Lesson 6: Logging

- [x] 6.1 Introduction
- [x] 6.2 Commands
- [x] 6.3 Kconfig
- [ ] 6.4 Exercise

What is logging?

The logging API provides a common interface to process messages issued by developers. Messages are passed through a frontend and are then processed by active backends. Custom frontends and backend can be used if needed. Default configuration uses built-in frontend and UART backend.

Logging is highly configurable at compile time as well as at run time. Using Kconfig options logs can be gradually removed from compilation to reduce image size and execution time when logs are no longer needed. During compilation logs can be filtered out on module basis and severity level.

Logs can also be compiled in but filtered on run time using dedicated API. Run time filtering is indepenent for each backend and each source of log messages. Source of log messages can be a module or specific instance of the module.

There are four severity levels available in the system: error, warning, info and debug. For each severity level the logging API (include/logging/log.h) has a set of dedicated macros. Logger API also has macros for logging data.

For each level following set of macros are available (change X by level ERR, WRN, INF or DBG):

- LOG_X for standard printf-like messages, e.g. LOG_ERR.
- LOG_HEXDUMP_X for dumping data, e.g. LOG_HEXDUMP_WRN.
- LOG_INST_X for standard printf-like message associated with the particular instance, e.g. LOG_INST_INF.
- LOG_INST_HEXDUMP_X for dumping data associated with the particular instance, e.g. LOG_HEXDUMP_INST_DBG.

There are configuration categories: configurations per module and global configuration. When logging is enabled globally, it works for modules. However, modules can disable logging locally. Every module can specify its own logging level. The module must define the LOG_LEVEL macro before using the API. Unless a global override is set, the module logging level will be honored. The global override can only increase the logging level. It cannot be used to lower module logging levels that were previously higher. It is also possible to globally limit logs by providing maximal severity level present in the system, where maximal means lowest severity (e.g. if maximal level in the system is set to info, it means that errors, warnings and info levels are present but debug messages are excluded).

Each module which is using the logging must specify its unique name and register itself to the logging. If module consists of more than one file, registration is performed in one file but each file must define a module name.

Logger's default frontend is designed to be thread-safe and minimizes time needed to log the message. Time consuming operations like string formatting or access to the transport are not performed by default when logging API is called. When logging API is called a message is created and added to the list. Dedicated, configurable buffer for pool or log messages is used. There are 2 types of messages: standard and hexdump. Each message contains source ID (module or instance ID and domain ID which might be used for multiprocessor systems), timestamp and severity level. Standard message contains pointer to the string and arguments. Hexdump message contains copied data and string.

How to use logging?

In order to use logging in the module, a unique name of a module must be specified and module must be registered using LOG_MODULE_REGISTER. Optionally, a compile time log level for the module can be specified as the second parameter. Default log level (CONFIG_LOG_DEFAULT_LEVEL) is used if custom log level is not provided.

#include <logging/log.h>
LOG_MODULE_REGISTER(foo, CONFIG_FOO_LOG_LEVEL);
If the module consists of multiple files, then LOG_MODULE_REGISTER should appear in exactly one of them. All the other files should use LOG_MODULE_DECLARE to declare its membership in the module. Optionally, a compile time log level for the module can be specified as the second parameter. Default log level (CONFIG_LOG_DEFAULT_LEVEL) is used if custom log level is not provided.

#include <logging/log.h>
/* In all files comprising the module but one */
LOG_MODULE_DECLARE(foo, CONFIG_FOO_LOG_LEVEL);
In order to use logging API in a function implemented in a header file LOG_MODULE_DECLARE macro must be used in the function body before logging API is called. Optionally, a compile time log level for the module can be specified as the second parameter. Default log level (CONFIG_LOG_LEVEL_DEFAULT_LEVEL) is used if custom log level is not provided.

```
#include <logging/log.h>

static inline void foo(void)
{
    LOG_MODULE_DECLARE(foo, CONFIG_FOO_LOG_LEVEL);

    LOG_INF("foo");
}
```

Dedicated Kconfig template (subsys/logging/Kconfig.template.log_config) can be used to create local log level configuration.

Example below presents usage of the template. As a result CONFIG_FOO_LOG_LEVEL will be generated:

```
module = FOO
module-str = foo
source "subsys/logging/Kconfig.template.log_config"
```

Controlling the logging

Logging can be controlled using API defined in include/logging/log_ctrl.h. Logger must be initialized before it can be used. Optionally, user can provide a function which returns time; if not, k_cycle_get_32 is used. log_process() function is used to trigger processing of one log message (if pending). Function returns true if there is more messages pending.

Following snippet shows how logging can be processed in simple forever loop.

```
#include <log_ctrl.h>

void main(void)
{
    log_init();

    while (1) {
        if (log_process() == false) {
            /* sleep */
        }
    }
}
```

X: ERR, WRN, INF or DGB

Command	Description
LOG_X	Writes an X level message to the log
LOG_PRINTK	Unconditionally print raw log message
LOG_INST_X	Writes an X level message associated with the instance to the log
LOG_HEXDUMP_X	Writes an X level hexdump message to the log
LOG_INST_HEXDUMP_X	Writes an X level hexdump message associated with the instance to the log
LOG_MODULE_REGISTER	Create module-specific state and register the module with Logger. This macro normally must be used after including <logging/log.h> to complete the initialization of the module.
LOG_MODULE_DECLARE	Macro for declaring a log module (not registering it).
LOG_LEVEL_SET	Macro for setting log level in the file or function where instance logging API is used.

Mode of operations:

Kconfig	Description
CONFIG_LOG	Deferred mode
CONFIG_LOG_MODE_DEFERRED	Deferred mode
CONFIG_LOG2_MODE_DEFERRED	Deferred mode v2
CONFIG_LOG_MODE_IMMEDIATE	Immediate (synchronous) mode
CONFIG_LOG2_MODE_IMMEDIATE	Immediate (synchronous) mode v2
CONFIG_LOG_MODE_MINIMAL	Minimal footprint mode

Filtering options:

Kconfig	Description
CONFIG_LOG_RUNTIME_FILTERING	Enables runtime reconfiguration of the filtering.
CONFIG_LOG_DEFAULT_LEVEL	Default level, sets the logging level used by modules that are not setting their own logging level.
CONFIG_LOG_OVERRIDE_LEVEL	It overrides module logging level when it is not set or set lower than the override value.
CONFIG_LOG_MAX_LEVEL	Maximal (lowest severity) level which is compiled in.

Processing options:

Kconfig	Description
CONFIG_LOG_PRINTK	Redirect printk calls to the logging
CONFIG_LOG_BUFFER_SIZE	Number of bytes dedicated for the message pool. Single message capable of storing standard log with up to 3 arguments or hexdump message with 12 bytes of data take 32 bytes. In v2 it indicates buffer size dedicated for circular packet buffer.

Formatting options:

Kconfig	Description
CONFIG_LOG_FUNC_NAME_PREFIX_ERR	Prepend standard ERROR log messages with function name. Hexdump messages are not prepended.
CONFIG_LOG_FUNC_NAME_PREFIX_WRN	Prepend standard WARNING log messages with function name. Hexdump messages are not prepended.
CONFIG_LOG_FUNC_NAME_PREFIX_INF	Prepend standard INFO log messages with function name. Hexdump messages are not prepended.
CONFIG_LOG_FUNC_NAME_PREFIX_DBG	Prepend standard DEBUG log messages with function name. Hexdump messages are not prepended.
CONFIG_LOG_BACKEND_SHOW_COLOR	Enables coloring of errors (red) and warnings (yellow).
CONFIG_LOG_BACKEND_FORMAT_TIMESTAMP	If enabled timestamp is formatted to hh:mm:ss:mmm,uuu. Otherwise is printed in raw format.