How to customize LTSI Test Project with LTP

Kenji TADANO and Kengo IBE
Mitsubishi Electric
13th, November 2015
Japan Technical Jamboree 55
Outline

• What is the LTSI Project?
  – LTSI Test Environment

• How to Customize?
  – Add New Board (e.g. Raspberry Pi2)
  – Add New Test Suite (e.g. LTP : Linux Test Project)

• Comparison result of running LTP for each case

• Conclusion
What is LTSI Test Project?

• LTSI Project:
  – The project creates and maintains Linux Kernel which is expected to be stable in quality for the typical lifetime of a consumer electronics product, typically 2 years.
  – LTSI-4.1 Developing now
    • **Closed Merge Window:** End of October

• LTSI Test Project
  – The project creates the LTSI Test Environment.
  – The LTSI Test Environment is Jenkins based automation test framework.
  – Include many test suites and kinds of target boards
    • 28 benchmarks and 33 functional test programs are already integrated
    • Minnow board (x86), koelsch (arm), quem-arm (QEMU) are already integrated
  – I hope to further increase the kind of target board, test suite.
  – I’m happy that many people will join this project.
LTSI Test Environment (Overview)

- The main page of GUI of LTSI Test Environment

![Test Automation Framework](image)

- History of Test Results
- Target Boards
- List of Test name
LTSI Test Environment (Flow)

1. Compile Test Suite

2. Send the test binary

3. Execute some tests on the target board

4. Get the results

5. Show the results on GUI

Target Board:
- (koslsch) (minnow)
- QEMU: (qemu-arm)

Test Framework

- jenkins
- Debian 7.X

SDK (Cross-tools)
- minnow-jta
- renesas
- qemu-arm

Test Suite
- Bench marks:
  - [bonnie]
  - [cyclictest]
  - [Dhrystone]
  - [himeno]:
- Functional:
  - [bzip2]
  - [LTP]
  - [expat]
  - [netperf]:

Target Board:
- (koslsch) (minnow)
- QEMU: (qemu-arm)
How to Customize ? (New Target)

- 3 step to add New Target
  - Make the SDK for the target
    - Using yocto project
  - Deploy the SDK into Test Framework
  - Set target Information by GUI

Test Framework

SDK (Cross-tools)
- minnow-jta
- renesas
- qemu-arm

NEW target

Test Suite

1. Make the SDK
2. Deploy the SDK

New Setting

3. Set info. by UI

Jenkins

Debian 7.X
How to Customize ?(Raspberry pi 2)

• 3 step to add New Target
  – Make the SDK for the target
    • Using yocto project
  – Deploy the SDK into Test Framework
  – Set target Information by GUI

Test Framework

SDK (Cross-tools)

- minnow-jta
- renesas
- qemu-arm

New target

1. Make the SDK
2. Deploy the SDK
3. Set info. by UI

New Setting

Test Suite

Jenkins

Debian 7.X
How to Customize ?(Raspberry pi 2)

• Make the SDK
  – Getting **poky** from Yocto project
    ```
    $ git clone git://git.yoctoproject.org/poky.git
    ```
  – Getting **meta-raspi** and **meta-jta**
    • **meta-raspi**: For making a OS image and SDK for Raspberry pi2
      ```
      $ git clone git://git.yoctoproject.org/meta-raspberrypi
      ```
    • **meta-jta**: For adding Headers and Libs for the Test Suite
      ```
      $ git clone https://bitbucket.org/cogentembedded/meta-jta.git
      ```
How to Customize ?(Raspberry pi 2)

• Make the SDK (Cont’d)
  – Configure the environment to build
    • Execute “oe-init-build-env” script in Poky Directory
      ```
poky$ source oe-init-build-env build-raspi2
      ```
    • then “build-raspi2” directory is created like this directory tree.
      ```
build-raspi2$tree
   └── conf
      └── bblayers.conf
      └── local.conf
      ```
  – “build-raspi2” includes a conf directory
  – There are “bblayers.conf” and “local.conf” in the conf directory
How to Customize ?(Raspberry pi 2)

• Make the SDK (Cont’d)
  – Setting to build(Cont’d)
    • Configure bblayers.conf for meta-raspberrypi & meta-jta
      BBLAYERS ?= " ¥
        /home/melco/sdk/yocto/poky/meta ¥
        /home/melco/sdk/yocto/poky/meta-yocto ¥
        /home/melco/sdk/yocto/poky/meta-yocto-bsp ¥
        /home/melco/sdk/yocto/poky/meta-raspberrypi ¥
        /home/melco/sdk/yocto/poky/meta-jta ¥
      
    • Configure local.conf for meta-raspberrypi & meta-jta
      
      #MACHINE ?= "genericx86-64"
      #MACHINE ?= "mpc8315e-rdb"
      #MACHINE ?= "edgerouter"
      MACHINE ?= "raspberrypi2"
      GPU_MEM = "16"

Adding the path of “meta-raspberrypi” & “meta-jta”

Setting MACHINE & GPU Memory size for raspi2
How to Customize *(Raspberry pi 2)*

• Make the SDK (Cont’d)

   – Build SDK

   ```
   melco@debian-7:~/sdk/yocto/poky/build-raspi2$ bitbake meta-toolchain
   Parsing recipes: 100%
   Parsing of 912 .bb files complete (0 cached, 912 parsed). 1341 targets, 61 skipped, 0 masked, 0 errors.
   NOTE: Resolving any missing task queue dependencies
   Build Configuration:
   BB_VERSION        = "1.27.1"
   BUILD_SYS         = "x86_64-linux"
   NATIVELSBSTRING   = "Debian-7.8"
   TARGET_SYS        = "arm-poky-linux-gnueabihf"
   MACHINE           = "raspberrypi2"
   DISTRO            = "poky"
   DISTRO_VERSION    = "1.8+snapshot-20150908"
   TUNE_FEATURES     = "arm armv7a vfp vfpv4 neon callconvention-hard vfpv4 cortexa7"
   TARGET_FPU        = "vfp-vfpv4-neon"
   meta
   meta-yocto
   meta-yocto-bsp    = "master:c1df471feacaf2590216aa476ce242908dac38cf"
   meta-raspberrypi  = "master:17dad9328b100beda1cf870c9075e509b5cbfa90"
   meta-jta          = "master:86387705bfe2ae9495bd661f0c4c7cead8fe06de"
   ```

   Execute “`bitbake meta-toolchain`” command in the build-raspi2 Directory

   To be able to verify “MACHINE”
   For raspi2

   To be able to verify that “`bblayers.conf`” works
How to Customize ?(Raspberry pi 2)

• Make the SDK (Cont’d)
  – Build SDK (Cont’d)
  • When building SDK finished, SDK install script is created at <Build Dir>/tmp/deploy/sdk/

```
melco@debian-7:~/sdk/yocto/poky/build-raspi2$ ls -al tmp/deploy/sdk/
合計 206104
drwxr-xr-x 2 melco melco  4096 9月 8 19:04 .
drwxr-xr-x 5 melco melco  4096 9月 8 14:45 ..
-rw------- 1 melco melco  9331 9月 8 19:04 poky-glibc-x86_x86_64-meta-toolchain-cortexa7hf-vfp-vfpv4-neon-toolchain-1.8+snapshot.host.manifest
-rw------- 1 melco melco 103547364 9月 8 19:04 poky-glibc-x86_64-meta-toolchain-cortexa7hf-vfp-vfpv4-neon-toolchain-1.8+snapshot.sh
-rw------- 1 melco melco  1866 9月 8 19:03 poky-glibc-x86_64-meta-toolchain-cortexa7hf-vfp-vfpv4-neon-toolchain-1.8+snapshot.target.manifest
```

This file is the SDK install script.
How to Customize ?(Raspberry pi 2)

• 3 step to add New Target
  – Make the SDK for the target
    • Using yocto project
  – Deploy the SDK into Test Framework
  – Set target Information by GUI

Test Framework

1. Make the SDK
2. Deploy the SDK
3. Set info. UI

New target

SDK (Cross-tools)

Test Suite

minnow-jta
renesas
qemu-arm

New Setting

Jenkins

Debian 7.X
How to Customize ?(Raspberry pi 2)

• Deploy the SDK into Test Framework
  – You can Deploy the SDK anywhere
    • This is the default Directory `/home/jenkins/tools/`. Minnow, qemu-arm and renesas-arm SDK are already in the directory.

```bash
melco@debian-7:~$ /sdk/yocto/poky/build-raspi2/tmp/deploy/sdk$ ./poky-glibc-x86_64-meta-toolchain-cortexa7hf-vfp-vfpv4-neon-toolchain-1.8+snapshot.sh -y -d /home/jenkins/tools/raspi2
Poky (Yocto Project Reference Distro) SDK installer version 1.8+snapshot
=====================================================================
The directory "/home/jenkins/tools/raspi2" already contains a SDK for this architecture.
If you continue, existing files will be overwritten! Proceed[y/N]? Y
[sudo] password for melco:
Extracting SDK...done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the environment setup script e.g.
```
```
How to Customize ?(Raspberry pi 2)

- Deploy the SDK into Test Framework(conf.)
  - how to Set the Test Framework for the SDK
    - Adding raspi2 configuration on /home/Jenkins/scripts/tools.sh
    - The Test Framework already includes here minnow, qemu-arm and renesas-arm configurations.

```bash
elif [ "${PLATFORM}" = "raspi2" ]; then
  SDKROOT=$JTA_ENGINE_PATH/tools/raspi2/sysroots/cortexa7hf-vfp-vfpv4-neon-poky-linux-gnueabi
  # environment script changes PATH in the way that python uses libs from sysroot which is not what we want, so save it and use later
  ORIG_PATH=$PATH
  PREFIX=arm-poky-linux-gnueabi
  source $JTA_ENGINE_PATH/tools/raspi2/environment-setup-cortexa7hf-vfp-vfpv4-neon-poky-linux-gnueabi
  HOST=arm-poky-linux-gnueabi
  unset PYTHONHOME
  env -u PYTHONHOME
```

“SDKROOT” is the path of the sysroot that there is in deploying the SDK Directory.

Setting "PREFIX" for cross compile

Setting this path written the file of environment variable. This file is in the directory deploying the SDK.

Set “HOST” for cross compile like “PREFIX”
How to Customize ? (Raspberry pi 2) 

• Deploy the SDK into Test Framework (conf.)
  – Set of Test Framework for the Target (raspi2)
    • Adding raspi2 target board configuration on
      /home/jenkins/overlays/boards/<targetname>.board
    • A Sample target board configuration file is template-dev.board
    • When you add a new board, you should use template-dev.board
How to Customize ?(Raspberry pi 2)

• Deploy the SDK into Test Framework(conf.)

```plaintext
inherit "base-board"
include "base-params"

IPADDR="set_ip_here"
LOGIN="root"
JTA_HOME="/home/a"
PASSWORD=""
PLATFORM="set platform here (see tools.sh)"
TRANSPORT="ssh"
ARCHITECTURE="set_ia32_or_arm_here"
SATA_DEV="/dev/sdb1"
SATA_MP="/mnt/sata"
USB_DEV="/dev/sda1"
USB_MP="/mnt/usb"
MMC_DEV="/dev/mmcblk0p2"
MMC_MP="/mnt/mmc"

LTP_OPEN_POSIX_SUBTEST_COUNT_POS="1319"
LTP_OPEN_POSIX_SUBTEST_COUNT_NEG="169"
EXPAT_SUBTEST_COUNT_POS="1769"
EXPAT_SUBTEST_COUNT_NEG="41"
```

Setting IP address of a target
Login user name
Directory to run some test
LOGIN user password
Setting Platform name
Setting Architecture name
Setting a device Information of the target board
like SATA, USB and MMC etc.

Setting configuration
For each test
like LTP and EXPAT etc.
How to Customize ?(Raspberry pi 2)

• Deploy the SDK into Test Framework(conf.)

– For example, <target name>.board for Raspi2

```plaintext
inherit "base-board"
include "base-params"
IPADDR="192.168.1.42"
LOGIN="root"
JTA_HOME="/home/a"
PASSWORD="pi"
PLATFORM="raspi2"
TRANSPORT="ssh"
ARCHITECTURE="arm"

MMC_DEV="/dev/mmcblk0p1"
MMC_MP="/mnt/mmc"
LTP_SYSCALL_COUNT_TPASS="4071"
LTP_SYSCALL_COUNT_TINFO="2776"
LTP_SYSCALL_COUNT_TCONF="140"
LTP_SYSCALL_COUNT_TFAIL="4"
LTP_SYSCALL_COUNT_TBROK="2764"
```

Setting IP address of a target
Login user name
Directory to run some test
LOGIN user password
Setting Platform name
Setting Architecture name
Setting a MMC Information of Raspberry Pi2
Setting the configuration for LTP
How to Customize ? (Raspberry pi 2)

• 3 step to add New Target
  – Make the SDK for target
    • Using yocto project
  – Deploy the SDK into Test Framework
  – Set target information by GUI

Test Framework

<table>
<thead>
<tr>
<th>SDK (Cross-tools)</th>
<th>Test Suite</th>
</tr>
</thead>
<tbody>
<tr>
<td>minnow-jta</td>
<td></td>
</tr>
<tr>
<td>renesas</td>
<td></td>
</tr>
<tr>
<td>qemu-arm</td>
<td></td>
</tr>
<tr>
<td>New target</td>
<td>1. Make the SDK</td>
</tr>
<tr>
<td></td>
<td>2. Deploy the SDK</td>
</tr>
</tbody>
</table>

New Setting

3. Set info. by UI

Jenkins

Debian 7.X
How to Customize ?(Raspberry pi 2)

• Set target Information by UI
  – Select “Targets Status” on top screen of Test Framework

Select “Targets Status”

The list of target board information.
How to Customize ?(Raspberry pi 2)

• Set target Information by UI(conf.)
  – Select “New Node”

  – Then, you can see a configuration form

Select “New Node”

Select “Copy Exiting Node” and Enter template-dev in Copy from
How to Customize ?(Raspberry pi 2)

• Set target Information by UI(conf.)
  – You enter just 2 forms as “Name” and “List of Key-values pairs”

Enter the target name

Enter the path of a file of target configuration

Click Save button
Set target Information by UI(conf.)

- You can see a target list that new target board was added.

Finish adding new target as Raspberry pi 2!!!
How to Customize ?(New Test Suite)

• 3 step to add New Test Suite
  – Create a script for running a new test suite
  – Deploy the script and a test suite tarball
  – Set the test suite information by GUI

Test Framework

SDK (Cross-tools)
- minnow-jta
- renesas
- qemu-arm

Test Suite

Benchmarks
- [bonnie]
- [cyclictest]
- [Dhrystone]
- [himeno]
- [new test]

Functional
- [bzip2]
- [LTP]
- [expat]
- [netperf]
- [new test]

Jenkins

Debian 7.X

New Setting

©2015 Mitsubishi Electric Corporation
How to Customize ? (New Test Suite)

• 3 step to add New Test Suite
  – Create a script for running a new test suite
  – Deploy the script and a test suite tarball
  – Set the test suite information by GUI
How to Customize ? (Linux Test Project)

• Create the script named “ltp-all.sh” (1)

```
tarball=ltp-full-20150420.tar.bz2

function test_build {
    make autotools
    ./configure CC="${CC}" AR="${AR}" RANLIB="${RANLIB}" LDFLAGS="${LDFLAGS}" --
    without-perl --without-python --target=$PREFIX --host=$PREFIX --
    prefix=`pwd`/target_bin --build=`uname -m`-unknown-linux-gnu
    make CC="${CC}" make install
}

function test_deploy {
    put -r target_bin /tmp/jta.$TESTDIR/
}

function test_run {
    safe_cmd "cd /tmp/jta.$TESTDIR/target_bin; ./runltp -f syscalls |
    tee $JTA_HOME/jta.$TESTDIR/$TESTDIR.log"
}
```

To describe tarball name of the adding test suite

To describe procedure of creating test module using cross compile.

To describe procedure of deploying the test module to the target.

To describe commands to execute the test module on target.

In this case, to show running LTP command and collecting the result log.
How to Customize ?(Linux Test Project)

• Create the script named “ltp-all.sh” (conf.)

```bash
function test_processing {
  ## To judge test result
  assert_define LTP_SYSCALL_COUNT_TPASS
  assert_define LTP_SYSCALL_COUNT_TINFO
  assert_define LTP_SYSCALL_COUNT_TCONF
  assert_define LTP_SYSCALL_COUNT_TFAIL
  assert_define LTP_SYSCALL_COUNT_TBROK

  TPASS_CRIT="TPASS :"
  TINFO_CRIT="TINFO :"
  TCONF_CRIT="TCONF :"
  TFAIL_CRIT="TFAIL :"
  TBROK_CRIT="TBROK :"

  log_compare "${TESTDIR}" $LTP_SYSCALL_COUNT_TPASS "${TPASS_CRIT}" "TPASS"
  log_compare "${TESTDIR}" $LTP_SYSCALL_COUNT_TINFO "${TINFO_CRIT}" "TINFO"
  log_compare "${TESTDIR}" $LTP_SYSCALL_COUNT_TCONF "${TCONF_CRIT}" "TCONF"
  log_compare "${TESTDIR}" $LTP_SYSCALL_COUNT_TFAIL "${TFAIL_CRIT}" "TFAIL"
  log_compare "${TESTDIR}" $LTP_SYSCALL_COUNT_TBROK "${TBROK_CRIT}" "TBROK"

  echo "test_processing done"
}
```
How to Customize ?(Linux Test Project)

• Functional.sh inherits from ltp-all.sh.
  – “functional.sh” is defined on LTSI test by default.

```
source $JTA_ENGINE_PATH/scripts/overlays.sh
set_overlay_vars

source $JTA_ENGINE_PATH/scripts/reports.sh
source $JTA_ENGINE_PATH/scripts/functions.sh

pre_test $TESTDIR

if $Rebuild; then
  build
fi

deploy

test_run

get_testlog $TESTDIR

test_processing
```

To include common scripts and execute overlay using Test plan and spec files. Test plan and Spec files provide the very flexibility in configuring tests to be run on different boards and scenarios in the Test Framework.

Standard sequence for running test script on the Test Framework.
- “Pre_test” is checking precondition like connection with target board and target version and so on before running test.
- “Build” is executing test_build function which is defined on “ltp-all.sh” as I explained.
- “Deploy” is executing test_deploy function.
- “Get_testlog” is getting the executing log after running the test..
- “test_run” and “test_processing” are defined on “ltp-all.sh”.

©2015 Mitsubishi Electric Corporation
How to Customize ?(New Test Suite)

• 3 step to add New Test Suite

  – Create a script for running a new test suite
  – Deploy the script and a test suite tarball
  – Set the test suite information by GUI
• Deploy the script and test suite tarball
  – To create work directory “Functional.LTP.all” under /home/jenkins/tests/.
    • Arbitrary directory name can be used but the above is standard.
  – To obtain tarball of LTP from the below site
    • https://github.com/linux-test-project/ltp/releases/tag/20150420
  – To put the created script and tarball under Functional.LTP.all.

  melco@debian-7:/home/jenkins/tests/ Functional.LTP.all$ ls
  ltp-all.sh ltp-full-20150420.tar.bz2
How to Customize ?(New Test Suite)

• 3 step to add New Test Suite
  – Create a script for running a new test suite
  – Deploy the script and a test suite tarball
  – Set the test suite information by GUI

<table>
<thead>
<tr>
<th>Test Framework</th>
<th>Test Suite</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDK (Cross-tools)</td>
<td>Benchmark Functional</td>
</tr>
<tr>
<td>minnow-jta</td>
<td>[bonnie] [bzip2]</td>
</tr>
<tr>
<td>renesas</td>
<td>[cyclic]test [LTP]</td>
</tr>
<tr>
<td>qemu-arm</td>
<td>[Dhrystone] [expat]</td>
</tr>
<tr>
<td></td>
<td>[himeno] [netperf]</td>
</tr>
<tr>
<td></td>
<td>[new test] [new test]</td>
</tr>
</tbody>
</table>

Debian 7.X

Jenkins

New Setting
How to Customize ?(Linux Test Project)

• Set test suite Information by GUI
  – To select “New Test” on the left side of screen of Test Framework
How to Customize (Linux Test Project)

• Set test suite information by GUI
  – To input Test name
  – To choose “Copy existing Test” and Copy from

Input Functional.LTP.all in “Test name”

Select “Copy existing Test” and input Functional.LTP.Open_Posix in “Copy from”
How to Customize ? (Linux Test Project)

• Set test suite Information by GUI
  – To input the created script path in “Command” field of “Execute shell” of “Test Run”

Input ltp-all.sh path in “Command” of Execute shell
How to Customize ?(Linux Test Project)

• Set test suite Information by GUI
  – You can see new test suite name in Functional Tab

Finish adding new test suite as Linux Test Project !!!
Run new LTP on Raspberry Pi2

• Test Environment

- Running Test Framework on Debian 7.8
- Show the test result by web browser
- Adding the SDK Raspberry Pi2
- Adding the LTP-20150420

PC

To Connect with Ethernet

Raspberry Pi2

- Running poky 1.8 from Yocto project
- Kernel version: 3.18
Run new LTP on Raspberry Pi2 (Cont’d)

• To select LTP-20150420

  – To choose Functional.LTP.all

  – To choose “Run Test Now”
Run new LTP on Raspberry Pi2 (Cont’d)

• To Run LTP-20150420

Chose “raspberrypi2” in the pulldown

Click and Run the test
Run new LTP on Raspberry Pi2 (Cont’d)

• We can Show the log with Console output at run time
Run new LTP on Raspberry Pi2 (Cont’d)

• To Show Test Results

Test History

Test Results is SUCCESS!

Console output

++ echo FUNCTIONAL.LTP.2015
+ upName=FUNCTIONAL_LTP_20150420_ALL
+ fcmname=FUNCTIONAL_LTP_20150420_ALL_FAIL_CASE_COUNT
+ fcc=
+ '[-z '']'
+ return 0

POST BUILD TASK : SUCCESS
END OF POST BUILD TASK : 0
Finished: SUCCESS

Complete Running Test!
Comparison result of running LTP

• For using OSS test suite effectively, we would like to find out the proper categorization to choose test case.

• We ran LTP for each cases and compared the results from the below perspectives.
  – Hardware difference: Minnow board vs Raspberry Pi2
  – Kernel difference: 3.18 vs 4.1
  – Userland difference: minimal vs with GUI
    • core-image-minimal vs core-image-sato (on Yocto Project)
  – Bit architecture difference: 32bit vs 64bit
## Result summary

<table>
<thead>
<tr>
<th>case</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>Minnow board (32bit)</td>
<td>Minnow board (64bit)</td>
<td>Raspberry Pi2</td>
<td>Raspberry Pi2</td>
<td>Raspberry Pi2</td>
</tr>
<tr>
<td>Kernel</td>
<td>4.1.8</td>
<td>4.1.8</td>
<td>3.18.11</td>
<td>4.1.10</td>
<td>4.1.10</td>
</tr>
<tr>
<td>Userland</td>
<td>core-image-sato</td>
<td>core-image-sato</td>
<td>core-image-sato</td>
<td>core-image-sato</td>
<td>core-image-minimal</td>
</tr>
<tr>
<td>TPASS</td>
<td>938</td>
<td>868</td>
<td>934</td>
<td>934</td>
<td>933</td>
</tr>
<tr>
<td>TWARN</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TCONF</td>
<td>64</td>
<td>134</td>
<td>70</td>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>TFAIL</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>TBROK</td>
<td>54</td>
<td>54</td>
<td>55</td>
<td>55</td>
<td>56</td>
</tr>
</tbody>
</table>

- **TPASS** - Indicates that the test case had the expected result and passed
- **TWARN** - Indicates that the test case experienced an unexpected or undesirable event that should not affect the test itself such as being unable to cleanup resources after the test finished.
- **TCONF** - Indicates that the test case was not written to run on the current hardware or software configuration such as machine type, or, kernel version.
- **TFAIL** - Indicates that the test case had an unexpected result and failed.
- **TBROK** - Indicates that the remaining test cases are broken and will not execute correctly, because some precondition not met, such as a resource not being available.
Checking about TWARN/TFAIL

<table>
<thead>
<tr>
<th>case</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>Minnow board (32bit)</td>
<td>Minnow board (64bit)</td>
<td>Raspberry Pi2</td>
<td>Raspberry Pi2</td>
<td>Raspberry Pi2</td>
</tr>
<tr>
<td>Kernel</td>
<td>4.1.8</td>
<td>4.1.8</td>
<td>3.18.11</td>
<td>4.1.10</td>
<td>4.1.10</td>
</tr>
<tr>
<td>Userland</td>
<td>core-image-sato</td>
<td>core-image-sato</td>
<td>core-image-sato</td>
<td>core-image-sato</td>
<td>core-image-minimal</td>
</tr>
<tr>
<td>TPASS</td>
<td>938</td>
<td>868</td>
<td>934</td>
<td>934</td>
<td>933</td>
</tr>
<tr>
<td>TWARN</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TCONF</td>
<td>64</td>
<td>134</td>
<td>70</td>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>TFAIL</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>TBROK</td>
<td>54</td>
<td>54</td>
<td>55</td>
<td>55</td>
<td>56</td>
</tr>
</tbody>
</table>

- TWARN 3 items: Occurred on Minnow board only.
- TFAIL 3 items: The results of all cases are same. There might be no dependency.
## Checking about TBROK

<table>
<thead>
<tr>
<th>case</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>Minnow board (32bit)</td>
<td>Minnow board (64bit)</td>
<td>Raspberry Pi2</td>
<td>Raspberry Pi2</td>
<td>Raspberry Pi2</td>
</tr>
<tr>
<td>Kernel</td>
<td>4.1.8</td>
<td>4.1.8</td>
<td>3.18.11</td>
<td>4.1.10</td>
<td>4.1.10</td>
</tr>
<tr>
<td>Userland</td>
<td>core-image-sato</td>
<td>core-image-sato</td>
<td>core-image-sato</td>
<td>core-image-sato</td>
<td>core-image-minimal</td>
</tr>
<tr>
<td>TPASS</td>
<td>938</td>
<td>868</td>
<td>934</td>
<td>934</td>
<td>933</td>
</tr>
<tr>
<td>TWARN</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TCONF</td>
<td>64</td>
<td>134</td>
<td>70</td>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>TFAIL</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>TBROK</td>
<td>54</td>
<td>54</td>
<td>55</td>
<td>55</td>
<td>56</td>
</tr>
</tbody>
</table>

- The results of each cases are same, excepting the below.
  - 1 item: **NOT** occurred on Minnow board (32bit) only.
  - 1 item: Occurred on Raspberry Pi2 only.
  - 1 item: Occurred on core-image-minimal only.
Checking about TCONF

<table>
<thead>
<tr>
<th>case</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td>Minnow board (32bit)</td>
<td>Minnow board (64bit)</td>
<td>Raspberry Pi2</td>
<td>Raspberry Pi2</td>
<td>Raspberry Pi2</td>
</tr>
<tr>
<td>Kernel</td>
<td>4.1.8</td>
<td>4.1.8</td>
<td>3.18.11</td>
<td>4.1.10</td>
<td>4.1.10</td>
</tr>
<tr>
<td>Userland</td>
<td>core-image-sato</td>
<td>core-image-sato</td>
<td>core-image-sato</td>
<td>core-image-sato</td>
<td>core-image-minimal</td>
</tr>
<tr>
<td>TPASS</td>
<td>938</td>
<td>868</td>
<td>934</td>
<td>934</td>
<td>933</td>
</tr>
<tr>
<td>TWARN</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TCONF</td>
<td>64</td>
<td>134</td>
<td>70</td>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>TFAIL</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>TBROK</td>
<td>54</td>
<td>54</td>
<td>55</td>
<td>55</td>
<td>56</td>
</tr>
</tbody>
</table>

- The results of each cases are same, excepting the below.
  - 10 items: **NOT** occurred on Minnow board (32bit) only.
  - 1 item: Occurred on Raspberry Pi2 only.
  - 2 items: Occurred on Minnow board only.
  - 66 items: Occurred on Minnow board (64bit) only.
  - 3 items: NOT occurred on Minnow board (64bit) only.
The details of test case

• The below test case might be depending on Hardware. *(7items)*
  • Raspberry Pi2 only
    – clock_getres01 (TCONF)
    – getrusage04 (TBROK)
  • Minnow board only
    – fanotify05, fanotify06 (TCONF)
    – Fanotify01, fanotify02, fanotify04 (TWARN)

  – The below test case might be depending on bit architecture. *(69items)*
    • Minnow 64bit only
      – bdflush01, chown01_16, chown02_16, chown03_16, chown05_16, fchown01_16, fchown02_16, fchown03_16, fchown05_16, fstatat01
      – fanotify05, fanotify06 (TCONF)
      – rmdir01, rmdir02, rmdir03, rmdir04, rmdir05, rmdir06, rmdir07, rmdir08, rmdir09, rmdir10, rmdir11, rmdir12
    • Other than Minnow 64bit
      – fork14, getcpu01, mmap15 (TCONF)

  – The below test case might be depending on User land. *(1item)*
    • core-image-minimal only.
      – Utimensat01 (TBROK)

  – The below test case might be depending on Minnow 32bit. *(11items)*
    • Other than Minnow 32bit
      – eventfd01, io_cancel01, io_destroy01, io_getevents01, io_setup01, io_submit01, readdir21, sgetmask01, set_thread_area01,
        ssetmask01 (TCONF)
      – syslog08 (TBROK)

# There is no items of depending on Kernel.
Next step

• We sorted out the result of running test and chose test cases which might be depending on HW or Kernel or Userland or Bit architecture.

• As next step, we will check the below to review source code of them.
  – To confirm if assumption of dependency is correct.
  – To find out the regularity of description of source code.
Conclusion

• Summary
  – LTSI Test Framework has already had many kinds of target boards and Test suites.
  – We showed How to Customize.
    • Adding a new target board such as Raspberry pi 2
    • Adding a new test suite such as LTP-20150420
  – We showed the result of running LTP for each case.

• Future Works
  – To add Kselftest to LTSI Test Project
  – To think about making a SDK without yocto
  – To think about how to choose proper test case.
    • Trying to check the result of running LTP
Reference

• LTSI project:
  – http://ltsi.linuxfoundation.org/

• LTSI Test project:
  – http://ltsi.linuxfoundation.org/ltsi-test-project
  – Test Framework:
    • https://bitbucket.org/cogentembedded/jta-public.git

• Linux Test Project
  – http://ltp.sourceforge.net/