A computer runs programs that are loaded in memory.

- The program is loaded from a storage device.
- The result of an execution is stored to a storage device.

Storage device is:

- Nonvolatile
- Larger than memory
- Slower than memory
Punch card / punch tape

- Stores program and data to load
  - Indexed by physical meaning (cabinet, ...)

History of storage devices
**History of storage devices**

**Magnetic tape**

- **UNIVAC / 1951**
- **tar**: sequential I/O

<table>
<thead>
<tr>
<th>header</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
<td>data</td>
</tr>
<tr>
<td>header</td>
<td>data</td>
</tr>
</tbody>
</table>
Magnetic disk

- 3 dimensional addressing (head, cylinder, sector)
- Still used as a main storage device
Magnetic disk characteristics

- Mechanical movement defines I/O latency
  
  \[
  T_{\text{seek}} + T_{\text{rot}} + \frac{\text{Volume}}{\text{Bandwidth}}
  \]

- \[ T_{\text{avg}} = \frac{1}{2 \cdot \text{RPM}} + \frac{\text{Volume}}{\text{Bandwidth}} \]

- Smaller the request, lower the throughput

  \[ T_{\text{avg}} = \frac{1}{2 \cdot \text{RPM}} \quad \text{vs.} \quad T_{\text{avg}} = \frac{\text{Volume}}{\text{Bandwidth}} \]

- 8~32MB I/O does the magic
Non-volatile memories

- **Flash memory**

- **Future:** PCM, STTM, ZRAM, ...

<table>
<thead>
<tr>
<th>Technology</th>
<th>Density</th>
<th>Latency</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SLC Flash</td>
<td>$4 \times F^2$</td>
<td>$4 \times F^2$</td>
<td>25 $\mu$s</td>
</tr>
<tr>
<td>PCM</td>
<td>$10 \times F^2$</td>
<td>$4 \times F^2$</td>
<td>67.5 ns</td>
</tr>
<tr>
<td>STTM</td>
<td>$64 \times F^2$</td>
<td>$15 \times F^2$</td>
<td>29.5 ns</td>
</tr>
<tr>
<td>DRAM</td>
<td>$6 \times F^2$</td>
<td>$4 \times F^2$</td>
<td>25 ns</td>
</tr>
</tbody>
</table>
Flash memory characteristics

• **Pros**
  - Small form factor
  - No mechanical parts: good random I/O throughput
  - Low power consumption

• **Cons**
  - Read, program (write), erase interface
    - Slow page programming, even slower block erase
  - Limited block erase cycle
  - Expensive (price / capacity)

• **Types and timing**
  - SLC / MLC / TLC
Storage device characteristics

- Non-volatile
- Slow and large
- **Storage hierarchy**
Storage abstraction

```
/  
|   |
usr home

Flat address

Application  Application  Application

File system
ext4  VFAT  NFS

Block I/O layer

Device driver
```
The Block I/O Layer
**Block device concepts**

- **Old HDD access method**: (head, cylinder, sector)
  - Differs for each disk
  - SSDs has different structure

- **Generic method**: contiguous address space
  - Let storage devices do the complex managements
    - (Head, cylinder) mapping, flash translation layer, ...
    - Bad block management

- **Access units**: sectors (512 bytes) / blocks (4KB)
I/O subsystem architecture

- read/write (device, addr, offset, nr sectors)
- Hard to manipulate
• **Split storage area**: safe-keeping, multi OS

<table>
<thead>
<tr>
<th>/</th>
<th>/usr/local</th>
<th>/home</th>
<th>swap</th>
</tr>
</thead>
</table>

• **Partition table types**
  – Master Boot Record
  – GUID Partition Table
    • Large device support
    • Partition name support
    • Consistency mechanism
Block I/O layer

- Operate on `struct block_device <linux/fs.h>`
- Allows random access to a block device
- Supports partitioning, I/O scheduling, ...

Virtual File System

- Ext4
- FAT
- YAFFS
- NFS
- FUSE

Page Cache

Block I/O layer

- read/write(sector, length, buffer)

Device mapper

- LVM
- RAID
- Cache

I/O scheduler

Partition support
Buffer head I/O mechanism

• Buffer head and BIO
  – submit_bh() vs. submit_bio()

• Buffer head
  – **Buffer**: less than a page size
  – Uses *virtual address* to map blocks
  – Deprecated API, but still used in some places
  – Uses BIO to execute I/O operations
  – **API**: alloc_buffer_head(), get_bh(), put_bh(), submit_bh()
• **BIO structure** (since 2.5)
  - An array of segments of pages (*bio_vec*)
  - Can use unmapped pages

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>block_device</td>
<td><em>bi_bdev</em></td>
</tr>
<tr>
<td>sector_t</td>
<td><em>bi_sector</em></td>
</tr>
<tr>
<td>struct bio *</td>
<td><em>next</em></td>
</tr>
<tr>
<td>long</td>
<td><em>bi_rw</em></td>
</tr>
<tr>
<td>int</td>
<td><em>bi_size</em></td>
</tr>
<tr>
<td>short</td>
<td><em>bi_vcnt</em></td>
</tr>
<tr>
<td>struct bio_vec*</td>
<td><em>bi_io_vec</em></td>
</tr>
<tr>
<td>bio_end_io_t</td>
<td><em>bi_end_io</em></td>
</tr>
<tr>
<td>void *</td>
<td><em>bi_private</em></td>
</tr>
</tbody>
</table>
• Life cycle of BIO

bio_alloc(gfp_mask, nr_vecs)

bio_get(bio)

bio_add_page(bio, page, offset, length)

bio_for_each_segment(bvl, bio, i) {

bio_put(bio)

submit_bio(rw, bio)

bio->bi_end_io(bio, errno)

bio_endio(bio, errno)

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>block_device</td>
<td>bi_bdev</td>
</tr>
<tr>
<td>sector_t</td>
<td>bi_sector</td>
</tr>
<tr>
<td>struct bio *</td>
<td>next</td>
</tr>
<tr>
<td>long</td>
<td>bi_rw</td>
</tr>
<tr>
<td>int</td>
<td>bi_size</td>
</tr>
<tr>
<td>short</td>
<td>bi_vcnt</td>
</tr>
<tr>
<td>struct bio_vec *</td>
<td>bi_io_vec</td>
</tr>
<tr>
<td>bio_end_io_t</td>
<td>bi_end_io</td>
</tr>
<tr>
<td>void *</td>
<td>bi_private</td>
</tr>
</tbody>
</table>
• **Request** and request_queue
  – **bios** merged by I/O scheduler

  ![Diagram of request queues]

  – **Request queue** operations
    • request_fn()
    • make_request_fn()

  ![Flowchart of request operations]

  Used for device mapper implementation

  Device mapper

  I/O scheduler
I/O scheduler

- Gathers block I/Os to make large request
  - Minimize seek time

- Scheduling algorithms in OS courses

- FIFO
- Scan
- Elevator
Linux I/O schedulers

• Based on (Linus) elevator algorithm
  – Merges BIOs and executes them

• Available schedulers
  – **noop**: simple elevator
  – **Deadline**: prevent starvation
  – **Anticipatory**: wait for a few msec to merge contiguous I/O
  – **Complete Fair Queuing**: *per process queue* for fairness

• Choosing and displaying I/O schedulers
  – `/sys/block/{device name}/queue/scheduler`
Block device data structure

- **Block device** (*block_device*)
  - Physical disk + partition + virtual mapped device
    Ex. /dev/sda, /dev/sda1, /dev/mapper/vol1, ...

- **General disk** (*gendisk*)
  - Physical/virtual device driver
    Ex. /dev/sda

- **Partitions**
  - *gendisk-*>part_tbl[]
  - *bdev-*>bd_part
Block device and gendisk

- **block_device**
  - `struct gendisk`
    - `part_tbl`
      - `part`
        - `bd_contains`
        - `bd_part`
        - `bd_disk`
- `/dev/sda`
  - `bd_contains`
  - `bd_part`
  - `bd_disk`
- `/dev/sda1`
  - `bd_contains`
  - `bd_part`
  - `bd_disk`
- `/dev/sda3`
  - `bd_contains`
  - `bd_part`
  - `bd_disk`

- `struct hd_struct`
  - `start_sect`
  - `nr_sects`
Creation / destruction

• **gendisk**
  - struct gendisk *alloc_disk(int minors);
  - int add_disk(gendisk);
  - del_gendisk(gendisk);

• **Block device**
  - struct block_device *bdget_disk(
    struct gendisk *disk, int partno);
  - int __invalidate_device(
    struct block_device *bdev,
    bool kill_dirty);

• **Partition**
  - int rescan_partitions(
    struct gendisk *disk,
    struct block_device *bdev)
gendisk operation

- **block_device_operations**
  - `open(struct block_device *, fmode_t);`
  - `release(struct gendisk *, fmode_t);`
  - `ioctl(struct block_device *, fmode_t, unsigned, unsigned long);`
  - `direct_access(struct block_device *, sector_t, void **, unsigned long *);`
  - `check_events(struct gendisk *disk, unsigned int clearing);`

- **request_queue->request_fn()**
  - Set by `blk_init_queue()`
  - Sends I/O requests to device driver (gendisk)
**Review: Block I/O layer**

- **Block device**: abstracting HDDs, SSDs, ...
  - `/dev/sd xn` >= 1, or none
  - a~z, aa~zz

- **Accessing block device**
  - Through file system
  - Through device node – allows file operation

- **Linux implementation layers**

<table>
<thead>
<tr>
<th>Virtual file system</th>
<th>Block I/O layer</th>
<th>Device driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext4</td>
<td>Device mapper</td>
<td>SATA</td>
</tr>
<tr>
<td>btrfs</td>
<td>LVM</td>
<td>SSD</td>
</tr>
<tr>
<td>FAT</td>
<td>RAID</td>
<td>HDD</td>
</tr>
<tr>
<td></td>
<td>I/O scheduler</td>
<td>SCSI</td>
</tr>
<tr>
<td></td>
<td>CFQ</td>
<td>LVM</td>
</tr>
<tr>
<td></td>
<td>NOOP</td>
<td>RAID</td>
</tr>
</tbody>
</table>

  ```
  fd = open("/dev/sdb2", O_RDWR);
  lseek(fd, 0, SEEK_SET)
  read(fd, sb, 4096);
  ```

  Reading super block

  Partition table