Functional Validation of Inter Processor Communication through Dynamic Analysis

Euteum Jo, Pyeongsoo Mah
Department of Computer Software, University of Science & Technology, Daejeon, Republic of Korea
Department of Embedded SW Research, ETRI, Daejeon, Republic of Korea

Summary
The major role of inter processor communication(IPC) driver is to transfer data between an application processor and a communication processor. In order to sustain a stabilized performance of the upper framework of radio interface layer and telephony application, the driver should be thoroughly examined. In traditional method, however, the driver software have been developed without any standard validation tools. As a result, developing an IPC driver was a time consuming process and it was difficult to obtain reliable validation result from the test. In this paper, we present a tool, called DLCTest, that can be used for the dynamic analysis of IPC driver software. The tool can enhance both the observability and the coverage of IPC driver testing. The tool is actually used during the development process of commercial smartphones structured with dual processors. The proposed tool enables an early validation of the IPC driver through easier and quicker detection of software defects.

Key words:
Radio Interface Layer, Inter Processor Communication, Smartphone Development, Device Driver Validation

1. Introduction

In dual-processor architectures for mobile devices, inter processor communication(IPC) takes a role of exchanging data between two processors. These two processors are usually the combination of an application processor(AP) and a communication processor(CP) in smartphones. In order to ensure the normal operation of the applications that is required for telephony functions, the IPC function should be stabilized in advance and validated thoroughly from the perspective of radio interface layer(RIL)[1] developers.

Since IPC driver works on kernel domain, system panic can occur when there are some defects in the IPC driver codes. Therefore, testing engineers have to understand the IPC driver codes completely and know various scenarios well.

By now, most of IPC driver testing methods have been performed manually by running telephony applications. With such a manual method, however, long-time repetitive tests are unavoidable and the possibility of detecting software defect is relatively low. In addition, performance related testing becomes very difficult in case the software is updated by adding several patch software. In that case, the response speed can be slower in tens of millisecond unit, even though the function of the updated software is correct. However, the slowed response speed is hard to detect without a testing tool.

Some testing tools are already used in other domain. Windbg[2] is used for kernel mode driver validation at Microsoft Windows and Memset86[3] is in use for memory failure detection. Iperf[4] which is a network testing tool and LDV[5] for Linux Driver are another examples that used automated tools in testing. However, to the best of my knowledge, a testing tool for IPC driver has never been published.

IPC validation tool should be independent from radio interface layer(RIL) framework as IPC device driver belongs to the lower level of RIL. To demonstrate the correctness of IPC function independently, communication processor(CP) and ATcommand should be transmitted and received without the RIL framework.

We present a tool, called DLCTest(Data Link Connection Test), that can be used for enhancing both the observability and the coverage of IPC driver testing. The proposed tool is designed to work on hardware abstraction layer(HAL) for Android smartphones. It contains ATcommand module for Vendor RIL not only for enabling IPC feature but also for checking whether communication processor(CP) is in a normal operating state or not. The tool we implemented is applied to the development process for smartphones with two processors. The proposed tool shows shortened validation time and improved driver performance.

2. Background

2.1 Two types of IPC architecture

There are two types of IPC architecture; direct communication scheme having interface bridge and indirect communication scheme using shared memory. SOCs that integrate an application processor(AP) and a communication processor(CP) into one chip use a shared memory mode and indirect communication schemes. The architecture shows better stability(debugging becomes much easier and outbreak of unidentified errors were less frequent) and high throughput.

Manuscript received November 5, 2014
Manuscript revised November 20, 2014

Nonetheless, direct communication scheme with interface bridge mode is often preferred to reduce the cost of mass production. This scheme has an advantage that various kinds of application processor(AP) and communication processor(CP) from different vendors can be combined.

Hardware designer may establish interface bridge mode with secure digital input output(SDIO)[6], serial peripheral interface(SPI)[7], mobile industry processor interface(MIPI)[8], or high speed inter-chip(HSIC)[9]. The interface bridge mode is determined according to minimum bandwidth requirement. CDMA smartphones usually use SPI for the interface bridge, whereas WCDMA smartphones use MIPI and LTE smartphones use HSIC.

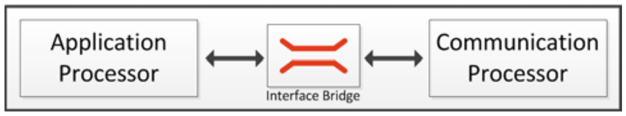

Figure 1. Direct communication scheme(Interface bridge)

2.2 Logical channels for various applications

The physical interface between application processor(AP) and communication processor(CP) provides only one channel in general. Therefore, a separate driver which logically establishes channels to the upper layer of physical interface is required for the provision of multiple channels for diverse user space applications like Phone Call, VT, SMS, Ethernet. Radio interface layer(RIL) incorporates this multiplexer driver into the range of IPC layer. The most common logical interfaces are 3GPP TS 27.010[10] Multiplexer and HSI Logical Channels. According to TS 27.010 standard, Linux kernel provides 32 TTY interfaces to the user space applications to communicate with communication processor(CP). Those interfaces correspond to 32 different TS 27.010 DLCs(Data Link Connections). The 32 TTY interfaces are controlled by Vendor RIL on user space. (Figure 2)

3. Necessity of an IPC validation tool in each development stage

Development process of IPC function in mobile devices can be classified into three stages; porting, patch and debugging, and validation.

Porting : A tool for IPC driver supports the validation of the normal operations for IPC device driver by controlling TTY device interface on user space in the absence of Vendor RIL. Without such a tool, it is still possible to test the IPC driver by Vendor RIL having minimum operational functions. However, the repetitive transmission of a certain ATcommand is difficult because the execution flow occurs only one time after booting. Without a tool, repetitive rebootings are unavoidable. To meet the time to market requirement, IPC function should be stabilized within a short period of time because Vendor RIL, RIL framework, and user applications can be tested after the test of IPC functions is completed.

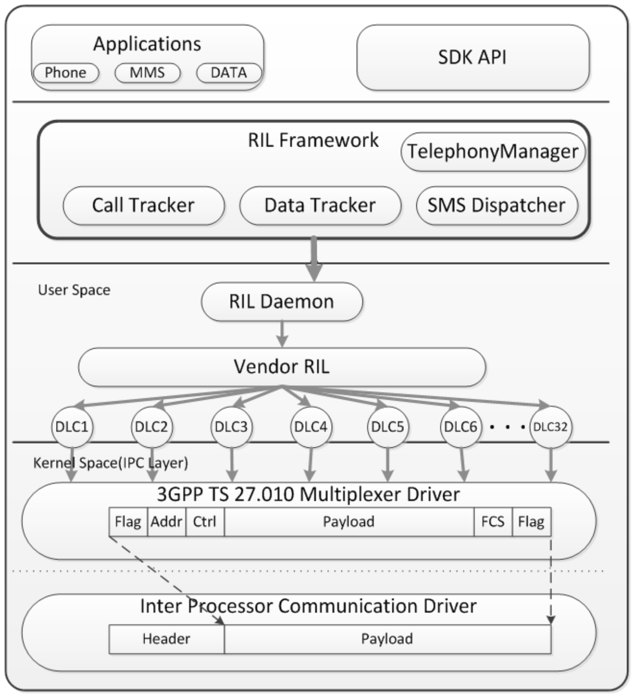

Figure 2. Radio Interface Layer(RIL) Architecture

Patch and Debugging : In traditional manual debugging method that uses Vendor RIL, only one time test is possible and repetitive rebooting is unavoidable. For example, if a new handler is added at a communication processor(CP) side for a new ATcommand, the modification of Vendor RIL at an application processor(AP) side is inevitable to test the new handler. As a result, rebuilding of AP images and reinstallation process should be followed. This process requires quite a bit of time.

Validation : Validation of IPC functions includes both examining its normal operation and checking stability of a communication processor(CP). In traditional method, user applications are used to validate IPC functions. The weakness of such method is that the test coverage is too broad and can only be done when the test of IPC and all the radio interface layer(RIL) related components are finished. From the perspective of IPC developer, the test range needs to be adjusted and narrowed down only to the IPC layer. The applicability of traditional test method that uses user applications is limited. If there are any defects indeed, debugging is hardly achievable because the reproducing ratio of the same error is very low.

A certain tool that can validate IPC function is needed when issues are related to radio interface layer(RIL). The developers of upper layer desire to validate IPC functions first when there are issues related to their applications because IPC is positioned on the lowest level of radio interface layer(RIL). Traditionally, the source code of Vendor RIL has to be rebuild and reinstalled again. It takes at least 30 minutes to do so even in the case that incremental build method is used.

4. Conceptual approach

The basic principle of DLCTest is transmitting and receiving data continuously via IPC interface. Empirically, the durability of data transmission is very important when it comes to IPC function test. Without a proper tool, it is almost impossible to detect all the errors in the IPC driver software. Figure 3 shows that DMA timeout errors can be detected only at a specific time.

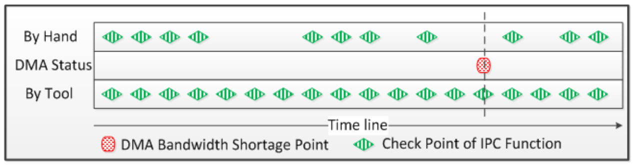

Figure 3. Possibility of DMA related error detection(By hand vs By tool)

Validation of IPC function is activated within physical interface and logical interface. In this study, logical interface will be explained based on 3GPP TS 27.010 standard and physical interface will be based on Android mobile smartphones having dual processors, which are composed of serial peripheral interface(SPI).

Requirements reflected for the development of the tool are as follows:

1. IPC should be controlled independently from Vendor RIL.
2. Must be implemented to execute on HAL.
3. Independent execution is necessary without connecting to PC host.
4. Random ATcommand transmission and validation of normality of its response should be possible.
5. Average response time for ATcommand should be measurable.
6. Log function is essential. The time and aspect should be recorded when a certain event or abnormality occurs.

5. Implementation details of the proposed tool

We implemented a dynamic analysis tool for IPC driver. It has a high portability and the dependency to Android platform is low because it is implemented with C and POSIX library. The structure of the implemented tool is shown in Figure 4.

The tool is divided into a server module and a client module. The server module is responsible for testing on device and the client module is used for real-time monitoring of testing status. The purpose of placing the client module separately is to obtain an easier detection of any related defects because, if USB cable is attached, it does not enter to sleep mode in case of aging testing. In the case of any other test except from aging test, the server module provides the developer the user console. The role of each module is summarized as follows.

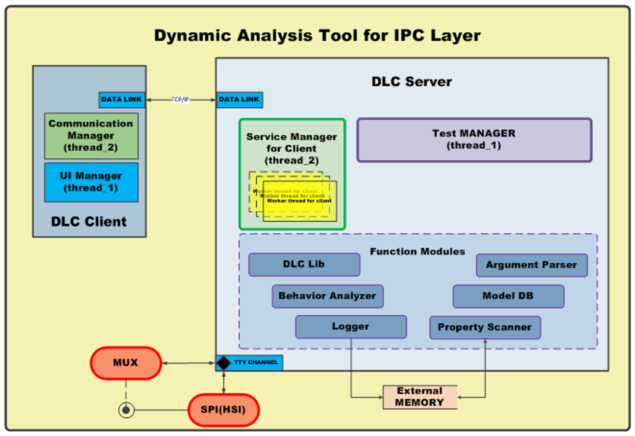

Figure 4. DLCTest Block Diagram

Test Manager : Thread which responsible for testing IPC function according test scenarios.
DLC Lib : Utility functions are implemented.
Behavior Analyzer : Predict the root cause of problems by analyzing resource states in system and kernel log with the patterns of problems, when abnormal symptoms are found during the test.
Logger : Collect and record all necessary information for the analysis of test result.
Model DB : Retaining hardware information for each model and parameters that is needed for the test appointed in prior accordingly.
Property Scanner : Parser is implemented to detect specific name of model from property files.

6. Use cases of the proposed tool

DLCTest provides an automated aging test. With this tool, we are able to detect abnormal operation of IPC layer. It can also perform a radio interface layer(RIL) scenario instead of a RIL framework before RIL framework is ready. With user console interface, the developer can generate any RIL commands and obtain responses from communication processor(CP) at any time without modifying the RIL framework. This means that the time for rebuilding and reinstalling is no longer necessary for testing purpose.

6.1 Major defects observed by DLCTest

6.1.1 Memory leak in TS 27.010 multiplexer driver
After 3 days of aging test, a bug on system memory leakage was found. It was very small size of memory leak that is very difficult to detect using the traditional manual method.

6.1.2 Packet stuck by binary semaphore in multiplexer driver
Lock related errors were detected and it is also very difficult to detect using the traditional manual method.

6.1.3 Delayed response time between multiplexer and SPI driver
Side effect can easily be detected after code patch. The tool allows to check digitalized figures.

6.1.4 Identifying reproduction scenario of specific issues based on log pattern analysis
In some cases, it is difficult to find out the reproduction scenario of some test cases. For example, when the communication processor(CP) is rebooted unexpectedly, there are only application processor(AP) logs. By using the proposed tool, it was possible to analyze the exchanged data between CP and AP and the unexpected error situation was reproduced.

We also found that the values of response parameters from CP in some erroneous situations are different from those of normal conditions. Such patterns that indicate the possibility of CP crash are utilized by the tester to correct hidden errors in the codes.

6.1.5 DMA timeout in SPI device driver
A quick check on abnormality that is related to DMA and power management is possible using the developed tool. The DMA related errors are almost impossible to reproduce through the traditional manual method.

6.2 Reduced time for problem solving

Due to independent control of IPC driver from radio interface layer(RIL) framework, the time consumed for inserting test codes to the framework and the time for rebuilding the modified IPC driver can be saved. Furthermore, it can be said that the overall time for solving many issues is reduced in consequence of a quicker diagnose of defects, compared to the traditional manual method.

6.3 Reduced Time for IPC driver porting

Development time can be reduced significantly, since the errors in the codes are corrected at an early stage of development with the use of proposed tool.

6. Conclusion

In this paper, we proposed a tool for IPC function validation that overcomes the limits of non-automatic testing method. By using the proposed tool, the development and validation time for IPC driver is shortened and the quality of the developed software product is improved significantly. More importantly, the effectiveness of the proposed tool is proved through the development of mass production model of commercial smartphones.

References

[1] Radio Interface Layer. (2014, May). Wikipedia. Retrieved November 20, 2014 from http://en.wikipedia.org/wiki/Radio_Interface_Layer
[2] Testing Device Drivers on Windows Platforms. (2012, April 24). Microsoft Developer Network. Retrieved November 20, 2014 from http://msdn.microsoft.com/en-us/library/dd873575.aspx
[3] An Advanced Memory Diagnostic Tool. (2013, September 27). Memset86 Official. Retrieved November 20, 2014 from http://www.memtest.org
[4] Iperf User Docs. (2014). Iperf forum. Retrieved November 20, 2014 from http://iperf.frs
[5] Linux kernel Space Verification. (2014). Verification Center of the Operating System Linux. Retrieved November 20, 2014 from http://linuxtesting.org
[6] SDIO Simplified Specification. (2011, February). SD Association. Retrieved November 20, 2014 from http://www.sdcard.org/downloads/pls/simplified_specs/archive/partE1_300.pdf
[7] Serial Peripheral Interface Bus. (2014, November). Wikipedia. Retrieved November 20, 2014 from http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus
[8] High-speed Syncronous Serial Interface(HSI) Specification. (2014, May). Mipi alliance. Retrieved November 20, 2014 from http://mipi.org/specifications/high-speed-syncronous-serial-interface-hsi
[9] High-Speed Inter-Chip. (2012, May). Universal Serial Bus. Retrieved November 20, 2014 from http://usb.org/developers/docs/usb20_docs
[10] Terminal Equipment to User Equipment (TE-UE) multiplexer protocol. (2014, September 17). The Mobile Broadband Standard. Retrieved November 20, 2014 from http://www.3gpp.org/DynaReport/27010.htm