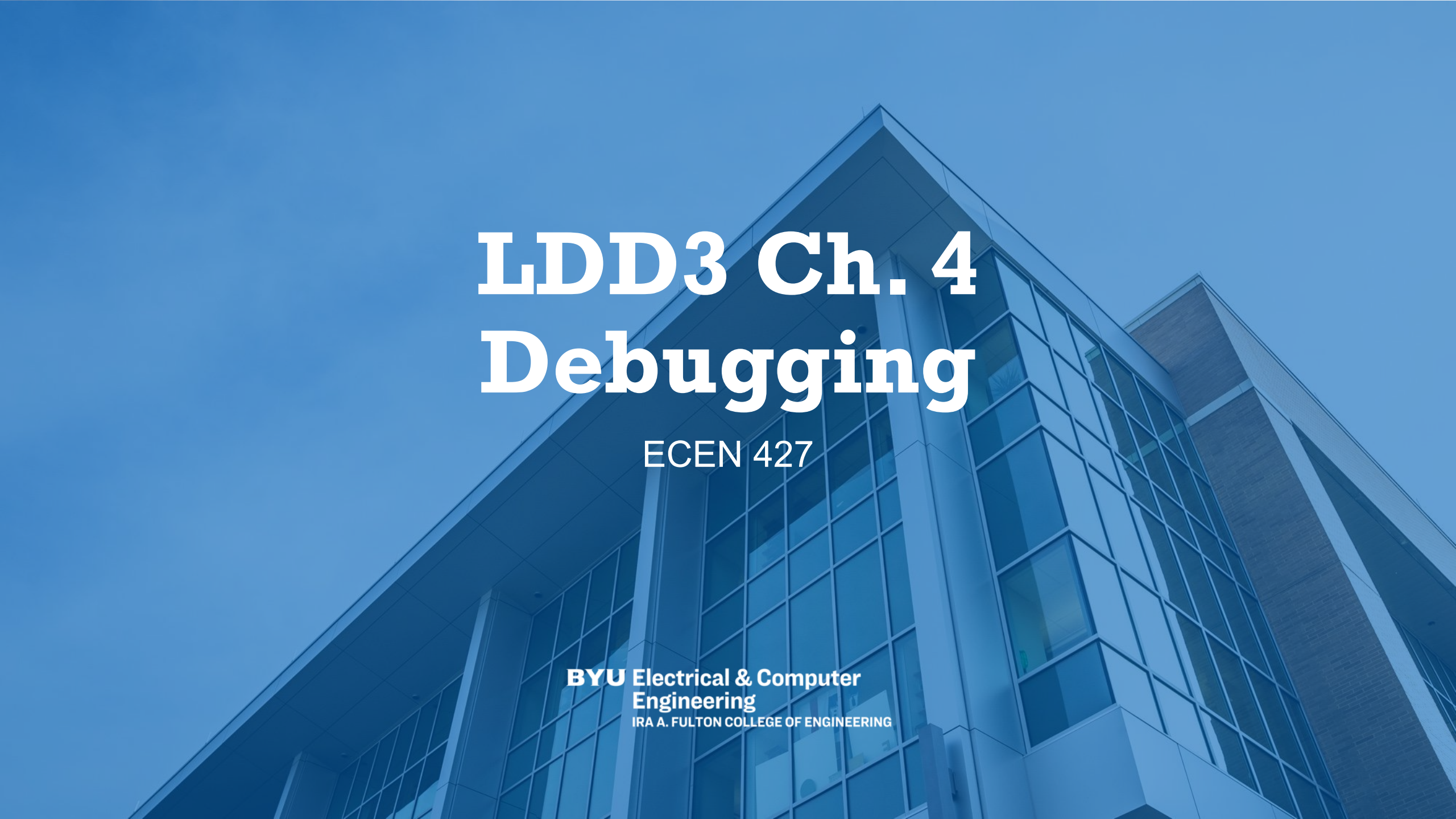




LDD3 Ch. 4

Debugging

ECEN 427

BYU Electrical & Computer
Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

One of the differences is that `printk` lets you classify messages according to their severity by associating different log levels, or priorities, with the messages. You usually indicate the log level with a macro.

There are eight possible loglevel strings, defined in the header `<linux/kernel.h>`; we list them in order of decreasing severity:

`KERN_EMERG`

Used for emergency messages, usually those that precede a crash.

`KERN_ALERT`

A situation requiring immediate action.

`KERN_CRIT`

Critical conditions, often related to serious hardware or software failures.

`KERN_ERR`

Used to report error conditions; device drivers often use `KERN_ERR` to report hardware difficulties.

`KERN_WARNING`

Warnings about problematic situations that do not, in themselves, create serious problems with the system.

`KERN_NOTICE`

Situations that are normal, but still worthy of note. A number of security-related conditions are reported at this level.

`KERN_INFO`

Informational messages. Many drivers print information about the hardware they find at startup time at this level.

`KERN_DEBUG`

Used for debugging messages.

```
printk(KERN_DEBUG "Here I am: %s:%i\n", __FILE__, __LINE__);  
printk(KERN_CRIT "I'm trashed; giving up on %p\n", ptr);
```

Printing Pointers

- Sometimes students use the “%p” format specifier to print their pointers.
- Since kernel 4.15, plain %p prints a hashed value so kernel addresses don't leak to user space
 - Example 00000000a3f1c2d9
 - The hash is stable within a boot, so it can still tell two pointers apart, but it is useless for checking an address against the memory map.

Printing pointers:

- %p prints a hashed value, not the real address. This is a security feature.
- %px prints the raw address (don't keep this in deployed code)
- %pa prints a phys_addr_t or resource_size_t. Pass a pointer to the variable: &res->start.
- %pR prints a whole struct resource, such as [mem 0x43c00000-0x43c0ffff].
- %pS prints the symbol name for a function pointer, such as my_isr+0x14/0x40.

Reference: <https://www.kernel.org/doc/html/v6.6/core-api/printk-formats.html#pointer-types>

Don't call printk directly!

<https://www.kernel.org/doc/html/v6.6/process/coding-style.html#printing-kernel-messages>

If you have a struct device*

- dev_err(), dev_warn(), dev_info(), and so forth
- https://elixir.bootlin.com/linux/v6.6/source/include/linux/dev_printk.h#L137

Otherwise, use:

- pr_notice(), pr_info(), pr_warn(), pr_err(), etc.
- <https://elixir.bootlin.com/linux/v6.6/source/include/linux/printk.h#L467>

These functions provide nicer formatting and more context information in the dmesg log.

- The `printk` function writes messages into a circular buffer that is `CONFIG_LOG_BUF_SHIFT` bytes long: a value from 4 KB to 32 MB chosen while configuring the kernel.
- The **`dmesg`** command can be used to look at the content of the buffer without flushing it; actually, the command returns to stdout the whole content of the buffer, whether or not it has already been read.
- If the circular buffer fills up, `printk` **wraps around** and starts adding new data to the beginning of the buffer, overwriting the oldest data

Turning Debug Prints Off (and On)

- LDD3's answer is a hand-rolled macro (scull's PDEBUG) that compiles away when a flag is off.
- The kernel now has this built in: `pr_debug()` and `dev_dbg()`.
 - Compiled out entirely unless `DEBUG` is defined.
 - All other levels (`pr_info`, `dev_err`, ...) always print.
- Turn them on for a whole module from the Makefile, or for one file in the source:

```
ccflags-y += -DDEBUG
```

```
#define DEBUG                /* must come before any #include */  
#include <linux/module.h>
```

If you are not careful, you can find yourself generating thousands of messages with `printk`, overwhelming the console and, possibly, overflowing the system log file....When using a slow console device (e.g., a serial port), an excessive message rate can also slow down the system or just make it unresponsive.

- What should you do?
 - In many cases, the best behavior is to set a flag saying, “I have already complained about this,” and not print any further messages once the flag gets set

```
if (!my_flag) {  
    printk(...);  
    my_flag = 1;  
}
```

Old method

```
pr_warn_once("message");  
dev_warn_once("message");
```

New method

- In others, though, there are reasons to emit an occasional “the device is still broken” notice. The kernel has provided a function that can be helpful in such cases:

```
if (printk_ratelimit())  
    printk(KERN_NOTICE "The printer is still on fire\n");
```

```
pr_notice_ratelimited("The printer is still on fire\n");
```

Oops Messages

- “Most bugs show themselves in NULL pointer dereferences or by the use of other incorrect pointer values. The usual outcome of such bugs is an oops message.”

```
Unable to handle kernel NULL pointer dereference at virtual address 00000000
printing eip:
d083a064
Oops: 0002 [#1]
SMP
CPU: 0
EIP: 0060:[<d083a064>] Not tainted
EFLAGS: 00010246 (2.6.6)
EIP is at faulty_write+0x4/0x10 [faulty]
eax: 00000000 ebx: 00000000 ecx: 00000000 edx: 00000000
esi: cf8b2460 edi: cf8b2480 ebp: 00000005 esp: c31c5f74
ds: 007b es: 007b ss: 0068
Process bash (pid: 2086, threadinfo=c31c4000 task=cfa0a6c0)
Stack: c0150558 cf8b2460 080e9408 00000005 cf8b2480 00000000 cf8b2460 cf8b2460
        ffffffff7 080e9408 c31c4000 c0150682 cf8b2460 080e9408 00000005 cf8b2480
        00000000 00000001 00000005 c0103f8f 00000001 080e9408 00000005 00000005
Call Trace:
[<c0150558>] vfs_write+0xb8/0x130
[<c0150682>] sys_write+0x42/0x70
[<c0103f8f>] syscall_call+0x7/0xb

Code: 89 15 00 00 00 00 c3 90 8d 74 26 00 83 ec 0c b8 00 a6 83 d0
```