Gabriele **Proietti Mattia**

Advanced Operating Systems and Virtualization

[Lab 05] Kernel Messaging & Debugging

DIAG
Department of Computer, Control and Management Engineering "A. Ruberti", Sapienza University of Rome

gpm.name · proiettimattia@diag.uniroma1.it

A.Y. 2020/2021 · v2

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:

- software 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()

V5.11

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 but 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.

```
8   #define KERN_EMERG      KERN_SOH "0"    /* system is unusable */
9   #define KERN_ALERT      KERN_SOH "1"    /* action must be taken immediately */
10  #define KERN_CRIT       KERN_SOH "2"    /* critical conditions */
11  #define KERN_ERR        KERN_SOH "3"    /* error conditions */
12  #define KERN_WARNING    KERN_SOH "4"    /* warning conditions */
13  #define KERN_NOTICE     KERN_SOH "5"    /* normal but significant condition */
14  #define KERN_INFO       KERN_SOH "6"    /* informational */
15  #define KERN_DEBUG      KERN_SOH "7"    /* debug-level messages */
16
17  #define KERN_DEFAULT    ""              /* the default kernel loglevel */
```

https://elixir.bootlin.com/linux/v5.11/source/include/linux/kern_levels.h#L8

Message Priority Management

There are 4 configurable parameters which determine how output messages are managed. They are associated with the following variables:

1. **`console_loglevel`**: level under which the messages are logged on the console device
2. **`default_message_loglevel`**: priority level that is associated by default with messages for which no priority value is specified
3. **`minimum_console_loglevel`**: minimum level to allow a message to be logged on the console device
4. **`default_console_loglevel`**: the default level for messages destined to the console device

For seeing and even changing these parameters you can look at the folder

`/proc/sys/kernel/printk`

You will find the 4 numbers referring to the 4 variables.

syslog()

```c
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()

Types

0. SYSLOG_ACTION_CLOSE (0) Close the log. Currently a NOP.
1. SYSLOG_ACTION_OPEN (1) Open the log. Currently a NOP.
2. SYSLOG_ACTION_READ (2) Read from the log.
3. SYSLOG_ACTION_READ_ALL (3) Read all messages remaining in the ring buffer, placing them in the buffer pointed to by bufp.
4. SYSLOG_ACTION_READ_CLEAR (4) Read and clear all messages remaining in the ring buffer.
5. SYSLOG_ACTION_CLEAR (5) The call executes just the "clear ring buffer" command.
6. SYSLOG_ACTION_CONSOLE_OFF (6) The command saves the current value of console_loglevel and then sets console_loglevel to minimum_console_loglevel
7. SYSLOG_ACTION_CONSOLE_ON (7) If a previous SYSLOG_ACTION_CONSOLE_OFF command has been performed, this command restores console_loglevel to the value that was saved by that command.

syslog()

Types

8. `SYSLOG_ACTION_CONSOLE_LEVEL (8)` The call sets `console_loglevel` to the value given in len, which must be an integer between `1` and `8` (inclusive).
9. `SYSLOG_ACTION_SIZE_UNREAD (9)` (since Linux 2.4.10) The call returns the number of bytes currently available to be read from the kernel log buffer
10. `SYSLOG_ACTION_SIZE_BUFFER (10)` (since Linux 2.6.6) This command returns the total size of the kernel log buffer.

dmesg

dmesg is a utility to print or control the kernel ring buffer from /proc/kmsg, by default. The utility, as syslog allows to read, read and clear, show specific log levels

```
DMESG(1)                                                                User Commands                                                                DMESG(1)

NAME
       dmesg - print or control the kernel ring buffer

SYNOPSIS
       dmesg [options]

       dmesg --clear
       dmesg --read-clear [options]
       dmesg --console-level level
       dmesg --console-on
       dmesg --console-off

DESCRIPTION
       dmesg is used to examine or control the kernel ring buffer.

       The default action is to display all messages from the kernel ring buffer.
```

Messaging Management Daemon

klogd is a system daemon which intercepts and logs Linux kernel messages. syslogd is a set of utilities that manages logging. In particular:

*System logging is provided by a version of **syslogd**(8) derived from the stock BSD sources. Support for kernel logging is provided by the **klogd**(8) utility which allows kernel logging to be conducted in either a standalone fashion or as a client of syslogd.*

*In Linux there are two potential sources of kernel log information: the /proc file system and the syscall (sys_syslog) interface, although ultimately they are one and the same. Klogd is designed to choose whichever source of information is the most appropriate. It does this by first checking for the presence of a mounted /proc file system. If this is found the **/proc/kmsg** file is used as the source of kernel log information. If the proc file system is not mounted **klogd** uses a system call to obtain kernel messages. The command line switch (**-s**) can be used to force klogd to use the system call interface as its messaging source.*

https://linux.die.net/man/8/klogd - https://linux.die.net/man/8/syslogd

How Messages Get Logged

The printk function writes messages into a circular buffer that is `__LOG_BUF_LEN` bytes long, a value from 4KB to 1MB (according to the configuration). The function then wakes any process that is waiting for messages:

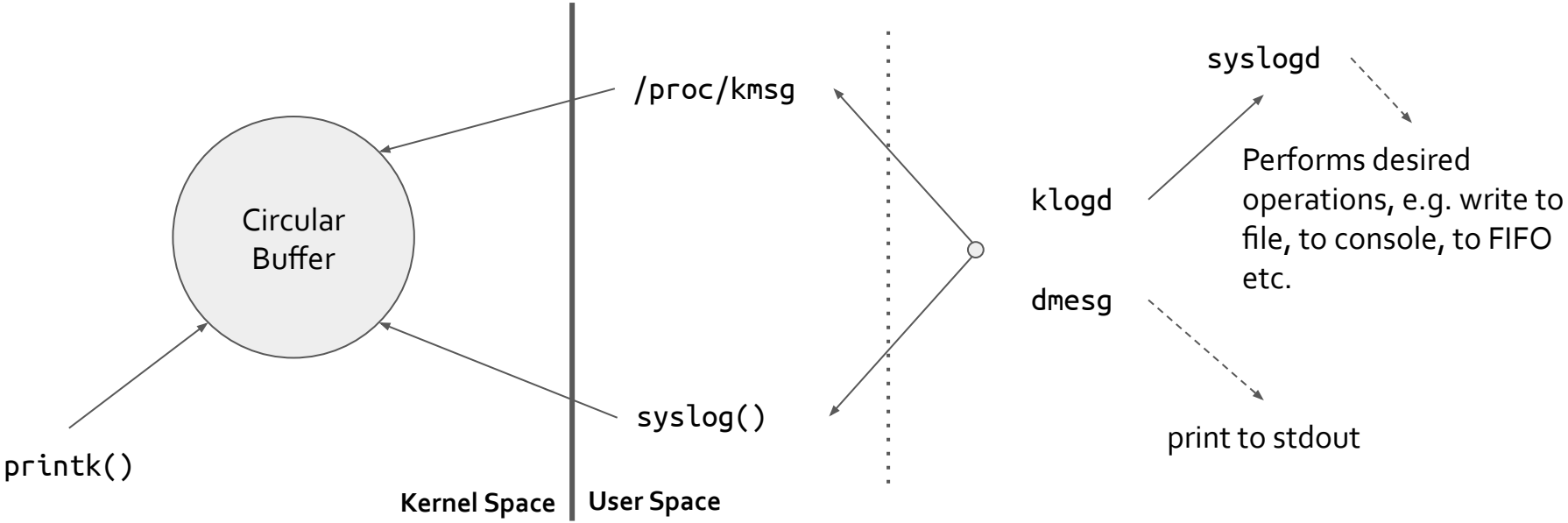
- `syslog` system call
- reading file `/proc/kmsg`

They are similar but reading from /proc/kmsg consumes the data, while syslog allows you to make the data available even for other processes. In general, reading from /proc is easier and it is the default behaviour of `klogd`. Obviously, if klog is running you cannot read from /proc/kmsg because you will contend the data.

In general, when circular buffer ends the printk wraps around and starts adding new data at the beginning of the buffer, overwriting the old data.

How Messages Get Logged

If the `klogd` process is running, it retrieves the kernel messages and dispatches them to syslogd which checks for the configuration in `/etc/syslog.conf` to find out how to deal with them. If klogd is not running, data remain in the circular buffer until someone reads it or the buffer overflows.



Management of Messages

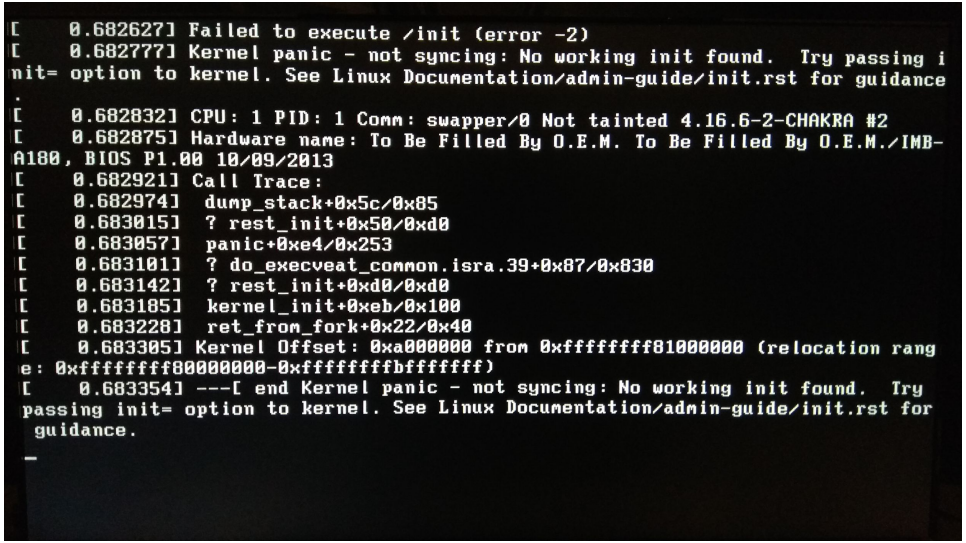
To enable the delivery of messages with the **exactly-once** semantic, printing on console is **synchronous** (recall that standard library functions only enable at-most-once semantic, due to asynchronous management).

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. This may impact performance.

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.



Ubuntu 20.10

On the latest version of Ubuntu there is a package called rsyslogd that is usually already installed in your machine. The demon unifies `klogd` and `syslogd`:

- default configuration is stored in `/etc/rsyslog.conf`
- configuration variation are stored in `/etc/rsyslog.d`
- always have a look at the linux manual, man `rsyslogd`

By default kernel messages are logged in the /var/log/kern.log folder.

Debugging

The Hanging Kernel

We already experienced what is a kernel oops and a kernel panic, for the former it is simple to find the error cause by looking at the kernel log. Using printk for debugging, is always recommended, but when there is a kernel panic or in general the kernel hangs, it could be difficult to understand where is the problem. In general, the kernel hangs for different reasons:

1. spinning indefinitely in a loop
2. waiting indefinitely on a lock, therefore we have deadlocks

The kernel panic is always generated by the panic() function and leads to reason #1 since the kernel cannot go on with that kind of error. In all of these cases, your system hangs and you have to **reset** your machine.

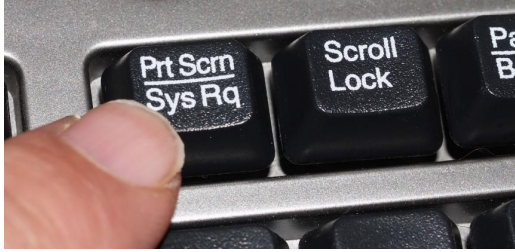
The Magic SysRq

The SysRq is a mechanism according to which you can communicate with the kernel, even if it is frozen by mean of particular keystrokes. But it must be manually enabled in your kernel.

- you can enable it permanently, by marking Magic System Request(SysRq) keys under "Kernel Hacking" (`CONFIG_MAGIC_SYSRQ`) Menu of Kernel Configuration, when compiling the kernel with make menuconfig
- you can enable it temporarily, by using `echo 1 > /proc/sys/kernel/sysrq`
- you can enable it at boot by modifying `/etc/sysctl.conf` and adding a line like `kernel.sysrq=1` (then `sysctl -p`)

The number that you specify represents the features that you want to enable, `1` stands for all the features.

The Magic SysRq



The SysRq capabilities are called by using the key sequence, in order

Alt + SysRq + <character>

By default, the SysRq button is Print Screen (99) but it is not present on all the keyboards. You can remap it to the **Windows** key for example (you can see the key sequence with command sudo showkey -s).

Windows key is `0xe0 0x5b` so you can associate it to SysRq by using the command

```
sudo setkeycodes e05b 99
```

This command must run every time you reboot the machine, but for simplicity you can add it to your `~/.bashrc` file.

https://www.kernel.org/doc/html/latest/admin-guide/sysrq.html

The Magic SysRq

Features

These are the capabilities associated to the keys that you can use with SysRq.

- **b - Will immediately reboot the system without syncing or unmounting your disks.**
- **c - Will perform a system crash by a NULL pointer dereference. A crashdump will be taken if configured.**
- d - Shows all locks that are held.
- e - Send a SIGTERM to all processes, except for init.
- f - Will call the oom killer to kill a memory hog process, but do not panic if nothing can be killed.
- g - Used by kgdb (kernel debugger)
- **h - Will display help** (actually any other key than those listed here will display help. but h is easy to remember :-)
- **i - Send a SIGKILL to all processes, except for init.**
- j - Forcibly "Just thaw it" - filesystems frozen by the FIFREEZE ioctl.
- k - Secure Access Key (SAK) Kills all programs on the current virtual console. NOTE: See important comments below in SAK section.

The Magic SysRq

Features

- **l - Shows a stack backtrace for all active CPUs.**
- **m - Will dump current memory info to your console**.
- n - Used to make RT tasks nice-able
- **o - Will shut your system off (if configured and supported).**
- **p - Will dump the current registers and flags to your console.**
- q - Will dump per CPU lists of all armed hrtimers (but NOT regular timer_list timers) and detailed information about all clockevent devices.
- r - Turns off keyboard raw mode and sets it to XLATE.
- s - Will attempt to sync all mounted filesystems.
- **t - Will dump a list of current tasks and their information to your console.**
- u - Will attempt to remount all mounted filesystems read-only.
- v - Forcefully restores framebuffer console
- v - Causes ETM buffer dump [ARM-specific]

The Magic SysRq

Features

- w - Dumps tasks that are in uninterruptable (blocked) state.
- x - Used by xmon interface on ppc/powerpc platforms. Show global PMU Registers on sparc64. Dump all TLB entries on MIPS.
- y - Show global CPU Registers [SPARC-64 specific]
- z - Dump the ftrace buffer
- **0-9 - Sets the console log level, controlling which kernel messages will be printed to your console. (0, for example would make it so that only emergency messages like PANICs or OOPSes would make it to your console.)**

Dumping Memory

With SysRq you can even trigger memory dumps for better debugging, but you need kdump to be installed. On ubuntu, for instance, you can install the following packet (say Yes to both questions)

```
sudo apt install linux-crashdump
```

After installing, you need to reboot then you can try to use the 'c' command with SysRq. By default, even with kernel oops, the machine will dump the memory and reboot. The reports are in `/var/crash` you can analyze them with the utility `crash`. For using crash you need the decompressed kernel image, aka `vmlinux`.

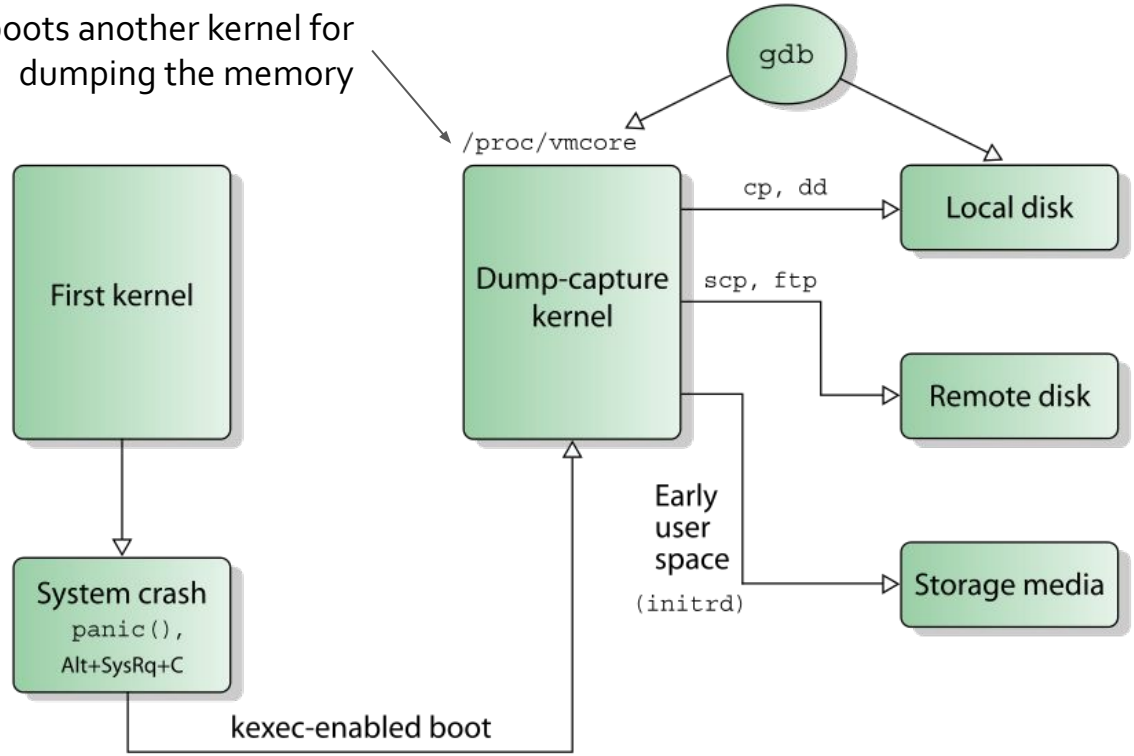
Remember that for debugging, the kernel must be compiled with `CONFIG_DEBUG_INFO=y`.

https://www.kernel.org/doc/Documentation/kdump/kdump.txt

https://ubuntu.com/server/docs/kernel-crash-dump

Dumping Memory

kdump boots another kernel for dumping the memory



https://en.wikipedia.org/wiki/Kdump_(Linux)

Advanced Operating Systems and Virtualization

[Lab 05] Kernel Messaging

LECTURER

Gabriele **Proietti Mattia**

BASED ON WORK BY

http://www.ce.uniroma2.it/~pellegrini/

gpm.name · proiettimattia@diag.uniroma1.it

DIAG