Linux Device Drivers

Lection 1

Roman Okhrimenko 2019

Questions to answer on this lection

1. What is OS kernel and how I can get one?
2. What are kernel functions in operating system?
3. What are Linux kernel considerations?
4. Which subsystems kernel consist of?
5. Monolithic vs microkernel?
6. What is a kernel module and how it is used?
7. Drivers are modules but not vice versa?
8. Different types of drivers
9. What does it mean ”Everything is a file”?
10. What are special purpose file systems and their use?

What is Kernel?

Kernel:

- It is a computer program
- it is a core application in all operating systems
- It boot/inits system and always stays in memory
- It handles low level system operations
- It handles memory management and distribution
- It handles I/O devices and peripherals
- It handles scheduling of processes and processor time quota
- Provides unified System Call interface for other processes

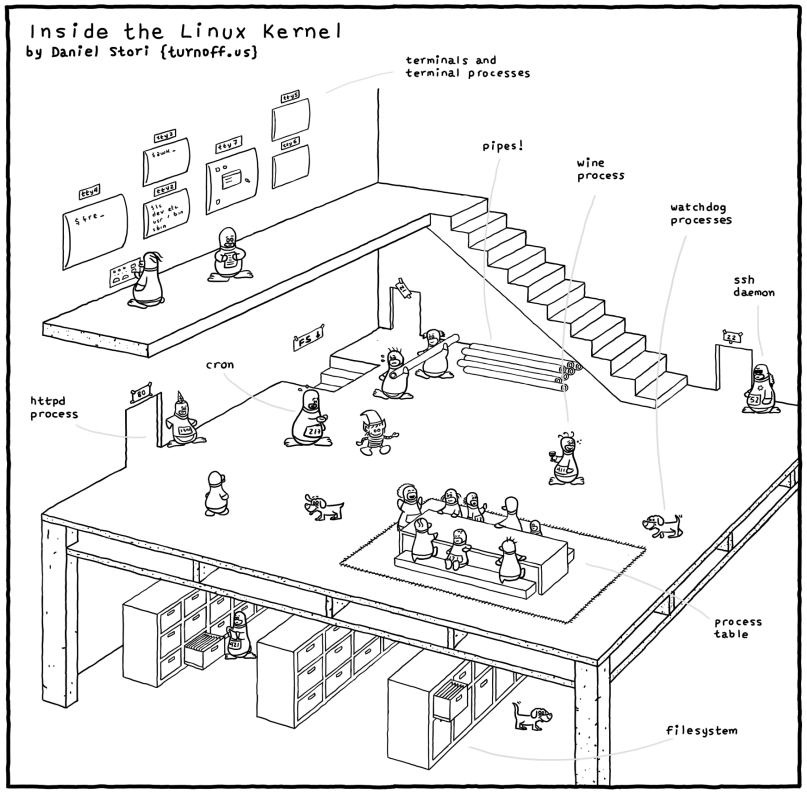
How I can get a Kernel?

1. Write your own - https://wiki.osdev.org/Getting_Started
2. Use one of existing
   - Linux - https://github.com/torvalds/linux
   - FreeBSD - https://github.com/freebsd/freebsd
   - uclinux - https://github.com/EmcraftSystems/linux-emcraft
   - Xv6 - https://pdos.csail.mit.edu/6.828/2019/xv6.html
3. There are microkernels also
   - http://www.microkernel.info/
4. And RTOS kernels
   - FreeRTOS - https://www.freertos.org/
   - Zephyr - https://www.zephyrproject.org/
   - Mbed-os - https://www.mbed.com/en/platform/mbed-os/

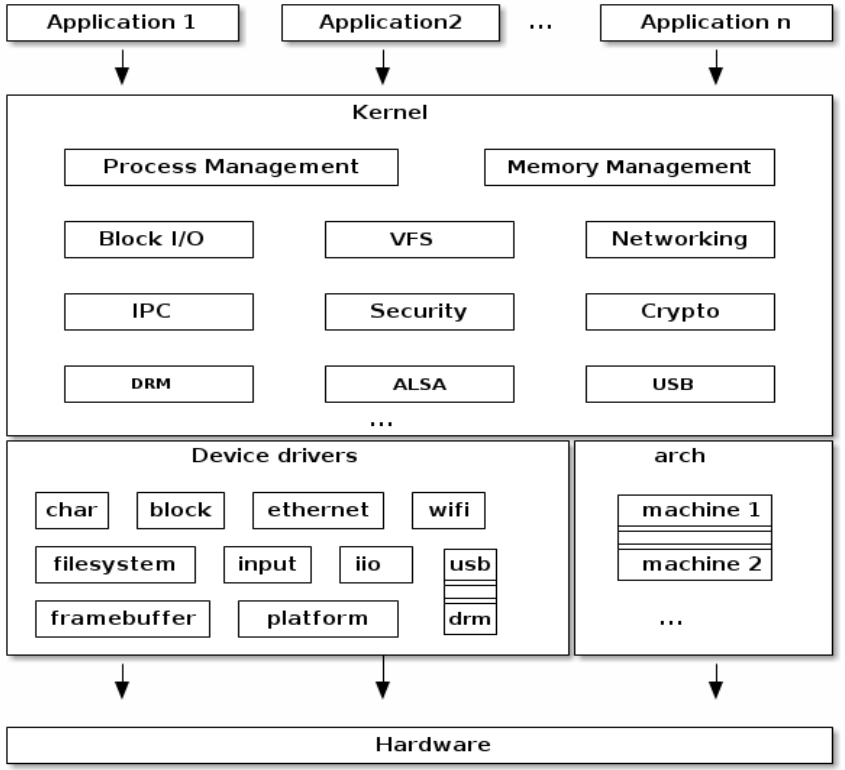
Linux Kernel features

- Written in C and assembler
- Portability and hardware support
- Runs on most architectures
- Scalability. Can run on super computers as well as on tiny devices (4 MB of RAM is enough)
- Compliance to standards and interoperability
- Exhaustive networking support
- Security
- It can’t hide its flaws. Its code is reviewed by many experts
- Stability and reliability
- Modularity. Can include only what a system needs even at run time
- Easy to dive in. You can learn from existing code. Many useful resources on the net

Linux Kernel features



http://turnoff.us/geek/inside-the-linux-kernel/



https://linux-kernel-labs.github.io/master/lectures/intro.html#monolithic-kernel

CYPRESS
EMBEDDED IN TOMORROW™

Linux Kernel considerations

- The kernel has to be standalone and can’t use user space code
- Floating point numbers in kernel code are never used
- No stable in-kernel API
  - The internal kernel API to implement kernel code can undergo changes between two releases
  - In-tree drivers are updated by the developer proposing the API change
  - An out-of-tree driver compiled for a given version may no longer compile or work
  - The kernel to user space API does not change
- No memory protection
- The kernel doesn’t try to recover from attemps to access illegal memory locations
- Fixed size stack (8 or 4 KB)
- No C library – no printf – printk instead

Linux Kernel subsystems

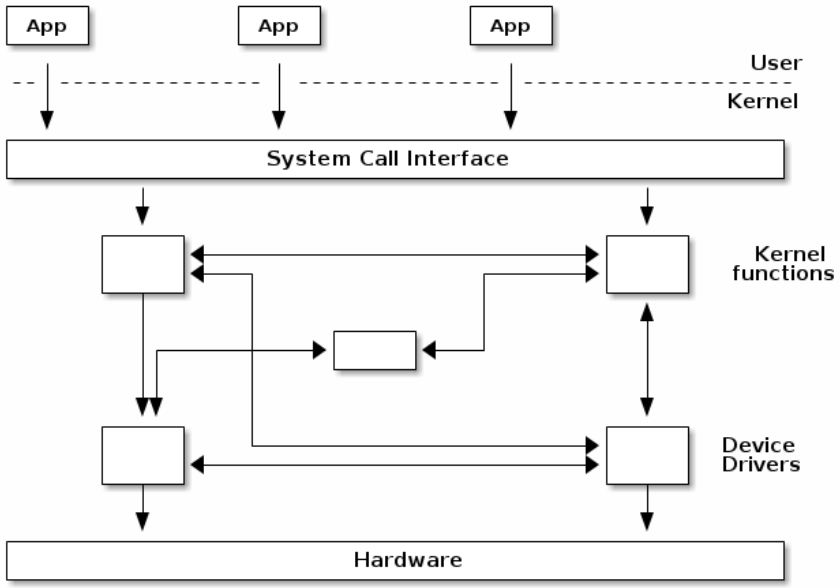
- **Scheduler** – share process time between lots of processes, implements SMP, ASMP, IPC and various mechanisms to support efficient load distribution in system
- **Memory management** (HIGHMEM, virtual memory) – defines memory spaces for kernel and user space processes by request, provides a mapping between process memory references and the machine's physical memory
- **Networking** – implements all the connectivity related protocols and hardware support
- **USB** – implements universal serial bus protocols and infrastructure
- **Virtual Filesystem –** implements software layer in the kernel that provides the filesystem interface to userspace programs. It also provides an abstraction within the kernel which allows different filesystem implementations to coexist
- **Hardware architecture** – implements platform specific code for kernel
- **Other** – power management, crypto, IPC, modules, init, interrupt handling

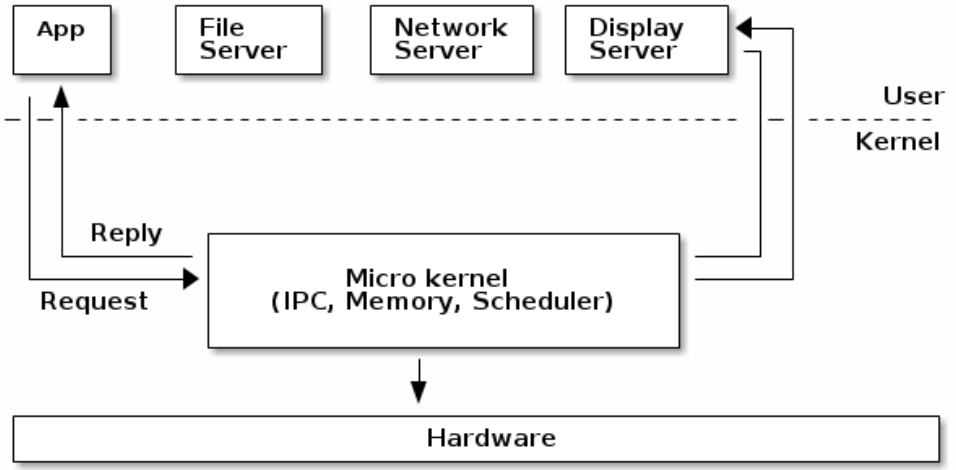
Monolithic vs Micro-kernel

- A **monolithic** kernel is one where there is no access protection between the various kernel subsystems and where public functions can be directly called between various subsystems.
- Most monolithic kernels do enforce a logical separation between subsystems especially between the core kernel and device drivers with relatively strict APIs (but not necessarily fixed in stone) that must be used to access services offered by one subsystem or device drivers. This, of course, depends on the particular kernel implementation and the kernel's architecture.

- A **micro-kernel** is one where large parts of the kernel are protected from each-other, usually running as services in user space. Because significant parts of the kernel are now running in user mode, the remaining code that runs in kernel mode is significantly smaller, hence micro-kernel term.
- One of the advantages of this architecture is that the services are isolated and hence bugs in one service won't impact other services.

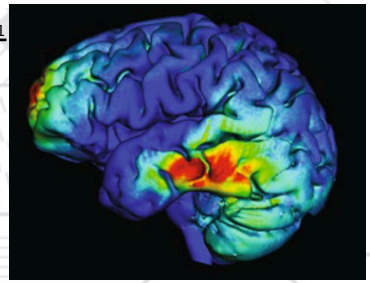
Monolithic vs Microkernel





https://linux-kernel-labs.github.io/master/lectures/intro.html

Monolithic



The Hive

Micro

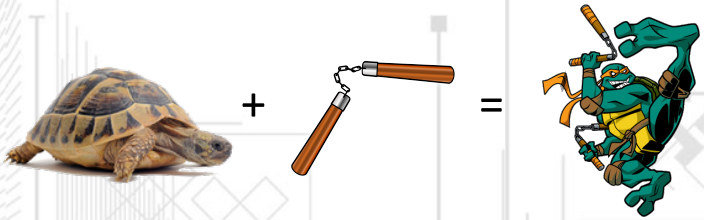
CYPRESS
EMBEDDED IN TOMORROW™

Monolithic vs Microkernel

Mono + modules + isolation => Micro

- Components can be enabled or disabled at compile time
- Support of loadable kernel modules (at runtime)
- Organize the kernel in logical, independent subsystems
- Strict interfaces but with low performance overhead: macros, inline functions, function pointers

Linux Kernel Modules



Module is a kernel object which can be compiled and installed in system on demand and independently of main kernel image

- Introduce new functionality to compiled and running kernel
- Help to keep the main kernel image size to the minimum
- Help to reduce boot time: you don’t spend time initializing devices and kernel features that you only need later.
- Easy to develop drivers: load, test, unload, rebuild, load...
- **Consideration**: have full control and privileges in the system
- Only the root user can load and unload modules
- May have dependencies on each other which implies dep management

https://www.pinclipart.com/downpngs/bowJxm_hd-wallpapers-online-coloring-pages-ninja-turtles-michelangelo/

Linux Kernel Modules

```
#include <linux/init.h>
#include <linux/module.h>
#include "md.h"

MODULE_LICENSE( "GPL" );
MODULE_AUTHOR(”Jonh Doe <john.doe@foobar.com>" );

char* md1_data = ”Hello world!";
extern char* md1_proc( void ) {
   return md1_data;
}
static char* md1_local( void ) {
   return md1_data;
}
extern char* md1_noexport( void ) {
   return md1_data;
}
EXPORT_SYMBOL( md1_data );
EXPORT_SYMBOL( md1_proc );
static int __init md_init( void ) {
   printk( "+ module md1 start!\n" );
   return 0;
}
static void __exit md_exit( void ) {
   printk( "+ module md1 unloaded!\n" );
}

module_init( md_init );
module_exit( md_exit );
```

```
#include <linux/init.h>
#include <linux/module.h>
#include "md.h"

MODULE_LICENSE( "GPL" );
MODULE_AUTHOR( “Jonh Doe <john.doe@foobar.com>");

static int __init md_init( void ) {
   printk( "+ module md2 start!\n" );
   printk( "+ data string exported from md1 : %s\n", md1_data );
   printk( "+ string returned md1_proc() is : %s\n", md1_proc() );
   return 0;
}

static void __exit md_exit( void ) {
   printk( "+ module md2 unloaded!\n" );
}

module_init( md_init );
module_exit( md_exit );
```

```
extern char* md1_data;
extern char* md1_proc( void );
```

Linux Kernel Modules

Important notes:

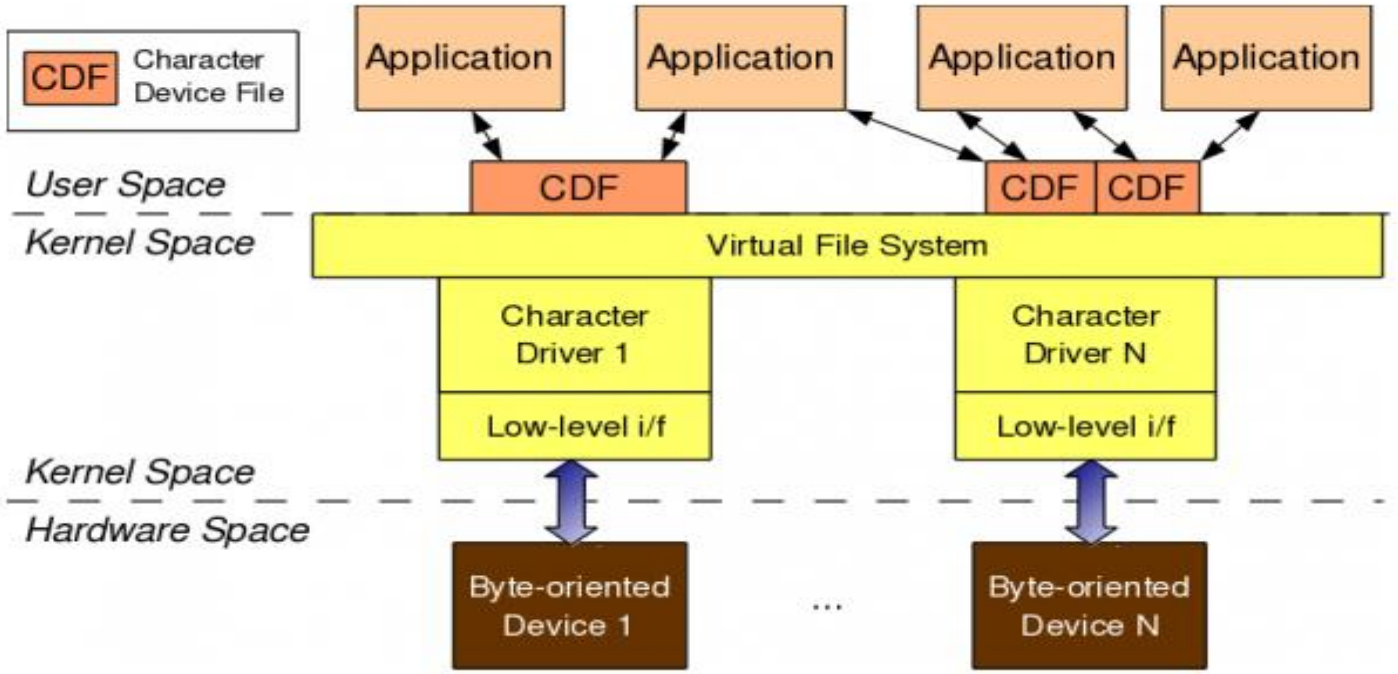
- Modules that use exported kernel symbols are linked to direct absolute address
- New modules should be compiled on target or cross compiled in recreated target environment

Device drivers types

- **Network devices** are represented as network interfaces and are visible when issuing ifconfig command from userspace.
- **Block devices** provide userspace applications with access to raw storage hardware devices (e.g. hard disks). These are visible to userspace as device files in /dev directory.
- **Character devices** - they provide userspace applications with access to other types of devices such as input/output, serial, graphics, or sound and more.

Character device drivers

Linux **character device drivers** are the kernel code that gets access to data from a hardware device sequentially.



http://www.opensourceforu.com/2011/02/linux-character-drivers

CYPRESS EMBEDDED IN TOMORROW™

Character device drivers

Major numbers are used to distinguish between different types of drivers.

Minor numbers are used to distinguish between different instances of a driver.

Character driver must implement and register entry points or methods that enable userspace applications to manipulate the device as a file

- **struct file_operations**
- **struct cdev**
- **create_class()**
- **device_create()**

Everything is a file

C language > Unix >  **> Linux**

- In the Unix design philosophy - **everything is a file.**
- This allows the kernel to represent system objects as files, and applications manipulate all system objects with files by means of normal file APIs, for example, open, read, write, and close
- Allows implementation of virtual filesystems concept that is widely used in operating systems

Special purpose file systems

- **/dev** - hosts device files - an interface to a device driver that appears in a file system as if it were an ordinary file
  - **/dev/null** - accepts and discards all input written to it; provides an end-of-file indication when read from
  - **/dev/random** – produces bytes generated by the kernel's cryptographically secure pseudorandom number generator
- **/sys** – hosts virtual file system sysfs. Contains various device and drivers information exposed by kernel
  - **/sys/bus** - contains subdirectories for each physical bus type supported in the kernel
  - **/sys/class** - directory contains every device class registered with the kernel
- **/proc**
  - **/proc/kallsyms** – contains dynamic table of loaded symbols
  - **/proc/sys/kernel/printk** – show/set level of debug messages

References

1. https://bootlin.com/doc/training/linux-kernel/
2. https://linux-kernel-labs.github.io/master/lectures/intro.html
3. https://www.kernel.org/doc/html/latest/filesystems/vfs.html
4. https://www.ibm.com/developerworks/ru/library/l-linux_kernel_01/index.html
5. http://www.opensourceforu.com/2011/02/linux-character-drivers
6. https://www.thegeekdiary.com/understanding-the-sysfs-file-system-in-linux/