How to use the kernel dynamic debug

Contents

1 Introduction 3
2 More technical information 4
3 Examples 5
4 Synchronous tracing on the console 6
5 Debug messages during boot process 7
6 References 8

1 Introduction

As prerequisite to reading this article, please refer to the Dmesg and Linux kernel log page.

"Dynamic debug is designed to allow you to dynamically enable/disable kernel code to obtain additional kernel information. Currently, if **CONFIG_DYNAMIC_DEBUG** is set, all **pr_debug()/dev_dbg()** calls can be dynamically enabled per-callsite." extracted from the Linux kernel documentation[1].

The related debugfs entry is usually:

```
/sys/kernel/debug/dynamic_debug/control
```

Note that the verbose **dev_vdbg()** calls cannot be dynamically activated.

When the dynamic debug traces are activated, the trace results are printed in **dmesg (or /proc/kmsg)**, and in the console if console loglevel is set to 8.

2 More technical information

The dynamic debug trace configuration is done through a **control** file in the **debugfs** filesystem: *<debugfs>/dynamic_debug /control*

The command includes keywords and flag elements (for details see the Linux kernel documentation[1]).

- Keywords

Possible keywords are:

```
func : function name
file : source filename
module : module name
format : format string
line : line number (including ranges of line numbers)
```

The colored keywords above are illustrated by examples in the next chapter.

- Flags

The flag specification comprises a change operation followed by one or more flag characters. The change operation is one of the characters:

```
- : remove the given flags
+ : add the given flags
= : set the flags to the given flags
```

Possible flags are:

```
f : Include the function name in the printed message
l : Include line number in the printed message
m : Include module name in the printed message
p : Causes a printk() message to be emitted to dmesg
t : Include thread ID in messages not generated from interrupt context
```

3 Examples

- Track all dev_*dbg/pr_debug() in a **file** (you can add several files if necessary):

```
Board $> mount -t debugfs none /sys/kernel/debug
Board $> echo "file stm32-adc.c +p" > /sys/kernel/debug/dynamic_debug/control
```

Note that just the file name or full file path can be given, here *stm32-adc.c* or *drivers/iio/adc/stm32-adc.c*

- Track only one **line** with dev_dbg() in a **file** (you can add several files and several lines if necessary, please use the last line number of the function call):

```
Board $> echo "file stm32-adc.c line 1438 +p" > /sys/kernel/debug/dynamic_debug/control
```

- For an entire "**module** (module means ~.ko, so not applicable for a statically linked driver)":

```
Board $> echo "module cfg80211 +p" > /sys/kernel/debug/dynamic_debug/control
```

- If you want to list all available traces *(warning: it is a long file so you may need to use "tee" or another solution to save it)*:

```
Board $> cat /sys/kernel/debug/dynamic_debug/control | tee /tmp/dynamic_log.log
```

- For instance, if you are looking for a particular **file** to find a particular **line**:

```
Board $> cat /sys/kernel/debug/dynamic_debug/control | grep adc
drivers/iio/adc/stm32-adc.c:1515 [stm32_adc]stm32_adc_conf_scan_seq =p "%s chan %d to %s%
d\012"
drivers/iio/adc/stm32-adc.c:1438 [stm32_adc]stm32_adc_awd_set =p "%s chan%d htr:%d ltr:%
d\012"
drivers/iio/adc/stm32-adc.c:2182 [stm32_adc]stm32_adc_dma_start =p "%s size=%d watermark=%
d\012"
drivers/iio/adc/stm32-adc.c:2304 [stm32_adc]stm32_adc_trigger_handler =p "%s bufi=%d\012"
drivers/iio/adc/stm32-adc.c:2443 [stm32_adc]stm32_adc_chan_of_init =p "Configured to use
injected\012"
drivers/iio/adc/stm32-adc.c:2364 [stm32_adc]stm32_adc_of_get_resolution =p "Using %u bits
resolution\012"
```

- Multiple commands can be written together, separated by ';' or '\n'.

```
Board $> echo "file stm32-adc.c +p; file stm32-adc-core.c +p" > /sys/kernel/debug
/dynamic_debug/control
```

- A another method is to use a wildcard. The match rule supports * (matches zero or more characters) and ? (matches exactly one character). For example, you can match all USB drivers:

```
Board $> echo "file drivers/usb/* +p" > /sys/kernel/debug/dynamic_debug/control
```

4 Synchronous tracing on the console

In the case of a crash, or impossibility to call dmesg, it is sometimes useful to have traces synchronously emitted on the console.
Only error, warning and informational traces are emitted synchronously on the console (that is, loglevel=5), so if you need to see the lower level traces too, you need to change the console loglevel to "8".

```
<enable the conditional traces>
Board $> echo 8 > /proc/sys/kernel/printk
or
Board $> dmesg -n 8
or
Board $> dmesg -n debug
```

Please follow this article to get a serial console for the target: How to get Terminal

As all traces are now synchronously emitted, real-time is affected

If you want to return to the default console log level, you have to get this default value from the procfs entry */proc/sys/kernel/printk*:

```
Board $> cat /proc/sys/kernel/printk
8       4       1       7
Board $> dmesg -n 7
Board $> cat /proc/sys/kernel/printk
7       4       1       7
```

5 Debug messages during boot process

In order to activate debug messages during the boot process, even before userspace and debugfs exist, use the kernel's command-line parameter: **dyndbg**

For instance, the kernel *bootargs* can be modified in the following ways:

- Mount a boot partition from the Linux kernel console, and then update the extlinux.conf file using the vi editor (see man page [2], or introduction page[3]). For example:

```
Board $> mount /dev/mmcblk0p4 /boot
Board $> vi /boot/mmc0_stm32mp157c-ev1_extlinux/extlinux.conf
```

or

- Edit the extlinux.conf file by using UMS (USB Mass Storage): see How to use USB mass storage in U-Boot for details.

To mount partitions (mmc 0:microSD card / mmc 1: eMMC):

- Press any key to stop at U-Boot execution when booting the board.

```
Board $> ...
Board $> Hit any key to stop autoboot:  0
Board $> STM32MP>
```

- Then

```
STM32MP> ums 0 mmc 0
```

- Check for the boot partition mounted on your host PC (/media/*$USER*/bootfs)
- Edit the extlinux file corresponding to your setup (/media/*$USER*/bootfs/mmc0_extlinux/stm32mp157f-dk2_extlinux.conf

- Update the kernel command line, adding the dyndbg parameter:

```
root=PARTUUID=e91c4e10-16e6-4c0e-bd0e-77becf4a3582 rootwait rw console=ttySTM0,115200 dynd
bg="file drivers/usb/core/hub.c +p"
```

Save and quit file update, and then reboot the board.

Note: to display these debug messages in the console, in addition to the dmesg, add **loglevel=8** in the kernel command line.

- Reboot the board and check for a kernel command-line, and that debug messages are present in the dmesg output

6 References

- [1.01.1] Documentation/admin-guide/dynamic-debug-howto.rst
- http://ex-vi.sourceforge.net/vi.html
- http://ex-vi.sourceforge.net/viin/paper.html

- Useful external links

Document link	Document Type	Description
The dynamic debugging interface (lwn.net)	User guide	http://lwn.net
Documentation/dynamic-debug-howto.txt (lwn.txt)	User guide	http://lwn.net
Dynamic debug howto (kernel.org)	Standard	http://www.kernel.org

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

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

User-space Mode Setting

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

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