LINUX KERNEL HACKING:

A CRASH COURSE

bitly

HELLO, MY NAME IS GEORGI

@GeorgiCodes

bitly

AGENDA

History of Linux
Kernel architecture
How to build a kernel module
How to build a device driver

bitly

debian

ubuntu

CentOS

CoreOS

kernel v4.0

8000 developers

800 companies

15 million lines of code

10 patches 7/365

2-3 months new release

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) …

– Linus Torvalds 1991

bitly

TIMELINE

1991: Linus Torvalds creates kernel prototype

1994: Linux version 1.0.0 released

Mid 1990s: Lots of Linux distributions

1996: Tux was born

1994-1997: Linux gets mainstream press

1998: Support from Google, Oracle, Intel & Netscape

.....

2015: Linux kernel version 4.0 released

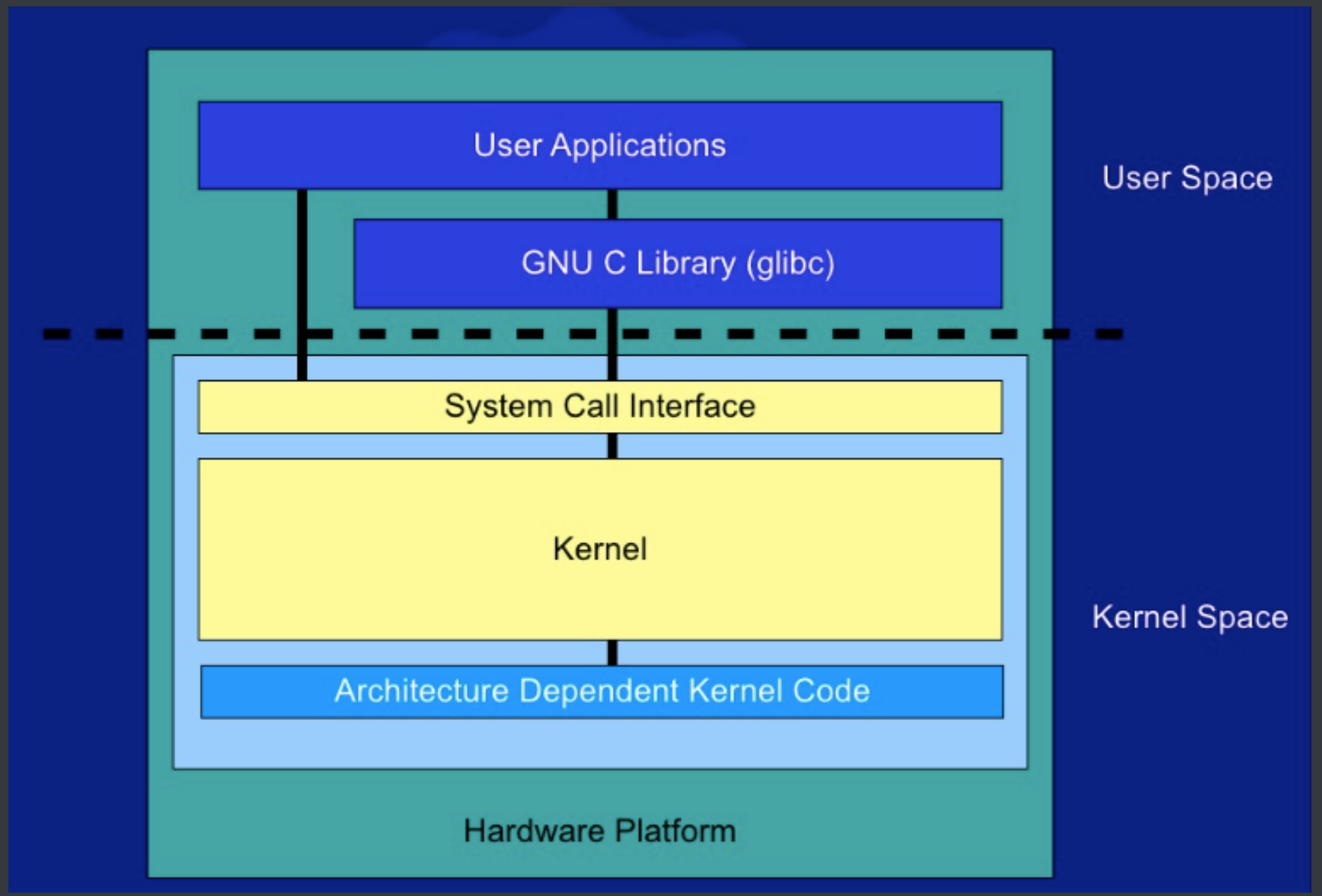
bitly

“… the kernel is a computer program that manages I/O (input/output) requests from software, and translates them into data processing instructions for the central processing unit and other electronic components of a computer.”

– Wikipedia

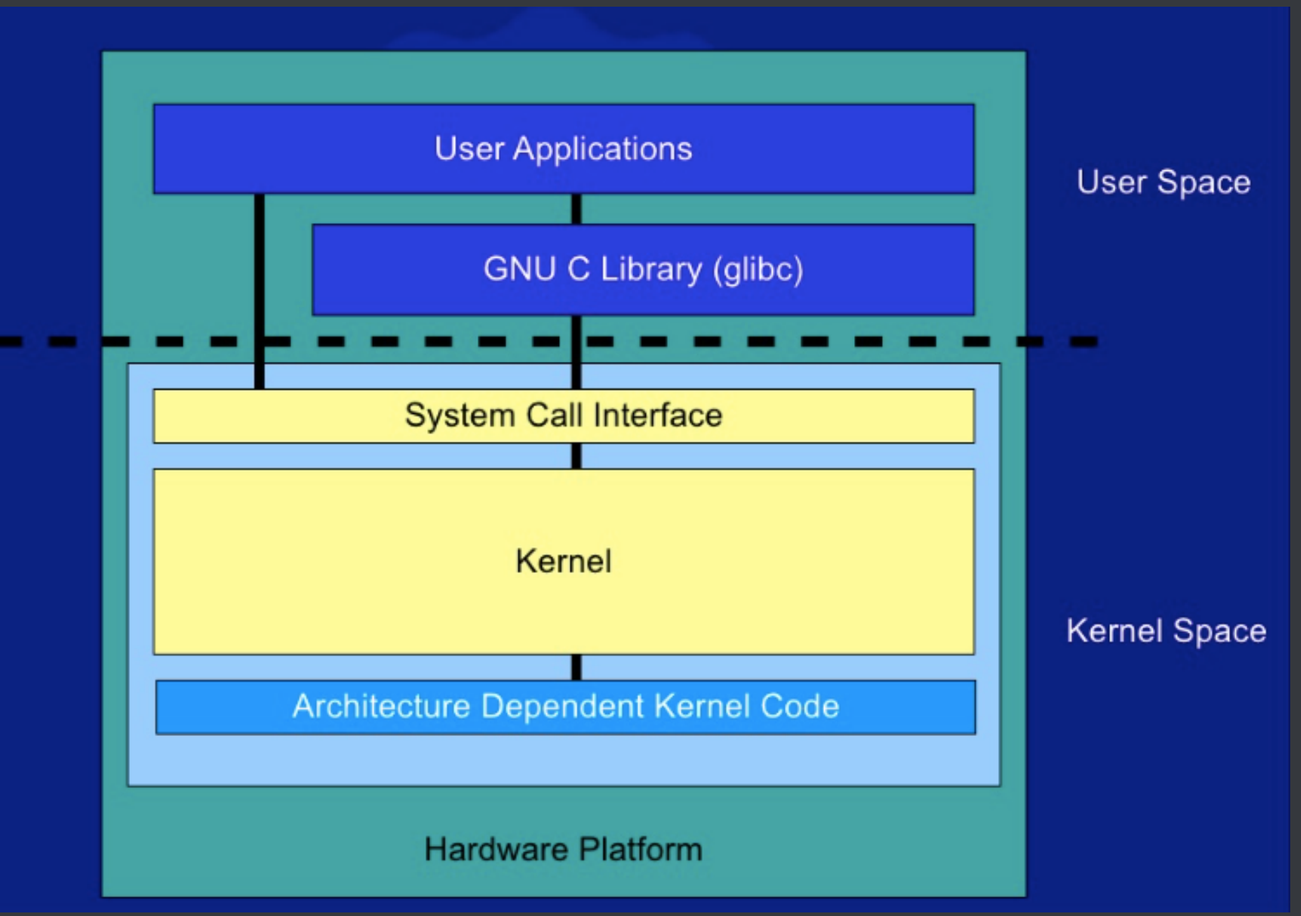
bitly

KERNEL ARCHITECTURE



bitly

KERNEL VS. USER SPACE



* User space restricts user programs so that they can't accidentally mess with the system.
* Kernel space is privileged and has full access to memory and resources.

bitly

WHO USES THE KERNEL?

The kernel provides a way for other programs to use the hardware via system calls.

* The kernel is not designed for direct human consumption (no UI).
* The kernel's users are other programs.

bitly

SYSTEM CALLS

* there are a few hundred sys calls with functions like `read()`, `write()`, `open()`

* When a system call is executed, the arguments are passed from user space to kernel space.

* A user process becomes a kernel process when it executes a system call.

bitly

```c
#include <fcntl.h>
int main()
{
    int fd, count; char buf[1000];
    fd=open("mydata", O_RDONLY);
    count = read(fd, buf, 1000);
    write(1, buf, count);
    close(fd);
}
```

bitly

WHERE DOES KERNEL CODE EXECUTE?

The kernel has two entry points:

1. When a running process/application that makes a system call.

2. Responding to a hardware interrupt
   - A key was pressed
   - A network packet just arrived
   - A time just ticked

bitly

/PROC

* window of communication between the kernel and user space

* dynamically generated files provide info on running system

bitly

WHAT ARE KERNEL MODULES?

* programs written in C built against the Linux kernel source tree
* run in kernel space
* core of kernel remains small where modules can be loaded and unloaded as required

To Build:

* need kernel source tree, gcc and make
* run the same version of kernel you built module with

bitly

```c
#include <linux/module.h>    // included for all kernel modules
#include <linux/kernel.h>    // included for KERN_DEUBG
#include <linux/init.h>      // included for __init and __exit macros

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Georgi");
MODULE_DESCRIPTION("A Simple Hello World module");

static int __init hello(void)
{
    printk(KERN_DEBUG ">>> Hello world! <<<\n");
    return 0;    // Non-zero return means that the module couldn't be loaded.
}

static void __exit goodbye(void)
{
    printk(KERN_DEBUG ">>> Goodbye world! <<<\n");
}

module_init(hello);
module_exit(goodbye);
```

BUILD YOUR KERNEL MODULE

```
ifneq ($(KERNELRELEASE),)
obj-m  := hello.o

else
    KDIR ?= /lib/modules/`uname -r`/build

default:
  $(MAKE) -C $(KDIR) M=$$PWD
endif
```

bitly

COMMANDS

Loading + Unloading

* `insmod hello.ko` - loads module
* `rmmod hello.ko` - unloads module

View kernel logs

* `dmesg`

Use modprobe to manage dependencies

* Copy `hello.ko` into `/lib/modules/$KERNEL_VERSION`
* `depmod`
* `modprobe hello` - loads module
* `modprobe -r hello` - unloads module
* `modinfo` - tells you info about the module

bitly

DEMO!

bitly

HOW IS KERNEL PROGRAMMING DIFFERENT?

1. kernel has no standard C headers and libraries
2. no memory protection!
3. a single big namespace
4. always multi-threaded

bitly

TYPES OF DEVICE DRIVERS

Char

* reads/writes character by character to the device
* operates in a blocking mode
* stream of bytes

Block

* reads/writes large amounts of data block by block.
* operates in a non-blocking mode

Network device

* Exchange data over network
* Understands packets and connections

USB device

bitly

LET'S BUILD A DEVICE DRIVER!

What will our char device driver do?

It will respond with *"Hello Code PaLOUsa!"* when its read from.

Steps

1. Write the code
2. Build and load our module
3. Create a device file

bitly

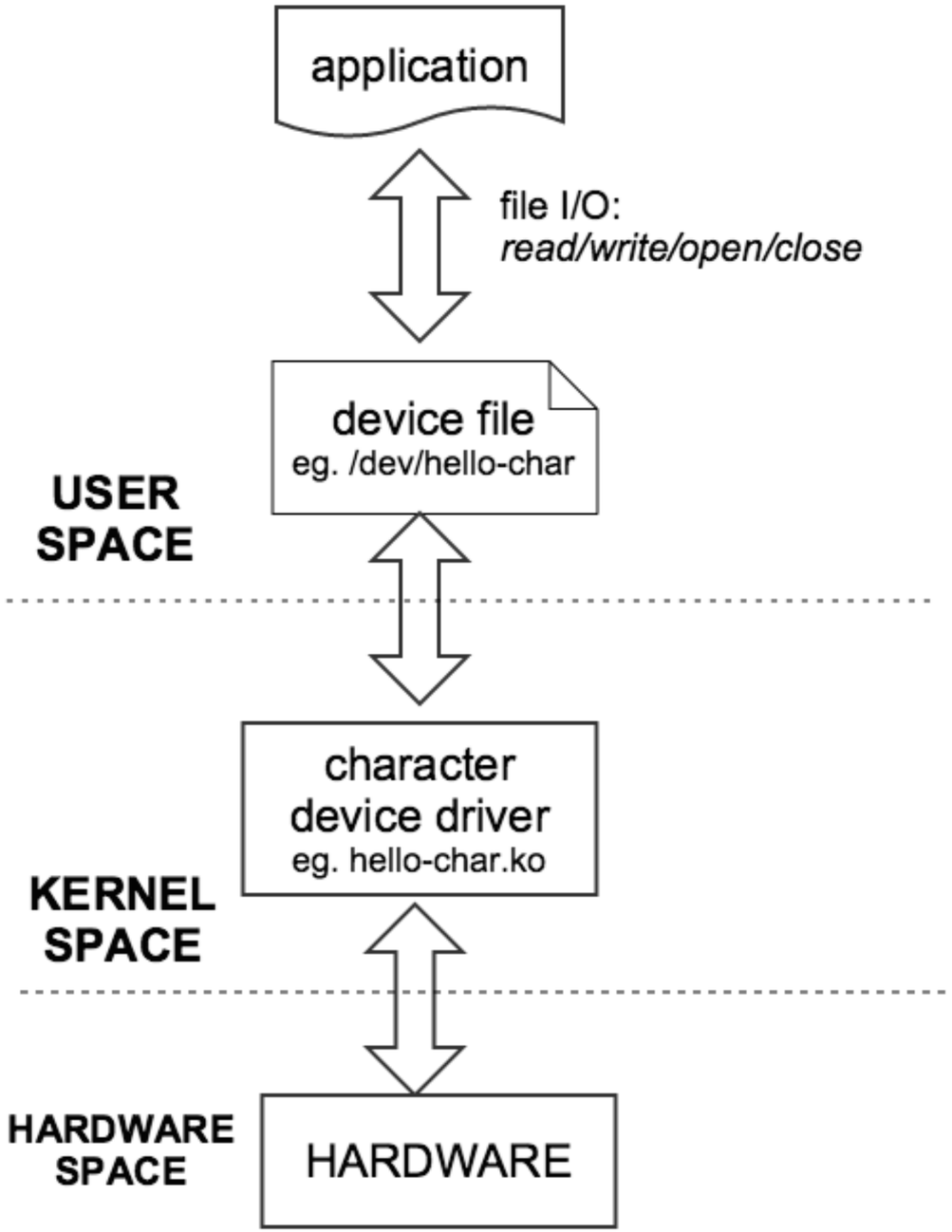
```
……

int init_module(void)
{
  major = register_chrdev(0, DEVICE_NAME, &fops);

  printk(KERN_INFO ">>> I was assigned major number %d. <<<\n", major);
  printk(KERN_INFO ">>> Run 'mknod /dev/%s c %d 0'. <<<\n", DEVICE_NAME, major);

  return 0;
}

void cleanup_module(void)
{
  unregister_chrdev(major, DEVICE_NAME);
}

……
```

bitly

```
static struct file_operations fops = {
  .read = read_dev,
  .write = write_dev,
  .open = open_dev,
  .release = close_dev
};
```

* interaction is through system calls: open(), close(), read(), write()
* “ops” struct pattern that allows you to fill in behaviors via callbacks

bitly



DEVICE FILES

* a device file is how a user program can access the physical device which lives all the way in kernel space.
* device files live in the `/dev` directory
* create device file:

```
mknod /dev/hello-char c 250 0
```

bitly

OUR DEVICE DRIVER IN ACTION!

Reading

```
cat /dev/hello-char
```

this will call the `read_dev()` function

bitly

DEMO!

bitly

LEVELLING UP YOUR LINUX SKILLS

* install native Linux and use it!
* configure and build your own kernel
* write your own kernel module and/or device driver
* do the Eudyptula Challenge

bitly

code + links + slides:

georgi.io/kernel-talk

BITLY.IS/HIRING

bitly