NanoK Documentation

Release 0.0.1

Jean Guyomarc'h

Oct 07, 2018

Documentation Contents

This is the root documentation of the NanoK kernel. Take a tour by going through the *Introduction* page. If you are interested in developping NanoK, please visit *NanoK Development*.

Warning: NanoK is under heavy development, and as such is not ready for anything but its own development.

CHAPTER 1

Introduction

Contents

1.1 What is NanoK?

NanoK is a nanoscopic kernel for micro-controllers. It aims at providing an efficient runtime tasks abstraction for hardware that does not come with many resources. It is not a standalone kernel: it must be compiled with the application that will run on the selected hardware. This allows the kernel to be tuned finely for a given usage. As such, NanoK can be seen as a unikernel

1.2 What hardware does NanoK support?

Warning: NanoK is still in an omega development phase. It is highly experimental and shall not be used for anything but its own development.

- STM32F4-DISCOVERY: ST's discovery board with an STM324f variant.

1.3 How to build NanoK?

Please refer to the *NanoKBS* page.

1.4 How to contribute to the development of NanoK?

Please refer to the *NanoK Development* page.

CHAPTER 2

NanoKBS

2.1 Introduction

As stated in the *Introduction* page, NanoK must be compiled with the application that will run on the targeted hardware. As such, NanoK's build shall be driven by the application. Given this requirement, NanoK should not be built on its own. So, NanoK does not provide a build system out-of-the-box. This may seem shocking at first, but this allows an more in-depth integration with the application. Third parties will be able to tweak NanoK more easily.

Instead, NanoK provides an exhaustive and comprehensive **description of its sources**, as well as a small (python3) script `nanokbs/nanokbs.py` that allows to generate a build system from an input template file. Third-parties will be able to write the template that better suit their needs. A basic ninja build template is provided by NanoK for its testing. It can be used as an example.

2.2 Command-Line Interface

`nanokbs/nanokbs.py` can be run as an executable python3 script. To consult the command-line interface of this tool, run the following command, from the top source directory of NanoK:

```
./nanokbs/nanokbs.py --help
```

`nanokbs/nanokbs.py` is run to generate a build system if the following arguments are provided:

- `--app` (or `-a`) to provide the path to a JSON (hjson) file that describes how the application is built (see *Describing a NanoK Application*);
- `--template` (or `-t`) to provide the path to a jinja2 template that will be used to generate the application build system; and
- `--output` (or `-o`) to specify the path where the generated build system shall be written.

It is likely to always be used as follows:

```
./nanokbs/nanokbs.py -a "<path/to/app.hjson>" -t "<path/to/template>" -o "<build>"
```

2.3 Describing a NanoK Application

An application shall be defined in an hjson file. It is a superset of JSON, so this description can be written in JSON instead of HJSON. Note however that HJSON offers a more understandable syntax with syntactic suggar.

This document must define a key `app`, which shall contain the following data:

- `arch`: a string that indicates the hardware architecture to be used. The possible options can be seen by looking at the directories in the `src/arch` folder. Note that `pc` is reserved to unit tests.
- `products`: it is a list of binaries compiled against NanoK. Each element within this list shall define the following data:
  - `name`: the name of the binary to be generated;
  - `path`: provide the path from with all strings in the field `c_sources` and `asm_sources` are relative to;
  - `c_sources` and `asm_sources`, which consists in the source files required by the product.
- and any other data that a user-defined template may want to use.

You can have a look at the following existing files:

- `tests/pc/unit.hjson`: used to generate the unit tests;
- `tests/target/disco.hjson`: used to generate an stm32f4-disco program.

CHAPTER 3

NanoK Development

3.1 Installing the build dependencies

NanoK requires python3 (probably at least 3.4, but this was not tested so earlier python3 versions may work). It relies on the hjson and jinja2 packages. Assuming that `pip3` is your pip program for python3, run the following to install the python dependencies for the current user:

```
pip3 install --user -r env/requirements.txt
```

Then, you need to install the following programs:

- ninja: the fast incremental and parallel build-system;
- gcc: the GNU Compiler Collection with **C11** support;
- diff: to compare files.

You can install the right packages using this one-liner, depending on your GNU/Linux distribution. It must be run from the top-source directory of NanoK:

Distribution	Command
Debian/Ubuntu	`cat env/debian_packages.txt | xargs sudo apt install`

If you plan to develop for the STM32f4 platform, you will need to install the following tools:

- the `arm-none-eabi-` GCC toolchain. You may find it here;
- the st-link utility, to flash and debug your applications on the hardware target.

See `scripts/gen-disco.sh` to have an idea of the build and installation procedure of these tools.

3.2 Building the Core Unit Tests

The core services provided by NanoK are written in a platform-agnostic module. This allows them to be tested on a Linux-hosted platform, so the underlying logic of NanoK can be unit-tested without the burden of maintaining hardware tests. As such, NanoK is first developed on a Linux-hosted platform, and then implemented on a physical platform.

You should first make sure you are able to compile and run the core test suite on a Linux platform. First generate the appropriate build system, then use the generated `build.ninja` to perform the tasks. Assuming a POSIX shell, run the following:

```
sh ./scripts/gen-unit-tests.sh
ninja check
```

To better understand what the tests do, and how to maintain them, please refer to the *NanoK Unit Tests* page.

3.3 Developping on the STM32f4

As previously mentioned, you will need the install the st-link utility. Run the following:

```
git clone https://github.com/texane/stlink.git
make -C stlink release
sudo make -C stlink/build/Release install
```

Note that this will install `st-link` on your system. This is required because udev rules are distributed.

Then, run the following to compile, flash and start to run the test application:

```
./scripts/gen-disco.sh
ninja gdb-test-run
```

This will compile the application and uploaded through a GDB server interface. You can now drive the application:

3.4 Development Workflow

NanoK is hosted on GitHub, and git is the source control management software used for its development.

The branch `master` is the stable development branch. All work shall be done in branches. When a feature, bugfix or any other kind of modification to the sources of NanoK is ready, a pull request shall be initiated with the `master` branch as being the destination. The commits will be rebased onto `master` and submitted to the continuous integration server. Once all the tests pass, the pull request containing your changes will be rebased onto `master`.

3.5 Coding Guidelines

NanoK is written in C11 with GNU extensions.

CHAPTER 4

NanoK Unit Tests

Unit tests reside in the *tests/pc/* directory. The following files are worth considering:

- `unit.hjson`: which describes the different applications (products) to be built. Each product is actually a test;
- `build-unit-tests.ninja.j2`: which is a ninja build template to compile and run the unit tests on a Linux-hosted platform; and
- `run-test.sh`: which is the unit tests runner. Each unit test is supposed to produce a given result in its standard output. This test compares the output to a known reference. Failure in executing the test or exactly matching the expected output will yield to a test failure. These references are stores in the *tests/pc/refs/* directory.