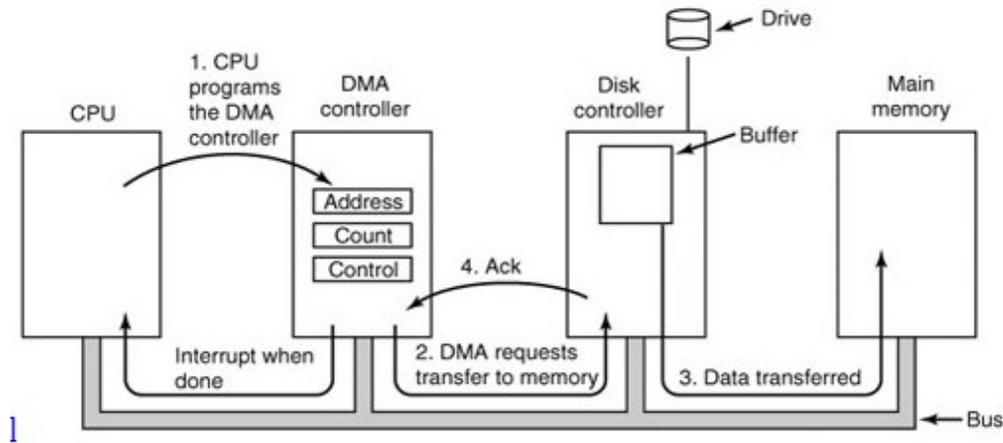


# DMA, LAYERS IN I/O AND DEVICE DRIVERS

## DMA: (Direct Memory Access)

Short for direct memory access, a technique for transferring data from main memory to a device without passing it through the CPU. Computers that have DMA channels can transfer data to and from devices much more quickly than computers without a DMA channel can. This is useful for making quick backups and for real-time applications.

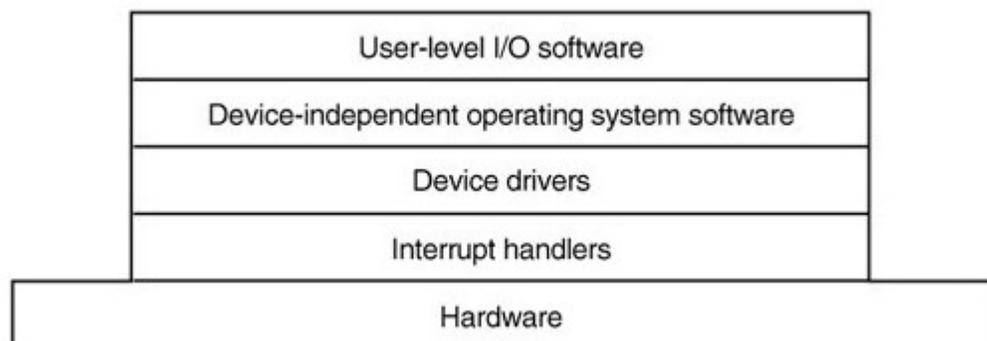
*Direct Memory Access (DMA) is a method of allowing data to be moved from one location to another in a computer without intervention from the central processor (CPU).*

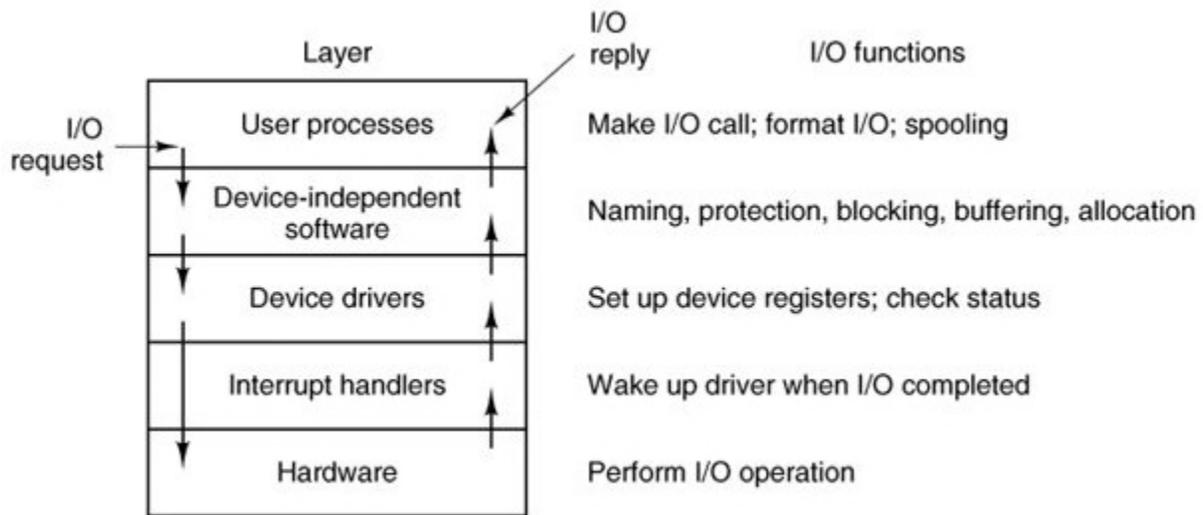


First the CPU programs the DMA controller by setting its registers so it knows what to transfer where (step 1 in Fig.). It also issues a command to the disk controller telling it to read data from the disk into its internal buffer and verify the checksum. When valid data are in the disk controller's buffer, DMA can begin.

The DMA controller initiates the transfer by issuing a read request over the bus to the disk controller (step 2). This read request looks like any other read request, and the disk controller does not know or care whether it came from the CPU or from a DMA controller. Typically, the memory address to write to is on the address lines of the bus so when the disk controller fetches the next word from its internal buffer, it knows where to write it. The write to memory is another standard bus cycle (step 3). When the write is complete, the disk controller sends an acknowledgement signal to the disk controller, also over the bus (step 4). The DMA controller then increments the memory address to use and decrements the byte count. If the byte count is still greater than 0, steps 2 through 4 are repeated until the count reaches 0. At this point the controller causes an interrupt. When the operating system starts up, it does not have to copy the block to memory; it is already there.

## Layers of the I/O software system:





**Fig. Layers of the I/O system and the main functions of each layer.**

The arrows in fig above show the flow of control. When a user program tries to read a block from a file, for example, the operating system is invoked to carry out the call. The device-independent software looks for it in the buffer cache, for example. If the needed block is not there, it calls the device driver to issue the request to the hardware to go get it from the disk. The process is then blocked until the disk operation has been completed.

When the disk is finished, the hardware generates an interrupt. The interrupt handler is run to discover what has happened, that is, which device wants attention right now. It then extracts the status from the device and wakes up the sleeping process to finish off the I/O request and let the user process continue.

### ***Device Driver:***

In computing, a device driver or software driver is a computer program allowing higher-level computer programs to interact with a hardware device.

A driver typically communicates with the device through the computer bus or communications subsystem to which the hardware connects. When a calling program invokes a routine in the driver, the driver issues commands to the device. Once the device sends data back to the driver, the driver may invoke routines in the original calling program. Drivers are hardware-dependent and operating-system-specific. They usually provide the interrupt handling required for any necessary asynchronous time-dependent hardware interface.

Each device controller has registers used to give it commands or to read out its status or both. The number of registers and the nature of the commands vary radically from device to device. For example, a mouse driver has to accept information from the mouse telling how far it has moved and which buttons are currently depressed. In contrast, a disk driver has to know about sectors, tracks, cylinders, heads, arm motion, motor drives, head settling times, and all the other mechanics of making the disk work properly. Obviously, these drivers will be very different.

Thus, each I/O device attached to a computer needs some device-specific code for controlling it. This code, called the device driver, is generally written by the device's manufacturer and delivered along with the device on a CD-ROM. Since each operating system needs its own drivers, device manufacturers commonly supply drivers for several popular operating systems.

Each device driver normally handles one device type, or one class of closely related devices. For example, it would probably be a good idea to have a single mouse driver, even if the system supports several different brands of mice. As another example, a disk driver can usually handle multiple disks of different sizes and different speeds, and perhaps a CD-ROM as well. On the other hand, a mouse and a disk are so different that different drivers are necessary.

Source : <http://dayaramb.files.wordpress.com/2012/02/operating-system-pu.pdf>