Kernel Messaging

Advanced Operating Systems and Virtualization

Alessandro Pellegrini

A.Y. 2019/2020

Linux Kernel Messaging System

- Kernel-level software can produce output messages related to events occurring during the execution (debugging by printing)
- The messages can be produced both during initialization and steady state operations, hence:
  - Sofware modules forming the messaging system cannot rely on I/O standard services (such as `sys_write()`)
  - No standard library function can be used for output production
- Management of kernel-level messages occurs via specific modules that take care of the following tasks:
  - Message print on the “console” device
  - Message logging into a circular buffer kept within kernel level virtual addresses

`printk()`

- The kernel level function to produce output messages is `printk()` defined in `kernel/printk/printk.c`
- It accepts a format string, similar to that used for the `printf()` standard library function
  - Floating point values are not allowed
- A message priority can be specified by relying on macros (expanded to strings) which tell how critic is a message

Message Priorities

- Priority levels are defined in `include/linux/kernel.h`

```
#define KERN_EMERG   "<0>"   /* system is unusable */
#define KERN_ALERT   "<1>"   /* action must be taken immediately */
#define KERN_CRIT    "<2>"   /* critical conditions */
#define KERN_ERR     "<3>"   /* error conditions */
#define KERN_WARNING "<4>"   /* warning conditions */
#define KERN_NOTICE  "<5>"   /* normal but significant condition */
#define KERN_INFO    "<6>"   /* informational */
#define KERN_DEBUG   "<7>"   /* debug-level messages */
```

```
printk(KERN_WARNING "message to print")
printk(KERN_INFO "%s: Module message\n", KBUILD_MODNAME);
```

Message Priority Management

- There are 4 configurable parameters which determine how output messages are managed
- They are associated with the following variables:
  - `console_loglevel`: level under which the messages are logged on the console device
  - `default_message_loglevel`: priority level that is associated by default with messages for which no priority value is specified
  - `minimum_console_loglevel`: minimum level to allow a message to be logged on the console device
  - `default_console_loglevel`: the default level for messages destined to the console device

Interacting with Log Level Settings

- `cat /proc/sys/kernel/printk`

  7 4 1 7

  (current default minimum boot-time-default)

- Write to this file to change the parameters (if supported by the specific kernel version/configuration)

- This is not a real stable storage file

syslog()

```
int syslog(int type, char *bufp, int len);
```

- This is the system call to perform management operation on the kernel-level circular buffer hosting output messages
- the `bufp` parameter points to the memory area where the bytes read from the circular buffer will be copied
- `len` specifies how many bytes we are interested in, or a flag (depending on the value of `type`)

syslog()'s type

```
/*
* Commands to sys_syslog:
*
*      0 -- Close the log.  Currently a NOP.
*      1 -- Open the log. Currently a NOP.
*      2 -- Read from the log.
*      3 -- Read up to the last 4k
                    of messages in the ring buffer.
*      4 -- Read and clear last 4k
                    of messages in the ring buffer
*      5 -- Clear ring buffer.
*      6 -- Disable printk's to console
*      7 -- Enable printk's to console
       8 -- Set level of messages printed to console
*/
```

`console_loglevel` can be set (to a value in the range 1-8) by calling:

```
syslog(8, dontcare, value)
```

Messaging Management Daemon

klogd - Kernel Log Daemon

SYNOPSIS

klogd [-c n] [-d] [-f fname] [-iI] [-n] [-o] [-p] [-s] [-k fname] [-v] [-x] [-2]

DESCRIPTION

klogd is a system daemon which intercepts and logs Linux kernel messages

Circular buffer features

- The circular buffer keeping the kernel output messages has size `LOG_BUF_LEN` and is stored in the array `__log_buf`, both in `kernel/printk/printk.c`
  - originally 4096 bytes
  - Since 1.3.54 moved to 8192 bytes
  - Since 2.1.113 moved to 16384 bytes
  - Today, it can be defined at compile time
- A unique buffer is used for any message, independently of the priority level
- The buffer content can be accessed by also relying on `dmesg`

Management of Messages

- To enable the delivery of messages with the exactly-once semantic, printing on console is synchronous (recall that standard library functions only enable at-most-once semantic, due to asynchronous management)
- Hence the `printk()` function does not return control until the message is delivered to any active console-device driver
- The driver, in its turn, does not return control until the message is sent to the (physical) console device
- This may impact performance
  - As an example, the delivery of a message on a serial console device working at 9600 bit/sec, slows down the system by 1 millisecond per char

panic()

- `panic()` is defined in `kernel/panic.c`
- This function prints the specified message on the console device (by relying on `printk()`)
- The string “Kernel panic:” is prepended to the message
- Further, this function halts the machine, hence leading to stopping the execution of the kernel
  - Indeed, threads enter an infinite loop