What is Device Driver ?

- A Device Driver is glue between an OS and its I/O devices.
- Device drivers communicate directly with devices.
- A device is a physical or logical entity that requires control, resource management, or both from the operating system (OS).
- A device driver is a software module that manages the operation of a virtual or physical device, a protocol, or a service.

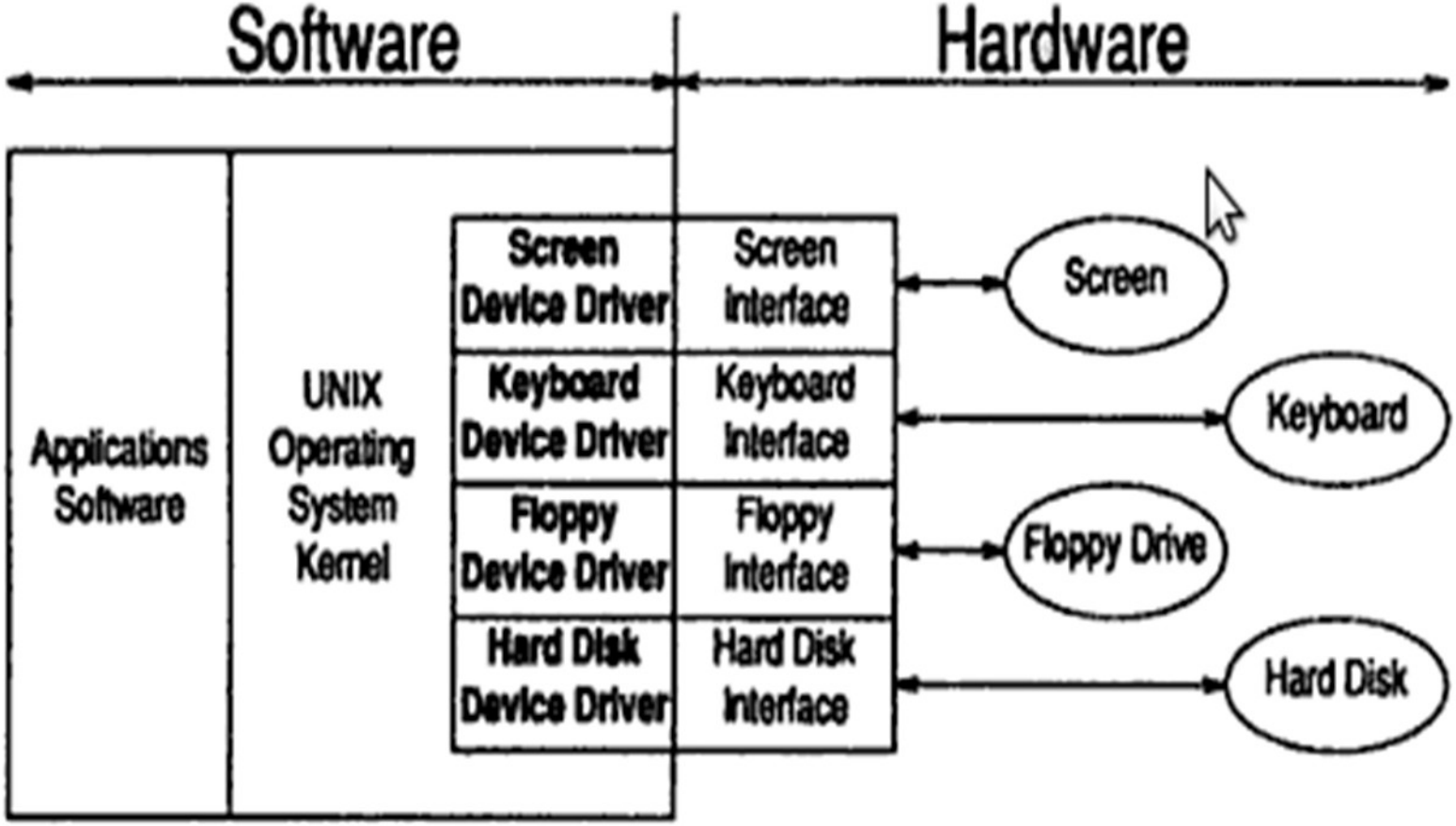
- Device driver act as translators, **converting the generic requests received from the operating system into commands that specific peripheral controllers can understand**.
- Provides software interface for hardware devices
- Helps OS & other computer programs to access hardware functions without knowing the implementation details of hardware.

Two parts of device driver

- Device specific part
  - This part remains the same across all OS
  - Used for understanding and decoding the device data than software programming
- Operating system specific part
  - In Linux device drivers provides system calls which is the boundary line between kernel space and user space of Linux.

What is Device Driver ?

FIGURE 1-1



Device Driver

1. The application software makes system calls to the operating system requesting services.
2. The operating system analyses these requests and when necessary issues request to appropriate device driver
3. The device driver in turn analyses these requests from OS and when necessary issues commands to the hardware interface to perform the operations needed to service the request

Device Drivers and OS

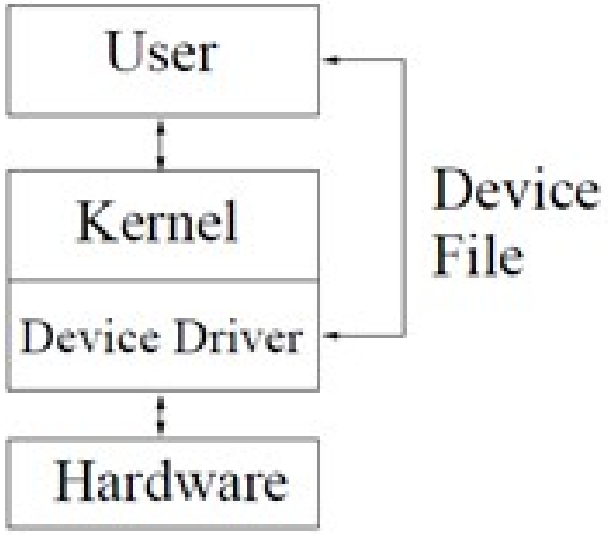
- Device drivers, simplifies the OS without directly interacting with hardware.
- The Device drivers on the other hand does not have to worry about the many issues related to general I/O management, as these are handled by the OS
- Finally
  - The OS can be written without any knowledge of the specific devices that will be connected
  - And the Device driver writer can connect any I/O device to the system without having to modify the OS.
- The result is a clean separation of responsibilities and the ability to add device drivers for new devices without changing the OS.

Functions of device driver

- Encapsulation:
  - Hides low-level device protocol details from the client
- Unification:
  - Makes similar devices look the same
- Protection (in cooperation with the OS):
  - Only authorised applications can use the device
- Multiplexing (in cooperation with the OS):
  - Multiple applications can use the device concurrently

Anatomy of a device driver

- Driver is a set of entry points (routines) that can be called by the OS.
- The driver can also contain:
  - data structures private to the driver;
  - references to kernel data structures external to the driver;
  - and routines private to the
- A device driver has three sides:
  - one side talks to the rest of the
  - one talks to the hardware,
  - and one talks to the user



Design Issue

A device driver should implement the following functions

1:OS/Driver communication

- Exchange information as command/data
- Supports functions that kernel provides

2:Driver/hardware communication

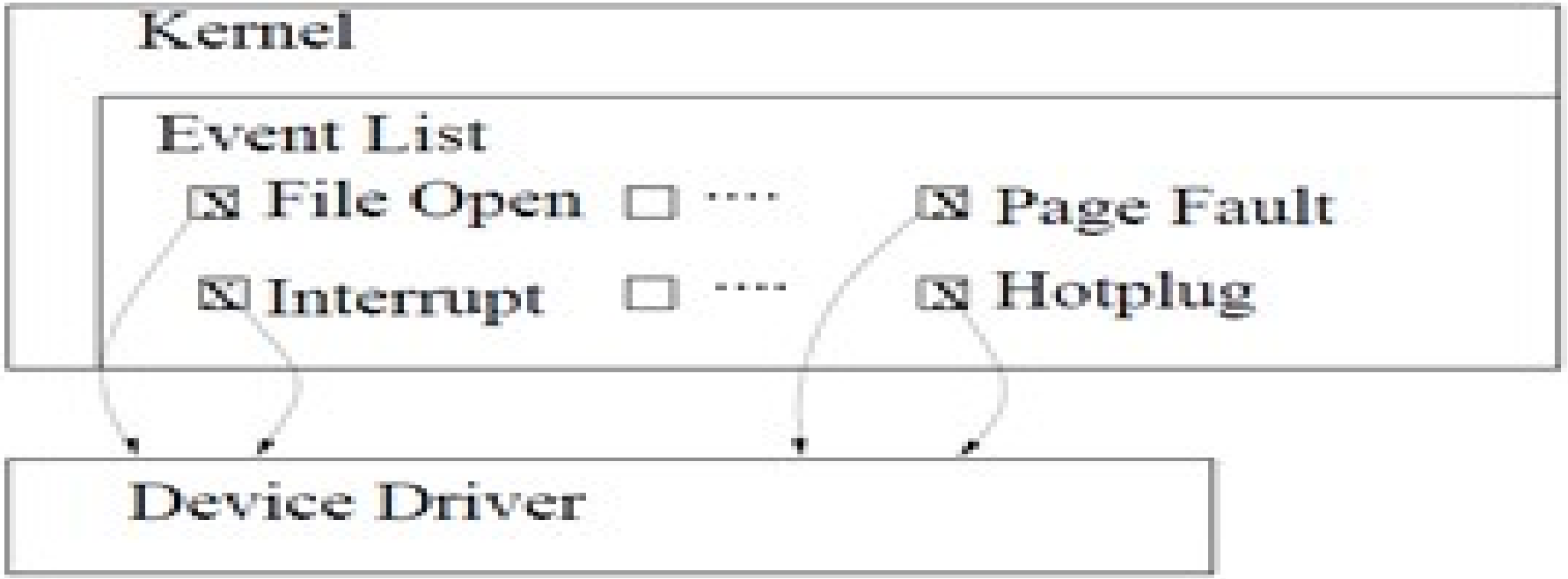
- Exchange information
- Software & hardware talks to each other

3:Driver operations

- Interrupting, scheduling, managing ,read , write , accepting…

Kernel interface of a device driver

- In order to talk to the kernel, the driver registers with the subsystem to respond to events.
- Such an event might be the opening of a file, a page fault, the plugging in of a



User interface of a device driver

- Since Linux follows the UNIX model, and in UNIX everything is a file, users talk with device drivers through device files.
- Device files are a mechanism, supplied by the kernel precisely for this direct User-Driver interface.
- Most device drivers are written as a single source file.

Prologue

- The initial part of the driver is sometimes called the *prologue*.
- The prologue is everything before the first routine and like most C programs contains:
  - #include directives referencing header files which define various kernel data types and structures
  - #define directives that provide mnemonic names for various constants used in the driver
  - Declarations of variables and data structures
- The remaining parts of the driver are the entry points (C functions referenced by the OS) & routines (C functions private to the driver)

Entry Points

Each device driver defines ENTRY POINTS

- Standard set of functions

3 parameters passed to device drivers through entry points

1. Device Parameters
2. Channel ID if it is multiplexed device driver
3. Extension parameter(Rarely used) – provides calls to extended subroutines

Entry Points

- init()
  - The init entry point is called by the kernel immediately after the system is booted.
  - It provides the driver with an opportunity to initialize the driver & the hardware as well as to display messages announcing the presence of the driver and hardware.
- start()
  - The start entry point is called by the kernel late in the boot strap sequence when more system services are available.
  - It provides the driver with an opportunity to perform initialization that requires more system

Entry Points

- open(dev, flag, id)
  - The open entry point is called by the kernel whenever a user process performs an open system call on a special file that is related to the driver.
  - It provides the driver with an opportunity to perform initialization that need to occur prior to handling read and write system calls.
- close(dev, flag, id)
  - The close entry point is called by the kernel when the last user process that has the driver open performs a close system call.
  - It provides the driver with an opportunity to release resources that may be needed only while the device

Entry Points

- halt()
  - The halt entry point is called by the kernel just before the system is shut down.
- intr(vector)
  - The intr entry point is called by the kernel whenever an interrupt is received from the hardware.
- read(dev)
  - The read entry point is called by the kernel whenever a user process performs a read system call on a special file that is related to the driver.
- write(dev)
  - The write entry point is called by the kernel whenever a user process performs a write system call on a special file that is related to the driver.

Entry Points

- ioctl(dev, cmd, arg, mode)
  - Input/output control
  - The ioctl entry point is called by the kernel whenever a user process performs an ioctl system call on a special file that is related to the driver.
  - ioctl calls are used to pass special requests to the driver or to obtain information on the configuration or status of the device and driver.

Types of Device Drivers

- **Block Drivers**
- **Character Drivers**
- Terminal Drivers
- Stream Drivers

Types of Device Drivers

- The kernel data structures that are accessed and the entry points that the driver can provide vary between the various types of drivers.
- These differences affect the type of devices that can be supported with each interface.

Device Driver Types

- Character Device driver
  - Communicates by sending and receiving single characters
    - Eg: Serial Port
    - Parallel port
    - Sound Card
    - Keyboard
  - Directly transfer data to and from users process, no need of buffering
  - Common type
  - Flexible
  - Data that is transferred should be read in order

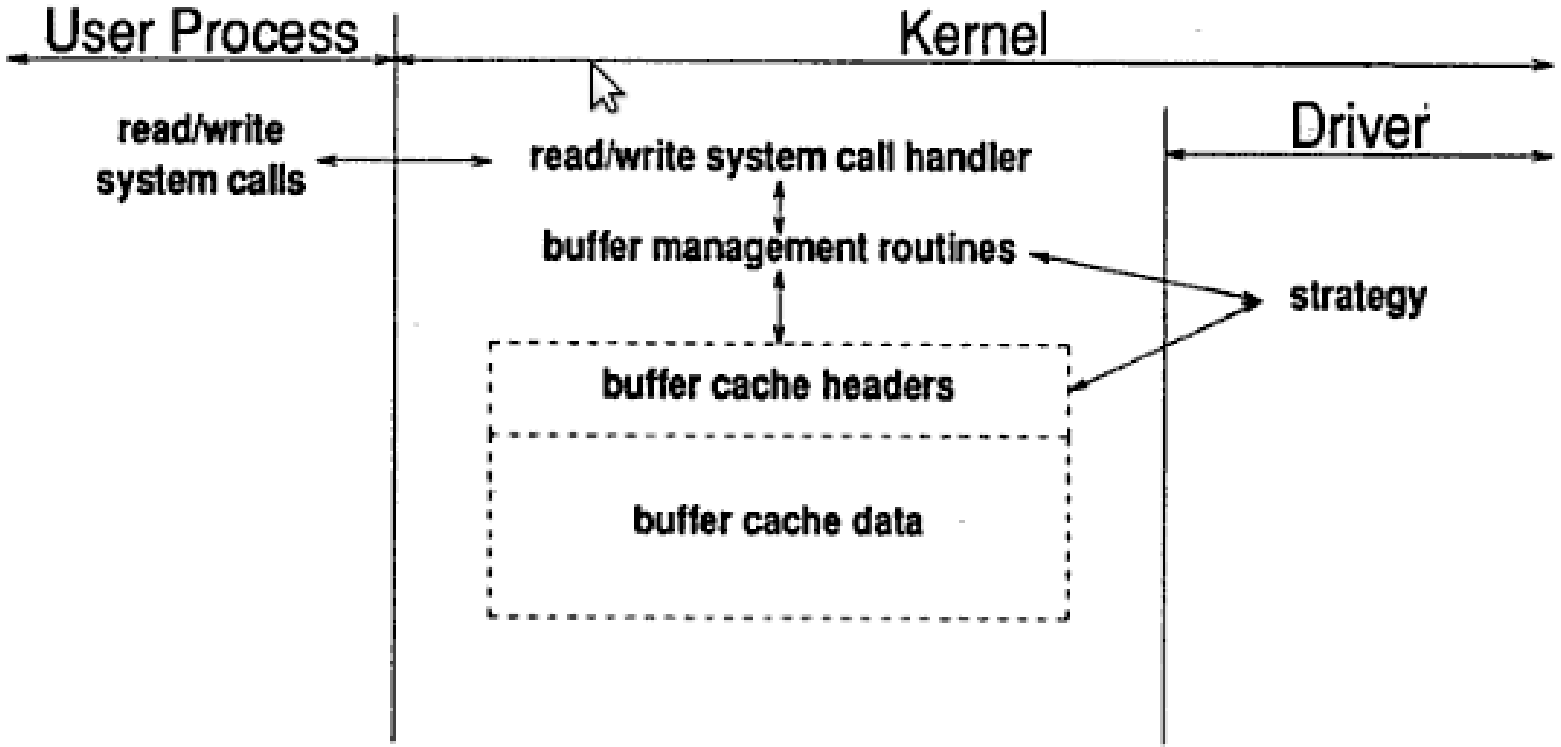
- Block Device Driver
- Communicates by sending and receiving entire blocks of data
  - Eg:Hard disk
  - Cameras

- Transfer: using buffer cache
- Used: to operate in I/O supporting block transfer of data
- Require block device drivers

Block Drivers

- It communicate with OS through a collection of fixed-sized buffers.
- For example, **disks** are commonly implemented as block devices.

FIGURE 1-2



Block device drivers

- A **block device driver** provides *structured* access to the underlying hardware.
- Block drivers support addressable block-oriented I/O (e.g., *read block number*, *write block number*) and exhibit persistence of data.
- Block I/O lends itself to caching frequently used blocks in memory.
- The **buffer cache** is a pool of kernel memory that is allocated to hold frequently used blocks from block devices.
- Block drivers are used primarly to support devices that can contain file systems

Block Device driver

- Should provide an interface to
  - Buffer cache & file operation interface
- Block device vector
  - Set of registered block devices maintained by OS
- Block device data structure
  - Buffer cache wishes to read and write from a registered device it adds request data structure on to the block device structure
  - Address of request data structure + pointer to a list of request from buffer cache , for the driver to read and write block of data
  - If a device completed a request it must remove each of the request from request structure

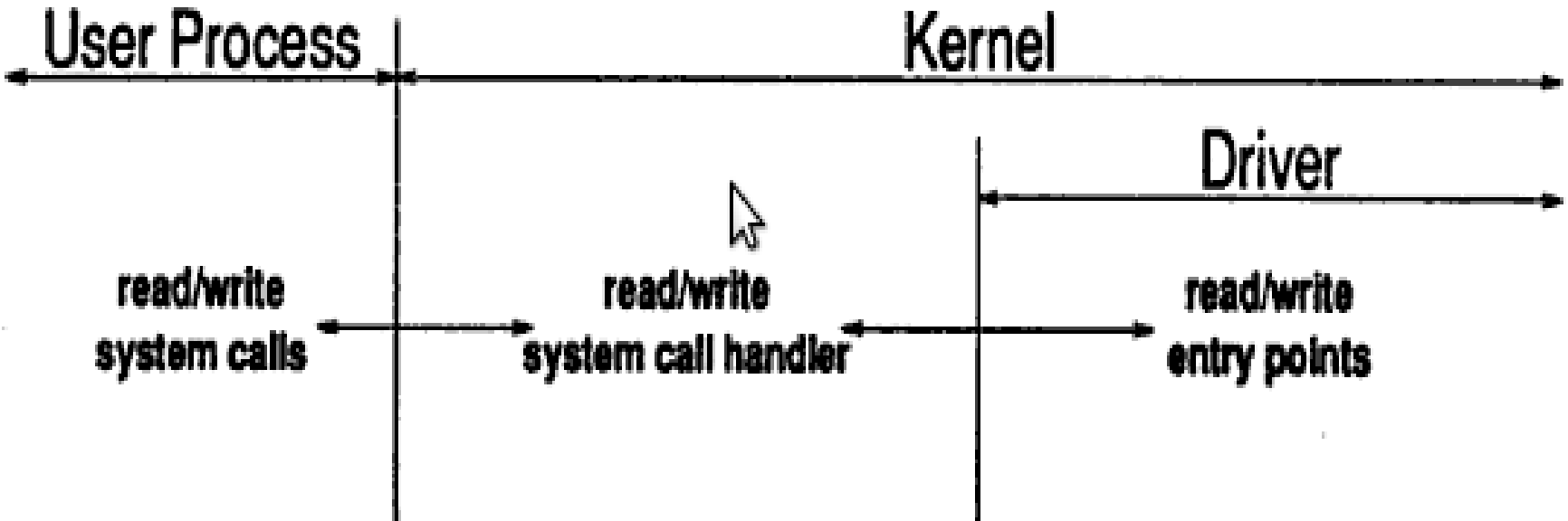
- **The Operating System/Driver Interface**
  - Block drivers may provide all of the entry points that are available to character drivers with the exception of read and write.
  - With block drivers, the function of these entry points is handled by a single strategy entry point that is **responsible for processing both read and write requests**. The strategy entry point is mandatory for all block drivers.
  - In addition, block drivers must supply a print entry point that may be used by the

Character Drivers

- It can handle I/O requests of arbitary size and can be used to support almost any type of device.
- Mostly used devices,Line Printers

- A character (char) device is one that can be accessed as a stream of bytes (like a file); a char driver is in charge of implementing this behavior.

FIGURE 1-3



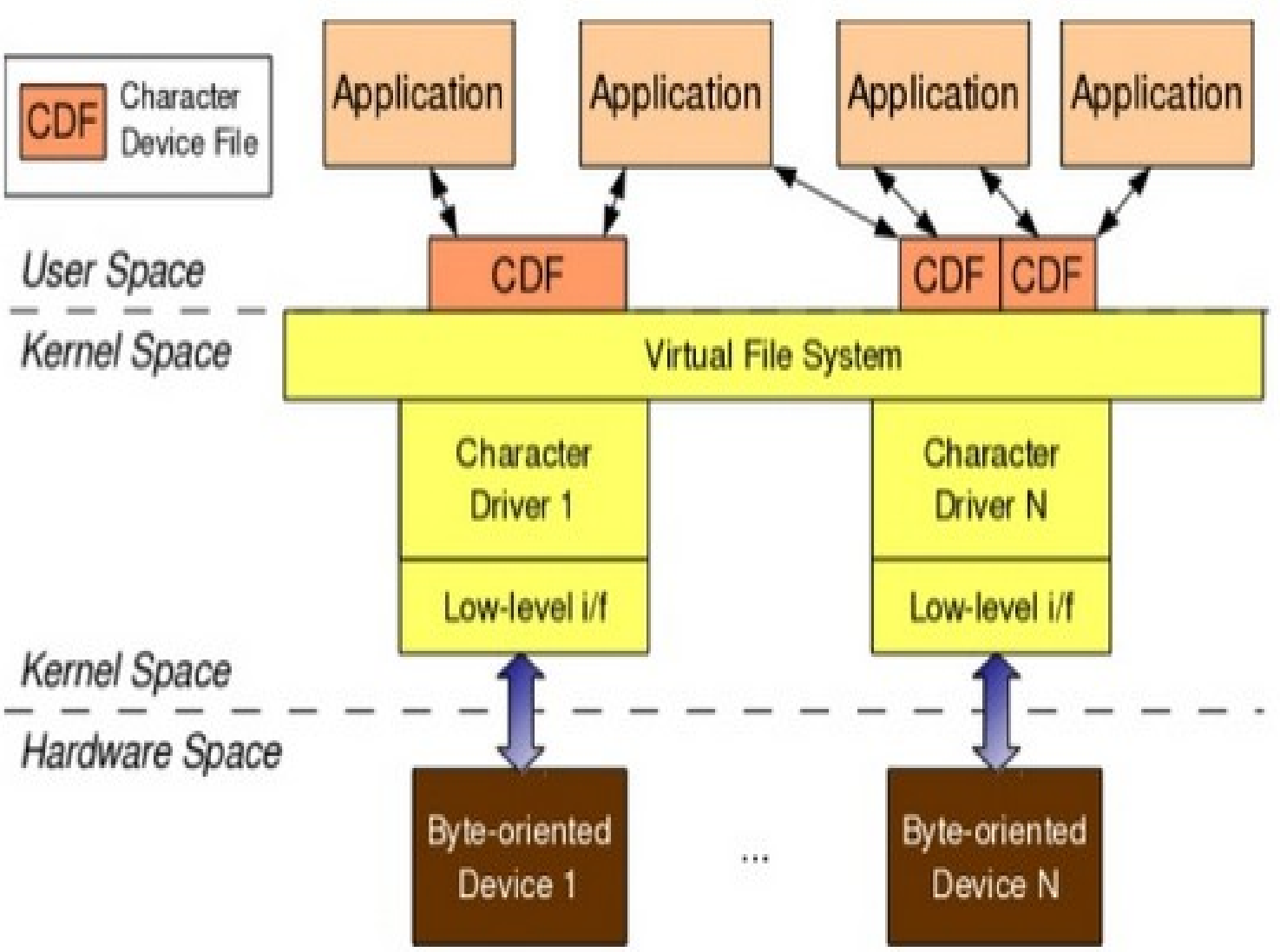
Character device driver

- Byte oriented or character oriented

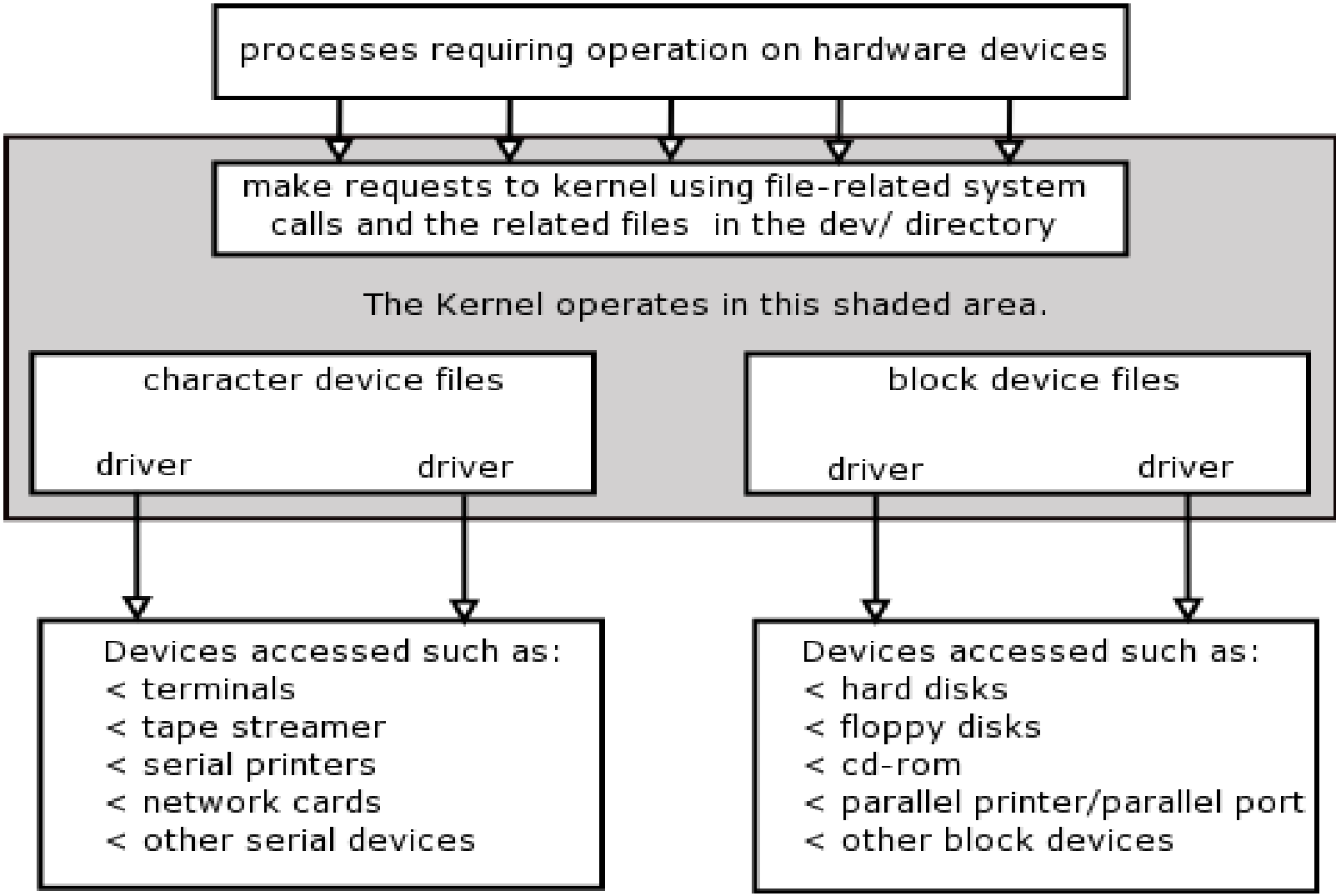
–Communicates by sending and receiving single characters

- Simplest and no need of buffer
- Accessed as files called as character device file(CDF)
    - Applications use standard system calls to access character device file to open, to read from ,to write to them and to close them
- Device driver registers to Linux Kernel
  – By adding an entry into the kernel
    - Entry: data structure containing list of character devices

Character device driver



Device Driver Types



Terminal Drivers

- Same as character drivers to deal with communication terminals that connect users to OS.
- Handle line editing, terminal functions etc.
  - E.g. MODEM

STREAM Drivers

- It can handle high speed communication devices such as networking adapters that deal with unusual sized chunks of data and that need to handle protocol.
- It is also known as Network Device Drivers.
- eg. Network Devices

Device Driver Design

- A device driver contains all the software routines that are needed to be able to use the device.
- Typically a device driver contains a number of main routines like a
  - **initialization routine**, that is used to setup the device,
  - a **reading routine** that is used to be able to read data from the device, and
  - a **write routine** to be able to write data to the device.
- The device driver may be either interrupt driven or just used as a polling routine.

- It is always desirable to split up the software into two parts,
  - one that is hardware independent and
  - one that is hardware dependent,
- It make it easier to replace one piece of the hardware without having to change the whole application.