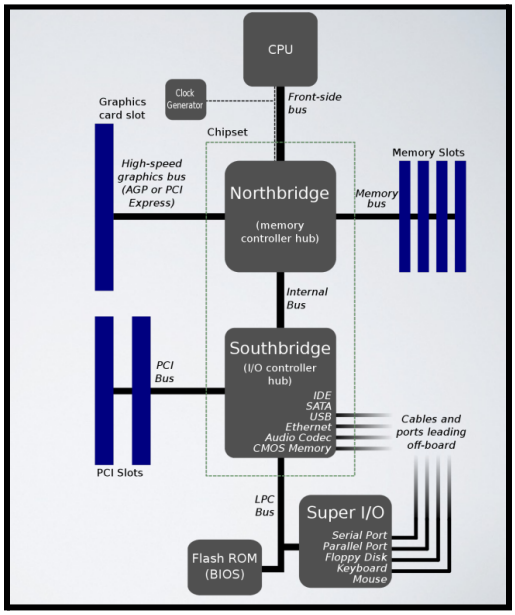
**Lecture Notes:**

- **I/O:**
- I/O devices vary greatly and new types of I/O devices appear frequently.
- There are various methods to control them and to manage their performances.
- Ports, buses, and device controllers connect to various devices.
- Some I/O Device interfaces are:
    - **Port:** A connection point for device.
      E.g. Serial Port
    - **Bus:** A daisy chain or shared direct access.
      E.g. Peripheral Component Interconnect Bus (PCI)
      E.g. Universal Serial Bus (USB)
    - **Controller (host adapter):** A set of electronics that operate ports, buses, devices, etc.
      It can be integrated or separated (host adapter).
      It contains processors, microcodes, private memory, bus controllers, etc.
      E.g. Northbridge, Southbridge, graphics controller, DMA, NIC, etc
- I/O Architecture:

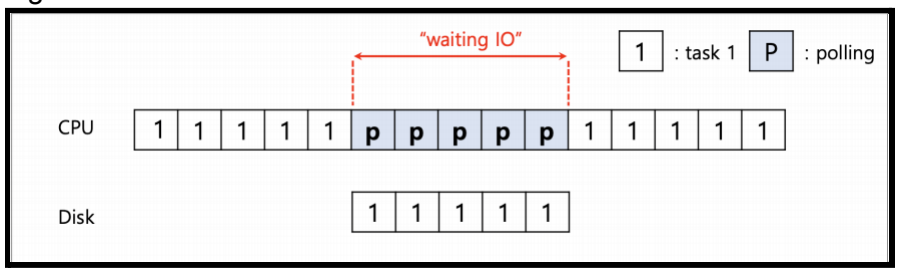



- Each device has three types of registers and the OS controls the device by reading or writing these registers.
  These registrars are:
    1. **Status register:** See the current status of the device.
    2. **Command register/Control register:** Tell the device to perform a certain task.
    3. **Data register:** Pass data to the device, or get data from the device.
- There are 2 ways to read/write with these registers:
    1. **I/O ports:** Uses in and out instructions on x86 to read and write devices registers.
    2. **Memory-mapped I/O:** Device registers are available as if they were memory locations and the OS can load (read) or store (write) to the device.

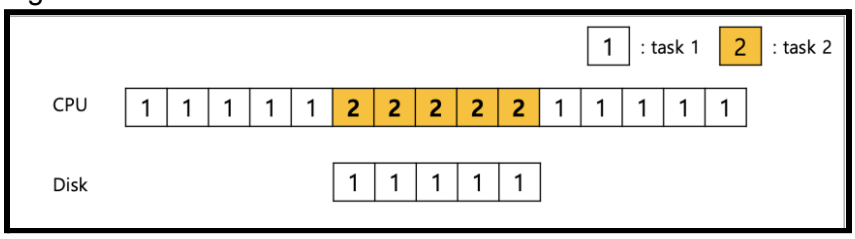
- Table of I/O Ports on PC:

| I/O address range (hexadecimal) | device |
|---|---|
| 000–00F | DMA controller |
| 020–021 | interrupt controller |
| 040–043 | timer |
| 200–20F | game controller |
| 2F8–2FF | serial port (secondary) |
| 320–32F | hard-disk controller |
| 378–37F | parallel port |
| 3D0–3DF | graphics controller |
| 3F0–3F7 | diskette-drive controller |
| 3F8–3FF | serial port (primary) |

- **Polling:**
- Here, the OS waits until the device is ready by repeatedly reading the status register.
- While it is simple and works, it wastes CPU time just waiting for the device.
- E.g.

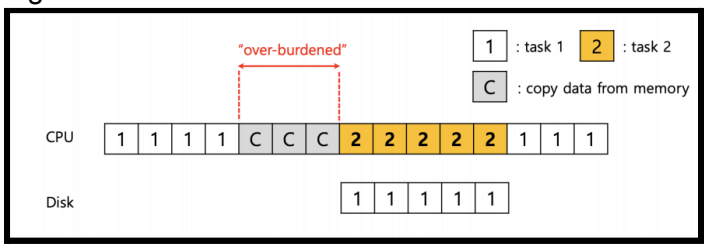



- **Interrupts:**
- With interrupts:
    1. Put the I/O request process to sleep and switch context.
    2. When the device is finished, send an interrupt to wake the process waiting for the I/O.
- The CPU is properly utilized.
- E.g.

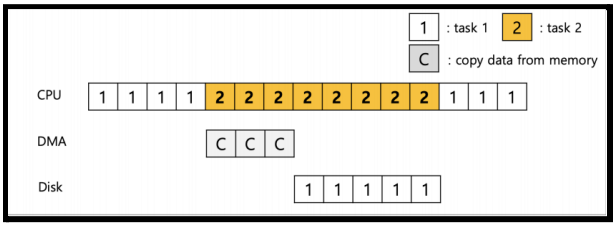



- **Polling vs Interrupts:**
- Interrupts are not always the best solution.
- If the device performs very quickly, interrupts will slow down the system.
- E.g. For a high network packet arrival rate:
    - Packets can arrive faster than the OS can process them.
    - Interrupts are very expensive (context switch).
    - Interrupt handlers have high priority.
    - In the worst case, it can spend 100% of time in the interrupt handler and never make any progress (receive livelock).
- The best practice is to do adaptive switching between interrupts and polling.

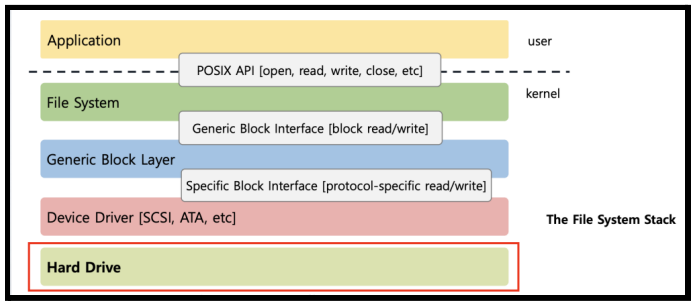
- **Data Copying:**
- When the OS copies data, the CPU wastes a lot of time in copying a large chunk of data from memory to the device.
- E.g.



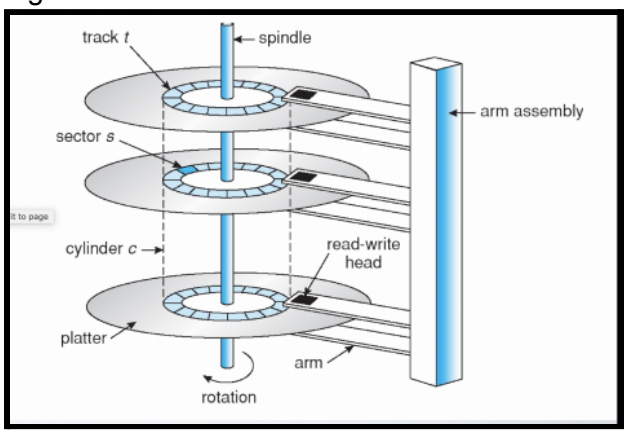

- As a result, we won't use interrupts or polling. We'll use **Direct Memory Access (DMA)**.
- **DMA (Direct Memory Access):**
- Only use the CPU to transfer control requests, not data, by passing buffer locations in memory.
- The device reads the list and accesses buffers through DMA.
- Descriptions sometimes allow for scatter/gather I/O.
- Here are the steps:
    1. OS writes DMA command block into memory.
    2. DMA bypasses CPU to transfer data directly between I/O device and memory.
    3. When completed, DMA raises an interrupt.
- E.g.




- **Note:** Variety is a challenge.
- The problem is that there are many devices and each has its own protocol.
    - Some devices are accessed by I/O ports or memory mapping or both.
    - Some devices can interact by polling or interrupt or both.
    - Some devices can transfer data by programmed I/O or DMA or both.
- The solution is to use an abstraction.
    - Build a common interface.
    - Write a device driver for each device.
- Drivers are 70% of the Linux source code.
- **File System Abstraction:**
- A file system specifics of which disk class it is using It issues block read and write requests to the generic block layer.
- I.e.



- **Hard Disk Drive (HDD):**
- **Platter (aluminum coated with a thin magnetic layer):**
  - A circular hard surface.
  - Data is stored persistently by inducing magnetic changes to it.
  - Each platter has 2 sides, each of which is called a surface.
- **Spindle:**
  - Spindle is connected to a motor that spins the platters around.
  - The rate of rotations is measured in RPM (Rotations Per Minute).
  - Typical modern values are: 7,200 RPM to 15,000 RPM.
- **Track:**
  - Concentric circles of sectors.
  - Data is encoded on each surface in a track.
  - A single surface contains many thousands and thousands of tracks.
- **Cylinder:**
  - A stack of tracks of fixed radius.
  - Heads record and sense data along cylinders.
  - Generally only one head is active at a time.
- E.g.




- The disk interface presents a linear array of sectors.
- Historically it is 512 Bytes but 4 KB in "advanced format" disks.
- Written atomically (even if there is a power failure).
- Disk maps logical sector numbers to physical sectors.
- The OS doesn't know the logical to physical sector mapping.
- Each time we write/read, we need to do 3 things:
  1. **Seek:**
     - Move the head to the above specific track.
       1. Speedup: Accelerate arm to max speed.
       2. Coast: At max speed (for long seeks).
       3. Slowdown: Stops arm near destination.
       4. Settle: Adjusts head to the actual desired track.
     - Seek is slow.
     - Settling alone can take 0.5 to 2ms.
     - An entire seek operation often takes 4 - 10ms.
  2. **Rotate:**
     - Rotate disk until the head is above the right sector.
     - This depends on rotations per minute (RPM).
     - With typical 7200 RPM it takes 8.3 ms / rotation.
     - The average rotation is slow (4.15 ms).

3. **Transfer:**
    - Data is either read from or written to the surface..
    - The speed depends on RPM and sector density.
    - With typical 100+ MB/s it takes 5µs / sector (512 bytes).
    - Pretty Fast.

- In summary:
    - Seeks are slow .
    - Rotations are slow.
    - Transfers are fast.
- What kind of workload is fastest for disks:
    - **Sequential:** Access sectors in order (transfer dominated).
    - **Random:** Access sectors arbitrarily (seek+rotation dominated).
- The Disk Scheduler decides which I/O request to schedule next:
    - First Come First Served (FCFS)
    - Shortest Seek Time First (SSTF)
    - Elevator Scheduling (SCAN)
- **Solid State Drive (SSD):**
- Completely solid state (there are no moving parts).
- It remembers data by storing charge (like RAM).
- Advantages:
    - Same interface as HDD (linear array of sectors).
    - No mechanical seek and rotation times to worry about. SSD are way faster than HDD.
    - Lower power consumption and heat (better for mobile devices).
- Disadvantages:
    - More expensive than HDD for now but it is getting cheaper.
    - Limited durability as charge wears out over time (but improving).
    - There is a limited number of overwrites possible.
      Blocks wear out after 10,000 (MLC) – 100,000 (SLC) erases.
      Requires **Flash Translation Layer (FTL)** to provide wear levelling, so repeated writes to logical block don't wear out physical block.
      FTL can seriously impact performance.