Persistence: I/O and Disk Devices

CS 537: Introduction to Operating Systems

Louis Oliphant

University of Wisconsin - Madison

Fall 2023

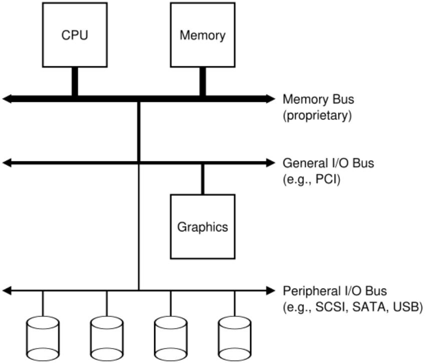
Administrivia

- Project 5 due Nov 7th @ 11:59pm
- Exam 2, Nov 9th 7:30-9pm
    - Bring ID and #2 Pencil, same format as Exam 1
    - Lec 001 – Humanities 3650
    - Lec 002 – Humanities 2340
    - McBurney – 5:45-8pm, CS 1325

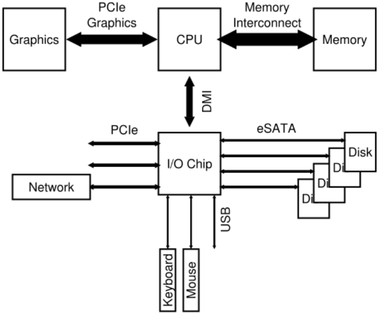
I/O Devices Agenda

- How OS interacts with I/O Devices
- How HDD is organized

Prototypical Systems Architecture



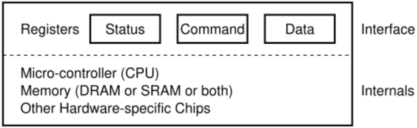
- Multiple Bus Levels
- Faster busses are shorter, more expensive



- Direct Media Interface
- Slow devices connect through an I/O chip

OS Communication with Cannonical Device

```
while (STATUS == BUSY)
  ; //wait until device is not busy
write data to DATA register
write command to COMMAND register
  (Doing so starts the device and executes the command)
while (STATUS == BUSY)
  ; //wait until device is done with request
```



Registers Status Command Data Interface

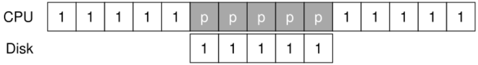
Micro-controller (CPU)
Memory (DRAM or SRAM or both)
Other Hardware-specific Chips

Internals

- OS uses **polling** to check status
- **Programmed I/O** (PIO) when main CPU controls data movement
- Motivates **Hardware Interrupts** for effeciency

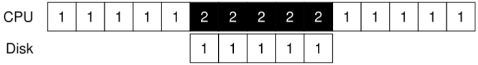
More Efficient I/O

- Polling



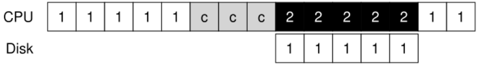
CPU	1	1	1	1	1	p	p	p	p	p	1	1	1	1	1
Disk						1	1	1	1	1					

- Interrupts (allow other process to run)



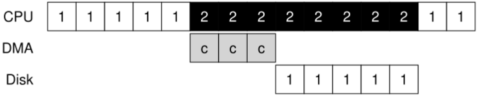
CPU	1	1	1	1	1	2	2	2	2	2	1	1	1	1	1
Disk						1	1	1	1	1					

- OS still copies data to device



CPU	1	1	1	1	1	c	c	c	2	2	2	2	2	1	1
Disk									1	1	1	1	1		

- OS uses **Direct Memory Access (DMA)** which handles the copy portion of IO
- Just pass data location and size to DMA



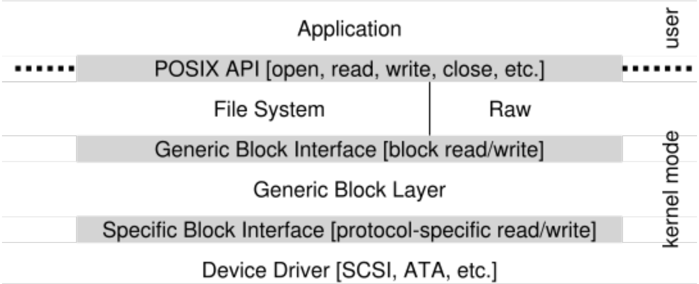
CPU	1	1	1	1	1	2	2	2	2	2	2	2	2	1	1
DMA						c	c	c							
Disk									1	1	1	1	1		

Methods of I/O Interactions

- Explicit I/O Instructions
  - on x86, the `in` and `out` instructions used to communicate with device
  - OS conrols register with data, and knows specific *port* which names the device, issues instruction.
- Memory-mapped I/O
  - Device appears as memory location
  - OS uses same load/store commands as for regular memory
  - Hardware routes the instruction to the device instead

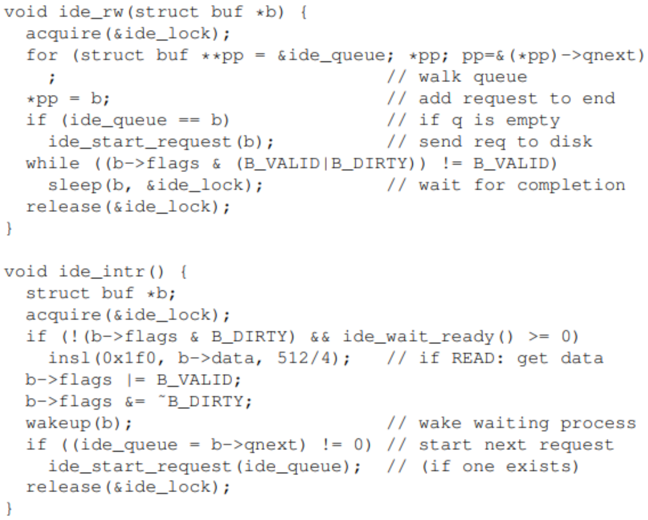
Device Driver

- Many, many devices, each has its own protocol
- **Device driver** for each device, rest of OS just interacts with driver
- OS often has **raw interface** to directly read and write blocks
- 70% of OS code is found in device drivers



Application
user
POSIX API [open, read, write, close, etc.]
File System
Raw
Generic Block Interface [block read/write]
Generic Block Layer
kernel mode
Specific Block Interface [protocol-specific read/write]
Device Driver [SCSI, ATA, etc.]

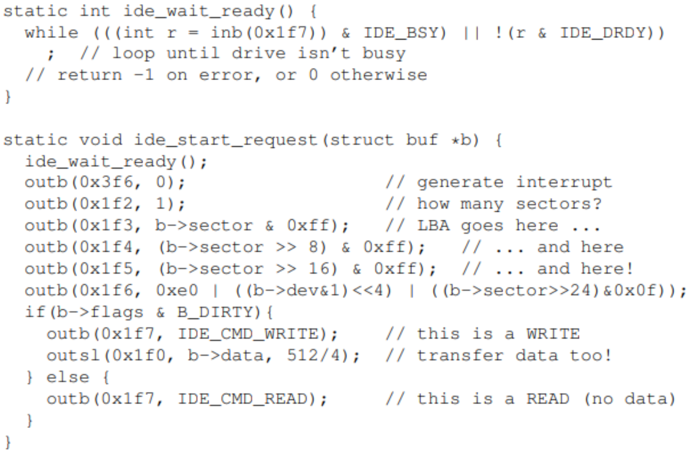
Simple IDE Disk Driver (**xv6**)



```
void ide_rw(struct buf *b) {
  acquire(&ide_lock);
  for (struct buf **pp = &ide_queue; *pp; pp=&(*pp)->qnext)
    ;                               // walk queue
  *pp = b;                          // add request to end
  if (ide_queue == b)               // if q is empty
    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);
}

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

Simple IDE Disk Driver (**xv6**) (cont.)

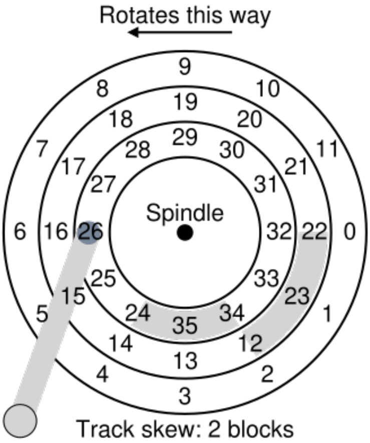


```
static int ide_wait_ready() {
  while (((int r = inb(0x1f7)) & IDE_BSY) || !(r & IDE_DRDY))
    ;  // loop until drive isn't busy
  // return -1 on error, or 0 otherwise
}

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 goes here ...
  outb(0x1f4, (b->sector >> 8) & 0xff);    // ... and here
  outb(0x1f5, (b->sector >> 16) & 0xff);   // ... and here!
  outb(0x1f6, 0xe0 | ((b->dev&1)<<4) | ((b->sector>>24)&0x0f));
  if(b->flags & B_DIRTY){
    outb(0x1f7, IDE_CMD_WRITE);      // this is a WRITE
    outsl(0x1f0, b->data, 512/4);    // transfer data too!
  } else {
    outb(0x1f7, IDE_CMD_READ);       // this is a READ (no data)
  }
}
```

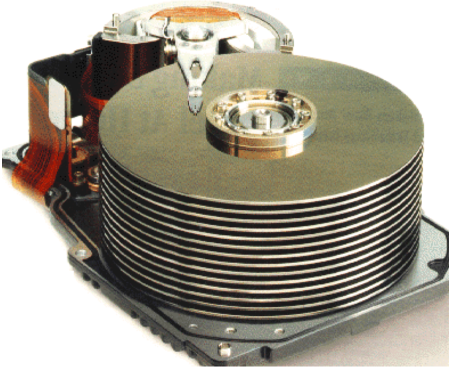
Hard Disk Interface

- Consists of sectors (512 byte blocks)
- Sectors numbered from 0 to $n-1$, **address space**
- Many file systems read/write 4KB at a time
- Sectors written along tracks
- Arm moves head as disk rotates
- Sectors have a *skew* from one track to another
- In **multi-zoned** disk, tracks in different zone have more sectors



Hard Disk Mechanics

- **Platters** has two surfaces and rotate around spindle
- Head and arm on each side of platter
- Rate of Rotation: RPM
- Time to read/write divided into three components:
    - Seek time
    - Rotation time
    - Transfer time



$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer}$$