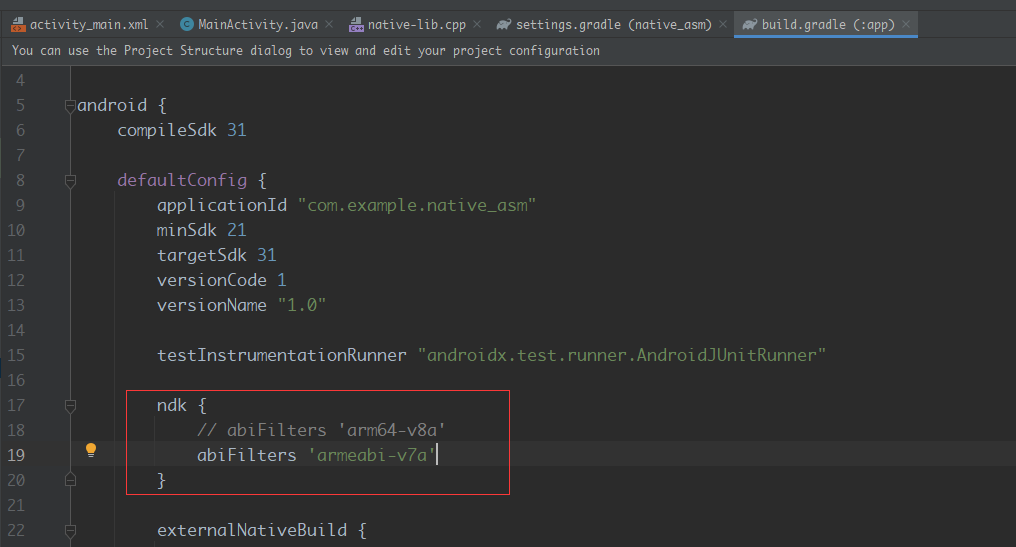
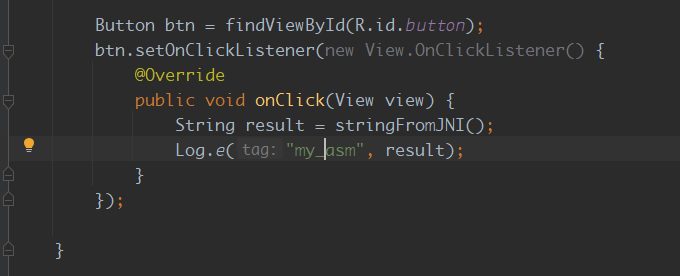
使用汇编重写一些系统函数可以防止直接从import符号中hook，增加逆向难度

# 使用内联汇编

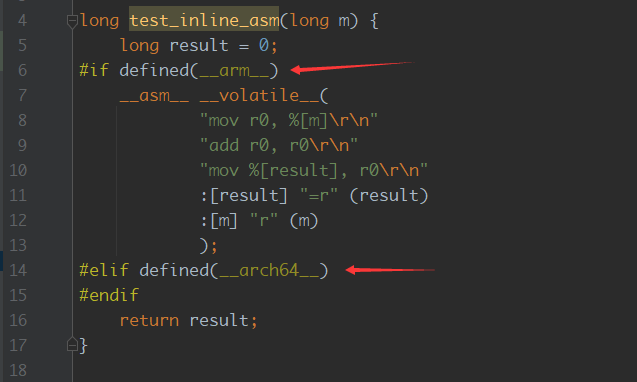
1. 首先创建一个Native C++工程
2. 在build.gradle(:app)中添加ndk的abi



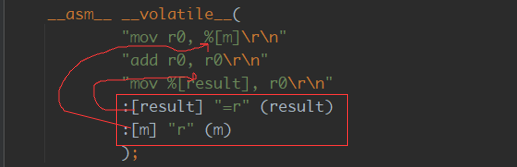
1. 这里创建一个按钮进行测试



1. 写一个输入\*2的汇编进行测试



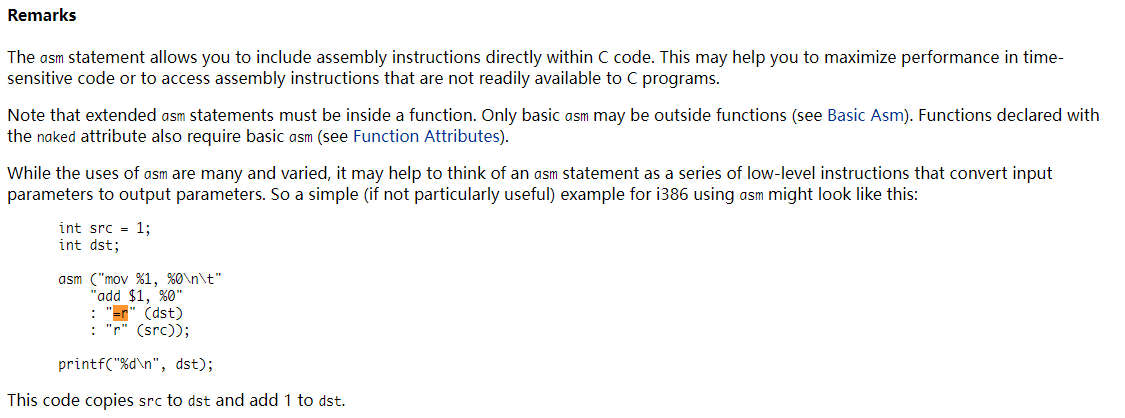
这里我们用#if defined(\_\_arm\_\_)和#elif defined(\_\_arch64\_\_)来分别写对应的arm和arm64的汇编，这个环境变量和我们第二部中的abi相对应。



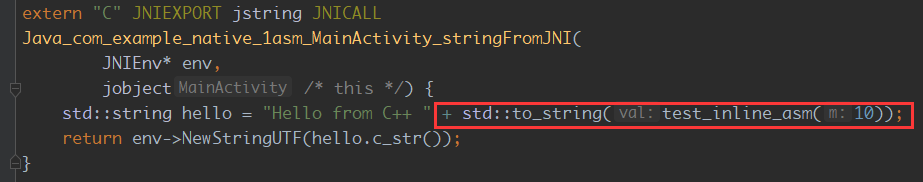
这两行是用来引入外部变量的，后面的圆括号代表外部的变量，前面的方括号代表asm中应用的别名（在前面加上%）。详情见：<https://gcc.gnu.org/onlinedocs/gcc/Extended-Asm.html>

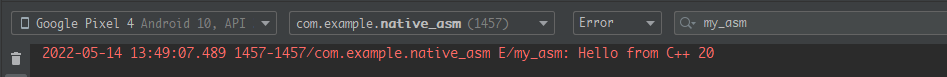
<https://gcc.gnu.org/onlinedocs/gcc/Constraints.html#Constraints>

<https://gcc.gnu.org/onlinedocs/gcc/Modifiers.html#Modifiers>

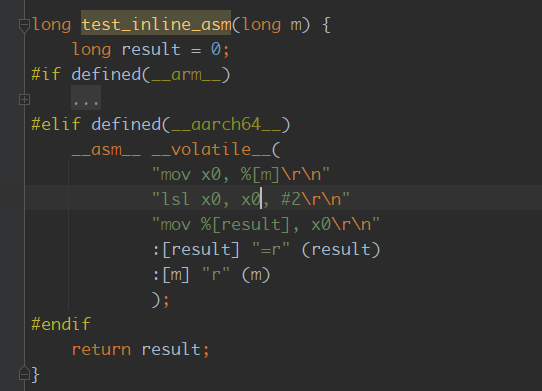


1. 绑定到button，并在log查看效果





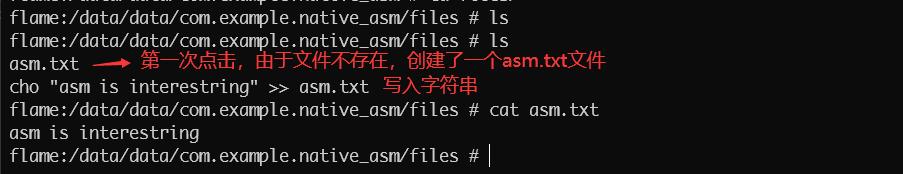
1. 同样的方法写个arm64的，将输入\*4

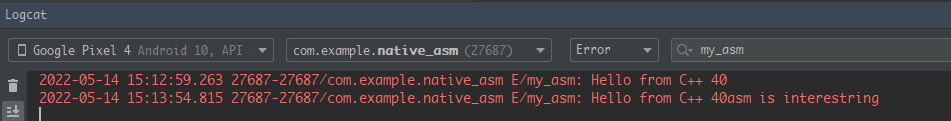


# Syscall读取一个文件

我们使用syscall调用openat，read，close来读取一个文件，这里我们给读和创建权限

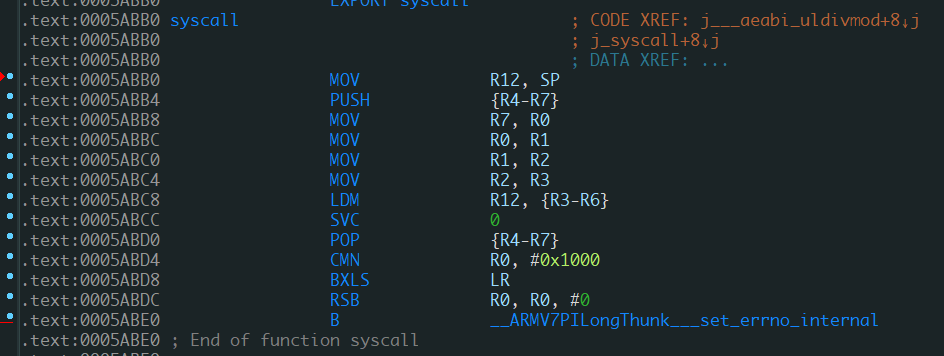






## 用asm代替syscall

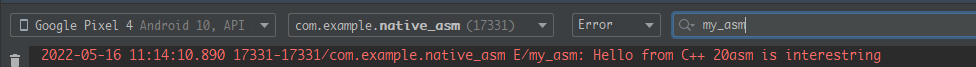
Syscall在libc.so中，32位和64位的分别在/system/lib和/system/lib64目录下，pull下来用ida打开



用这个汇编，自定义一个syscall函数。

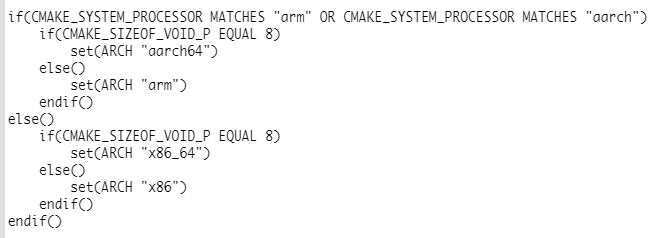


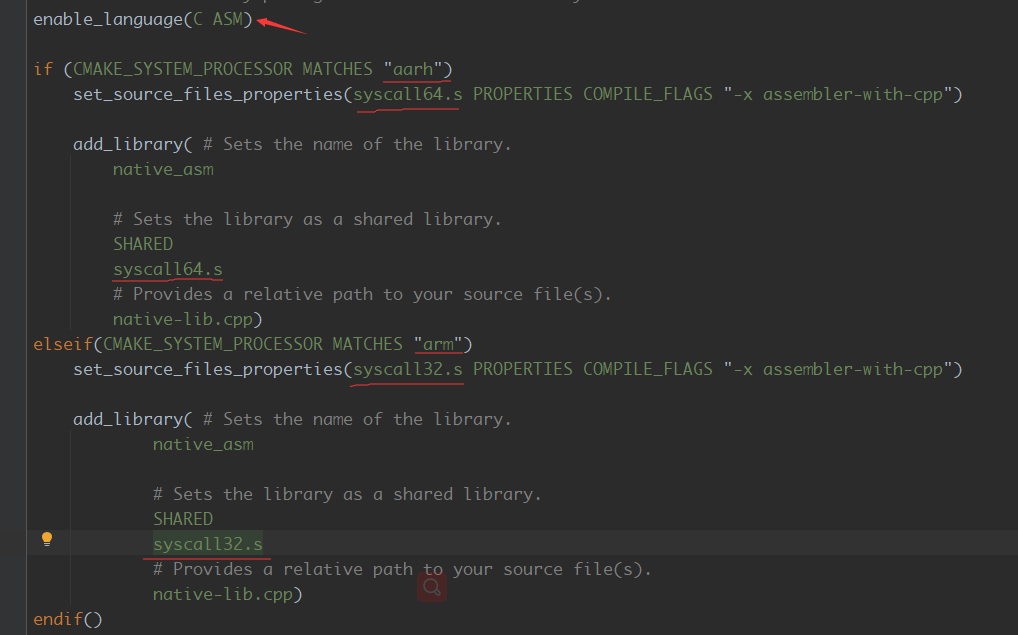
成功执行。



## 将32位和64位syscall分别放在单独文件

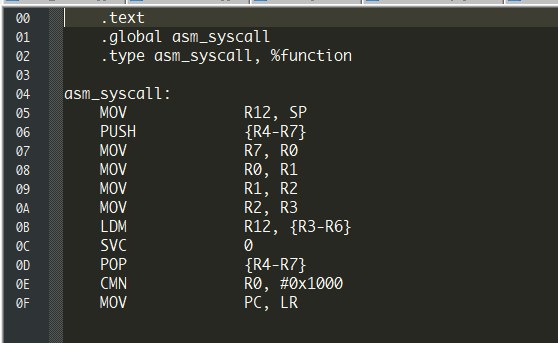
关于架构的判断主要用CMAKE\_SYSTEM\_PROCESSOR，参考源码http://androidxref.com/8.1.0\_r33/xref/external/swiftshader/CMakeLists.txt

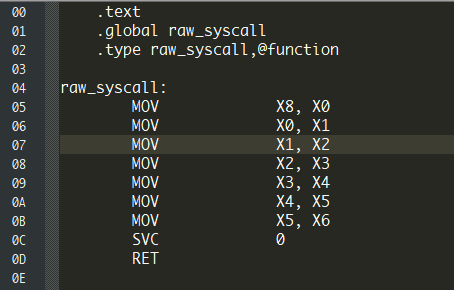




在CMakeLists.txt中加入上面的配置

分别创建syscall32.s和syscall64.s





使用外部 调用来引入

