I/O Software

- The I/O software should provide interface between devices and the rest of the system and also the interface should be device independent.
- The other issues to be considered in design of I/O software are uniform naming of devices, error handling, blocking/ non-blocking transfer of data, handling shared or exclusive devices.
- The synchronization between CPU and I/O controller is performed by the interrupt processing or by busy wait handshaking. In some cases, data may be directly transferred from device to memory or vice versa.

Performing I/O

Polling/Programmed I/O

- CPU directly controls the I/O operation by issuing commands on behalf of a process to an I/O module.
- It constantly monitors a command execution by reading the controller status register.
- Process is busy-waiting for the operation to complete

Interrupt driven I/O

- CPU issues a command on behalf of a process to an I/O module and continues to execute next instructions from the same process if the I/O operation is non-blocking else if the I/O operation is blocking then OS suspends the current process, and assigns CPU to another process.
- The controller has an embedded interrupt circuit and after completion of I/O operation, it interrupts the CPU on command process completion and CPU reads the command execution status information from the status ports.

Direct memory access (DMA)

- When a large amount of data is to be transferred between the host system and an I/O controller, the direct mode of data transfer by CPU (byte by byte or word by word) is inefficient and interrupt handling adds substantial overhead in data transfer.
- If handshake mode of data transfer is used CPU spends a considerable amount of time reading the device status and output ports.
- So to transfer large volumes of data without involving CPU, an additional hardware (Direct Memory Access device) is needed.
- DMA module controls exchange of data between memory and an I/O module.
- A DMA device assists the I/O controllers in transferring data between them and with the main memory without involving CPU.
- Processor is interrupted only after entire block has been transferred

The steps involved are:

- CPU sets up a memory buffer for data transfer and sends a request to transfer a block of data to the DMA module. The information sent are
  - Request type – read or write; using the read or write control line between the CPU and DMA module
  - Address of the I/O device, on the data line
  - Starting location in memory for read/write; communicated on data lines and stored by DMA module in its address register
  - Number of words to read/write; communicated on data lines and stored in the data count register
- CPU engages in some other activities.
- The DMA module takes over control of system bus to perform the data transfer to/from the buffer and on completion, I/O controller interrupts the CPU.

Three popular modes of DMA transfer:

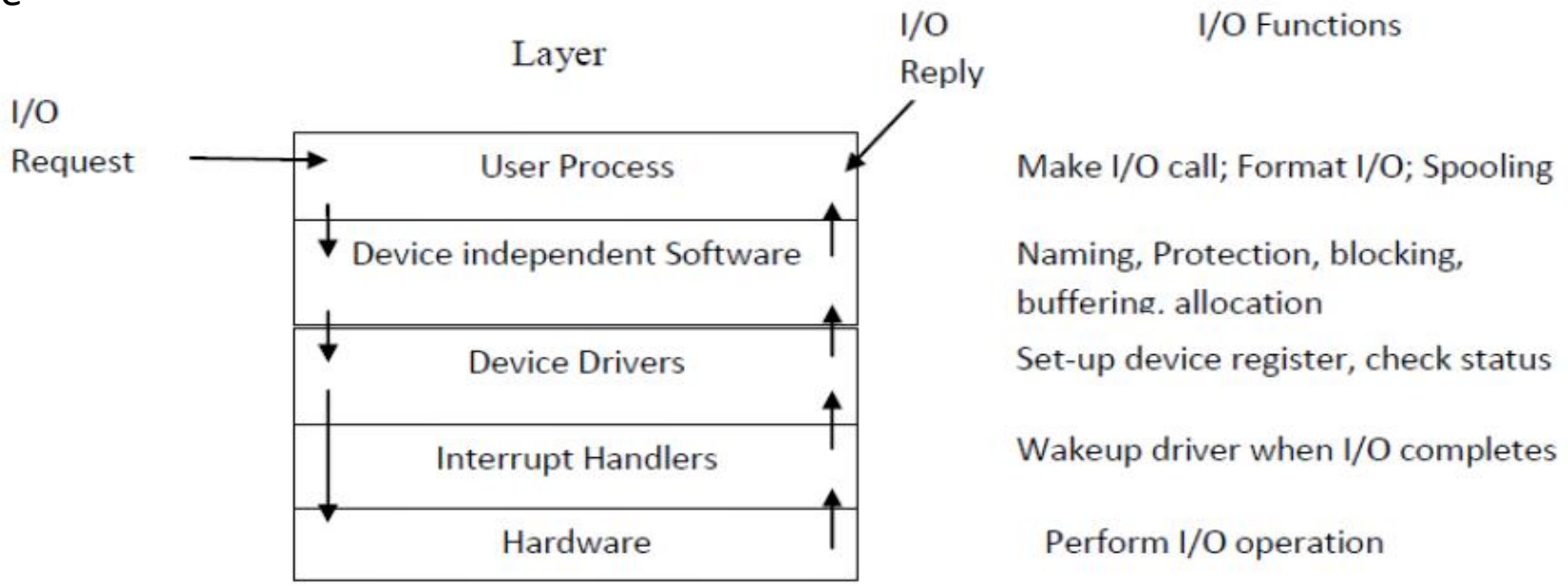
- Cycle stealing: A DMA device transfers data using bus interleaving, i.e. it steals the host bus for its purpose.
- Burst mode or Block mode: The DMA controller transfers a block of data making uninterrupted use of the bus. Other devices are not permitted to access the bus during the burst mode of transfer.
- Fly by mode or Single access mode: This mode transfers data at high speed between source and destination. The data transfer is done in a single cycle and during the transfer, the DMA controller simultaneously enables control signals to both the source and destination.

I/O Software Layer

Layers of the I/O system and the main function of each layer

Four layers

- Interrupt Handler
- Device Driver
- Device independent software
- User level I/O software



Interrupt Handler:

- Interrupt driven I/O are commonly used and they should be hidden from OS. The best way to hide them is to block the driver that start the I/O operation until the I/O has completed and the interrupt occurs.
- When the interrupt happens the interrupt procedure does whatever it has to do in order to handle the interrupt. Then it can unblock the driver that started it.

Steps that must be performed in software after the hardware interrupt has completed.

1. Save the contents of all registers (including PSW).
2. Set up a context for the interrupt service procedures. (i.e. setting up the TLB, MMU, page table etc.)
3. Set up a stack for the interrupt service procedure.
4. Acknowledge the interrupt controller and re-enable the interrupts if there is no centralized interrupt controller.
5. Copy the registers values to the process table.

6. Run the interrupt service procedure. It will extract information from the interrupting device controller’s register.

7. Choose the next process to run.

8. Set up the MMU context for the process to run next.

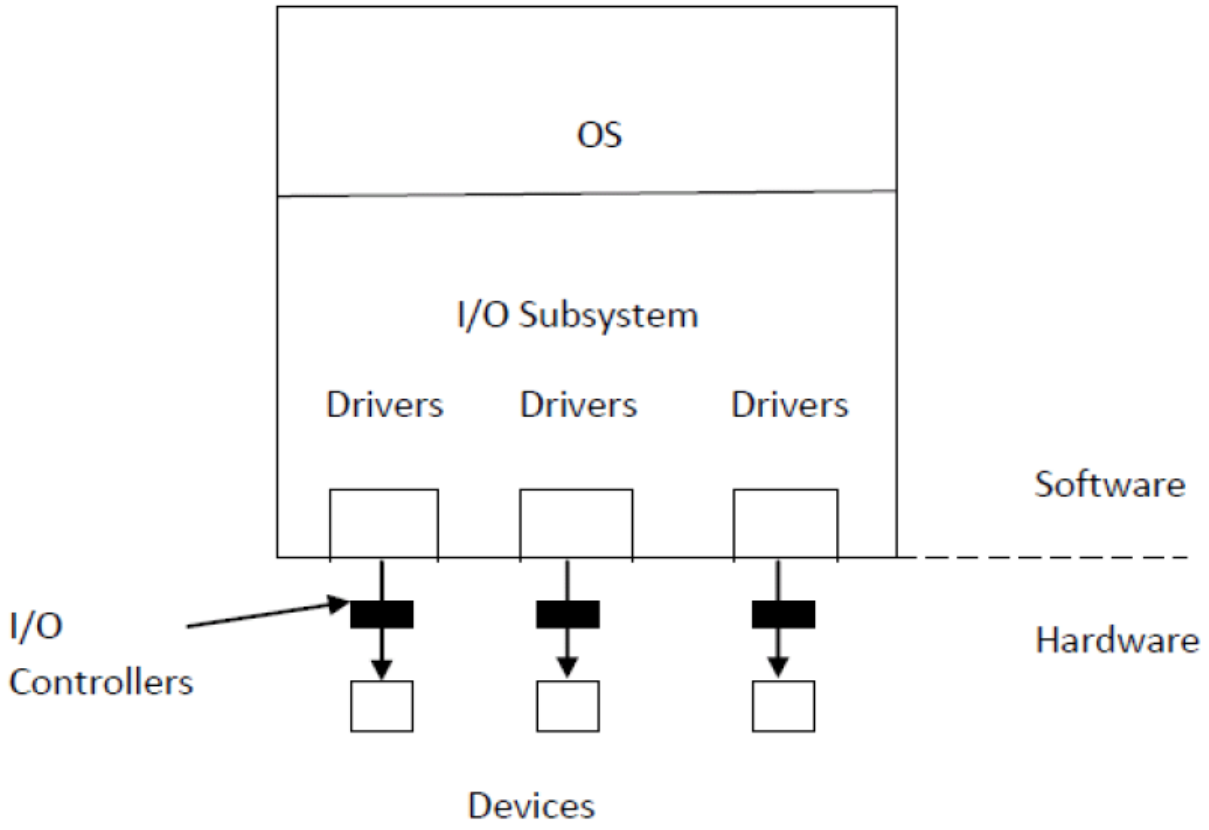
9. Load the new process registers including PSW.

10. Start running the new process.

Device drivers

- Device drivers reside in the kernel space and interact with the rest of the kernel through a specific interface.
- A device driver manipulates the I/O controller and the devices connected to it.
- It implements the software interface that enables the OS to access data from I/O devices by executing the driver program
- An interface is a collection of routines and data structures that operate in a well-defined way.
- Each driver is customized to a specific I/O controller. The drivers implement uniform interface for the OS by hiding the differences among the I/O controllers.

Device driver interface to I/O Devices



- A device driver can be visualized as an abstract object or a monitor accessed by a predefined set of operations.
- Device driver software consists of a set of private data structures, a set of device dependent routines and Kernel Device Interface Model **(**KDIM) device independent routines.
- When the driver routines are executed, the driver is said to be controlling activities of the I/O controller and its connected devices.

- The advantages of extra layer between hardware and applications are:
- Users need not study low-level programming characteristics of hardware devices and so programming is easier;
- System security is increased; kernel can check accuracy of request at the interface level before attempting to satisfy it
- Uniform interface makes programs more portable; programs compiled and executed correctly on every kernel that offers the same set of interfaces

Device independent I/O software

Some parts of I/O software is device specific, where as other parts are device independent. The exact boundary between the drivers and the device independent software is system dependent.

Functions of the Device independent I/O software:

- Uniform interfacing for device drivers
  - How to make all I/O devices and drivers look more or less the same
- All drivers have the same interface.
  - How I/O devices are named
- In Unix, i-node contains major device number which is used to locate appropriate driver
  - How does the system prevent users from accessing devices that they are not entitled to access
- Protection using permissions to access.

- Buffering
  - is needed to reduce the number of interrupts for reading and writing data from /to I/O devices such as block and character devices
  - is part of user space and/or kernel space.
- Error Reporting
  - Many errors are device specific and must be handled by the appropriate driver, but the framework for error handling is device independent.
  - Errors may be programming errors or I/O errors
  - Different ways of handling errors are ignoring errors, killing the calling process or displaying error messages and terminating the I/O operation.

- Allocating and releasing dedicated drivers.
  - Some devices such as CD-ROM recorders are exclusive in nature. It is upto the OS to examine requests for device usage and accept or reject them, depending on whether the requested device is available or not.
- Providing a device independent block size.
  - Hiding the different sector sizes of disks; number of bytes transferred by devices such as modem. Network interfaces etc.

User – space I/O software

- Small portion of the I/O software either consists of libraries linked together with the user programs (I/O system calls) or spooling system for managing serially accessible devices.