

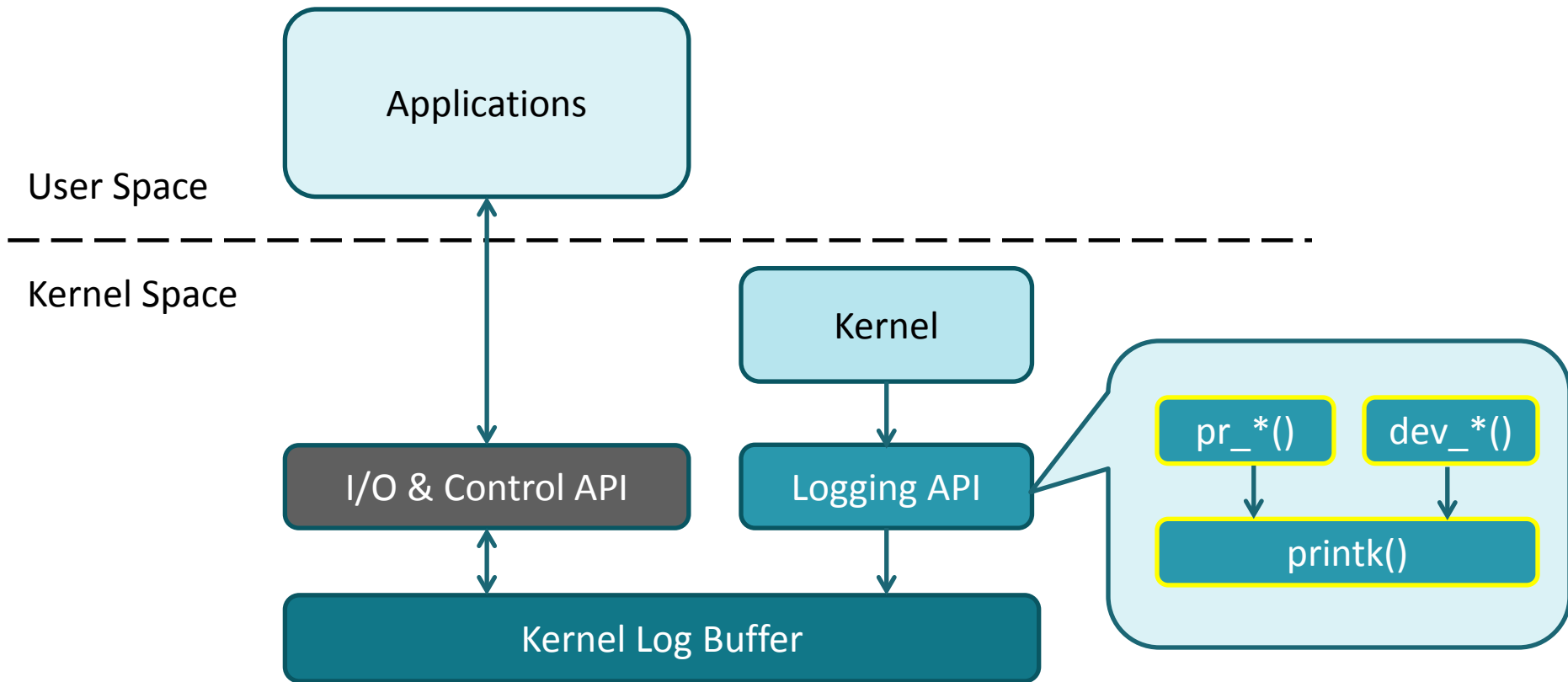
Debugging Embedded Linux Systems: printk and Variations

Debugging Embedded Linux Training Series [Part 3]

Debugging Embedded Linux Training Series

- Part 1: Linux/Kernel Overview
- Part 2: Kernel Logging System Overview
- **Part 3: printk and Variations**
- Part 4: Dynamic Debug
- Part 5: Locate Device Driver Source Code
- Part 6: Understand Kernel Oops Logs

Kernel logging system architecture



Agenda

- printk()
- Kernel Log Message Levels
- pr_*(), dev_*()
- pr_debug(), dev_dbg()
- dev_vdbg()
- Enable Debug Log Messages
- Case Study

printk()

- Format and print data/message into kernel log buffer
- Foundation of Logging API
- Works in a similar way as `printf()` in user space
- Example:

```
printk(KERN_ALERT "DBG: passed %s %d\n", __FUNCTION__, __LINE__)
```

- Kernel-specific conversion specifiers:

Ex.: “%pS” - print symbol name with offset: `versatile_init+0x0/0x110`

Log message level

```
printk(KERN_ALERT "DBG: passed %s %d\n", __FUNCTION__, __LINE__)
```

Log message levels

Macro*	Level
KERN_EMERG	"<0>"
KERN_ALERT	"<1>"
KERN_CRIT	"<2>"
KERN_ERR	"<3>"
KERN_WARNING	"<4>"
KERN_NOTICE	"<5>"
KERN_INFO	"<6>"
KERN_DEBUG	"<7>"

*Defined in `include/linux/kern_levels.h`

Used by the kernel to determine the importance of a message

```
printk(KERN_ALERT "DBG: passed %s %d\n",  
        __FUNCTION__, __LINE__);
```

<1>DBG: passed

Filtering log messages

- Userspace tool can filter messages based on the message level.
- Examples:

```
dmesg -n 5
```

Set console logging filter to KERN_WARNING or more severe.

```
dmesg -l warn
```

Only print the logs of KERN_WARNING in the kernel ring buffer.

pr_*(), dev_*()

- pr_emerg, pr_alert, pr_crit, pr_err, pr_warning, pr_notice, pr_info
pr_debug
 - Macro definitions for the printk calls with respective message level
 - Should be used in **kernel (not device drivers)** instead of calling printk() directly
- dev_emerg, dev_alert, dev_crit, dev_err, dev_warn, dev_notice, dev_info
dev_dbg
 - Macro definitions for the printk calls with respective message level
 - Should be used in **device drivers** instead of calling printk() directly

Logging API

Level	Base	For Kernel	For Drivers
0	printk(KERN_EMERG...)	pr_emerg()	dev_emerg()
1	printk(KERN_ALERT...)	pr_alert()	dev_alert()
2	printk(KERN_CRIT...)	pr_crit()	dev_crit()
3	printk(KERN_ERR...)	pr_err()	dev_err()
4	printk(KERN_WARNING...)	pr_warning()	dev_warn()
5	printk(KERN_NOTICE...)	pr_notice()	dev_notice()
6	printk(KERN_INFO...)	pr_info()	dev_info()
7	printk(KERN_DEBUG...)	pr_debug()	dev_dbg(), dev_vdbg()

pr_debug(), dev_dbg()

- Macros defined with the lowest message level 7 - *KERN_DEBUG*
- Used for printing *debug* messages in kernel or device drivers, respectively.
- Only enabled when DEBUG compiler macro is defined.
- Kernel production build would have DEBUG macro undefined.

```
#ifndef DEBUG
    #define pr_debug(...)      printk(KERN_DEBUG ...)
    #define dev_dbg(...)      dev_printk(KERN_DEBUG ...)
#else
    #define pr_debug(...)      ({} )
    #define dev_dbg(...)      ({} )
#endif
```

dev_vdbg()

- Print verbose debug messages
- Macro definition for dev_dbg()
- Controlled by VERBOSE_DEBUG compiler macro

```
#ifdef VERBOSE_DEBUG
    #define dev_vdbg    dev_dbg
#else
    #define dev_vdbg    ({} )
#endif
```

Enable DEBUG/VERBOSE_DEBUG macro

- Three options to enable DEBUG macro:
 1. Kernel config option
 2. Add -DDEBUG in Makefile ccflags
 3. Add “#define DEBUG” in *.c
Above **any** #include <...> lines
- Three options to enable VERBOSE_DEBUG macro:
 1. Kernel config option
 2. Add -DVERBOSE_DEBUG in Makefile ccflags
 3. Add “#define VERBOSE_DEBUG” in *.c
Above **any** #include <...> lines

Case study: Locate Kconfig option for debug/vdebug

- Procedure:
 1. Find the kernel config option referred for DDBUG/DVERBOSE_DEBUG in the Makefile.
 2. (Optional) Find the corresponding config option in Kconfig.
 3. Use the search function to locate the config option in kernel menuconfig.
- Example: Enable DDBUG in GPIO drivers.

Case study: Locate debug Kconfig option in GPIO drivers (1)

./drivers/gpio/Makefile

```
1 #generic gpio support: platform drivers, dedicated expander chips, etc
2
3 ccflags-$(CONFIG_DEBUG_GPIO) += -DDEBUG
4
5 obj-$(CONFIG_GPIO_DEVRES) += devres.o
6 obj-$(CONFIG_GPIOLIB) += gpiolib.o
7 obj-$(CONFIG_GPIOLIB) += gpiolib-legacy.o
8 obj-$(CONFIG_OF_GPIO) += gpiolib-of.o
9 obj-$(CONFIG_GPIO_SYSFS) += gpiolib-sysfs.o
10 obj-$(CONFIG_GPIO ACPI) += gpiolib-acpi.o
```

Case study: Locate debug Kconfig option in GPIO drivers (2)

```
$ find . -name Kconfig -exec grep -Hn '\<config DEBUG_GPIO\>' {} \;
```


Case study: Locate debug Kconfig option in GPIO drivers (2)

```
$ find . -name Kconfig -exec grep -Hn '\<config DEBUG_GPIO\>' {} \;  
./drivers/gpio/Kconfig:63:config DEBUG_GPIO
```

Case study: Locate debug Kconfig option in GPIO drivers (2)

```
$ find . -name Kconfig -exec grep -Hn '<config DEBUG_GPIO>' {} \;  
./drivers/gpio/Kconfig:63:config DEBUG_GPIO
```

```
59 config GPIOLIB_IRQCHIP  
60     select IRQ_DOMAIN  
61     bool  
62  
63 config DEBUG_GPIO  
64     bool "Debug GPIO calls"  
65     depends on DEBUG_KERNEL  
66     help  
67     Say Y here to add some extra checks and diagnostics to GPIO calls.
```

Case study: Locate debug Kconfig option in GPIO drivers (3)

\$ make menuconfig

/DEBUG_GPIO

```
.config - Linux/arm 4.4.44 Kernel Configuration
-----
Search Configuration Parameter
Enter (sub)string or regexp to search for (with or without "CONFIG_")
DEBUG_GPIO
-----
< Ok > < Help >
```

Case study: Locate debug Kconfig option in GPIO drivers (3)

```
.config - Linux/arm 4.4.44 Kernel Configuration
> Search (DEBUG_GPIO) -----
                                Search Results
Symbol: DEBUG_GPIO [=y]
Type : boolean
Prompt: Debug GPIO calls
Location:
  -> Device Drivers
(1)  -> GPIO Support (GPIOLIB [=y])
Defined at drivers/gpio/Kconfig:63
Depends on: GPIOLIB [=y] && DEBUG_KERNEL [=y]

                                (100%)
< Exit >
```

Case study: Locate debug Kconfig option in GPIO drivers (3)

```
.config - Linux/arm 4.4.44 Kernel Configuration
> Device Drivers > GPIO Support -----
GPIO Support
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus
---->). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M>
modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search.
Legend: [*] built-in [ ] excluded <M> module < > module capable
-----
--- GPIO Support
[*] Debug GPIO calls
[*] /sys/class/gpio/... (sysfs interface)
Memory mapped GPIO drivers --->
I2C GPIO expanders --->
MFD GPIO expanders --->
SPI or I2C GPIO expanders --->
USB GPIO expanders ----
-----
<Select> < Exit > < Help > < Save > < Load >
```

Case study: Locate debug Kconfig option in GPIO drivers (3)

```
.config - Linux/arm 4.4.44 Kernel Configuration
> Device Drivers > GPIO Support -----
                                Debug GPIO calls +
CONFIG_DEBUG_GPIO:
Say Y here to add some extra checks and diagnostics to GPIO calls.
These checks help ensure that GPIOs have been properly initialized
before they are used, and that sleeping calls are not made from
non-sleeping contexts. They can make bitbanged serial protocols
slower. The diagnostics help catch the type of setup errors
that are most common when setting up new platforms or boards.

Symbol: DEBUG_GPIO [=y]
Type : boolean
Prompt: Debug GPIO calls
Location:
  -> Device Drivers
    -> GPIO Support (GPIOLIB [=y])
                                ( 86%) +
                                < Exit > +
```

Summary

- `printk()` is the foundation of the kernel logging API. But it should not be used directly in kernel modules or device drivers.
- `pr_*()` are macros wrapping `printk()` for kernel module logging.
- `dev_*()` are macros wrapping `printk()` for device driver logging.
- `pr_debug()`, `dev_dbg()/dev_vdbg()` are for logging debug messages.
 - Disabled in production build
 - Controlled by `DEBUG` or `VERBOSE DEBUG` compiler macro

For more information

- Processor SDK Training Series:
<http://training.ti.com/processor-sdk-training-series>
- Debugging Embedded Linux Training Series:
<http://training.ti.com/debug-embedded-linux-training-series>
- Processor SDK Linux Getting Started Guide:
http://processors.wiki.ti.com/index.php/Processor_SDK_Linux_Getting_Started_Guide
- Download Processor SDK Linux for Embedded Processors:
<http://www.ti.com/processorsdk>
- For questions about this training, refer to the E2E Embedded Linux Community Forum: <http://e2e.ti.com/support/embedded/linux/f/354>



©Copyright 2017 Texas Instruments Incorporated. All rights reserved.

This material is provided strictly “as-is,” for informational purposes only, and without any warranty.
Use of this material is subject to TI’s **Terms of Use**, viewable at TI.com