National University of Singapore

CS2106 Operating System

Second Half Summary Notes

Dong Shaocong A0148008J

April 26, 2018

1 I/O System

Definition 1.1. I/O Devices

- **Communication devices**: Input only (mouse, keyboard); output only (display); Input/output (network card)
- **Storage devices**: Input/output (disk, tape); Input only (CD-ROM)

Definition 1.2. Main tasks of I/O System

- Present **logical** (abstract) view of devices (**hide**: details of hardware interface and error handling)
- Facilitate **efficient** use: overlap CPU and I/O
- Support **sharing** of devices: protection when device is shared (disk), scheduling when exclusive access needed (printer)

Definition 1.3. Block-Oriented Device Interface

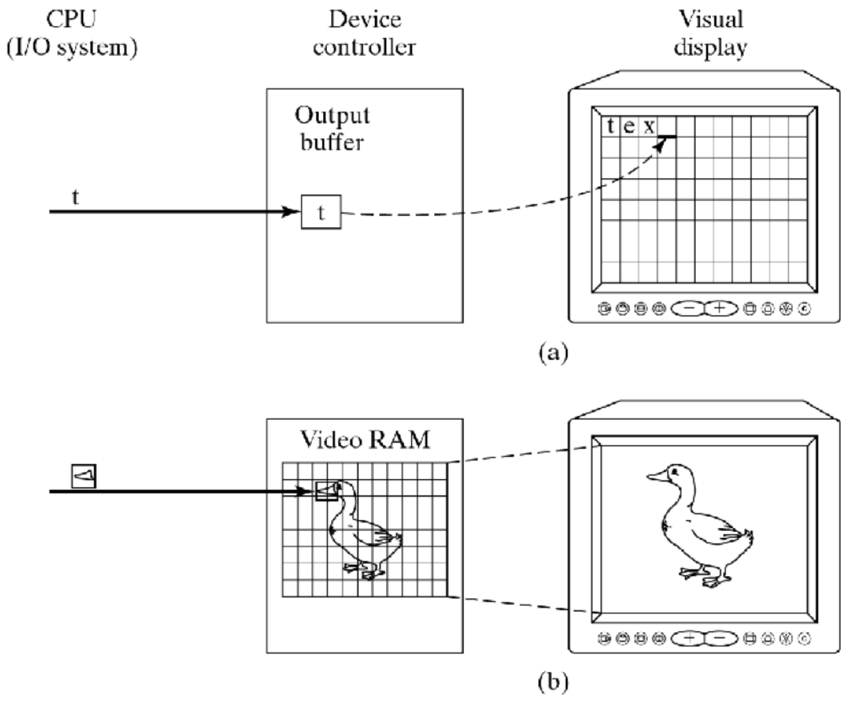
- **Description**: direct access, contiguous blocks, usually fixed block size
- **Operation**:
  - **Open**: verify device is ready, prepare it for access
  - **Read**: Copy a block into main memory
  - **Write**: Copy a portion of main memory to a block
  - **Close**: Release the device
  - ***Note**: these are lower level than those of the FS
- **Application**: Used by File System and Virtual Memory System; Applications typically go through the File System

Definition 1.4. Stream-Oriented Device Interface

- **Description**: character-oriented, sequential access
- **Operation**:
  - **Open**: reserve exclusive access
  - **Get**: return next character of input stream
  - **Put**: append character to output stream
  - **Close**: release exclusive access
  - ***Note**: these too are different from those of the FS but some systems try to present a uniform view of files and devices

Definition 1.5. I/O Devices - Ouput

- **Display monitors**:
  - character or graphics oriented
  - Different data rates: 25 x 80 characters vs 800 x 600 pixels (1B allows 256 colors) Refresh 30-60 times/s for video
- **Printers (ink jet, laser)**
- **Interface**:
  - **write** to controller buffer
  - **wait** for completion
  - handle **errors**

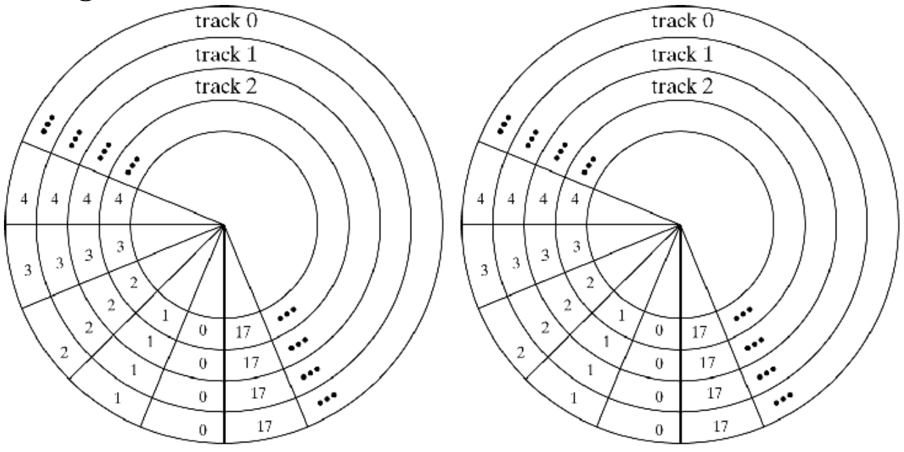


Definition 1.6. I/O Devices - Input Keyboards, pointing devices (mouse, trackball, joystick), scanners. **Interface**:

- device generates interrupt when data is ready
- read data from controller buffer
- low data rates, not time-critical

Definition 1.7. I/O Devices - Storage

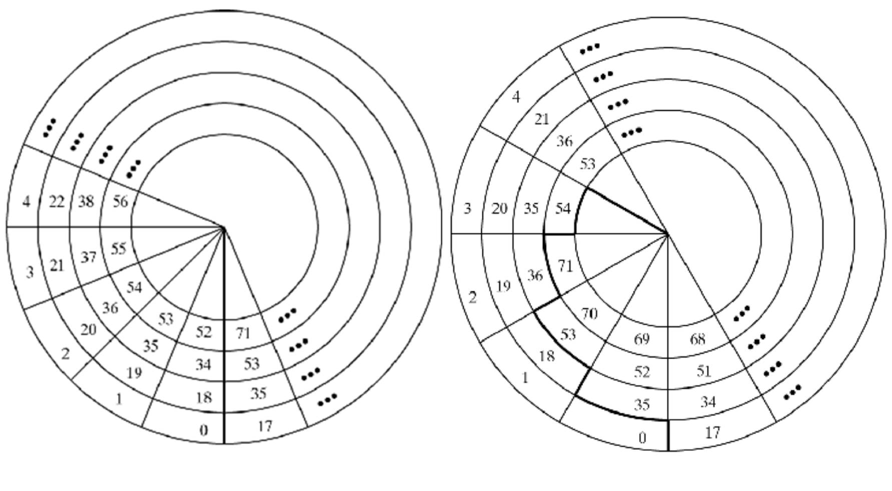
- Surface, tracks/surface, sectors/track, bytes/sector
- All sectors numbered sequentially $0..(n-1)$, device controller provides mapping



Track skew: account for seek-to-next-track to minimise rotational delay

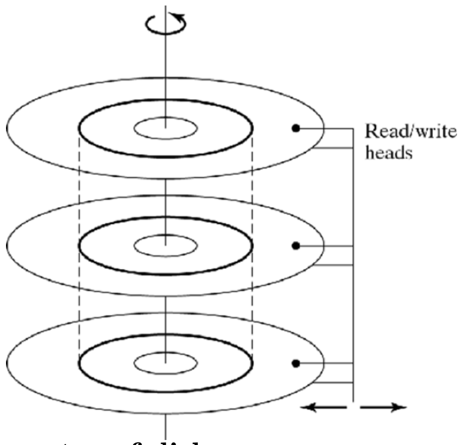
$$\frac{\text{track to track seek time}}{\text{rotational time per track}} \times \text{sectors per track} = \text{offset}$$

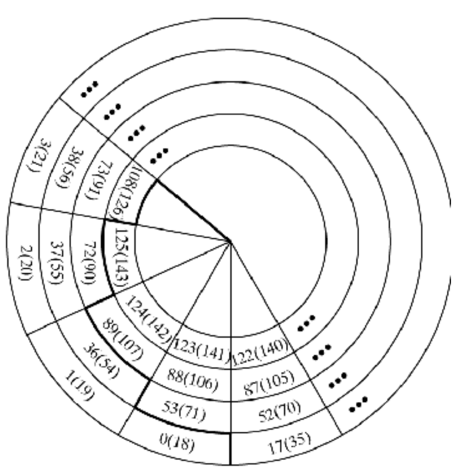
Round up the result



Double-sided or multiple surfaces

- Tracks with same diameter = **cylinder**
- Sectors are numbered within cylinder consecutively to **minimise seek time**





Critical issue: data transfer rates of disks

- **Sustained** rate: continuous data delivery
- **Peek** rate: : transfer once read/write head is in place; depends on rotation speed and data density

Example 1.1. Transfer rate calculation: 7200 rpm, 100 sectors/track, 512 bytes/sector

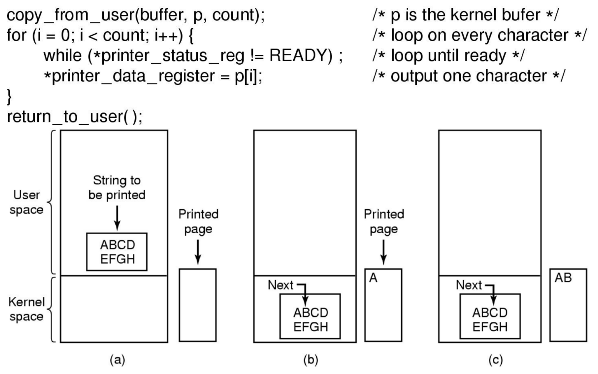
- What is the **peak** transfer rate?

$$\frac{7200}{60} \times 100 \times 512\text{byte/s}$$

- What is the **sustained** transfer rate? - Depends on file organization

Definition 1.8. I/O programming - access the I/O devices

- **Polling** (You with a broken ringer)
  - Consider a process that prints ABCDEFGH on the printer: The OS then copies character by character onto the printers latch, and the printer prints it out.
    1. Copy the first character and advance the buffers pointer.
    2. Check that the printer is ready for the next character. If not, wait. This is called "busy-waiting" or "polling".
    3. Copy the next character. Repeat until buffer is empty.

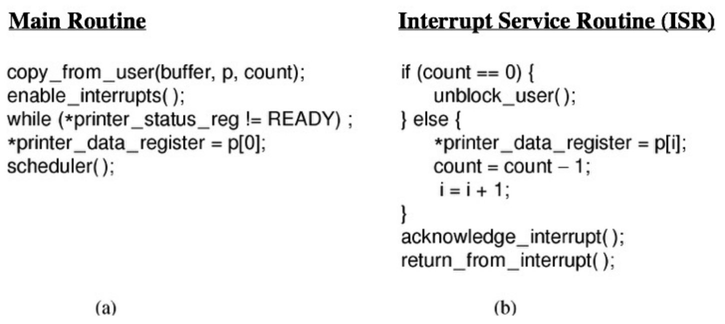


```
copy_from_user(buffer, p, count);                    /* p is the kernel bufer */
for (i = 0; i < count; i++) {                        /* loop on every character */
     while (*printer_status_reg != READY) ;          /* loop until ready */
     *printer_data_register = p[i];                  /* output one character */
}
return_to_user( );
```

  Issue: It takes perhaps 10ms to print a character. During this time, the CPU will be busy-waiting until the printer is done printing. On a 3.2 GHz processor this is equivalent to wasting 320,000,000 instructions! Extremely inefficient use of CPU as most polls are likely to fail. On the other hand not polling risks losing data.

- **Interrupt-driven I/O** (You with a fully functioning phone)

- After the string is copied, the OS will send a character to the printer, then switch to a task.
- When the printer is done, it will interrupt the CPU by asserting one of the interrupt request (IRQ) lines on the CPU.
- **Comment**: Some overhead when the hardware is ready, but much less than with polling.



Main Routine

```
copy_from_user(buffer, p, count);
enable_interrupts( );
while (*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler( );
```

(a)

Interrupt Service Routine (ISR)

```
if (count == 0) {
    unblock_user( );
} else {
    *printer_data_register = p[i];
    count = count – 1;
     i = i + 1;
}
acknowledge_interrupt( );
return_from_interrupt( );
```

(b)

- **Direct memory access** (You have an answering machine)
  - **Driver (CPU) operation to input sequence of bytes:**

```
write_reg(mm_buf, m); // give parameters
write_reg(count, n);
write_reg(opcode, read); // start op
block to wait for interrupt;
```

    * Writing opcode triggers DMA controller
    * DMA controller issues interrupt after n chars in memory
  - **Cycle Stealing**:
    * DMA controller competes with CPU for memory access
    * generally not a problem because: 1. Memory reference would have occurred anyway; 2. CPU is frequently referencing data in registers or cache, bypassing main memory.

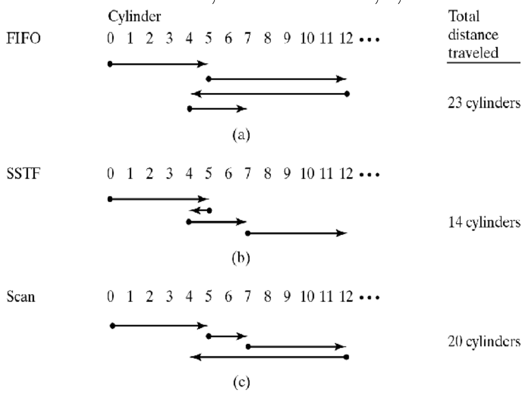
Definition 1.9. Device Management

- **Disk Scheduling**: Requests for different blocks arrive concurrently from different processes
- Minimize **rotational delay**: re-order requests to blocks on each track to access in one rotation
- Minimize **seek time**: Conflicting goals: Minimize total travel distance; Guarantee fairness

Algorithm 1.1. Device Management

- **FIFO**: requests are processed in the order of arrival: simple, fair, but inefficient
- **SSTF** (Shortest Seek Time First): most efficient but prone to starvation
  - always go to the track thats nearest to the current positions
- **Scan** (Elevator): fair, acceptable performance
  - maintain a direction of travel
  - always proceed to the nearest track in the current direction of travel
  - if there is no request in the current direction, reverse direction

Example: assume moving from 0 to 5; then 12,4,7 arrive



Remark. block size trade off Larger block sizes means fewer blocks that the OS needs to manage, less overhead. Disadvantage is that there will be greater wastage within each block (internal fragmentation)

Method 1.1. Disk access time calculation Time is takes to read one block of data:

1. **Switch time**: This is the time it takes for the drive to choose the correct side of the correct platter. (can be negligible)
2. **Seek time**: This is the time it takes a drive's arm to move to the correct track.
3. **Rotational delay**: This is the time it takes for the correct block to move underneath the head. Conventionally, taken to be $\frac{T_r}{2}$, where T_r is the time for one revolution.
4. **Transfer delay**: This is the time taken to actually transfer the block. If the disk can transfer M bytes per second and a block is B bytes long, then this time is $\frac{B}{M}$.

- **Peak data transfer rate**: The drive cannot be transferring more in one second than the amount of data in one track, multiplied by the number of times that track goes past the head per second

$$\text{Blocks / Track} = 16, \text{Bytes / Block} = 32768$$
$$\text{Bytes / Track} = 16 \times 32768 = 524288 \text{ bytes}, 7200rpm = 120rps$$
$$\text{peak throughput} = 120 \times 524288 = 62.9\text{Megabytes/Sec}(1\text{megabyte/sec} = 10^6\text{bytes/sec})$$

- **Time taken on average to read one block of data**:

$$\text{Average disk read time} = \text{switching time+seek time+rotational delay+transfer time}$$
$$\text{rotational delay} = \frac{1}{2} \times \frac{1}{7200/60} = 0.0042s$$
$$\text{Data through put} = 50 \text{ megabits per second}$$
$$\text{data transfer time} = 32768 \times 8/(50,000,000) = 0.0052s = 5.2ms$$

Note: For storage we conventionally use 1KB = 1024 bytes, not 1000 bytes. However we use 1KB = 1000 bytes for throughput.

Remark. Why interrupts have overheads? Overheads involved include detecting the interrupt, consulting the vector table to get the interrupt handler address, pushing the current PC to the stack, switching to kernel mode, and vectoring to the interrupt handler and switching back to user mode.

Example 1.2. Compare different I/O Programming methods
Question: Printing 250 characters, how many cycles are wasted on polling?
Assumption: Assuming the printer is currently free, (polling) time for the first character should be negligible. Almost all CPUs can execute at least 1 instruction per cycle

- **Polling**: For subsequent 249 characters, there is a 10ms poll time between characters. Each clock cycle is 10ns, so this corresponds to approximately 1,000,000 clock cycles per poll, or total of about 249,000,000 cycles to print the entire 250 characters.
- **Interrupt Driven**: For subsequent 249 characters we have total 300ns interrupt time, = 74,700 ns. Each clock cycle is 10ns, so this is equal to 7,470 clock cyles, a great reduction from the original 249,000,000 cycles.
- **DMA**: Total time taken = setup time + end of transfer interrupt processing time = 700ns. This is equal to 70 instructions.

Example 1.3. Disk access time calculation

- **What is the total number of blocks read by this program?**
Bytes per block =128; Total number of bytes read and written (calculated from the program) = 32768; Number of blocks to be read is $\frac{32768}{128} = 256$. Assuming one directory block read and one inode block read per file, then total = 256 + 2 = 258.

- **Assume that there is no caching or buffering (so not affected by the caching policies), and that the disk blocks are *not skewed*. What is the total time in milliseconds that this program spends reading and writing the files?**
  - **Calculation of # of cylinders to read**
    # of tracks per cylinder = 4; # of blocks per cylinder = 16 × 4= 64;
    # of cylinders to read = 256 / 64 = 4 (for the file itself)
  - **Time to read directory and inode**
    Seek time to directory block: 12ms
    Rotational delay to directory block: 3600rpm=60rps. Delay=1/(2×60)=8.3ms
    Transfer delay (to read directory / inode Block): 128 / 100,000,000=0.00128ms
    Total: (12+8.3+0.00128)×2=40.6ms
  - **Time to read first cylinder**
    Seek time to first cylinder: 12ms
    Rotational delay to first block: 8.3ms
    Bytes per cylinder = 64 × 128 = 8192
    Time to read one cylinder = 8192 / 100,000,000=0.08192ms
    Total: (12+8.3+0.08192)=20.382ms
  - **Time to read the remaining 3 cylinders**
    Seek time to adjacent track: 2ms
    Rotation delay still 8.3ms; Total time to read data still: 0.08192ms
    Total: $3 \times (2 + 8.3 + 0.08192) \times 3 = 31.146$ms

  Total time taken: 40.6+20.382+31.146=92.128ms
- **Once again assuming that there is no caching or buffering, repeat part b. assuming that the blocks are *skewed* such that rotational delay is completely eliminated when the head moves to the next cylinder to continue writing.**
  Same timing, except no rotational delay for additional 3 cylinders. $92.128 - 3 \times 8.3 = 67.228$ms