  
            field ^= toggle[8];  
            break;  
        case KeyEvent.KEYCODE_P:  
            field ^= toggle[9];  
            break;  
        case KeyEvent.KEYCODE_1:  
            field ^= toggle[10];  
            break;  
        case KeyEvent.KEYCODE_2:  
            field ^= toggle[11];  
            break;  
        case KeyEvent.KEYCODE_3:  
            field ^= toggle[12];  
            break;  
        case KeyEvent.KEYCODE_4:  
            field ^= toggle[13];  
            break;  
    }
```

```
char keypad_fpga_keycode[16] = {  
    KEY_Q, KEY_W, KEY_E, KEY_R,  
    KEY_T, KEY_Y, KEY_U, KEY_I,  
    KEY_O, KEY_P, KEY_1, KEY_2,  
    KEY_3, KEY_4, KEY_5, KEY_6  
};
```

Keypad의 KeyEvent를 처리합니다.

```

clear.java x
1  package com.example.jnittest;
2
3  import ...
6
7  public class clear extends AppCompatActivity {
8      test t;
9      Thread myThread3;
10
11  @Override
12  protected void onCreate(Bundle savedInstanceState) {...}
44
45  @Override
46  public void onBackPressed(){...}
56
}
```

모든 게임을 성공적으로 마치면 이동하는 clear activity입니다.

```

timeover.java x
1  package com.example.jnittest;
2
3  import ...
7
8  public class timeover extends AppCompatActivity {
9      test t;
10
11  @Override
12  protected void onCreate(Bundle savedInstanceState) {...}
17
18
19  public void retry(View v){...}
32
33  @Override
34  public void onBackPressed(){...}
47
48
}
```

시간이 초과하면 이동하는 timeover activity 입니다.

```

test.java x
1 package com.example.jntest;
2
3 public class test {
4     public native int textIcd(String input);
5     public native void led(int data);
6     public native int dotmatrix(String input);
7     static {
8         System.loadLibrary( libname: "JNIString");
9     }
10 }
11

```

Hardware module에 대한 입출력 함수입니다.

```

JNIEXPORT jint JNICALL Java_com_example_jntest_test_dotmatrix (JNIEnv *env, jobject obj, jstring input) {
    int dev, i, j, offset = 20, ch, len;
    char result[600], tmp[2];
    jboolean isSucceed;
    const char *string = (env)->GetStringUTFChars(input, &isSucceed);

    dev = open("/dev/fpga_dotmatrix", O_WRONLY);

    if (dev != -1) {
        //printf("Input text : ");

        //scanf("%s", input);
        //char input[] = "WARNING!";
        len = strlen(string);

        for (j = 0; j < 20; j++)
            result[j] = '0';

        for (i = 0; i < len; i++) {
            ch = string[i];

            ch -= 0x20;

            for (j = 0; j < 5; j++) {
                sprintf(tmp, "%x", font[ch][j] / 16, font[ch][j] % 16);

                result[offset++] = tmp[0];
                result[offset++] = tmp[1];
            }
            result[offset++] = '0';
            result[offset++] = '0';
        }

        for (j = 0; j < 20; j++)
            result[offset++] = '0';

        for (i = 0; i < (offset - 18) / 2; i++) {
            for (j = 0; j < 20; j++) {
                write(dev, &result[2 + i], 20);
            }
        }
    } else {
    }
    close(dev);
    return 0;
}

```

문자열을 입력 받아 dotmatrix에 출력하는 JNI입니다.

```

JNIEXPORT void JNICALL Java_com_example_initest_test_led(JNIEnv *env, jobject obj, jint data) {
    int fd;
    int ret;
    fd = open("/dev/fpga_led", O_RDWR);
    if(fd < 0) return;
    if(fd > 0) {
        data &= 0xff;
        write(fd, &data, 1);
        close(fd);
    }
}
}

```

Jint를 입력 받아 led를 출력하는 JNI입니다.

```

JNIEXPORT jint JNICALL Java_com_example_initest_test_textlcd(JNIEnv *env, jobject obj, jstring input) {
    int fd;

    fd = open("/dev/fpga_textlcd", O_WRONLY);
    assert(fd != -1);

    jboolean isSucceed;
    const char *string = (env->GetStringUTFChars(input, &isSucceed));
    ioctl(fd, TEXTLCD_INIT);
    ioctl(fd, TEXTLCD_CLEAR);
    ioctl(fd, TEXTLCD_LINE1);
    write(fd, string, strlen(string));
    ioctl(fd, TEXTLCD_OFF);
    close(fd);
    return 0;
}

```

문자열을 입력 받아 textlcd에 출력해주는 JNI입니다.

V. Build Environment

이 프로젝트에서 사용하는 안드로이드 커널은 Linux(Assam) 서버에서 make 하여 zImage 를 생성하였습니다. 윈도우 OS 에서 fastboot 프로그램을 사용하여 안드로이드 보드에 생성한 zImage 를 안드로이드 커널에 입력하였습니다. 어플리케이션은 Window OS 에 안드로이드 스튜디오에서 구현 및 컴파일을 하였고, Hanback Electronics SM5-S4210-M3 보드의 안드로이드 OS 에서 실행시켰습니다.

VI. Problem & Solution

- **Problem:** Matrix, Piezo, 타이머에 사용하는 7-Segment 모두 각각 하드웨어입니다. 각각의 하드웨어가 모두 동작하려고 하기 때문에 하나의 디바이스가 동작 중에는 다른 디바이스들이 동작하지 못하고 순서대로 디바이스가 끝나면 다음 디바이스가 동작하게 됩니다.
 - **Solution:** 이러한 디바이스들끼리의 동시 사용 문제를 해결하기 위해서 디바이스를 각각 thread 를 사용하여 동작하도록 하였습니다. 이를 통해 동시에 다수의 디바이스들이 동작하도록 하였습니다.
- **Problem:** 여러 액티비티에서 공통으로 사용되는 모듈의 JNI 가 메인 액티비티에서 사용할 수 있게 되어있는데, 다른 액티비티에서 사용하기 위해 static method 로 선언했을 때 문제가 되었습니다.
 - **Solution:** 공통으로 사용되는 모듈을 클래스로 묶어 사용했습니다. 모듈이 필요하면 그 클래스를 생성하여 사용하도록 하였습니다.
- **Problem:** Keypad 의 이벤트 처리를 JNI 를 이용하려고 하였으나 실패하였습니다.
 - **Solution:** JNI 를 사용하지 않고 android 에서 제공되는 onKeyDown 으로 처리하였습니다.

- **Problem:** 초기 커널을 생성할 때 기본으로 설정된 keypad 설정 값 중 몇몇 값이 인식이 되지 않는 문제가 있었습니다.
 - **Solution:** 인식이 되는 키를 알아내어 Keypad 디바이스 드라이버 파일을 수정하였습니다.
- **Problem:** 처음 Dip Switch JNI 를 구현했을 때 read 한 Dip Switch 의 데이터 값이 예상 값과 차이가 있었습니다. 반환된 Dip Switch 데이터를 관찰을 해본 결과 오른쪽 8 개의 Dip Switch 가 0xFF0000 자리에 있었고 왼쪽 8 개의 Dip Switch 가 0x0000FF 자리에 있었습니다. 또한 4 비트씩 반전되어 있었습니다.
 - **Solution:** 문제를 해결하기 위해 Dip Switch JNI 에서 데이터를 read 한 직후 비트 쉬프팅과 비트 연산을 하여 예측한 데이터와 동일한 데이터를 반환 시키게 했습니다.
- **Problem:** 쓰레드 종료 문제로 게임 실패했을 때 리트라이버튼을 누르고 시작화면에서 시작버튼을 두 번 눌러야 게임이 재 시작하는 문제가 발생했습니다.
 - **Solution:** Thread 가 정상적 종료되지 않아 생긴 문제였습니다. Retry 버튼을 누르면 Thread 를 종료하도록 만들었습니다. 또한 retry 버튼을 눌러 시작 화면(main activity)으로 넘어가면 main activity 에서는 onResume 이 호출된다는 사실을 이용하여 onResume 에 onCreate 에서 해주는 task 를 일부 해주었습니다. 그런데 onResume 은 activity 의 lifecycle 을 따라 activity 가 시작되고 실행될 때까지 한 번은 거치는 과정이기 때문에 이것이 초기 생성에 의해 호출된 onResume 인지 retry 에 의해 호출된 onResume 인지를 구분되지 않는 문제가 생겼습니다. 이를 구분하기 위하여 flag 변수를 하나 두어 구분하였습니다.
- **Problem:** 디바이스를 사용하기 위해서는 디바이스 드라이버인 소프트웨어의 도움을 받아야하는데 이 때 처음에 디바이스를 open 하는 과정에서 dev 디렉토리 밑에 있는 디바이스들을 사용하여 open 을 합니다. 하지만 fd 인 file

descriptor 를 open 후 확인해보면 0 보다 작은 값을 가지고 있어 디바이스를 찾지 못하는 error 가 발생했습니다.

➤ **Solution:** 이런 문제를 위해 디바이스 모듈이 들어있는 디렉토리인 dev 를 확인하였고 우리가 알고있는 디바이스의 이름이 잘못되어 부르지 못한 것을 확인 할 수 있었습니다. 이를 확인한 후 입력되어 있는 이름으로 사용하여 open 을 하였고 error 없이 진행되었습니다.

- **Problem:** 타이머를 위한 스레드가 하나였을 경우 타이머의 시간 측정 매우 부정확 했습니다. (thread, sleep, 디바이스 드라이버)

➤ **Solution:** 실질적으로 잔여 시간을 감소시키는 thread 와 잔여 시간을 출력해주는 thread 를 각각 생성하였습니다.

- **Problem:** Clear 는 어플리케이션에서 작동하는 모든 문제들을 해결하면 성공을 의미하는 activity 입니다. 이 때 성공을 하는 경우에 대해 남은 시간이 정지하지 않고 계속 진행하여 최종적으로 clear 에서 다시 timeover activity 로 넘어가는 현상이 발생했습니다.

➤ **Solution:** 이 문제를 해결하기 위해 clear activity 로 들어가는 경우 time 을 감소하는 thread 는 동작을 중지시키고 time 을 출력해주는 thread 는 계속 진행하도록 하여 모든 문제가 끝난 후 플레이어가 얼마만큼의 시간이 남은 상태에서 문제를 해결하였는지 보여 주었습니다.

- **Problem:** Main activity 에서 실행할 때 Dot Matrix 로 사용하는 thread 에서 두가지 문장을 선택할 수 있습니다. 하나는 문제를 푸는 과정에서 warning 이라는 경고 문구이며 다른 하나는 남은 시간이 없어 게임이 종료된 후의 timeover 입니다. 이 후에 게임을 성공적으로 마친 경우에 대해서 success 를 출력하려고 하였을 때 어플리케이션이 나가지는 error 가 발생하였습니다.

➤ **Solution:** 위의 문제도 다른 문제들과 비슷한 thread 문제입니다. 이미 warning 이나 timeover 를 위해 thread 로 사용하고 있는 디바이스를

success 문구를 출력하기 위해 다시 open 을 하였기 때문에 충돌이 났습니다. 이를 해결하기 위해서 이전에 Dot Matrix 를 출력하던 thread 를 종료시키고 success 를 출력하기 위해 다시 thread 를 생성하도록 하였습니다.

- **Problem:** Serial 통신을 통해 보드가 실행 중에 log 를 확인해 보면 디바이스들을 init 하는 과정이 들어가있는 것을 볼 수 있습니다. 처음 디바이스를 init 하는 과정에서는 error 가 발생하지 않지만 이미 init 한 디바이스들을 다시 init 하는 과정에서 error 메시지들이 출력됩니다.

➤ **Solution:** Menuconfig 에서 각각의 사용하는 디바이스를 built-in 으로 설정하면 자동적으로 생성할 때 커널에 모듈을 삽입합니다. 하지만 module 의 경우에는 모듈을 직접 insmod 해 주어야 합니다. 하지만 한백전자에서 제공하는 Makefile 에서 자동적으로 모듈들을 다시 init 해 주는 스크립트가 작성되어 built-in 으로 이미 init 한 모듈을 다시 init 하도록 합니다. 이러한 점을 방지하기 위해 이미 존재하는 디바이스의 이름을 바꾸어 같은 디바이스 이름을 피해 error 를 나지않게 할 수도 있고 스크립트로 작성된 파일을 Makefile 을 수정하여 실행하지 않도록 하는 방법도 있습니다.

- **Problem:** 사용자가 오답을 냈을 경우 잔여 시간을 감소시켜야 할 필요가 있었습니다.

➤ **Solution:** 사용자가 오답을 냈을 때 잔여 시간을 감소시키기 위하여 잔여 시간 등을 static 변수로 두었습니다.

- **Problem:** 어플리케이션의 문제를 모두 해결하거나 문제를 해결하지 못하여 종료한 경우 다시 새로운 게임을 시작할 때 실행 중인 thread 를 재 생성하면 충돌하여 문제가 발생합니다. 이러한 점을 해결하기 위해 thread 를 종료하기 위해 thread.stop()을 사용하였지만 thread.stop()은 불안정하게 thread 를 정리하여 남거나 error 의 요소가 강하다는 이유로 이미 deprecate 되었습니다.

- **Solution:** `thread.stop()`이 없어진 대신 `thread.interrupt()`를 사용하여 `thread` 를 종료시킬 수 있습니다. 하지만 이때 `thread` 를 종료시키는 과정에서 주의해야하는 점들이 있습니다. `Thread.interrupt()` 메소드는 `thread` 가 일시 정지 상태에 있을 때 `InterruptedException` 예외를 발생시키는 역할을 합니다. 이 때문에 `interrupt()` 메소드를 사용하기 위해서는 종료시키고 싶은 메소드가 일시 정지 상태일 때 종료시킬 수 있습니다. `Thread` 가 실행 대기 또는 실행 상태에 있을 때 `interrupt()` 메소드가 실행되면 즉시 `InterruptedException` 예외가 발생하지 않습니다. `Thread` 가 미래에 일시 정지 상태가 되면 `InterruptedException` 예외가 발생합니다. 이 때문에 반드시 `thread` 를 일시 정지 상태로 변환해 주어야 합니다.
- **Problem:** CPP 파일로 생성한 JNI 함수들에 대해서 오버로딩을 사용하려고 하였지만 문제가 발생하였습니다.
 - **Solution:** 원인을 찾은 결과 `javah` 를 통해 생성한 header 파일에서 `_cplusplus` 가 이미 정의되어 `extern "C"`를 실행하도록 합니다. 여기서 `extern "C"`의 의미는 아래의 함수 호출 또는 선언들에 대해서 C 언어 규약 조건을 따른다는 의미를 나타냅니다. 따라서 OOP 개념이 존재하지 않는 C에서는 오버로딩을 사용할 수 없었습니다.
- **Problem:** 보드에 있는 하드웨어 버튼 중 뒤로 가기 버튼을 사용하면 그전 activity 에서 사용 중이던 `thread` 가 이미 실행이 되어있고 다시 다음 단계로 진행하기 위해서는 이미 사용중인 스레드가 다시 생성합니다. 이런 경우에 충돌이 발생하여 어플리케이션이 종료됩니다.
 - **Solution#1:** 이런 경우를 막기위해서 2 가지 방법을 선택할 수 있습니다. 매 경우 `onCreate` 에서 실행하던 `thread` 를 죽이고 다시 생성하는 경우가 하나의 방법입니다. 이런 방법은 앱의 처음부터 실행되고 이후에도 계속 유지가 되는 `thread` 가 있을 때 이 `thread` 를 죽이면 이후 다시 살려야 하므로 무의미합니다.

- **Solution#2:** 다른 방법은 뒤로 돌아가는 하드웨어 기능을 임의로 사용하지 못하게 만드는 방법이 있습니다. 우리가 만든 어플리케이션은 단계별로 진행하는 게임이며 뒤로 돌아갈 필요가 없는 어플리케이션입니다. 이러한 stage 별 어플리케이션 문제이기 때문에 뒤로 가는 하드웨어를 지우고 마지막 어플리케이션을 종료시키는 경우에만 뒤로 가기 하드웨어에 실행 중이던 thread 를 모두 종료하고 어플리케이션을 종료하게 만들었습니다.
- **Problem:** Time 이 줄어드는 동안 led 을 켤 때 남은 시간의 나머지가 0 임을 확인하여 횃수가 증가할 때마다 led 를 하나씩 증가하며 출력하였습니다. 하지만 문제를 풀며 오답에 의한 페널티를 부여할 때 마다 시간을 삭감하였고 이 때문에 조건 문을 확인하는 횃수가 줄어 led 가 켜지는 경우가 줄었습니다.
 - **Solution:** 일반적인 경우로 시간이 감소했을 때는 led 가 잘 켜졌지만 페널티로 인한 시간 감소인 경우는 led 가 정상적으로 켜지지 않았습니다. 이를 해결하기 위하여 기존에는 조건 문을 이용해서 나머지가 몇 인지에 따라 led 를 켤지를 결정했습니다. 이를 개선하여 남은 시간에 따른 led 를 계산하여 그 개수만큼 led 를 켜도록 개선하였습니다.
- **Problem:** Dot matrix 를 사용함에 있어서 지나가는 속도가 너무 느려 문장을 확인할 때 오래 걸렸습니다.
 - **Solution:** 이를 해결하기 위해 디바이스 드라이버를 확인해 보았고 확인한 결과 m_delay 를 사용하여 문구가 지나가는 속도를 나타내었습니다. 이를 확인하여 기존에 m_delay(3)을 m_delay(1)로 수정하여 문구가 빠르게 지나갈 수 있도록 하였습니다.
- **Problem:** 어플리케이션을 실행할 때 어떠한 에러로 어플리케이션이 종료되었는지 알기 힘들었습니다
 - **Solution:** 로그를 사용하여 디버깅 창에 원하는 데이터를 출력하였습니다.

VII. Personal Feelings

이도경 – 디바이스 드라이버와 JNI 를 다루는 것은 처음 해보는 경험이었습니다. 초반에는 생소한 개념으로 인해 잠시 진도 나가는게 늦었지만 중반부터는 개념에 대한 이해가 조금씩 생기고 흥미도 생겨 진도가 빠르게 나갔습니다. 어플리케이션을 만드는 것 특히 디자인하는 부분이 많이 편해진 것 같습니다. Main activity 에서 onCreate 부분과 onResume 부분의 코드 중복을 없애지 못한 것이 아쉬웠습니다.

이대훈 – 이번 과제를 통해 운영체제 과목에서 하지 못한 디바이스 드라이버를 볼 수 있는 기회를 가져서 큰 도움이 되었습니다. 디바이스 드라이버의 동작 원리 및 코드를 볼 수 있었고 직접 수정하여 빌드한 커널을 사용하여 완성한 어플리케이션을 실행해 볼 수 있었습니다. 어플리케이션을 만들어 보는 것보다는 디바이스 드라이버를 보고 사용하는 것에 더 큰 흥미를 가졌습니다. 또한 Java 를 처음 사용해 봄으로써 OOP 개념을 좀더 이해할 수 있었습니다. 특히 어플리케이션등의 안드로이드 스튜디오에서는 Java 만을 사용한다는 인식을 깰 수 있는 NDK 를 배움으로써 좀 더 넓은 방향이 있다는 것을 알 수 있는 시간이었습니다.

이종원 – 이번 과제는 참 신기했습니다. 저희가 직접 디바이스 드라이버 코드를 수정하고 menuconfig 에서 사용할 디바이스를 선정하고 커널을 make 하고 안드로이드 기기에 탑재시켜 보드의 주변기기에 원하는 행동을 하는 것을 봤을 때 아주 신기했습니다. 이번 프로젝트가 안드로이드 스튜디오를 처음 사용한 기회여서 많은 어려움을 겪었지만 조원들의 도움을 받아 극복할 수 있었습니다.