Run Tests with kunit_tool

We can either run KUnit tests using kunit_tool or can run tests manually, and then use kunit_tool to parse the results. To run tests manually, see: Documentation/dev-tools/kunit/run_manual.rst. As long as we can build the kernel, we can run KUnit.

kunit_tool is a Python script which configures and builds a kernel, runs tests, and formats the test results.

Run command:

```
./tools/testing/kunit/kunit.py run
```

We should see the following:

```
Generating .config...
Building KUnit kernel...
Starting KUnit kernel...
```

We may want to use the following options:

```
./tools/testing/kunit/kunit.py run --timeout=30 --jobs=`nproc --all
```

- `--timeout` sets a maximum amount of time for tests to run.
- `--jobs` sets the number of threads to build the kernel.

kunit_tool will generate a `.kunitconfig` with a default configuration, if no other `.kunitconfig` file exists (in the build directory). In addition, it verifies that the generated `.config` file contains the `CONFIG` options in the `.kunitconfig`. It is also possible to pass a separate `.kunitconfig` fragment to kunit_tool. This is useful if we have several different groups of tests we want to run independently, or if we want to use pre-defined test configs for certain subsystems.

To use a different `.kunitconfig` file (such as one provided to test a particular subsystem), pass it as an option:

```
./tools/testing/kunit/kunit.py run --kunitconfig=fs/ext4/.kunitconfig
```

To view kunit_tool flags (optional command-line arguments), run:

```
./tools/testing/kunit/kunit.py run --help
```

Create a `.kunitconfig` File

If we want to run a specific set of tests (rather than those listed in the KUnit `defconfig`), we can provide Kconfig options in the `.kunitconfig` file. For default .kunitconfig, see: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/tools/testing/kunit/configs/default.config. A `.kunitconfig` is a `minconfig` (a .config generated by running `make savedefconfig`), used for running a specific set of tests. This file contains the regular Kernel configs with specific test targets. The `.kunitconfig` also contains any other config options required by the tests (For example: dependencies for features under tests, configs that enable/disable certain code blocks, arch configs and so on).

To create a `.kunitconfig`, using the KUnit `defconfig`:

```
cd $PATH_TO_LINUX_REPO
cp tools/testing/kunit/configs/default.config .kunit/.kunitconfig
```

We can then add any other Kconfig options. For example:

```
CONFIG_LIST_KUNIT_TEST=y
```

kunit_tool ensures that all config options in `.kunitconfig` are set in the kernel `.config` before running the tests. It warns if we have not included the options dependencies.

> **Note**
>
> Removing something from the `.kunitconfig` will not rebuild the `.config file`. The configuration is only updated if the `.kunitconfig` is not a subset of `.config`. This means that we can use other tools (For example: `make menuconfig`) to adjust other config options. The build dir needs to be set for `make menuconfig` to work, therefore by default use `make O=.kunit menuconfig`.

Configure, Build, and Run Tests

If we want to make manual changes to the KUnit build process, we can run part of the KUnit build process independently. When running kunit_tool, from a `.kunitconfig`, we can generate a `.config` by using the `config` argument:

```
./tools/testing/kunit/kunit.py config
```

To build a KUnit kernel from the current `.config`, we can use the `build` argument:

```
./tools/testing/kunit/kunit.py build
```

If we already have built UML kernel with built-in KUnit tests, we can run the kernel, and display the test results with the `exec` argument:

```
./tools/testing/kunit/kunit.py exec
```

The `run` command discussed in section: **Run Tests with kunit_tool**, is equivalent to running the above three commands in sequence.

Parse Test Results

KUnit tests output displays results in TAP (Test Anything Protocol) format. When running tests, kunit_tool parses this output and prints a summary. To see the raw test results in TAP format, we can pass the `--raw_output` argument:

```
./tools/testing/kunit/kunit.py run --raw_output
```

If we have KUnit results in the raw TAP format, we can parse them and print the human-readable summary with the `parse` command for kunit_tool. This accepts a filename for an argument, or will read from standard input.

```
# Reading from a file
./tools/testing/kunit/kunit.py parse /var/log/dmesg
# Reading from stdin
dmesg | ./tools/testing/kunit/kunit.py parse
```

Run Selected Test Suites

By passing a bash style glob filter to the `exec` or `run` commands, we can run a subset of the tests built into a kernel . For example: if we only want to run KUnit resource tests, use:

```
./tools/testing/kunit/kunit.py run 'kunit-resource*'
```

This uses the standard glob format with wildcard characters.

Run Tests on qemu

kunit_tool supports running tests on qemu as well as via UML. To run tests on qemu, by default it requires two flags:

- `--arch`: Selects a configs collection (Kconfig, qemu config options and so on), that allow KUnit tests to be run on the specified architecture in a minimal way. The architecture argument is same as the option name passed to the `ARCH` variable used by Kbuild. Not all architectures currently support this flag, but we can use `--qemu_config` to handle it. If `um` is passed (or this flag is ignored), the tests will run via UML. Non-UML architectures, for example: i386, x86_64, arm and so on; run on qemu.
- `--cross_compile`: Specifies the Kbuild toolchain. It passes the same argument as passed to the `CROSS_COMPILE` variable used by Kbuild. As a reminder, this will be the prefix for the toolchain binaries such as GCC. For example:
  - `sparc64-linux-gnu` if we have the sparc toolchain installed on our system.
  - `$HOME/toolchains/microblaze/gcc-9.2.0-nolibc/microblaze-linux/bin/microblaze-linux` if we have downloaded the microblaze toolchain from the 0-day website to a directory in our home directory called toolchains.

If we want to run KUnit tests on an architecture not supported by the `--arch` flag, or want to run KUnit tests on qemu using a non-default configuration; then we can write our own``QemuConfig``. These `QemuConfigs` are written in Python. They have an import line `from..qemu_config import QemuArchParams` at the top of the file. The file must contain a variable called `QEMU_ARCH` that has an instance of `QemuArchParams` assigned to it. See example in: `tools/testing/kunit/qemu_configs/x86_64.py`.

Once we have a `QemuConfig`, we can pass it into kunit_tool, using the `--qemu_config` flag. When used, this flag replaces the `--arch` flag. For example: using `tools/testing/kunit/qemu_configs/x86_64.py`, the invocation appear as

```
./tools/testing/kunit/kunit.py run \
        --timeout=60 \
        --jobs=12 \
        --qemu_config=./tools/testing/kunit/qemu_configs/x86_64.py
```

To run existing KUnit tests on non-UML architectures, see: Documentation/dev-tools/kunit/non_uml.rst.

Command-Line Arguments

kunit_tool has a number of other command-line arguments which can be useful for our test environment. Below the most commonly used command line arguments:

- `--help`: Lists all available options. To list common options, place `--help` before the command. To list options specific to that command, place `--help` after the command.

  > **Note**
  >
  > Different commands (`config`, `build`, `run`, etc) have different supported options.

- `--build_dir`: Specifies kunit_tool build directory. It includes the `.kunitconfig`, `.config` files and compiled kernel.
- `--make_options`: Specifies additional options to pass to make, when compiling a kernel (using `build` or `run` commands). For example: to enable compiler warnings, we can pass `--make_options W=1`.
- `--alltests`: Builds a UML kernel with all config options enabled using `make allyesconfig`. This allows us to run as many tests as possible.

  > **Note**
  >
  > It is slow and prone to breakage as new options are added or modified. Instead, enable all tests which have satisfied dependencies by adding `CONFIG_KUNIT_ALL_TESTS=y` to your `.kunitconfig`.