POK Developper Guide

POK Team

May 25, 2013

Contents

1 About this manual

This manual provides information about the development of POK. It indicates coding rules and naming convention so that everybody could improve POK by modifying its source code.

1.1 About POK

POK is a free operating system compliant with the ARINC653 and MILS standards. It provides time and space partitioning to isolate software in partitions. POK is released under the BSD licence and thus, could be modified and used for commercial as well as non-commercial use. To have more information about the licence of the projet, see POK website[1].

2 Coding guidelines

There are our coding guidelines:

1. Prefix for types: `pok_`
2. Prefix for functions : `pok_`
3. Prefix for maccro: `POK_` but not for conditional compiling:
   (a) When the code needs a functionnality, we define a maccro with the prefix `POK_NEEDS_my-functionnality`
   (b) When a maccro configures the kernel or user code, it has the prefix `POK_CONFIG_my-config-directive`
4. Indentation for ANY loop/condition
5. Commits must be as small as possible
6. Reduce machine-dependent code as more as possible
7. Each header-file must begin with `#ifdef __POK_SUBCATEGORY_FILENAME_H__`
8. Loop and condition style is :

```
condition
{
}
```

   and NOT

```
condition {
}
```

9. To commit, do **not** invoke `svn commit` but issue `make commit` at the root of the sources directory. **To commit, lftp is required**.
10. If you introduce a new function for the userland, you must add relevant documentation in the `doc/userguide/` directory.

[1] http://pok.gunnm.org

3 Source organization

At the root directory, two main directories are available: `kernel` and `libpok`. We detail the organization and the guidelines for each subdirectory of kernel and libpok.

3.1 kernel

In the kernel, sources files are supposed to contain few lines of code. In consequence, there is one file for each service.

- `arch`: contains arch-dependent code. There is one directory for each architecture and one subdirectory for each BSP. For example, files for the x86 architecture and the x86-qemu BSP are located in the `arch/x86/x86-qemu` directory.
- `core`: contains the core functionnality of POK - threads, partitions, health monitoring, ...
- `include`: contains headers files. The organization of header files is the same than source files. So, you will find `core`, `middleware` or `arch` directories in the `include` directory.
- `libc`: provides some functionnalities for printing things. Functions located in this directory are here mainly for debugging purposes.
- `middleware`: contain the code for inter-partitions communication (sampling and queueing ports). It also contains some functionnalities about virtual ports routing.

3.2 libpok

In libpok, sources files are supposed to contain more code than in kernel. So, there is one file for each functions. There is the organization and purpose of each directory.

- `arch`: contains architecture dependent files. Unlike the kernel, there is no need to separate each BSP so there is no subdirecties for each architecture.
- `arinc653`: contains the implementation of the ARINC653 layer.
- `core`: contains the main functionnality of POK. It contains the thread service, lockobjects, semaphores, events.
- `drivers`: contains device drivers implemented in POK.
- `include`: contains header files. As in the kernel, the structure of this directory follow the structure of the sources.
- `libc`: contains the C-library of POK (stdio, stdlib and so on).
- `libm`: contains the libmath backported from NetBSD.

- `middleware`: contains sources for sampling and queueing ports (interfacing with the kernel - inter-partition communication) but also blackboard and buffers (intra-partition communication)

4 Optimization (about the `POK_CONFIG_OPTIMIZE_FOR_GENERATED_CODE`

Systems generated with POK must be lightweight and keep a small memory footprint to be compliant with embedded requirements and ensures a good code coverage. When a system is written by hand, the libpok layer contains all its functionnalities. It is more convenient for the developper, he does not have to specify which functions he needs.

However, when a system is generated from AADL models, it defines the maccro `POK_CONFIG_OPTIMIZE_FOR_GENERATED_CODE` and sets its values to 1. Its means that the code specifies precisely which functions are used. Then, the generated code specifies which services it needs using `POK_NEEDS*` maccros. For example, the `POK_NEEDS_LIBC_STDIO` specifies that it needs all functions of `libc/stdio`.

Then, each function of libpok is surrounded with a `POK_CONFIG_NEEDS_FUNC*` or `POK_CONFIG_NEEDS_*`. You have to introduce that in your code when you introduce new services in POK.

Then, the file in `include/core/dependencies.h` specifies which functions are needed for each service. When the `POK_CONFIG_OPTIMIZE_FOR_GENERATED_CODE` is not set, all functions are enabled (default behavior). But is defined, functions are carefully activated, depending on their service.

5 Documentation

5.1 User Guide

Each improvement and enhancement in kernel or libpok must be documented in the userguide (see `doc/userguide` in the POK sources) to keep a consistency between the documentation and the sources.

5.2 Code documentation

The code **must be** documented using doxygen. At each release, we issue a documentation in HTML and PDF using code documentation. The following paragraphs indicate at least what information should be included in the sources **at least**. Keep in mind that the more the code is documented, the best it is for users.

Beginning of a file

Specify the file, the author, the data and a brief description. You can have an example in `kernel/core/thread.c`. For example, the following comments provide these informations. It should be located at the beginning of the file.

```
/**
 * \file    core/thread.c
 * \author  Julien Delange
 * \date    2008-2009
 * \brief   Thread management in kernel
 */
```

Functions

You **MUST** document each function and details what the function do. You specify that with a comment just before the function. The comment must begin with `/**`. There is an exemple for the function `pok_thread_init`:

```
/**
 * Initialize threads array, put their default values
 * and so on
 */
void pok_thread_init(void)
{
...
```

Global variables

Each global variable **must be** documented. As functions, you put a comment just before the global variable. This comment **must begin** with `/**`. There is an example for the global variable `pok_threads`:

```
/**
 * We declare an array of threads. The amount of threads
 * is fixed by the software developper and we add two theads
 *    - one for the kernel thread (this code)
 *    - one for the idle task
 */
pok_thread_t                    pok_threads[POK_CONFIG_NB_THREADS];
```

6 Submit a patch

If you found a bug or just want to send us an improvement, you can reach us at the following address: `pok-devel at listes dot enst dot fr`. Please send an email with the patch. We will answer and potentially merge your patch in the current version of POK.

7 Algorithms guidelines

Before introducing new functions or modifying existing ones, please qualify your code in terms of complexity, memory overhead, computation overhead, determinism. POK targets safety-critical systems, and so, its functions must provide high confidence to the user and must address these problems in its functions.

Moreover, we always follow the moto *Keep It Simple, Stupid* for each function: code must be understandable and documentation to be spread over users or developpers.

8 GDB'ing POK with QEMU

POK allows you to attach a remote GDB to monitor the kernel *or* its partitions.

To do so, go to your example directory and run the system in debug mode.

```
$ cd $POK_PATH/examples/partition-threads
$ make run-gdb
```

QEMU should be paused. Now run GDB using the kernel image.

```
$ gdb generated-code/kernel/kernel.elf
...
(gdb) target remote :1234
Remote debugging using :1234
0x0000fff0 in ?? ()
(gdb) continue
```

You're all set if you want to debug the kernel, but what if you want to instrumentate a partition?

In GDB, we first have to let the kernel know about the symbols of the partition. But we also need to know where they are loaded in kernel space. Let's say we want to debug partition #1. One way to know where it was relocated would be:

```
(gdb) p pok_partitions[0].base_addr
$1 = 1175552
```

Please note that `pok_partition_init` *must* have completed or the array won't be initialized yet.

Now we can load the symbol table with the correct offset.

```
(gdb) add-symbol-file generated-code/cpu/part1/part1.elf 1175552
add symbol table from file "generated-code/cpu/part1/part1.elf" at
        .text_addr = 0x11f000
(y or n) y
Reading symbols from /home/laurent/pok/examples/partitions-threads/generated-code/cpu/part1/pa
(gdb) b user_hello_part1
Breakpoint 1 at 0x11f17a: file ../../../hello1.c, line 21.
```

```
(gdb) continue
Continuing.

Program received signal SIGTRAP, Trace/breakpoint trap.
0x0000017a in ?? ()
```

You will notice debug symbols are missing, although we loaded them above. This is because the memory mapping is not the same in kernel end userland. We have to load the symbol file again in place of the kernel.

```
(gdb) symbol-file generated-code/cpu/part1/part1.elf
Load new symbol table from "/home/laurent/pok/examples/partitions-threads/generated-code/cpu/p
Reading symbols from /home/laurent/pok/examples/partitions-threads/generated-code/cpu/part1/pa
(gdb) bt
#0  user_hello_part1 () at ../../../hello1.c:21
#1  0xc4830845 in ?? ()
```

9 Commit on the SVN

You **MUST** commit by ussing `make commit` at the root of the sources. This make target will build all examples on all architectures/platforms supported by POK to verify that modified sources seem to be consistent and do not introduce a regression.

For that, you have to install compilers for every platform supported by POK.

10 Join the POK developper network !

If you want to join the POK team, please send us an email (`pok-devel at listes dot enst dot fr`. We are always looking for developpers with strong skills in C, ASM and low-level programming.

If you are interested and think you can improve the project, you're welcome!