Doing structured logging from kernel

For obvious reasons it’s impossible to use the sd_journal C API from kernel to do structured logging. However, there are mechanisms for passing key=value pairs as extra data for log messages.

The standard way of reporting/logging from kernel is the printk() function. As its name suggests, it’s very similar to the well-known printf() function from libc, but it has some specialties related to the fact that it’s actually being used in kernel (address) space.

Actually, printk is a whole family of functions varying in the number and types of arguments and the way variadic arguments are passed. The most important way for our case is:

```c
int printk_emit(int facility, int level,
               const char *dict, size_t dictlen,
               const char *fmt, ...);
```

facility and level are the common logging parameters just like the ones passed to the syslog() function. fmt and the variadic arguments are like the respective ones for printf()/printk() functions. Finally, the most important ones for our case are dict and dictlen. The latter one of course just specifies the length of the former one. And that’s needed, because the dict is a dictionary of key=value pairs that are separated by \0-bytes.

Here’s a trivial example of a kernel module using the structured logging:

```c
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Vratislav Podzimek");
MODULE_DESCRIPTION("Testing structured logging from kernel");

static int __init strl_test_init(void)
{
    printk_emit (0, LOGLEVEL_INFO, "TEST=test\0TEST2=test2", 22, "Module loaded");
    return 0;    // Non-zero return means that the module couldn’t be loaded.
}

static void __exit strl_test_cleanup(void)
{
    printk(KERN_INFO "Cleaning up module.
");
```
module_init(strl_test_init);
module_exit(strl_test_cleanup);

and this is how the respective journal entries look like (in the JSON format):

```
{
   "__CURSOR" : "s=7ab355e44efe46b38bc5db3bbaffa43e;i=2ecde;b=b5cef4a829854151af7c0ee85b5a8ee3",
   "__REALTIME_TIMESTAMP" : "1500546073417124",
   "__MONOTONIC_TIMESTAMP" : "75646751532",
   "_BOOT_ID" : "b5cef4a829854151af7c0ee85b5a8ee3",
   "_PRIORITY" : "6",
   "_MACHINE_ID" : "a128b9a3c70b44e6898984060de3a76f",
   "_HOSTNAME" : "localhost.localdomain",
   "_TRANSPORT" : "kernel",
   "SYSLOG_FACILITY" : "0",
   "SYSLOG_IDENTIFIER" : "kernel",
   "KERNEL_TEST" : "test",
   "MESSAGE" : "Module loaded",
   "KERNEL_TEST2" : "test2",
   "_SOURCE_MONOTONIC_TIMESTAMP" : "75638333296"
}
```

```
{
   "__CURSOR" : "s=7ab355e44efe46b38bc5db3bbaffa43e;i=2ecde;b=b5cef4a829854151af7c0ee85b5a8ee3",
   "__REALTIME_TIMESTAMP" : "1500546073426647",
   "__MONOTONIC_TIMESTAMP" : "75646761055",
   "_BOOT_ID" : "b5cef4a829854151af7c0ee85b5a8ee3",
   "_PRIORITY" : "6",
   "_MACHINE_ID" : "a128b9a3c70b44e6898984060de3a76f",
   "_HOSTNAME" : "localhost.localdomain",
   "_TRANSPORT" : "kernel",
   "SYSLOG_FACILITY" : "0",
   "SYSLOG_IDENTIFIER" : "kernel",
   "MESSAGE" : "Cleaning up module.",
   "KERNEL_TEST2" : "test2",
   "_SOURCE_MONOTONIC_TIMESTAMP" : "75638343612"
}
```

It can be seen that the entries come from kernel and that the TEST and TEST2 items are stored as _KERNEL_TEST and _KERNEL_TEST2. This is a transformation that always happens for items coming from kernel. At the same time, keys coming from user-space cannot start with the _KERNEL prefix. That’s how authenticity of the data coming from kernel is ensured.

That means that any standard items/keys we specify for storage events/actions reporting will have to be supported both in their kernel-space and user-space forms, i.e. with the _KERNEL_ prefix and without it respectively.