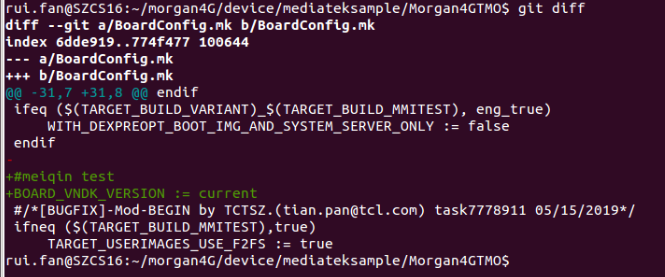
**VNDK编译及HIDL**

1. **启用NDK**

请在BoardConfig.mk添加BOARD\_VNDK\_VERSION定义。目前我们的项目都没有启用VNDK。

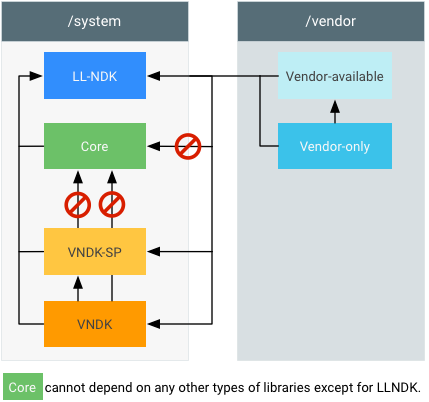


代码全编后在对应的lib目录下生成vndk相关的目录。

C:\Users\Tablet\AppData\Local\Temp\1562640650(1).png

1. **VNDK编译**

系统包含多种类型的对象，其中包括库（共享、静态或标头）和二进制文件：



1. core：位于system.img中，由系统进程使用。vendor、vendor\_available、vndk 或 vndk-sp 库不能使用此类库。

cc\_library {

name: "libThatIsCore",

...

}

1. vendor-only：位于vendor.img中，由vendor 进程使用。

cc\_library {

    name: "libThatIsVendorOnly",

    vendor: true,

      ...

}

1. vendor\_available：位于vendor.img中，由vendor进程使用（从代码上验证被编译到system.img中）。

cc\_library {

name: "libThatIsVendorAvailable",

    vendor\_available: true,

    ...

}

1. vndk：位于system.img中，由vendor进程使用（vendor\_available 的子集）。

cc\_library {

name: "libThatIsVndk",

    vendor\_available: true,

    vndk: {

        enabled: true,

    }

    ...

}

1. vndk-sp：位于system.img中，由system 进程间接使用（core 的子集）。

cc\_library {

name: "libThatIsVndkSp",

    vendor\_available: true,

    vndk: {

        enabled: true,

        support\_system\_process: true,

    }

...

}

1. llndk：同时由system 进程和vendor进程使用。

llndk\_library {

name: "libThasIsLlndk",

}

当一个库被标记为 vendor\_available:true 时，它将编译两次：

一次是为平台编译（因此被安装到 /system/lib 中）。

一次是为供应商编译（因此被安装到 /vendor/lib、/system/lib/vndk 或 /system/lib/vndk-sp 中）。

例如：

cc\_library\_shared {

name: "libtextclassifier\_hash",

……

**vendor\_available: true,**

……

}

输出路径：

./system/lib[64]/

libtextclassifier\_hash.so

cc\_library\_shared {

name: "android.hardware.bluetooth@1.0-impl-mediatek",

defaults: ["mtk\_bt\_hal\_defaults"],

**vendor: true,**

……..

输出路径：

./vendor/lib[64]/hw/android.hardware.bluetooth@1.0-impl-mediatek.so

cc\_library\_shared {

name: "libldacBT\_enc",

**vendor\_available: true,**

**vndk: {**

**enabled: true,**

**},**

……..

输出路径：

./system/lib[64]/libldacBT\_enc.so

./system/lib[64]/vndk-28/libldacBT\_enc.so

|  |  |  |  |
| --- | --- | --- | --- |
| **vendor\_available** | **vndk已启用** | **vndk**  **support\_same\_process** | **供应商变体说明** |
| true | false | false | 供应商变体为 VND-ONLY。共享库将安装到 /vendor/lib[64] 中。 |
| true | 无效（编译错误） |
| true | false | 供应商变体为 VNDK。共享库将安装到 /system/lib[64]/vndk-${VER} 中。 |
| true | 供应商变体为 VNDK-SP。共享库将安装到 /system/lib[64]/vndk-sp-${VER} 中。 |
| false | false | false | 没有供应商变体。此模块为 FWK-ONLY。 |
| true | 无效（编译错误） |
| true | false | 供应商变体为 VNDK-Private。共享库将安装到 /system/lib[64]/vndk-${VER} 中。供应商模块不得直接使用这些变体。 |
| true | 供应商变体为 VNDK-SP-Private。共享库将安装到 /system/lib[64]/vndk-sp-${VER} 中。供应商模块不得直接使用这些变体。 |

1. **定义VNDK扩展库**

在Android9及更高版本中，Android.bp本身支持 VNDK扩展。要编译VNDK扩展，需要定义另一个具有vendor:true 和 extends 属性的模块。

VNDK扩展库包括vendor:true ，vndk.enabled:true和 extends属性

Extends属性： 必须指定原始VNDK共享库的名称（或VNDK-SP共享库名称）。

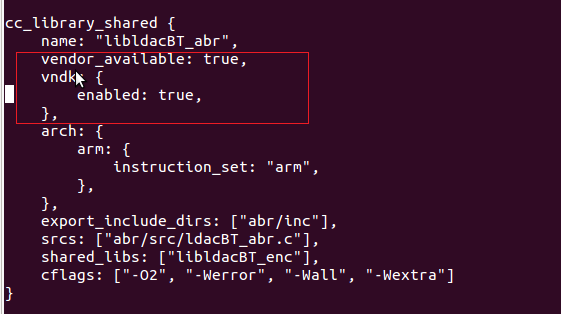
VNDK扩展库的名称是基于原始VNDK(VNDK-SP)共享库名称命名。例如：libldacBT\_abr\_ext的输出文件为libldacBT\_abr.so,而不是libldacBT\_abr\_ext.so。

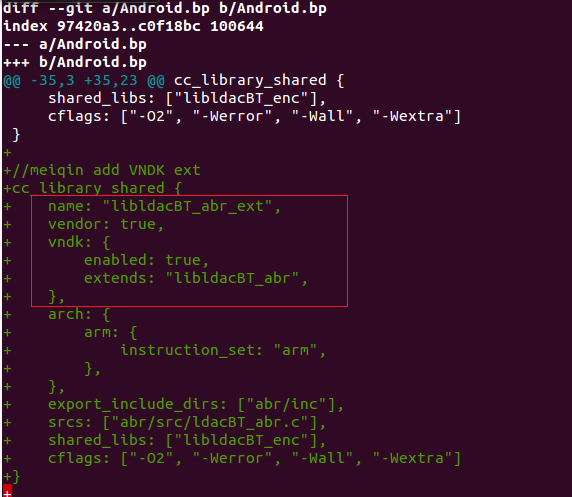
VNDK[sp]扩展编译到/vendor/lib[64]/vndk[-sp]中。

原始共享库必须同时具有vndk.enabled:true 和 vendor\_available:true。

例如：

1. 在原有模块libldacBT\_abr的基础上新增模块libldacBT\_abr\_ext





依赖的其他库

生成的so路径：

./vendor/lib/vndk/libldacBT\_abr.so

./vendor/lib64/vndk/libldacBT\_abr.so

2．在新增模块libldacBT\_abr\_ext添加cflag LIBLDACBT\_EXT\_ENABLE



3.条件编译

新增API

#ifdef LIBLDACBT\_ABR\_EXT\_ENABLE

void new\_add\_api()

#endif

修改API(new和old相比，接口名称一样，内容不一样)

#ifdef LIBLDACBT\_ABR\_EXT\_ENABLE

void new\_api()

#else

void old\_api()

#endif

1. VNDK限制下的解决方案

1.编译错误log：

error: fota\_util (native:platform) should not link to libnvram (native:vendor)

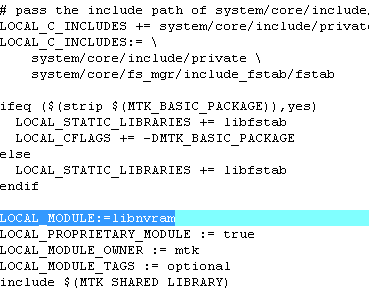
F linker: CANNOT LINK EXECUTABLE "/system/bin/xxx": library "libxxx.so" not found

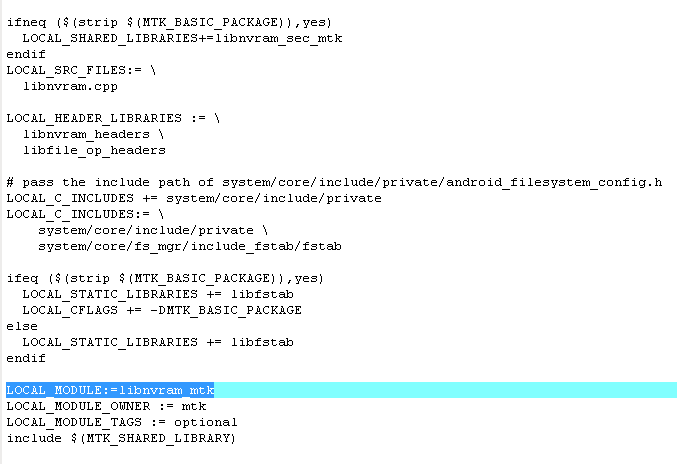
方案1.如果vendor下的资源比如so库，能够从vendor下彻底脱离，也可以拷贝一份到system下，这样合理的避开了这个问题，而且还省掉了开发供应商进程的成本，以及hidl或binder通信的消耗。

例如：MTK提供的libnvram.so分别生成在vendor/lib[64]及system/lib[64]目录下。

相关代码：

vendor/mediatek/proprietary/external/libnvram\_ext/Android.mk





方案2.把vendor下的资源通过供应商进程的方式暴露出来，框架进程通过hidl或者硬件binder的方式与供应商进程通信达到访问vendor资源的目的。

**总结：**在框架进程必须要加载这个库的情况下，如果vendor下的so库能彻底脱离vendor，拷贝一份在system是最佳的方案。

为什么不是直接挪到system下，而是要拷贝一份呢？因为so放在vendor下就是因为vendor要用，而vendor进程只能加载一部分谷歌允许的框架so，所以这个库如果不是谷歌允许的so只放在system下那vendor也加载不到了。这种情况两边都要放。

1. **HIDL简介**

HIDL是一种**（Interface Description Language）**接口定义语言，描述了HAL和它的用户之间的接口。全称是HAL interface definition language(硬件抽象层接口定义语言)。

在AIDL机制中Android 会提供一系列工具会将用户定义的\*.aidl文件编译生成Client端代码与Service端代码，用户仅仅需要1）在Service端实现所需要实现的接口。2）在Client端调用相关接口。

详细介绍请阅读：<https://source.android.com/devices/architecture/hidl/>

包名和代码位置关系：

| 包前缀 | 位置 |
| --- | --- |
| android.hardware.\* | hardware/interfaces/\* |
| android.frameworks.\* | frameworks/hardware/interfaces/\* |
| android.system.\* | system/hardware/interfaces/\* |
| android.hidl.\* | system/libhidl/transport/\* |
| vendor.mediatek.hardware.\* | vendor/mediatek/proprietary/hardware/interfaces/\* |

1. **hidl-gen工具**

系统定义的所有的.hal接口，都是通过hidl-gen工具转换成对应的代码。

使用方法：hidl-gen -o output-path -L language (-r interface-root) fqname

例如：hidl-gen -o hardware/interfaces/hidltest/1.0/default/ -L c++-impl -randroid.hardware:hardware/interfaces -randroid.hidl:system/libhidl/transport android.hardware.hidltest@1.0

参数含义：

**-o** ： 存放hidl-gen产生的中间文件的路径。

**-L**： 语言类型，包括c++, c++-headers, c++-sources, export-header, c++-impl, java, java-constants, vts, makefile, androidbp, androidbp-impl, hash等。hidl-gen可根据传入的语言类型产生不同的文件。

**fqname**： 完全限定名称的输入文件。比如本例中android.hardware.hidltest@1.0，要求在源码目录下必须有hardware/interfaces/ hidltest /1.0/目录。

对于单个文件来说，格式如下：package@version::fileName，比如android.hardware.power@1.0::types.Feature。

对于目录来说。格式如下package@version，比如android.hardware. hidltest @1.0。

**-r**： 格式package:path，可选，对fqname对应的文件来说，用来指定包名和文件所在的目录到Android系统源码根目录的路径。如果没有制定，前缀默认是：android.hardware，目录是Android源码的根目录。

对于实例来说，fqname是：android.hardware.hidltest@1.0，包名是android.hardware，文件所在的目录是hardware/interfaces。例子中的命令会在hardware/interfaces/hidltest/1.0/default/下面产生对应的c++文件。

1. **项目实例**

以morgan4G代码为例，进行hidl实例开发。

1. **准备工作：**

保证全套代码已经全编译，原生代码全编译命令

* source build/envset.sh
* lunch
* make

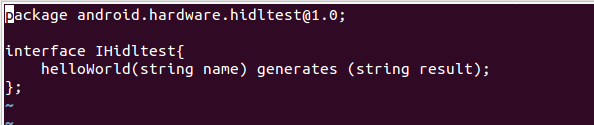
   hidl-gen工具已经安装，安装命令

* make hidl-gen

1. **添加android.hardware.hidltest**

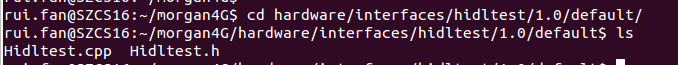
**a.在hardware/interfaces/目录下新建hidltest/1.0目录，并在1.0目录中创建接口IHidltest.hal。目录结构如下：**

C:\Users\Tablet\AppData\Local\Temp\1564033274(1).png

IHidltest.hal文件里面只有一个接口IHidltest和一个方法helloWorld(string name)，具体实现如下：

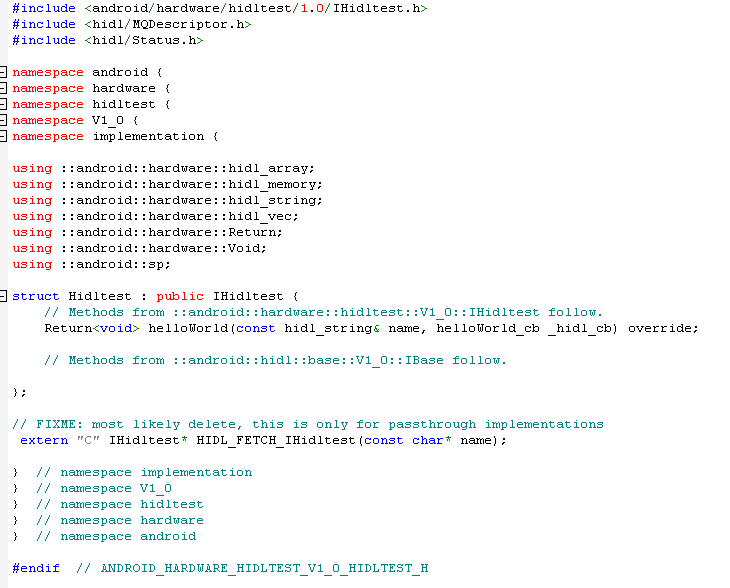
**b.执行下面命令会自动生成对应的c++文件**

rui.fan@SZCS16:~/morgan4G$ hidl-gen -o hardware/interfaces/hidltest/1.0/default/ -Lc++-impl -randroid.hardware:hardware/interfaces -randroid.hidl:system/libhidl/transport android.hardware.hidltest@1.0



default 是新生成的目录，Hidltest.cpp和Hidltest.h是新生成的两个文件。

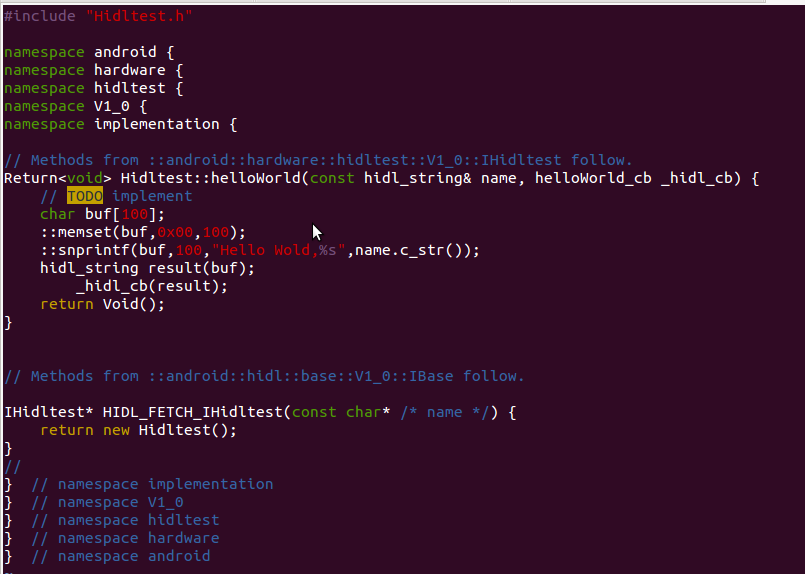
Hidltest.h内容如下：



Hidltest.cpp内容如下：

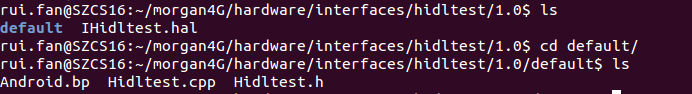


修改Hidltest.cpp，添加简单的字符处理，修改后内容如下：



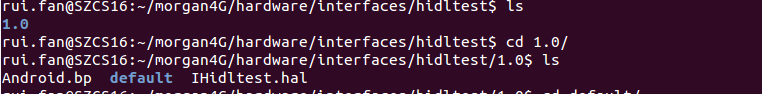
**c.执行下面命令，生成Android.bp文件。**

rui.fan@SZCS16:~/morgan4G$ hidl-gen -o $LOC -Landroidbp-impl -randroid.hardware:hardware/interfaces -randroid.hidl:system/libhidl/transport $PACKAGE



然后使用脚本update-makefiles.sh来更新Makefile，自动在hardware/interfaces/hidltest/1.0目录生成Android.bp。命令如下：

**rui.fan@SZCS16:~/morgan4G$ ./hardware/interfaces/update-makefiles.sh**

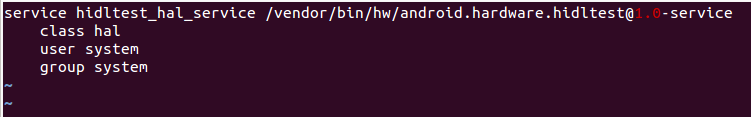
****

**d.在hardware/interfaces/hidltest/1.0/default目录下新建service.cpp跟android.hardware.hidltest@1.0-service.rc，其中android.hardware.hidltest@1.0-service.rc是程序的入口函数。**

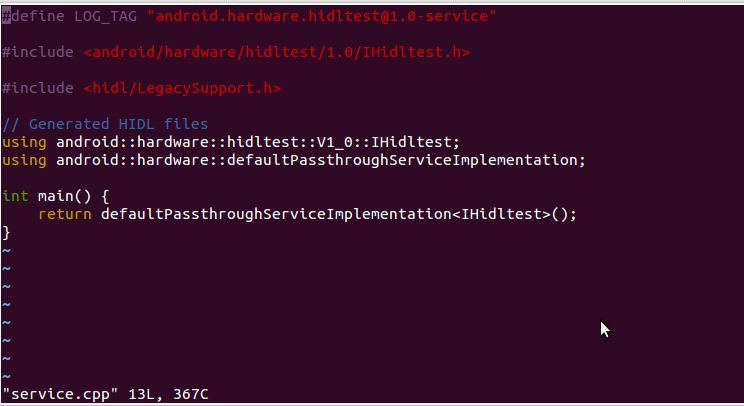
目录结构如下：

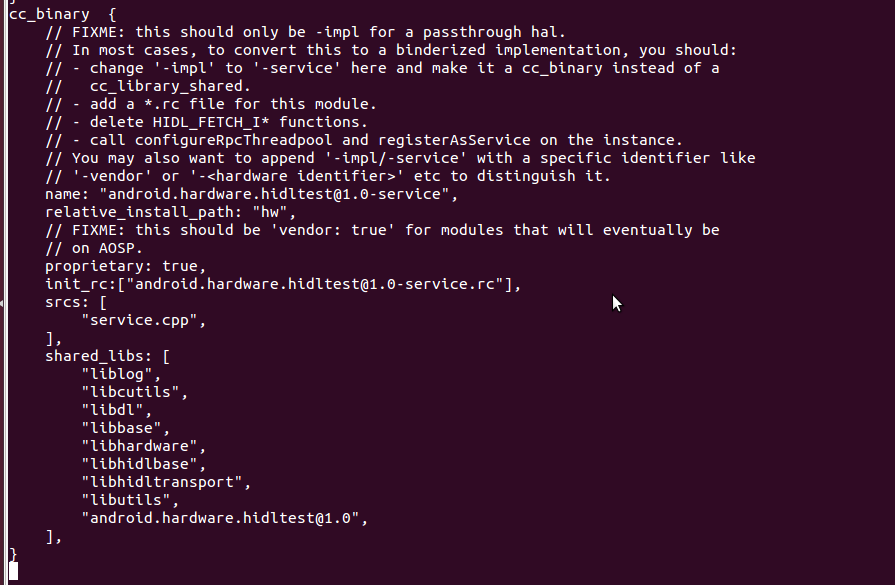
C:\Users\Tablet\AppData\Local\Temp\1564039248(1).png

android.hardware.hidltest@1.0-service.rc内容如下：



Service.cpp内容如下：



打开hardware/interfaces/hidltest/1.0/default目录下的Android.bp，添加编译service.cpp成为可执行文件的代码。具体添加内容如下：

运行以下命令：

rui.fan@SZCS16:~/morgan4G$ ./hardware/interfaces/update-makefiles.sh

rui.fan@SZCS16:~/morgan4G$ mmm hardware/interfaces/hidltest/1.0/

输出结果：

rui.fan@SZCS16:~/morgan4G/out/target/product/Morgan4GTMO/system$ find -name android.hardware.hidltest\*

./lib/vndk-28/android.hardware.hidltest@1.0.so

./lib/android.hardware.hidltest@1.0.so

./lib/android.hardware.hidltest@1.0-adapter-helper.so

./framework/android.hardware.hidltest-V1.0-java.jar

./lib64/vndk-28/android.hardware.hidltest@1.0.so

./lib64/android.hardware.hidltest@1.0.so

./lib64/android.hardware.hidltest@1.0-adapter-helper.so

rui.fan@SZCS16:~/morgan4G/out/target/product/Morgan4GTMO/vendor$ find -name android.hardware.hidltest\*

./etc/init/android.hardware.hidltest@1.0-service.rc

./lib/android.hardware.hidltest@1.0-adapter-helper.so

./lib/hw/android.hardware.hidltest@1.0-impl.so

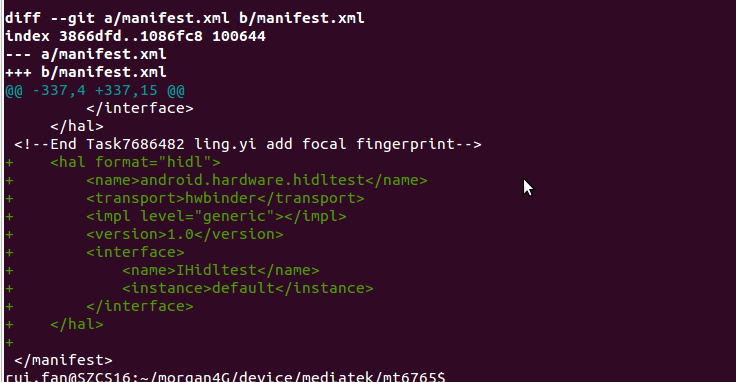
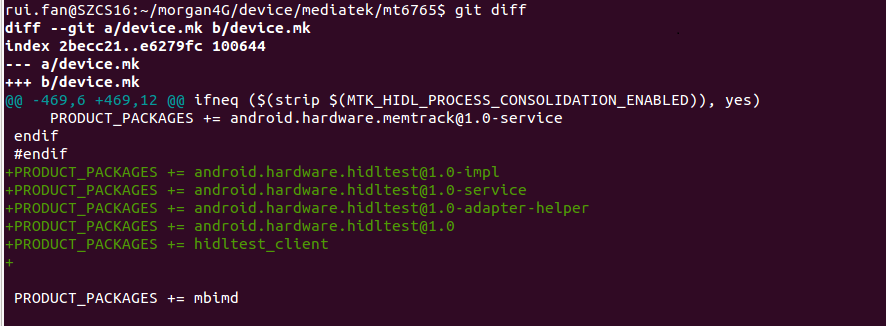
./lib/hw/android.hardware.hidltest@1.0-service.so

./lib64/android.hardware.hidltest@1.0-adapter-helper.so

./lib64/hw/android.hardware.hidltest@1.0-impl.so

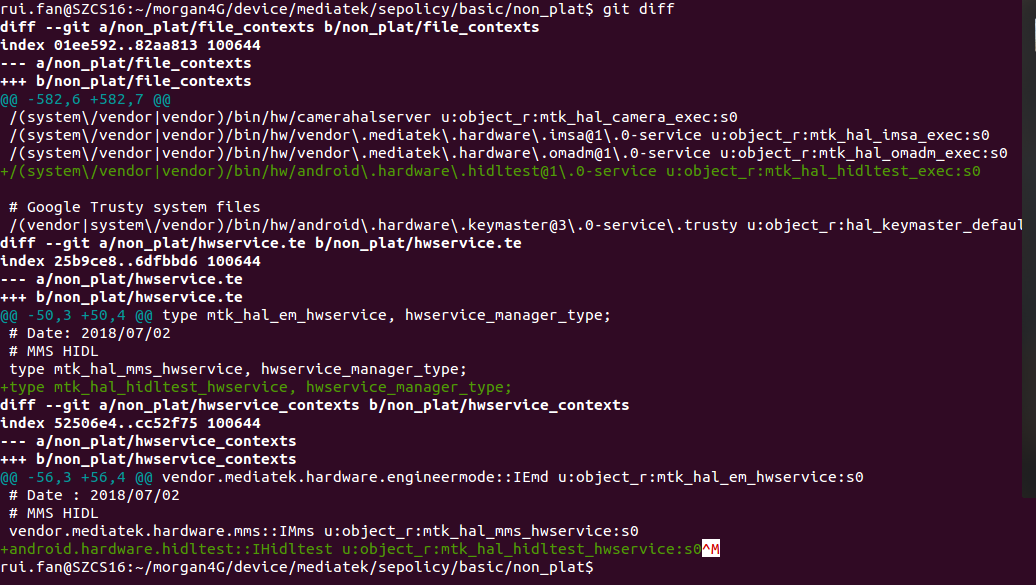
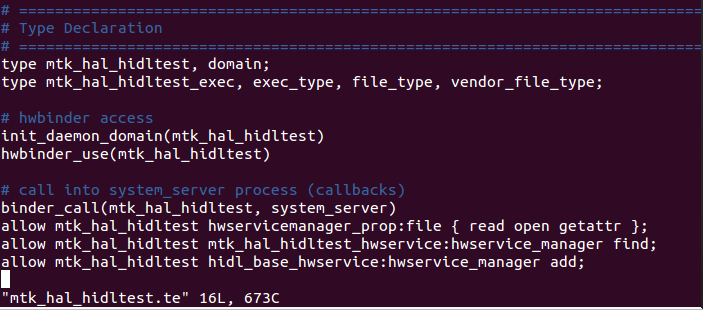
./lib64/hw/android.hardware.hidltest@1.0-service.so

**e.在morgan4G/device/mediatek/mt6765/目录下的manifest文件添加对应的内容，不然在client端是没法拿到service的。如下：**



**f.添加selinux权限**

添加mtk\_hal\_hidltest.te文件，内容如下：



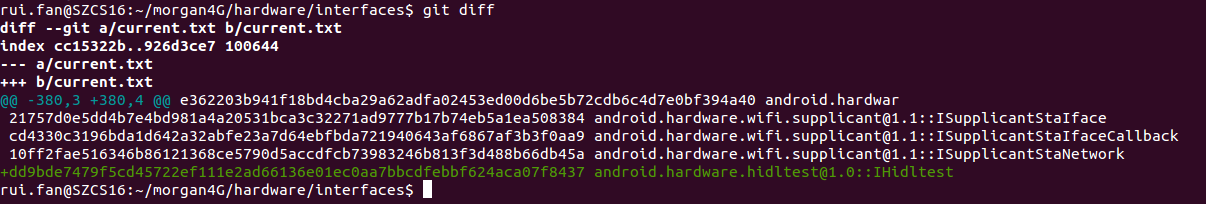
**g.生成hash**

rui.fan@SZCS16:~/morgan4G$ PACKAGE=android.hardware.hidltest@1.0

rui.fan@SZCS16:~/morgan4G$ LOC=hardware/interfaces/hidltest/1.0/default/

rui.fan@SZCS16:~/morgan4G$ hidl-gen -L hash -r android.hardware:hardware/interfaces -r android.hidl:system/libhidl/transport $PACKAGE >> hardware/interfaces/current.txt

输出结果：



**h.使用C++实现客户端调用**

在hardware/interfaces/hidltest/1.0目录下新建test目录，并且在test目录下新建Android.bp跟HidlTest.cpp文件，这两个文件的内容如下：

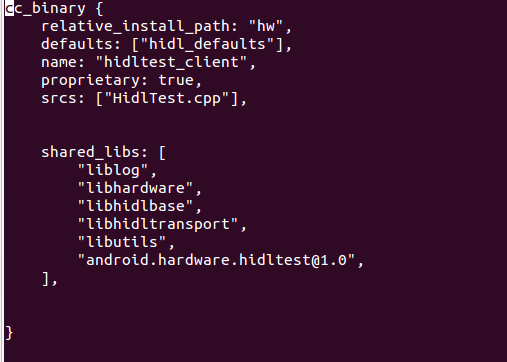
HidlTest.cpp：



**调用HAL层定义的接口**

**获取服务**

Android.bp:



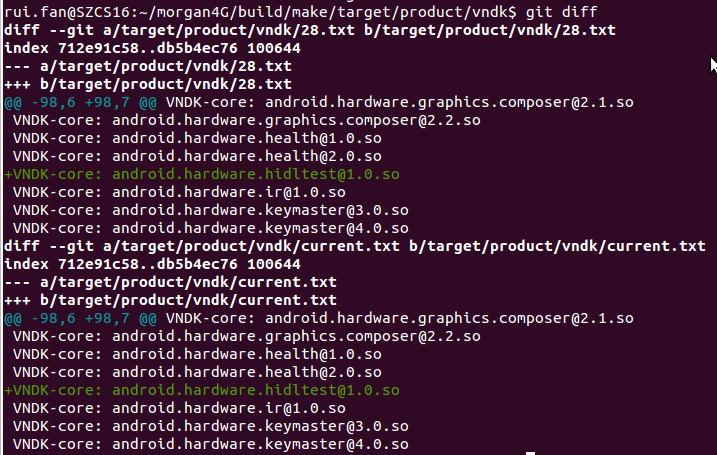
注：打开VNDK后，全编可能报错：

Added VNDK-core: android.hardware.hidltest@1.0.so

error: VNDK library list has been changed.

Changing the VNDK library list is not allowed in API locked branches.

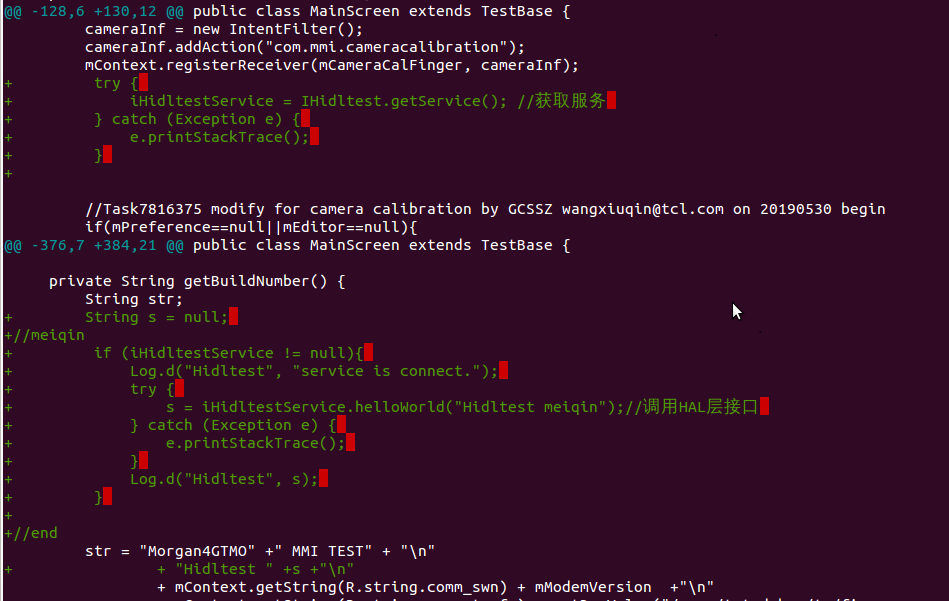
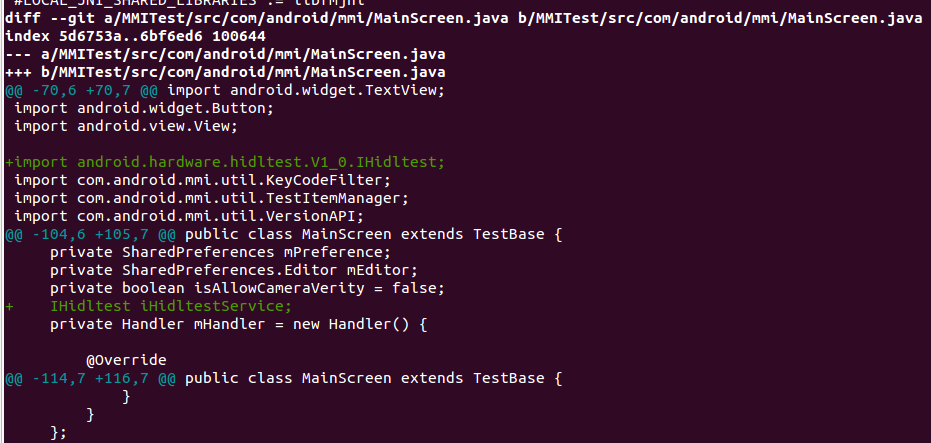
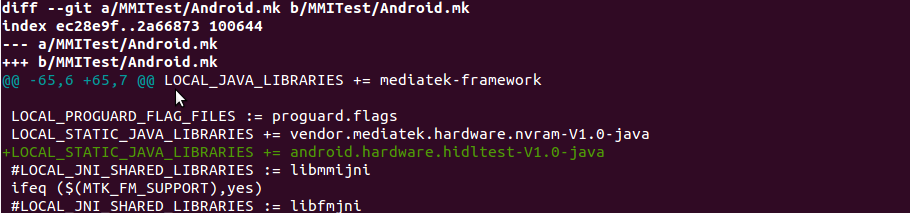
build/make/target/product/vndk/目录下，在28.txt和current.txt中加入VNDK-core android.hardware.hidltest@1.0.so，严格按照字母顺序加。（需要确认是否能过CTS,因为我们添加的HIDL接口都有vendor下面， 所以不会存在这种情况。）



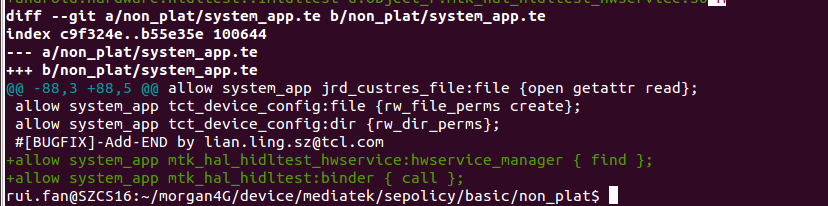
C++客户端调试：

先启动./android.hardware. hidltest @1.0-service服务，然后启动hidltest \_client程序，执行结果如下：C:\Users\Tablet\AppData\Local\Temp\1564451478(1).pngC:\Users\Tablet\AppData\Local\Temp\1564451438(1).png

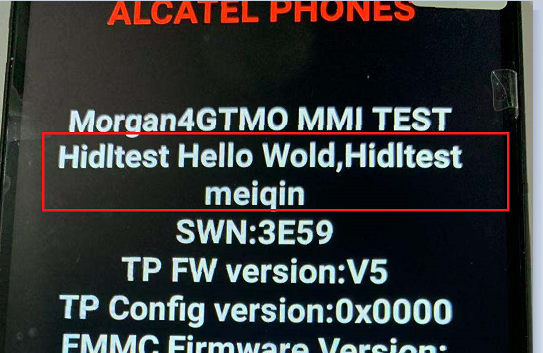
**i.使用JAVA实现客户端调用**



添加selinux权限：

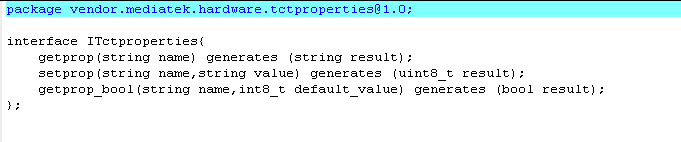


调试结果：



1. **添加vendor.mediatek.hardware.tctproperties，实现vendor/system相互访问property。**

**a.在vendor/mediatek/proprietary/hardware/interfaces下新建tctproperties目录，在tctproperties目录下新建1.0目录。在1.0目录下新建ITctproperties.hal文件，内容如下：**

****

**调用相关的命令生成Android.bp, Tctproperties.cpp及Tctproperties.h文件。**

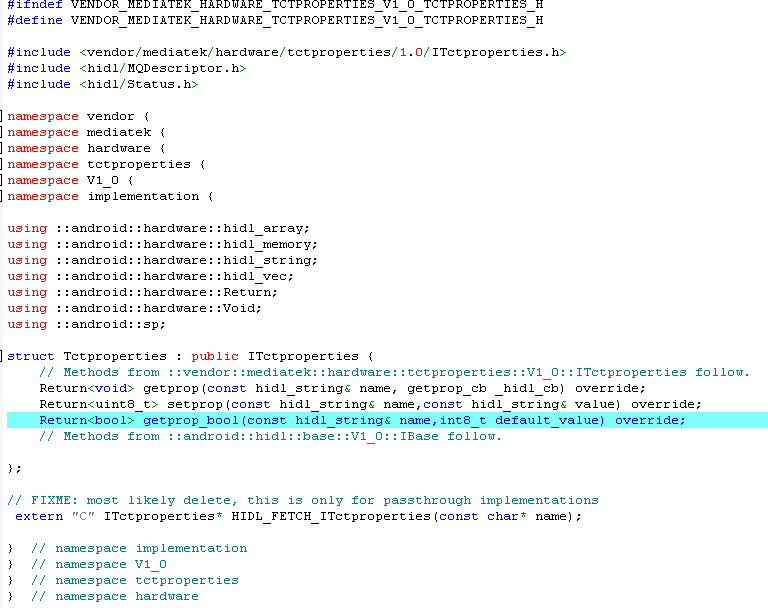
$ hidl-gen -o vendor/mediatek/proprietary/hardware/interfaces/tctproperties/1.0/default/ -Lc++-impl -rvendor.mediatek.hardware:vendor/mediatek/proprietary/hardware/interfaces -randroid.hidl:system/libhidl/transport [vendor.mediatek.hardware.tctproperties@1.0](mailto:vendor.mediatek.hardware.tctgetprop@1.0)

$ hidl-gen -o vendor/mediatek/proprietary/hardware/interfaces/tctproperties/1.0/default/ -Landroidbp-impl -rvendor.mediatek.hardware:vendor/mediatek/proprietary/hardware/interfaces -randroid.hidl:system/libhidl/transport [vendor.mediatek.hardware. tctproperties @1.0](mailto:vendor.mediatek.hardware.tctgetprop@1.0)

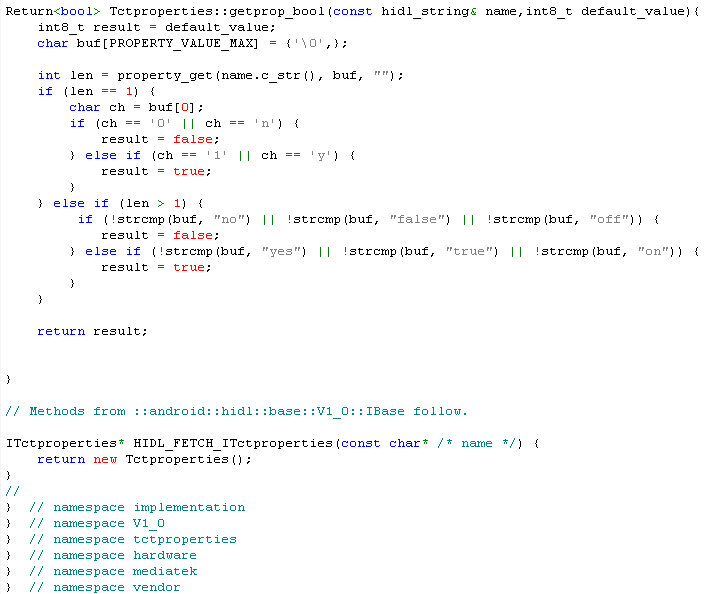
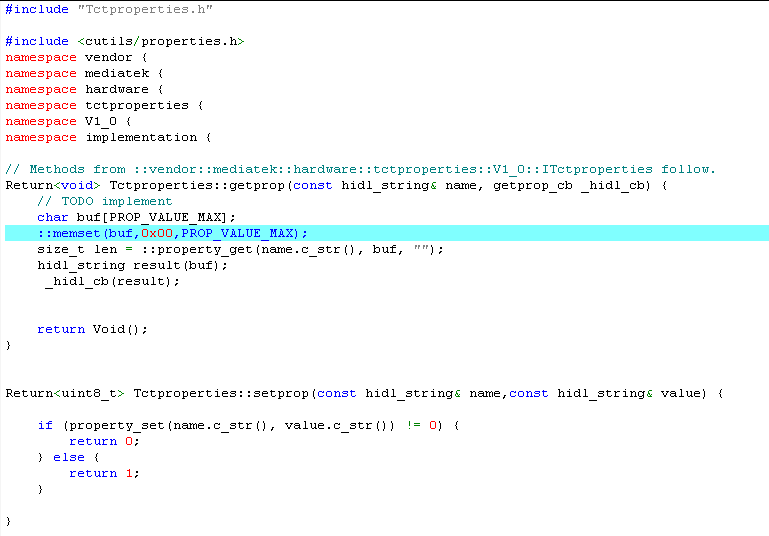
$./vendor/mediatek/proprietary/hardware/interfaces/update-makefiles.sh

**b.实现相关接口。**

**Tctproperties.h**

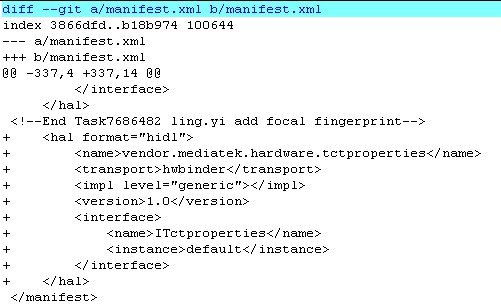
****

**Tctproperties.cpp：**



**c.在vendor/mediatek/proprietary/hardware/interfaces/tctproperties/1.0/default/目录下新建service.cpp跟vendor.mediatek.hardware.tctproperties@1.0-service.rc，其中vendor.mediatek.hardware.tctproperties@1.0-service.rc是程序的入口函数。修改vendor/mediatek/proprietary/hardware/interfaces/tctproperties/1.0/default/下的Android.bp文件**

**d. 修改device/mediatek/mt6765 /manifest.xml,及添加selinux相关权限**

****

**e.生成hash**

hidl-gen -o vendor/mediatek/proprietary/hardware/interfaces/tctproperties/1.0/default/ -Lhash -rvendor.mediatek.hardware:vendor/mediatek/proprietary/hardware/interfaces -randroid.hidl:system/libhidl/transport vendor.mediatek.hardware.tctproperties@1.0 >> vendor/mediatek/proprietary/hardware/interfaces/current.txt

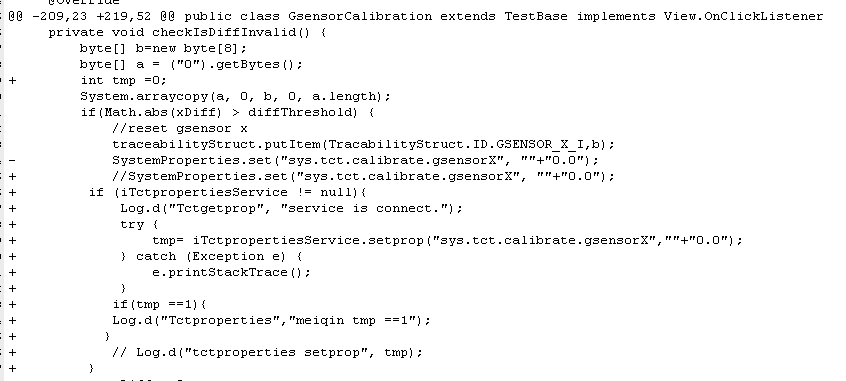
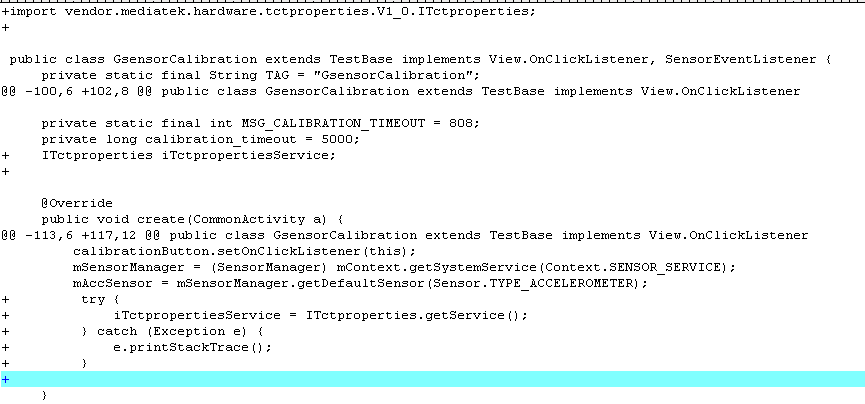
**注：修改了hidl接口都需要更改hash值，否则会报类似以下的错误信息。**

ERROR: vendor/mediatek/proprietary/hardware/interfaces/current.txt differs from expected: ; cat out/target/product/Morgan4GTMO/obj/FAKE/check-mtk-hidl\_intermediates/diff.txt; false) ) && (touch out/target/product/Morgan4GTMO/obj/FAKE/check-mtk-hidl\_intermediates/diff.txt )"

ERROR: vendor/mediatek/proprietary/hardware/interfaces/current.txt differs from expected:

> vendor.mediatek.hardware.tctproperties@1.0::ITctproperties

**f.MMITest中调用相关接口：**



**g. vendor/mediatek/proprietary/hardware/sensor/sensors-1.0中调用getprop**

