International Journal of Latest Trends in Engineering and Technology
Vol.(8)Issue(3), pp.222-225
DOI: http://dx.doi.org/10.21172/1.83.033
e-ISSN:2278-621X

SERIAL COMMUNICATION AS AN ALTERNATIVE MECHANISM FOR ADB IN TEST AUTOMATION OF ANDROID/LINUX DUT

Karthik B S[1] and Dr. S Gayathri[2]

Abstract- Automated test execution requires execution of commands at the remote device under test and also may require transfer of files for the successful execution of tests. Usually any test framework uses ADB interface to implement these tests for an Android DUT. ADB is a third party tool and hence has a dependency on Android. This paper intends to propose a mechanism to remove this dependency by carrying out successful execution of automated tests by transfer of the required commands and files using a serial interface which is present in all the DUT's. This makes the tests more robust and the same tests can be used in a Linux only environment as well.

Keywords – Android Debug Bridge(ADB), Device Under Test(DUT)

I. INTRODUCTION

The Android Debug Bridge (ADB) is a toolkit included in the Android SDK package. It consists of both client and server-side programs that communicate with one another. The ADB is typically accessed through the command-line interface, although numerous graphical user interfaces exist to control ADB. It facilitates a variety of device actions, such as installing and debugging apps, and it provides access a Unix shell that can be used to run a variety of commands on the connected device. The existing test framework uses ADB for the transmission of commands and files to the DUT.

The DUT's in this case may be Android or Linux based systems. Since the test mechanism totally depends on ADB functionality the scope of the test framework becomes very limited. This paper intends to propose a mechanism in which dependency on ADB can be removed in the test framework such that the test mechanism works seamlessly both in the presence and the absence of ADB support.

The proposed mechanism intends to achieve this independency by using the serial communication interface which is available both for Linux and Android systems and thus can be used as an alternative mechanism for the test framework whenever ADB cannot be used either due to its unavailability (in the case of Linux only systems) or due to some issue in using the feature.

The basic idea is to use serial port communication for test automation as an alternative when ADB is unavailable. There are many serial file transfer protocols available such as Kermit[3], which actually achieves the command/file transfer seamlessly as required but they have a requirement that the destination machine needs to be running as client and then the host machine runs as a server and achieves the command/file transfer[1]. The requirement is to have an end to end automated process which can be used on any DUT without any dependency on it.

The proposed mechanism intends to achieve command/file transfer through the serial interface which is neither dependent on the ADB nor have any dependencies on the destination machine. The user is free to connect an

[1] *M.Tech, Industrial Electronics Sri Jayachamarajendra College of Engineering, Mysuru, Karnataka, India*
[2] *Department of Electronics and Communication Engineering Sri Jayachamarajendra College of Engineering, Mysuru, Karnataka, India*

Android/Linux DUT of which baud rate or the port on which it is connected to the host are unknown and still the proposed mechanism must work in a totally automated manner.

II. Objective

To develop a mechanism to be used as an alternative to ADB, which can be used for test automation of Linux/Android DUT's. The developed mechanism needs to work in an automated manner and be independent of ADB as well as the DUT.

III. Implementation

The aim is to have an end to end automated process. The steps to achieve this are the following.

A. Port and Baud Rate Identification

The first step in achieving this is to detect the port to which the DUT is connected and the baud rate at which it is operating (Figure 1). This is achieved by the following algorithm.

1. The DUT connected to the serial port in Linux gets listed under /dev/ttyUSB*. All the active USB ports are listed in a file.
2. Have a look up table of all the possible baud rates.
3. Select an active USB port from the file and choose a baud rate from the LUT.
4. Send a test word and receive back the word in the same USB port previously selected.
5. If the received word matches the sent word, the identified USB port and the baud rate are populated to a file and goto step 9.
6. If all the possible baud rates for a given port are not tested, choose another baud rate from the LUT and goto step 4.
7. If all the active USB port are not tested, choose the next active USB port from the file and goto step 4.
8. Error in Finding the port.
9. Stop.

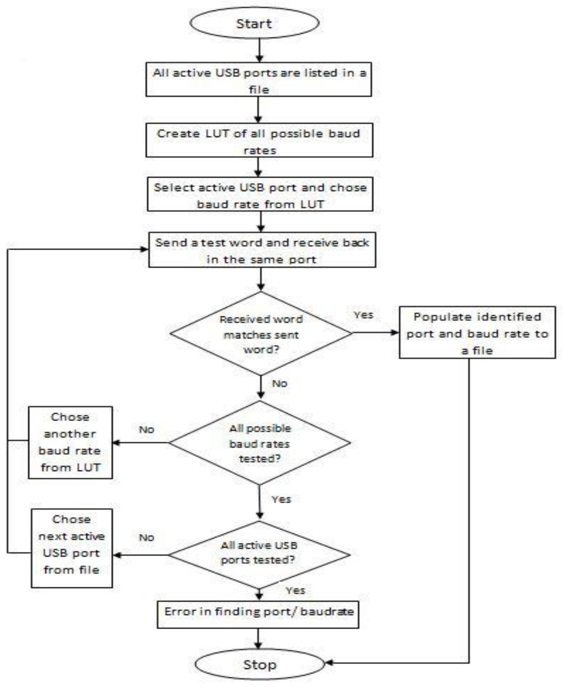
Start
All active USB ports are listed in a file
Create LUT of all possible baud rates
Select active USB port and chose baud rate from LUT
Send a test word and receive back in the same port
Received word matches sent word?
Yes
Populate identified port and baud rate to a file
No
All possible baud rates tested?
No
Chose another baud rate from LUT
Yes
All active USB ports tested?
No
Chose next active USB port from file
Yes
Error in finding port/ baudrate
Stop

Figure 1. Port and baud rate identification algorithm

B. Command Transmission

Once the port and baud rate has been identified the next step is to configure the port with all the required settings. The identified port is opened with the required settings and baud rate using the 'stty' command. The link for communication between the linux host and a DUT is set up with above mentioned steps.

The command transmission to the DUT can be achieved using ‘termios’. ‘termios’ is the Unix API for terminal I/O. The necessary declarations and constants for termios can be found in the header file <termios.h>. A simple code can achieve this. It needs the port name, baud rate and command to be transmitted to the DUT as input to the code.

C. *Acknowledgement*

The next objective is to have an acknowledgement mechanism to verify the successful execution of the command at the DUT and have a retransmission mechanism in place on detecting failure[2]. This is achieved by the following algorithm (Figure 2).

1. The command to be executed at the DUT is transmitted.
2. Any command executed in the shell of the DUT can be viewed by using the‘cat /dev/ttyUSB*’ command.
3. The output is put into a file and a parsing logic is used to verify the command executed at the DUT.
4. If the command executed matches the command transmitted then goto step 7.
5. Else check if maximum number of retransmissions has been reached. If yes then indicate failure of the mechanism and goto step 7.
6. Else goto step 1.
7. Stop.

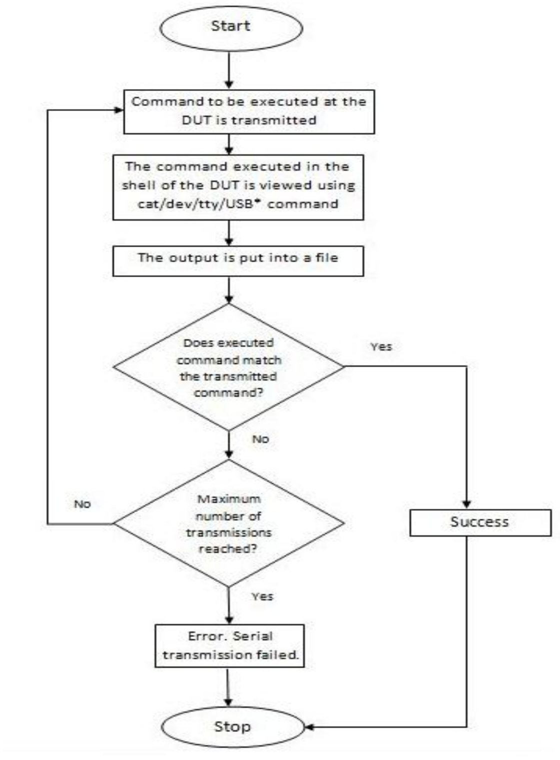
Start
Command to be executed at the DUT is transmitted
The command executed in the shell of the DUT is viewed using cat/dev/tty/USB* command
The output is put into a file
Does executed command match the transmitted command?
Yes
No
Maximum number of transmissions reached?
No
Yes
Success
Error. Serial transmission failed.
Stop

Figure 2. Transmission and Acknowledgement algorithm

D. *Retrieving the Results*

The next objective will be to have the result of the command executed at the DUT. For example an ‘ls’ command executed at the DUT lists all the files/folders present in the current folder. This information will have to be retrieved and brought back to the host machine. This is also achieved using a similar logic to the previous step i.e, using the ‘cat /dev/ttyUSB*’ command. Once the command has been successfully executed the following steps are used.

1. The output of the cat /dev/ttyUSB* is put into a log file.
2. The logging has to be active as long as the output of the executed command is coming in the shell. This logic is put in place by looking out for the shell name. Assume the shell name is ‘my_shell#’. So we constantly look out for ‘my_shell#’ in the log file. Once it is found the logging is stopped.
3. In the log file the first line would be the command executed and the last line would be the shell name. These two lines are removed out.

4. The remaining lines in between these two lines is the command output which is required.

IV. CONCLUSION

All the above steps integrated into a single process gives an end to end automated mechanism which can be used as an alternative to ADB, and can be used to transfer commands to an Android/Linux DUT. The proposed mechanism has been implemented and tested successfully. The DUT can be visualized as virtually a black box about which no details like the baud rate or the port to which it is connected is known. The proposed mechanism works seamlessly and in a completely automated manner.

REFERENCES

[1] Chetan Patil, "Development of a Simple Serial Communication Protocol for Microcontrollers (SSCPM)", International Journal of Scientific and Research Publications, Volume 1, Issue 1, December 2011.

[2] M. Srilatha, CH. Tejashree, SV. Kishore, P. Chandrasekhar and S.R.Pankaj Kumar, "A Real Time Implementation of Serial Communicatin between Graphical User Interface and Simulator Boarduing RS-232", International Journal of Engineering Trends and Technology, Volume 4, Issue 8, August 2013.

[3] Frank Da Cruz, "Kermit, A File Transfer Protocol", ISBN 0-932976-88-6, February 1985, Copyright 1987.

.