

[RFC] New kernel-message logging API

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-09/msg07226.html>

- *From:* "Vegard Nossum" <vegard.nossum@xxxxxxxx>
 - *Date:* Sat, 22 Sep 2007 21:27:29 +0200
-

After recent discussions on LKML and a general dissatisfaction at the current `printk()` kernel-message logging interface, I've decided to write down some of the ideas for a better system.

Requirements

=====

* Backwards compatibility with `printk()`, `syslog()`, etc. There is no way the whole kernel can be converted to a new interface in one go. `printk()` is used all over the kernel, in many different ways, including calls from assembly, multi-line prints spread over several calls, etc.

* Extensibility. Features like timestamping or file/line recording [1] should be both selectable at compile-time and (if compiled in) at run-time.

API

===

```
#define printk(fmt, ...)
```

The main part of the `kprint` interface should be the `kprint()` function. The string must be a single line of information (ie. it must not contain any newlines). Calling `kprint("Hello world")` is equivalent to calling `printk("Hello world\n")`. `kprint()` may be implemented as a macro, and must not be called from assembly.

To support the different log-levels, there exists one `kprint_*()` function for each log-level, for example `kprint_info()`. The string must be a single line of information. Calling `kprint_emerg("Oops.")` is equivalent to calling `printk(KERN_EMERG "Oops.\n")`. These functions may also be implemented as macros, and must also not be called from assembly. The different log-levels (and their functions) are:

```
enum kprint_loglevel {
```

[RFC] New kernel–message logging API

```
KPRINT_EMERG, /* kprint_emerg() */
KPRINT_ALERT, /* kprint_alert() */
KPRINT_CRIT, /* kprint_crit() */
KPRINT_ERROR, /* kprint_error() and/or kprint_err() */
KPRINT_WARNING, /* kprint_warning() and/or kprint_warn() */
KPRINT_NOTICE, /* kprint_notice() */
KPRINT_INFO, /* kprint_info() */
KPRINT_DEBUG, /* kprint_debug() */
};
```

In order to print several related lines as one chunk, the emitter should first allocate an object of the type struct `kprint_buffer`. This buffer is initialized with the function `kprint_buffer_init()` which takes as arguments a pointer to an object of the type struct `kprint_buffer` followed by the log–level number. The emitter may then make as many or as few calls to `kprint_buffer()` that is desired. `kprint_buffer()` is a function that takes a literal string followed by a variable number of arguments, similarly to the `kprint()` function. A final call to `kprint_buffer_flush()` appends the messages to the kernel message log in a single, atomic call. After it has been flushed, the buffer is not usable again (unless it is re–initialized). If for any reason the buffer should be de–initialized without writing it to the log, a call to `kprint_buffer_abort()` must be made.

```
Example: {
struct kprint_buffer buf;

kprint_buffer_init(&buf, KPRINT_DEBUG);
kprint_buffer(&buf, "Stack trace:");

while(unwind_stack()) {
kprint_buffer(&buf, "%p %s", address, symbol);
}

kprint_buffer_flush(&buf);
}
```

It may happen that certain parts of the kernel might wish to emit messages to the log (and console, if any) in an early part of the boot procedure, for example before the main memory allocation routines have been set up properly. For this purpose, a function `kprint_early()` exists. This "early" is a minimal way for the kernel to log its functions, and may as such not include all the features of the full `kprint` system. When the kernel is beyond the critical "early" point, the messages (if any) in the "early" log may be moved into the main logging store and `kprint_early()` must not be used again. `kprint_early()` is a function and may be called from assembly.

To allow non–early calls from assembly, a function `kprint_asm()` exists. This function takes a log–level number followed by a string

literal followed by a variable number of arguments.

The then legacy `printk()` function must be replaced by a backwards-compatible but different interface. In short, `printk` should parse messages, remove (and convert) initial log-level tokens, remove any newlines (splitting the string into several lines), and call the appropriate `kprint()`-system functions.

Internals

=====

The `kprint()` and its log-level variants are implemented as macros in order to be able to transparently pass extra information into the main `kprint()` machinery. As an example, it might happen that an extra feature is added that prepends the current file and line (the `__FILE__` and `__LINE__` macros) to the message, but in such a way that it can be discarded at run-time, or recorded along with the messages. Additionally, this would allow the compiler to completely optimize out calls that are below a certain log-level severity level [2][3].

With such a modular interface, message attributes (for example the current time) and arguments can be recorded separately from the actual format string, instead of written formatted to a ring buffer as a sequence of characters. Parameters would be formatted to their own strings (regardless of the original type) and saved in an array.

```
Example: {
struct kprint_message {
const char *format;

unsigned int argc;
char **argv;

#ifdef KPRINT_TIMESTAMP
unsigned long long timestamp;
#endif

#ifdef KPRINT_LOCATION
const char *file;
unsigned int line;

const char *function;
#endif
};
}
```

This can be a great help, for example in (user-space) localisation of kernel messages [4], since the "static" message (ie. format string) can be translated separately and the arguments re-attached at

[RFC] New kernel-message logging API

run-time, possibly in a different order. A new kernel-/user-space interface would be needed to retrieve the messages in this format.

The syslog() and /proc/kmsg interfaces can retain backwards compatibility by formatting messages as they are requested from user-space.

Considerations

=====

This scheme is obviously much more complex than the printk() is today. But at the same time, it is also much more powerful, extensible, and clearly/cleanly defined.

One of the difficult things with kernel messages is that the interface should be fail-proof. We must be able to tell the user that our memory allocator just blew. In the current printk() interface, a ring buffer of fixed size is used. With a fixed, static buffer, we are certain to never run into memory-related problems, even if the rest of the system is unstable or unusable.

I think an easy solution would be to check for EMERG messages and treat these with special care, like using a static reservoir.

The message entry and a message's arguments are kept separately. Most likely, there will be a huge number of tiny allocations. I am not sure how well this is handled in the kernel. Or we could try to merge the allocation of the struct kprint_message and its arguments into a single allocation by looking at the arguments before they're formatted. After all, the arguments cannot exist without the message or vice versa. Alternatively, a statically-sized ring buffer of struct kprint_message objects could be used, and then only arguments would need to be allocated dynamically. Either way, I think it should be possible to come up with a fairly memory-efficient system even for this method of storing the kernel log.

Also undealt with are dev_printk(), sdev_printk(), and ata_dev_printk(), but our backwards compatible printk() should handle this. Alternatively, the macros could be changed, though as I know close to nothing about them, I'll leave this entire field to somebody else.

References:

- [1] <http://lkml.org/lkml/2007/9/21/267> (Joe Perches <joe@xxxxxxxxxxxx>)
- [2] <http://lkml.org/lkml/2007/9/20/352> (Rob Landley <rob@xxxxxxxxxxxx>)
- [3] <http://lkml.org/lkml/2007/9/21/151> (Dick Streefland <dick.streefland@xxxxxxxxxx>)
- [4] <http://lkml.org/lkml/2007/6/13/146> (holzheu <holzheu@xxxxxxxxxxxxxxxxxxxx>)

[RFC] New kernel-message logging API

Vegard

–

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@xxxxxxxxxxxxxxxx

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>