I/O Devices - IDE (ch. 36+37)

Operating Systems

Based on: Three Easy Pieces by Arpaci-Dusseaux

Moshe Sulamy

Tel-Aviv Academic College

Case Study

- IDE disk
  - Four types of registers:
    - Control, command block, status, and error
  - Available at specific "I/O addresses"
    - Using `in` and `out` instructions

Case Study

- The IDE interface:

```
Control Register:
    Address 0x3F6 = 0x08 (0000 1RE0): R=reset,
                    E=0 means "enable interrupt"

Command Block Registers:
    Address 0x1F0 = Data Port
    Address 0x1F1 = Error
    Address 0x1F2 = Sector Count
    Address 0x1F3 = LBA low byte
    Address 0x1F4 = LBA mid byte
    Address 0x1F5 = LBA hi byte
    Address 0x1F6 = 1B1D TOP4LBA: B=LBA, D=drive
    Address 0x1F7 = Command/status

Status Register (Address 0x1F7):
    7       6       5       4       3   2       1       0
    BUSY    READY   FAULT   SEEK    DRQ CORR    IDDEX   ERROR

Error Register (Address 0x1F1): (check when ERROR==1)
    7   6   5   4       3   2       1       0
    BBK UNC MC  IDNF    MCR ABRT    T0NF    AMNF
    BBK = Bad Block
    UNC = Uncorrectable data error
    MC = Media Changed
    IDNF = ID mark Not Found
    MCR = Media Change Requested
    ABRT = Command aborted
    T0NF = Track 0 Not Found
    AMNF = Address Mark Not Found
```

Case Study

- **Wait for device to be ready**: read Status Register (0x1F7) until READY and not BUSY
- **Write parameters to command registers**: write the sector count, logical block address (LBA) of the sectors to be accessed, and drive number (master=0x00 or slave=0x10, as IDE permits just two drives) to command registers (0x1F2-0x1F6)
- **Start the I/O**: by issuing read/write to command register. Write READ—WRITE command to command register (0x1F7)
- **Data transfer (for writes)**: wait until drive status is READY and DRQ (drive request for data); write data to data port
- **Handle interrupts**: in the simplest case, handle an interrupt for each sector transferred; more complex approaches allow batching and thus one final interrupt when the entire transfer is complete
- **Error handling**: after each operation, read the status register. If the ERROR bit is on, read the error register for details

xv6 ide driver: wait

```
static int ide_wait_ready() {
    while (((int r = inb(0x1f7)) & IDE_BSY) ||
           !(r & IDE_DRDY));
}
```

xv6 ide driver: start

```
static void ide_start_request(struct buf *b) {
    ide_wait_ready();
    outb(0x3f6, 0); // generate interrupt
    outb(0x1f2, 1); // how many sectors?
    outb(0x1f3, b->sector & 0xff); // LBA
    outb(0x1f4, (b->sector >> 8) & 0xff); // ...
    outb(0x1f5, (b->sector >> 16) & 0xff); // ...
    outb(0x1f6, 0xe0 | ((b->dev&1)<<4) |
                ((b->sector>>24)&0x0f));
    if (b->flags & B_DIRTY) {
        outb(0x1f7, IDE_CMD_WRITE); // WRITE
        outsl(0x1f0, b->data, 512/4); //
    } else {
        outb(0x1f7, IDE_CMD_READ); // this is a READ
            (no data)
    }
}
```

xv6 ide driver: rw

```
void ide_rw(struct buf *b) {
    acquire(&ide_lock);
    for (struct buf **pp = &ide_queue; *pp;
                          pp = &(*pp)->qnext);

    *pp = b;

    if (ide_queue == b)
        ide_start_request(b); // send req to disk
    while ((b->flags & (B_VALID|B_DIRTY)) != B_VALID)
    sleep(b, &ide_lock); // wait for completion
    release(&ide_lock) ;
}
```

xv6 ide driver: isr

```
void ide_intr() {
    struct buf *b;
    acquire(&ide_lock);
    if (!(b->flags & B_DIRTY) &&
        ide_wait_ready() >= 0)
            insl(0x1f0, b->data, 512/4);
    b->flags |= B_VALID;
    b->flags &= ~B_DIRTY;
    wakeup(b); // wake waiting process
    if ((ide_queue = b->qnext) != 0) //
        ide_start_request(ide_queue);
    release(&ide_lock);
}
```

Summary (Hard Disk Drives)

- 512-byte sectors
  - **Platter** with two **surfaces**, bound around the **spindle**
  - Fixed rate of **RPM**
  - Data encoded in **tracks**, read and write by **disk head**
- **Rotational delay**: wait for sector to reach head
- **Seek**: move disk arm to correct track
  - Acceleration $\rightarrow$ coasting $\rightarrow$ deceleration $\rightarrow$ settling
- **I/O time**: seek $\rightarrow$ wait for rotational delay $\rightarrow$ transfer
- **Cache** holds read/write data
  - **Write-through**: acknowledge on write to disk
  - **Writeback**: acknowledge when data is in cache
- Disk scheduling
  - **SSTF**, **NBF**, **Elevator** (**sweep**, **F-SCAN**, **C-SCAN**), **SPTF**
  - **I/O merging**: merge requests for consecutive sectors
  - **Work-conserving**: wait before issuing I/O to disk