# Devices, Input, and Output

# Example: Disk Device Driver

- Must provide an illusion of a linear array of sectors that are numbered like elements of an array
- Sector *s* on track *t* in cylinder *c* is numbered

$$a = ((c \times (\#\text{tracks/cylinder}) + t) \times (\#\text{sectors/track}) + s$$

  so rather than referring to ($c$, $t$, $s$), kernel can refer to $a$

- Also must reduce effect of latencies of accessing disk
  - Overlap I/O and computation
  - Arrange large objects only one seek is needed to read/write them
  - Order outstanding disk requests

# Ordering Disk Requests: Assumptions

- Only 1 disk drive
- All I/O requests are for single equal-size blocks
- Requested blocks distributed over disk
- Disk has only 1 moveable arm with all heads on it
- Seek latency is linear in the number of tracks crossed
  - Ignores disk controller using sectors from tracks at end of disk to replace bad sectors
- Disk controller does not introduce appreciable delays
- Read, write requests are equally fast

# Key Points for Evaluating Disk Access Policies

- How long must requests wait as a function of load
  - Frequency of requests, measured in requests per time unit
- Mean, variance of waiting time for each request
  - Low mean, high variance means some requests will take a long time

# Disk Access Policies

- First come, first serve (FCFS)
- Pickup
- Shortest seek time first (SSF, SSTF)
- SCAN
- N-Step SCAN
- C-SCAN

# First Come First Serve (FCFS)

- Requests cannot be starved; all get serviced eventually
- Fairly low variance, but becomes saturated easily
  - Load becomes greater than driver can handle, so requests always waiting
- Problems
  - Every request is likely to require a seek
  - Great for low loads, but for high loads the latencies increase the mean of waiting time

# Pickup

- FCFS but, as the head moves to the next track, any queued requests for the tracks it passes over are serviced
- For high loads, this decreases the mean waiting time a bit

# Shortest Seek Time First (SSF, SSTF)

- Service the request lying on the closest track
  - Saturates at the highest load of any of these policies
- Problems:
  - Starvation; this means the disk can't keep up with disk I/O requests, usually indicating a more severe problem such as thrashing
  - Variance larger than that of FCFS as innermost and outermost tracks are serviced less frequently than others

# SCAN

- Head moves from outermost track to innermost, then back, etc., servicing requests along the way
- It reduces the problem of the innermost and outermost tracks getting less service
  - So it lowers the variance
- Problem: still subject to starvation

# N-Step SCAN

- Like SCAN, but when a sweep begins (going in or out), the requests in the device queue *at that time* are the only ones serviced
- Arrivals after that wait for the next sweep to begin
- Starvation not possible
- Reduces variance even more

# Circular SCAN (C-SCAN)

- Like SCAN, but requests are serviced only when the head is moving from outermost to innermost track
  - It “jumps back” from the innermost track to the outermost track
- Eliminates problem of innermost, outermost tracks getting less frequent service
- Waiting times also more uniform

# LOOK variants

- With SCAN, heads always goes to the innermost and outermost tracks, even if there are no requests for service involving those tracks
- LOOK variants have the heads go only as far as there are outstanding requests, and then have the head reverse direction
- Example: 200 track disk, requests for tracks 150, 90 and 70, with heads currently at 110 and moving inward
  - Handle request for track 90, then track 70
  - Change direction at track 70 rather than continue inward

# Comparison

- Disk has 200 tracks (tracks numbered from 199 to 0)
- Head is currently at track 53
- Set of requests in the queue is

  98 183 37 122 14 124 65 67

- For LOOK and SCAN, assume head is moving inward

# Order of Service

| policy | order of servicing | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| FCFS | 98 | 183 | 37 | 122 | 14 | 124 | 65 | 67 |
| PICKUP | 65 | 67 | 98 | 122 | 124 | 183 | 37 | 14 |
| SSTF | 65 | 67 | 37 | 14 | 98 | 122 | 124 | 183 |
| SCAN | 37 | 14 | 65 | 67 | 98 | 122 | 124 | 183 |
| LOOK | 37 | 14 | 65 | 67 | 98 | 122 | 124 | 183 |
| C-SCAN | 37 | 14 | 183 | 124 | 122 | 98 | 67 | 65 |
| C-LOOK | 37 | 14 | 183 | 124 | 122 | 98 | 67 | 65 |

# Head Motion

Number of cylinders heads move over to service each request

| policy | number of cylinders moved over | | | | | | | | total | mean | std dev |
|---|---|---|---|---|---|---|---|---|---|---|---|
| FCFS | 45 | 85 | 146 | 85 | 108 | 110 | 59 | 2 | 640 | 80.00 | 44.47 |
| PICKUP | 12 | 2 | 31 | 24 | 2 | 59 | 146 | 23 | 299 | 37.38 | 47.57 |
| SSTF | 12 | 2 | 30 | 23 | 84 | 24 | 2 | 59 | 236 | 29.50 | 28.62 |
| SCAN | 16 | 23 | 79 | 2 | 31 | 24 | 2 | 59 | 236 | 29.50 | 26.97 |
| LOOK | 16 | 23 | 51 | 2 | 31 | 24 | 2 | 59 | 208 | 26.00 | 20.72 |
| C-SCAN | 16 | 23 | 231 | 59 | 2 | 24 | 31 | 2 | 388 | 48.35 | 75.93 |
| C-LOOK | 16 | 23 | 169 | 59 | 2 | 24 | 31 | 2 | 326 | 40.75 | 54.89 |

# Waiting Time

Time each request has to wait for service, in terms of cylinders crossed

| policy | cumulative number of cylinders moved over | | | | | | | | total | mean | std dev |
|---|---|---|---|---|---|---|---|---|---|---|---|
| FCFS | 45 | 130 | 276 | 361 | 469 | 579 | 638 | 640 | 3138 | 392.25 | 228.78 |
| PICKUP | 12 | 14 | 45 | 69 | 71 | 130 | 276 | 299 | 916 | 114.5 | 113.24 |
| SSTF | 12 | 14 | 44 | 67 | 151 | 175 | 177 | 236 | 876 | 109.50 | 85.60 |
| SCAN | 16 | 39 | 118 | 120 | 151 | 175 | 177 | 236 | 1032 | 129.00 | 73.12 |
| LOOK | 16 | 39 | 90 | 92 | 123 | 147 | 149 | 208 | 864 | 108.00 | 62.37 |
| C-SCAN | 16 | 39 | 270 | 329 | 331 | 355 | 386 | 388 | 2114 | 264.25 | 150.91 |
| C-LOOK | 16 | 39 | 208 | 267 | 269 | 293 | 324 | 326 | 1742 | 217.75 | 123.33 |

# Optimizations

- Sector queueing
  - Policy to minimize rotational latency
  - Order requests for the same track so they can be serviced with a minimum number of rotations of disk
  - Implementation: each sector has its own queue for requests
  - Used only when there are extremely heavy loads
- Caching
  - Read extra sectors following the one you want

# Process Interface

- Concept of file underlies interface
    - More about this next
- Enables processes to interact with devices
    - Also kernel structures such as /dev/null and /proc
- Need at least 1 special system call to handle device-specific functions

# System Calls: *open*, *close*

- *open* makes file accessible to process
- Form: descriptor = *open*(file, how, . . . )
  - Now process uses descriptor to refer the file
  - If device not ready, process may block or call may return error code
  - Call also checks privileges to ensure user can open the file
- *close* disassociates file from process
- Form: *close*(descriptor)
  - Device driver does any needed clean-up

# System Calls: *seek*

- *seek* positions pointer associated with descriptor as instructed
- Form: *seek*(descriptor, where)
  - Read/write pointer repositioned to where
  - Examples: go to arbitrary location in file, position on tape
- Linux: *lseek*(descriptor, offset, whence)
  - whence indicates if offset is from beginning or end of descriptor, or current position of read/write pointer
  - Returns new position on success, –1 on error; but –1 may be valid value
  - Disambiguate using *errno*

# System Calls: *seek*

- Linux: *lseek* example

```
external int errno;
. . .
errno = 0;
if (lseek(desc, offset, SEEK_SET) == -1 && errno != 0){
        /* handle error */
}
else{
        /* handle success */
}
```

# System Calls: *read*

- Transfers data from descriptor object to memory
- Form: nread = *read*(descriptor, memory address, amount)
  - Reads nread bytes, which is at most amount
  - Returns 0 on end of file, error code on error
- Form: nread = *readv*(descriptor, memory list, list length)
  - Like read, but reads data into multiple memory locations
  - Locations given in memory list; also number of bytes for each
  - Returns number of bytes read, or 0 on end of file, error code on error

# System Calls: *write*

- Transfers data from memory to descriptor object
- Form: nbyte = *write*(descriptor, memory address, amount)
  - Outputs nbyte bytes, which is at most amount
  - Returns error code on error
- Form: nbyte = *writev*(descriptor, memory list, list length)
  - Like write, but writes data from multiple memory locations
  - Locations given in memory list; also number of bytes for each
  - Returns number of bytes written, error code on error

# Blocking vs. Non-Blocking Read and Write

- Blocking transfer is synchronous
  - So when the next statement is executed, transfer has been completed
- Non-blocking transfer is asynchronous
  - So next statement executed whether or not transfer has been completed
- Two ways to determine when non-blocking transfer completes:
  - Use polling by checking an indicator
  - Use interrupts

# Non-Blocking Read and Write

- Process requests interrupt from kernel when transfer completes
  - System call may arrange this; on Linux, it's SIGIO
- Process must arrange to catch interrupt and process it
  - Usually a system call like *handler*(signal, function)
- If process does need to block until transfer is complete, need a system call like *wait*(descriptor, timeout)
  - Blocks until transfer to or from descriptor completes
  - If not completed by timeout, then wake up and continue
- Never modify memory involved in transfer until transfer completes
  - Results are undefined

# System Calls: *control*

- Used for device-specific actions
- Form: *control*(descriptor, action, . . .)
  - action is device specific and may require other parameters
- Linux example: make FAT file system read-only:

  attrmask = ATTR_RO;

  *ioctl*(desc, FAT_IOCTL_SET_ATTRIBUTES, &attrmask)

# Linux Examples

- Insert ch into (terminal) input queue:

    toinsert = ch;

    *ioctl*(desc, TIOCSTI, &toinsert)

- Give up role of controlling terminal:

    *ioctl*(desc, TIOCNOTTY)