Dmesg and Linux kernel log

A quality version of this page, accepted on *24 September 2019*, was based off this revision.

Contents

1 Article purpose

The purpose of this article is to provide information about the Linux® kernel log including configuration, and to detail usage of dmesg command.

2 Introduction

Linux kernel is able to print log and trace messages, which are by default stored in a ring buffer.

The same messages can also be displayed, applying filter, on uart/console using serial port. This is defined in the kernel command-line, with the "console" parameter. See [1] for detail.

dmesg is a shell command on the kernel console, which also displays the content of the ring buffer, with filter or not (default).

3 printk function

The simplest way to get some debug information from the kernel code is by printing out various information with the kernel's equivalent of printf - the printk function and its derivatives.

```
printk("My Debugger is Printk\n");
```

See elinux.org[2] for reference. This information will be sent to the console, and also stored in a ring buffer.

You can also check to the printk-format.txt[3] document provided in the Linux kernel package to get detail about syntax and formatting.

4 Linux kernel ring buffer

The Linux kernel also manages a ring buffer to store log and trace messages.

The size of the buffer cannot be modified in runtime, and its default size value is 2^CONFIG_LOG_BUF_SHIFT bytes.

To change it, there are 3 possibles ways:

- Modify CONFIG_LOG_BUF_SHIFT value in defconfig file or use the config fragment file:

```
In example for 64K : CONFIG_LOG_BUF_SHIFT=16
```

- or use the Linux kernel menuconfig update

```
 Location:
   -> General setup
     -> Kernel log buffer size (16 => 64KB, 17 => 128KB)
```

- Or modify kernel arguments[4] in kernel command-line (via bootargs value in device tree, or directly in extlinux uboot config file)

```
bootargs = "root=/dev/mmcblk0p5 rootwait rw console=ttySTM0,115200 log_buf_len=65536";
```

This ring buffer can be displayed using *dmesg* command (see dmesg).

5 Loglevels

As reference, please see elinux.org[5].

The log level is used by the kernel to determine the importance of a message and to decide whether it should be presented to the user immediately, by printing it to the current console.

For this, the kernel compares the log level of the message to the console_loglevel (a kernel variable) and if the priority is higher (i.e. a lower value) than the console_loglevel, the message will be printed to the current console. As example, if console_loglevel=5, all messages with log level 0 to 4 will be displayed.

Please note that all messages with loglevel lower or equal to KERN_INFO level are stored in the ring buffer.

5.1 loglevels values

Name	String	Meaning	alias functions	dev alias function
KERN_EMERG	"0"	Emergency messages, system is about to crash or is unstable	pr_emerg	dev_emerg
KERN_ALERT	"1"	Something bad happened and action must be taken immediately	pr_alert	dev_alert
KERN_CRIT	"2"	A critical condition occurred like a serious hardware/software failure	pr_crit	dev_crit
KERN_ERR	"3"	An error condition, often used by drivers to indicate difficulties with the hardware	pr_err	dev_err
KERN_WARNING	"4"	A warning, meaning nothing serious by itself but might indicate problems	pr_warning	dev_warn
KERN_NOTICE	"5"	Nothing serious, but notably nevertheless. Often used to report security events	pr_notice	dev_notice
KERN_INFO	"6"	Informational message e.g. startup information at driver initialization	pr_info	dev_info
KERN_DEBUG	"7"	Debug messages	pr_debug, pr_devel if DEBUG is defined	dev_dbg

"Loglevels table"

Important: please note that Higher priority message is loglevel 0

5.2 Set loglevel filter value for console

5.2.1 Default values

To determine your current console_loglevel on the target you can verifiy with the following command:

```
Board $> cat /proc/sys/kernel/printk
7       Template:Blue           1       7
current Template:Blue minimum default_console
```

The first integer shows you the current console_loglevel; the second the default log level, see Use loglevel in the kernel source for log and trace.

This is defined at compilation:

- Current console loglevel via CONFIG_CONSOLE_LOGLEVEL_DEFAULT=7 (defined in file *lib/Kconfig.debug*)
- Template:Blue loglevel via Template:Blue (defined in file *lib/Kconfig.debug*)
- Minimum console loglevel via #define CONSOLE_LOGLEVEL_MIN 1 (defined in file *include/linux/printk.h*)
- **Default console** loglevel is **equal to CONFIG_CONSOLE_LOGLEVEL_DEFAULT**

5.2.2 Using kernel command-line

The console loglevel can be also set via a kernel command-line parameter if you want to use a different value than one specify by CONFIG_CONSOLE_LOGLEVEL_DEFAULT.
For example:

```
root=/dev/mmcblk0p5 rootwait rw console=ttySTM0,115200 loglevel=4
```

In that case only messages with a higher priority than KERN_WARNING (means < 4, KERN_EMERG to KERN_ERR) will be displayed on the console.

2 ways to add this command-line parameter which is set in the extlinux.conf file of boot partition:

- If using SD card, this is possible to edit the file on host PC:

```
Insert SD card on host PC
Check for mounting boot partition (i.e. /media/$USER/bootfs)
Check for your HW config (i.e. booting on mmc0 (SD Card) with ev1 board)
PC $> cd /media/$USER/bootfs/mmc0_stm32mp157c-ev1_extlinux/
PC $> gedit extlinux.conf
Add loglevel=8 at the end of APPEND line
Save and insert SD card on the board
```

- If using SD Card or eMMC, this is possible to edit the file directly on the board side:

```
When software is boot
Mount boot partition
Board $> mount /dev/mmcblk0p4 /boot (if not already done)
Update the kernel command line
Board $> cd /boot
Board $> cd mmc0_stm32mp157c-ev1_extlinux (case SD card on ev1 board)
Modify extlinux.conf to add loglevel=8 at the end of APPEND line by using ‘vi’ editor
Save and reboot the board
```

5.2.3 Using sysfs in runtime

To change your current console_loglevel simply write to this file:

```
Board $> echo <loglevel> > /proc/sys/kernel/printk
```

or using dmesg command.

As example:

```
Board $> echo 8 > /proc/sys/kernel/printk          # Temporary increase loglevel to
display messages up to loglevel 8
```

In that case, every kernel messages will appear on your console, as all priority higher than 8 (lower loglevel values) will be displayed.

Please note that after reboot, this configuration is reset.

5.2.4 Using menuconfig before compilation

As values are defined first at compilation step, this is also possible to set them (**CONFIG_CONSOLE_LOGLEVEL_DEFAULT** and **CONFIG_MESSAGE_LOGLEVEL_DEFAULT**) using the Linux kernel Menuconfig tool (Menuconfig or how to configure kernel):

```
Symbol: CONSOLE_LOGLEVEL_DEFAULT [=7]
Location:
  Kernel hacking --->
    printk and dmesg options --->
      (7) Default console loglevel (1-15)

Symbol: MESSAGE_LOGLEVEL_DEFAULT [=4]
Location:
  Kernel hacking --->
    printk and dmesg options --->
      (4) Default message log level (1-7)
```

5.3 Use loglevel in kernel source for log and trace

5.3.1 Using printk

A loglevel information can be added in the printk function call, with the following syntax.

```
printk(KERN_ERR "something went wrong, return code: %d\n",ret);
```

When not present, default loglevel value is given by CONFIG_MESSAGE_LOGLEVEL_DEFAULT (usually "4" =KERN_WARNING)

5.3.2 Using dedicated functions

In the loglevels table above, there are some alias functions *pr_* and *dev_*.

These functions are defined to replace *printk + loglevel info inside*, in order to simplify syntax.

```
pr_err("something went wrong, return code: %d\n",ret);
```

*dev_* functions are taken one more parameter to provide more information about current device or driver where message is coming from.

- Example for *pr_info*

```
pr_info("%s%s%s at %s (irq = %d, base_baud = %d) is a %s\n",
        port->dev ? dev_name(port->dev) : "",
        port->dev ? ": " : "",
        port->name,
        address, port->irq, port->uartclk / 16, uart_type(port));
```

will display information below:

```
[ 0.919488] 40010000.serial: ttySTM0 at MMIO 0x40010000 (irq = 41, base_baud = 6046875)
is a stm32-usart
```

- Example for *dev_info*

```
dev_info(&pdev->dev, "interrupt mode used for rx (no dma)\n");
```

will display information below, including device reference automatically:

```
[    1.046700] stm32-usart 40010000.serial: interrupt mode used for rx (no dma)
```

6 earlyprintk

earlyprintk is a Linux kernel debug feature useful to get traces for kernel issues which happen before the normal console is initialized.

- Linux kernel configuration

In order to enable earlyprintk feature, the Linux kernel configuration must activate **CONFIG_DEBUG_LL**, **CONFIG_STM32MP1_DEBUG_UART** and **CONFIG_EARLY_PRINTK** using the Linux kernel Menuconfig tool (Menuconfig or how to configure kernel):

```
Symbol: DEBUG_LL
Location:
  Kernel hacking --->
    [*] Kernel low-level debugging functions

Symbol: STM32MP1_DEBUG_UART
Location:
  Kernel hacking --->
    [*] Kernel low-level debugging functions
      (*) Use STM32MP1 UART for low-level debug

Symbol: EARLY_PRINTK
Location:
  Kernel hacking --->
    [*] Early printk
```

- Serial port configuration

When enabling the Linux kernel configuration **CONFIG_STM32MP1_DEBUG_UART**, it configures the addresses of the UART registers to be used.

By default, on STM32MP1 boards, UART4 is used for console for Linux kernel and by extension at all boot stages.

In case the UART port is different on a new board, you must apply the following changes:

- Update value for **CONFIG_DEBUG_UART_PHYS**, to select the UART port for the debug console

```
Symbol: DEBUG_UART_PHYS [=0x40010000]
Location:
  Kernel hacking --->
    [*] Kernel low-level debugging port (Use STM32MP1 UART4 for low-level debug)
    (0x40010000)  Physical base address of debug UART
```

- Update value for **CONFIG_DEBUG_UART_VIRT**, to define the associated virtual address to be used

```
Symbol: DEBUG_UART_VIRT [=0xFE010000]
Location:
  Kernel hacking --->
    [*] Kernel low-level debugging port (Use STM32MP1 UART4 for low-level debug)
    (0xFE010000)  Virtual base address of debug UART
```

Following rules to be respected for defining the virtual address:

- The 20 low weight bits (21 in case LPAE is enabled) must be kept in order to align region size of 1MB (2MB in LPAE is enabled).
- It must be mapped at the upper address of the vmalloc area, in order to not be overwritten by kernel which is stating from lower addresses: i.e here we select 0xFE0xxxxx

```
CONFIG_DEBUG_UART_PHYS: 0x40010000 /* UART4 */
CONFIG_DEBUG_UART_VIRT: 0xFE010000
```

- Please find below table for USART/UART of STMP32MP1:

Name	Physical base address	Virtual base address
USART 1	5c000000	**FE0**00000
USART 2	4000e000	**FE0**0e000
USART 3	4000f000	**FE0**0f000
UART4	40010000	**FE0**10000
UART5	40011000	**FE0**11000
USART 6	44003000	**FE0**03000
UART7	40018000	**FE0**18000
UART8	40019000	**FE0**19000

Note that the UART port used for console must be aligned for all components of the boot chain: FSBL(TF-A), SSBL-U-Boot) and Linux kernel

Especially because the Linux kernel do not configure all the setting registers for the UART port, as this is done by SSBL (U-Boot - How to debug)
See also TF-A - How to debug for FSBL changes)

- Get trace

Earlyprintk traces are pushed automatically to the serial console defined as seen previously, and also added to the kernel ring log buffer.

7 dmesg command

As reference, please see man page[6].

The Kernel ring buffer can be displayed using *dmesg* command. It will display on the console all the content of the ring buffer.

- It is possible to filter messages following the loglevels:

```
Board $> dmesg -n <loglevel>
```

In that case, only messages with a value lower (**not lower equal**) than the console_loglevel will be printed.

Here, *<loglevel>* can be a numeric value, but also a string:

```
Supported log levels (priorities):
  emerg (0)
  alert (1)
   crit (2)
    err (3)
   warn (4)
 notice (5)
   info (6)
  debug (7)
```

As example:

```
Board $> dmesg -n 8          # Temporary change loglevel to display messages up to debug
level
or
Board $> dmesg -n debug
```

In that case, every kernel messages will appear on your console, as **all priority higher than 8 (lower loglevel values)** will be displayed.

- It is possible to clear the dmesg buffer

```
Board $> dmesg -c         # Display the full content of dmesg ring buffer, and then clear
it
Board $> dmesg -C         # Clear the dmesg ring buffer
```

8 /var/log/messages file system entry

An other way to display the content of the Linux kernel log is to look at the content of the file /var/log/messages.

It contains general system activity messages from the start-up. It also provides useful information about origin of the message, and log level.

9 Dynamic debug message

These messages are using the loglevel 7 (KERN_DEBUG).

Please see How to use the kernel dynamic debug article.

10 References

- https://www.kernel.org/doc/html/latest/admin-guide/serial-console.html
- https://elinux.org/Debugging_by_printing
- https://www.kernel.org/doc/Documentation/printk-formats.txt
- https://www.kernel.org/doc/html/latest/admin-guide/kernel-parameters.html
- https://elinux.org/Debugging_by_printing#Log_Levels
- http://man7.org/linux/man-pages/man1/dmesg.1.html

SD memory card (https://www.sdcard.org)

former spelling for e•MMC ('e' in italic)

System File System (See https://en.wikipedia.org/wiki/Sysfs for more details)

Low layer of STM32Cube

Universal Asynchronous Receiver/Transmitter

Universal Synchronous/Asynchronous Receiver/Transmitter

First Stage Boot Loader

Trusted Firmware for Arm® Cortex®-A

Second Stage Boot Loader

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)