CMPSCI 577 Operating Systems Design and Implementation **Spring 2020**

Lecture 15: Mar 16, 2020

Lecturer: ***Prashant Shenoy*** *Scribe:* ***Pavan Reddy Bommana***

15.1 Architecture of an I/O system

The key components of the I/O system are :

System bus: The interface or the interconnect that allows the processor to communicate with memory or IO devices to communicate with the CPU as well as read or write data from memory.

Port: The device port has 4 registers for communication. These include status register, control register, data-in and data-out registers.

Controller: Each device has a controller which receives commands from the processor which is written to the control register and then it's going to translate them to device actions. All communication with the I/O device happens through the controller for that specific device.

15.2 Communication

The way I/O devices interface in the rest of the OS includes two layers. A device independent layer which is called the kernel I/O subsystem that is used to communicate with devices in a fashion independent of the specifics of the devices. Underneath the device independent layer is the device dependent layer which consists of all device drivers. Each device driver has all of the code needed to communicate with that specific device.

15.2.1 Communication using Polling

Device driver through the CPU is going to write a command to the command register. It is also going to write the byte that it wishes to output to the data-out register (if the operation is data out operation). Then it is going to set the status register to command-ready, which tells the controller that the device driver through CPU has actually sent a new command to the device. Once the controller sees this command in the register, its going to set the status register to busy which means its now executing that I/O operation and then the controller is going to read the command register. If it is an output command, then it is going to take the byte out from the data-out register and write to the I/O device. The device driver in the CPU keeps polling the device in busy-wait fashion checking the status register to see if the controller has completed the I/O command. Since the I/O devices are slow, it might take some time for this operation to complete thereby wasting CPU cycles as it is just doing busy-wait.

15.2.2 Communication using Interrupts

This method of communication reduces the busy-waits. In this method, the device driver is not going to busy-wait, waiting for the command to complete, instead it is going to wait for an interrupt. Except for this, all of the operations that we talked about in polling still hold for this method. Once the device has finished its operation, it raises an interrupt. In the meanwhile, CPU executes checks for this interrupt signal between the instructions. This interrupt then goes to the interrupt handler which is in the device driver which processes the data and returns from interrupt. Thereafter, CPU resumes to process the other requests.

15.2.3 Direct Memory Access(DMA)

The above two methods assume that you are interacting with I/O devices one byte at a time. DMA is helpful in performing I/O for large volumes of data at a time. In order to write to a device, we write that data to a block in memory using DMA controller. The CPU tells the DMA controller the locations of the source and destination of the transfer. DMA controller interrupts the CPU when the transfer is complete. Here we are transferring blocks of data at once instead of sending a byte at a time.

15.3 Abstraction

An OS is going to provide a much higher level abstraction for I/O devices and typically much of the lower level details are encapsulated in the device driver. The device independent layer within the OS is going to use higher level abstractions to read or write to these devices. Each device is abstracted using a few different dimensions such as transfer unit, access method, timing, speed etc.,

15.4 I/O Buffering

An important characteristic of device drivers is to somehow manage the mismatch between speed of I/O device on one hand and speed of the processor on the other. There are devices which are operated at different speeds. There is processor that is much faster than these I/O devices. Device read-write speed might be much different from CPU speed. I/O buffering is used to tackle this problem.
In buffering, there is a piece of memory both on the device and some memory used by OS on RAM. The device does read-write from the memory on the device. The DMA controller provides that piece of memory to the OS by reading that data from the memory on the device and writing it to the physical memory. The controller interrupts the CPU when the transfer is done. The buffer on the OS side will also minimize the time for which CPU has to remain in the loop for communication with the IO device.

15.5 Caching

The OS buffer is going to act as cache. The most useful purpose of having the cache is when we are using filesystem which does a lot of I/O operations to the disk. Whenever a process in trying to read a file, we are going to first see if disk blocks of that file are in OS buffer cache. Only when we have a cache miss, we are going to go to the disk to retrieve the data thereby reducing the slower disk operations.

15.5.1 Write-through policy

The changes to the data are made both on the buffer cache and the disk block immediately, making the change to propagate all the way down. This makes the policy reliable. But this process is slow as we need to make changes at multiple places(involving a disk operation).

15.5.2 Write-back policy

Under this policy, the changes to the data are made on the buffer cache initially, and to the slower memories later on. This makes the write-back policy faster than write-through. But this policy is not reliable because when a machine has a failure, any unwritten data on buffer cache is lost.

15.6 Memory-Mapped I/O (MMIO)

In MMIO, I/O devices addressing is integrated with the memory addressing. So there is a single address space. You simply read/write to a memory location which happens to be a special address(I/O address), to read/write to an I/O device. Many embedded devices use MMIO but more sophisticated devices essentially provide port mapped IO. Some machines also have a hybrid structure which supports both port-mapped and memory-mapped IO.