Required tools

Running the MutekH testsuite have the following requirements:

- A compiler toolchain,
- A python 2 interpreter, available in most GNU/Linux and BSD operating systems,
- Some target platform simulators,
- The `testwrap` tool, a modified version of GNU coreutils timeout command,
- Mercurial to clone the testsuite repository.

Depending on the target you wish to run the testsuite on, you may not need to install all of these tools.

The Install page explains how to install these requirements.

Usage

Getting the test suite

Once the required tools are installed and the MutekH source code is fetched, you need to get the testsuite:

```
cd .../mutekh
hg clone https://www.mutekh.org/hg/tests
```

The python module path must point to the tests/lib/python directory:

```
export PYTHONPATH=tests/lib/python/
```

Test makefile generation

Some tests applications are located in tests/pool. The mt.generator python module tool is designed generate a makefile which runs a set of tests:

- It first reads tests descriptions from all directories passed on the command line,
- then detects which backend tools are available on your system (do not forget to set PATH),
- and finally generates a makefile ready to run the tests.

Each test application will be used to generate many test targets in the makefile by exploration of associated configurations space.

```
$ python -m mt.generator tests/pool/hello
Test: hello
  Total tests count:     42
  Available tests count: 42
Writing 'tests.mk' makefile.
```

The generated makefile can then be used to start the previously selected tests:

```
make -f tests.mk
```

This will generate kernel binaries, log files and other files, all with the '`TEST`' prefix. Each passed test target generates a stamp file which must be deleted to start the test again.

Test infrastructure

The testsuite repository contains the following directories:

- `pool/` : Tests source code and description,
- `lib/` : python testsuite modules,
- `doc/` : python code documentation,
- `tools/` : additional tools source code.

The tests/pool directory contains a sub-directory for each test application. Other user test applications may reside elsewhere but a test application must always be packaged in its own directory.

Test application details

This part requires good knowledge of the BuildSystem usage.

A test is always packaged in a directory. It's a regular MutekH application module with an additional test description file. It is composed of:

- Some '.c' source files of the test application,
- An associated '`Makefile`' file,
- A '`config`' file. This file contain configuration sections as described in BuildSystem#Advancedsyntax,
- A '`test`' description file. This python file contains the test description.

The '`test`' description file contains details about how to build and run the test application:

- It specifies configuration sections which may be passed to the build system when building the test application.
- It associates backend names to some of the configuration sections to allow selection of the right execution platform.
- It specifies configuration space to explore for the test.
- It specifies ordered test actions (configuration, build, execute, ...).
- It specifies expected results.

The test will be built and executed for each possible configuration in the configuration test space.

Example test files

Parts of the test files are detailed here as an example. Refer to tests/pool/hello/ for full content.

Configuration sections

The test description file is a python script which relies on python modules found in the tests/lib/python directory.

An instance of the `Config` class is used to identify a valid configuration section which can be passed in the BUILD variable of the build command line. Such an instance must exist for each configuration section available in the '`config`' file which is triggered during exploration of the configuration space.

For instance, if the '`config`' file contains the following lines:

```
%section test_ipi
  CONFIG_HEXO_IPI defined
%else
```

```
CONFIG_HEXO_IPI undefined
```

the following line in the 'test' file allows triggering of the inter-processor interrupts (IPI) feature in the test configuration space:

```
# test configuration           BUILD sections

ipi                  = Config("test_ipi")
```

Target architecture configuration sections need to be associated with one or more backends to enable the test generation tool to chose the right execution platform (eg simulator) and check cross compiler availability. Test will be run on available backends and unavailable backends will be skipped.

```
# test configuration           BUILD sections                              test backends

soclib_mips32el      = Config("soclib-mips32el:pf-tutorial",           "soclib-mips32el-1-tuto")
soclib_mips32el_smp4 = Config("soclib-mips32el:pf-tutorial:test_smp4", "soclib-mips32el-4-tuto")
ibmpc_x86            = Config("ibmpc-x86",                             "ibmpc-x86-1-*")
```

The backend pattern must be of the form `arch-cpu-cpucount-simulator`. When muliple backends match the pattern, all available matching backends are used in generated test. Available backends are defined for each stage in tests/lib/python/mutekh/stages.py.

Test description

Once all `Config` instances have been created, an instance of the `Environment` class must be created which describes the test. This instance have several properties:

- `name`: The test unique name,
- `test_space` : A list of configuration dimensions describing the test configuration space.
- `actions` : A list of actions to perform for this test.
- `success_grep`: A grep expression to search for in the execution output for the test to pass,
- `timeout`: simulation timeout delay,

The `test_space` property is a list of `Dimension` and `Exclude` instances used to finely describe the configuration space based on previously declared `Config` objects.

The following example shows how to defines a two dimensional space with the first dimension being the target architecture and the second dimension triggering the IPI feature. An additional `Exclude` rule excludes configurations where inter-processors interrupts would be used in single processor platforms.

```
test_space = [
    Dimension(soclib_mips32eb, soclib_mips32eb_smp4, ibmpc_x86),
    Dimension(ipi, None),
    Exclude(ipi & ~soclib_mips32eb_smp4)
    ],
```