Slide 1

Week 7

COMP3231 Operating Systems

2005 S2

- IO Management — Part 2
- Scheduling

Slide 2

I/O Management

- ➜ Categories of I/O devices and their integration with processor and bus
- ➜ Design of I/O subsystems
- ➜ I/O buffering
- ➜ Disk scheduling
- ➜ RAID

Slide 3

Disk Scheduling

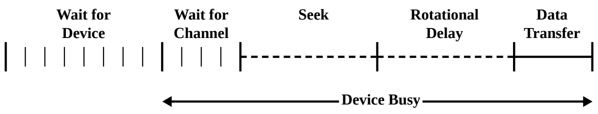
- ➜ Disk performance is critical for system performance
- ➜ Management and ordering of disk access requests have strong influence on
  - access time
  - bandwidth
- ➜ Important to optimise because:
  - huge speed gap between memory and disk
  - disk throughput extremely sensitive to
    - request order ⇒ disk scheduling
    - placement of data on disk ⇒ file system design
- ➜ Request scheduler must be aware of disk geometry

Slide 4

Disk performance parameters:

- ➜ Disk is moving device ⇒ must position correctly for I/O
- ➜ Execution of a disk operation involves:
  - Wait time: the process waits to be granted device access
    - Wait for device: time the request spends in a wait queue
    - Wait for channel: time until a shared I/O channel is available
  - Access time: time the hardware needs to position the head
    - Seek time: position the head at the desired track
    - Rotational delay (latency): spin disk to the desired sector
  - Transfer time: sectors to be read/written rotate below the head

Slide 5



Slide 6

PERFORMANCE PARAMETERS

- Seek time T_s: Moving the head to the required track
  - not linear in the number of tracks to traverse:
    - startup and settling time
  - Typical average seek time: a few milliseconds
- Rotational delay:
  - rotational speed, r, of 5,000 to 10,000rpm
  - At 10,000rpm, one revolution per 6ms $\Rightarrow$ average delay 3ms
- Transfer time:
  - to transfer b bytes, with N bytes per track:

$$T = \frac{b}{rN}$$

  - Total average access time:

$$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

Slide 7

A Timing Comparison:

- $T_s = 2$ ms, $r = 10,000$ rpm, 512B sect, 320 sect/track
- Read a file with 2560 sectors (= 1.3MB)
- File stored compactly (8 adjacent tracks):

Read first track	
Average seek	2ms
Rot. delay	3ms
Read 320 sectors	6ms
	11ms

$\Rightarrow$ All sectors: $11 + 7 * 9 = 74ms$

- Sectors distributed randomly over the disk:

Read any sector	
Average seek	2ms
Rot. delay	3ms
Read 1 sector	0.01875ms
	5.01875ms

$\Rightarrow$ All: $2560 * 5.01875 = 12,848ms$

Slide 8

DISK SCHEDULING POLICY

Observation from the calculation:

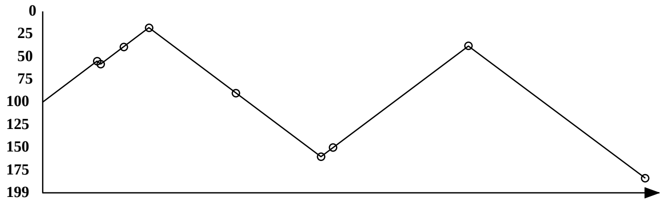
- Seek time is the reason for differences in performance
- For a single disk there will be a number of I/O requests
- Processing in random order leads to worst possible performance
- We need better strategies

Slide 9

First-in, first-out (FIFO):

- ➜ Process requests as they come in

Request tracks: 55, 58, 39, 18, 90, 160, 150, 38, 184



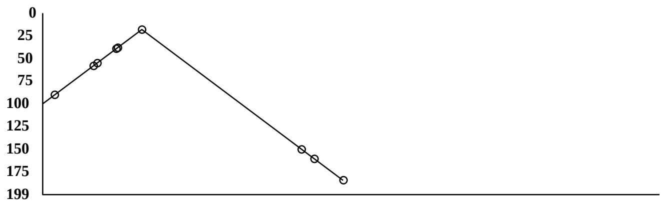
- ➜ Fair (no starvation!)
- ➜ Good for few processes with clustered requests
- ➜ deteriorates to random if there are many processes

Slide 10

Shortest Service Time First (SSTF):

- ➜ Select the request that minimises seek time

Request tracks: 55, 58, 39, 18, 90, 160, 150, 38, 184



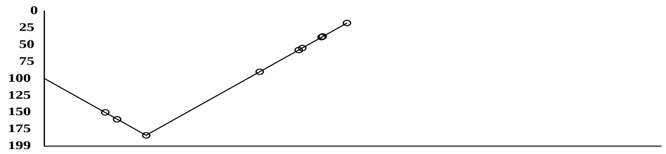
- ➜ service order: 90, 58, 55,39,18, 150,160,184
- ➜ Minimising locally may not lead to overall minimum!
- ➜ Can lead to starvation

Slide 11

SCAN (Elevator): Move head in one direction

- ➜ services requests in track order until it reaches last track, then reverse direction

Request tracks: 55, 58, 39, 18, 90, 160, 150, 38, 184



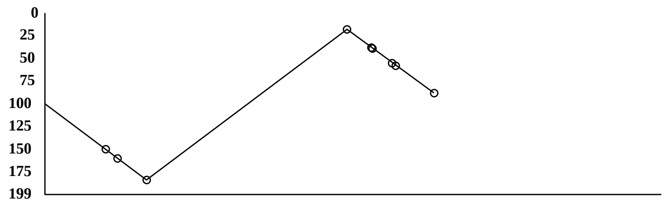
- ➜ service order: 150,160,184, (200), 90, 58, 55,39,18, (0)
- ➜ Similar to SSTF, but avoids starvation
- ➜ LOOK: variant of SCAN, moves head only to last request of one direction: 150,160,184, 90, 58, 55,39,18
- ➜ SCAN/LOOK are biased against region most recently traversed
- ➜ Favour innermost and outermost tracks
- ➜ Makes poor use of sequential reads (on down-scan)

Slide 12

Circular SCAN (C-SCAN):

- ➜ Like SCAN, but scanning to one direction only
  - When reaching last track, go back to first non-stop

Request tracks: 55, 58, 39, 18, 90, 160, 150, 38, 184



- ➜ Better use of locality (sequential reads)
- ➜ Better use of disk controller's read-ahead cache
- ➜ Reduces the maximum delay compared to SCAN

Slide 13

N-step-SCAN:

- ➜ SSTF, SCAN & C-SCAN allow device monopolisation
  - process issues many requests to same track
- ➜ N-step-SCAN segments request queue:
  - subqueues of length N
  - process one queue at a time, using SCAN
  - added new requests to other queue

Slide 14

FSCAN:

- ➜ Two queues
  - one being presently processed
  - other to hold new incoming requests

Slide 15

Disk scheduling algorithms:

Name	Description	Remarks
Selection according to requestor		
RSS	Random scheduling	For analysis and simulation
FIFO	First in, first out	Fairest
PRI	By process priority	Control outside disk magmt
LIFO	Last in, first out	Maximise locality & utilisation
Selection according to requested item		
SSTF	Shortest seek time first	High utilisation, small queues
SCAN	Back and forth over disk	Better service distribution
C-SCAN	One-way with fast return	Better worst-case time
N-SCAN	SCAN of N recs at once	Service guarantee
FSCAN	N-SCAN (N=init. queue)	Load sensitive

Slide 16

DISK SCHEDULING

- ➜ Modern disks:
  - seek and rotational delay dominate performance
  - not efficient to read only few sectors
  - cache contains substantial part of currently read track
- ➜ assume real disk geometry is same as virtual geometry
- ➜ if not, controller can use scheduling algorithm internally

So, does OS disk scheduling make any difference at all?

Linux 2.4.

Slide 17

- ➜ Used a version of C-SCAN
- ➜ no real-time support
- ➜ Write and read handled in the same way — read requests have to be prioritised

Linux 2.6.

Slide 18

Deadline I/O scheduler:

- ➜ two additional queues: FIFO read queue with deadline of 5ms, FIFO write with deadline of 500ms
- ➜ request submitted to both queues
- ➜ if request expires,scheduler dispatches from FIFO queue
- ➜ Performance:
  - ✔ seeks minimised
  - ✔ requests not starved
  - ✔ read requests handled faster
  - ✘ can result in seek storm, everything read from FIFO queues

Slide 19

Anticipatory Scheduling:

- ➜ Same, but anticipates dependent read requests
- ➜ After read request: waits for a few ms
- ➜ Performance
  - ✔ can dramatically reduce the number of seek operations
  - ✘ if no requests follow, time is wasted

Performance

Slide 20

- ➜ Writes
  - similar for writes
  - deadline scheduler slightly better than AS
- ➜ Reads
  - deadline: about 10 times faster for reads
  - as: 100 times faster for streaming reads

Slide 21

Disk Cache

- ➜ Buffer in main memory for disk sectors
- ➜ Contains a copy of some of the sectors on the disk

Design Considerations:

- ➜ transfer of data from cache to process memory
- ➜ using shared memory approach to map memory area into process memory

Slide 22

Least recently used:

- ➜ The block that has been in the cache the longest with no reference to it is replaced
- ➜ The cache consists of a stack of blocks
- ➜ Most recently referenced block is on the top of the stack
- ➜ When a block is referenced or brought into the cache, it is placed on the top of the stack
- ➜ The block on the bottom of the stack is removed when a new block is brought in
- ➜ Blocks don't actually move around in main memory
- ➜ A stack of pointers is used

Slide 23

Least frequently used:

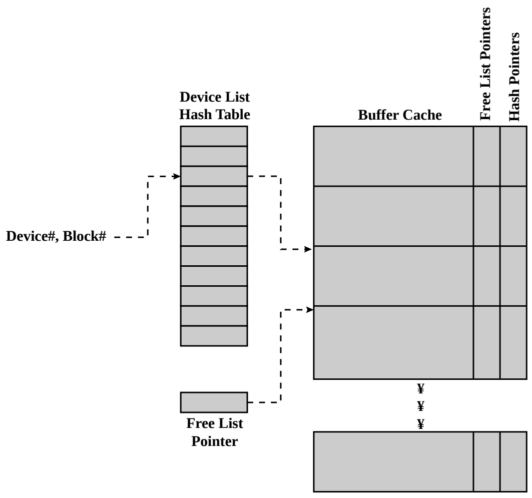
- ➜ The block that has experienced the fewest references is replaced
- ➜ A counter is associated with each block
- ➜ Counter is incremented each time block accessed
- ➜ Block with smallest count is selected for replacement
- ➜ Some blocks may be referenced many times in a short period of time and then not needed any more

Slide 24

UNIX Buffer Cache: Three lists maintained to manage buffer:

- ➜ Free list: free slots in the cache (LRU)
- ➜ Device list: all buffers associated with each disk
- ➜ Driver I/O queue: list of all buffers waiting for the completion of an I/O request

Slide 25



Slide 26

RAID

- ➜ CPU performace has improved exponentially
- ➜ disk performance only by a factor of 5 to 10
- ➜ huge gap between CPU and disk performance

Parallel processing used to improve CPU performance.

Question: can parallel I/O be used to speed up and improve reliability of I/O?

Slide 27

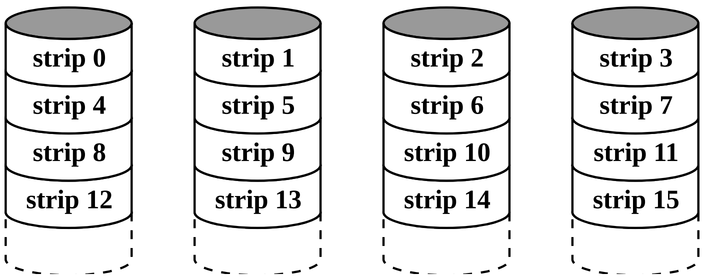
RAID: Redundant Array of Inexpensive/Independent Disks

Multiple disks for improved performance or reliability:

- ➜ Set of physical disks
- ➜ Treated as a single logical drive by OS
- ➜ Data is distributed over a number of physical disks
- ➜ Redundancy used to recover from disk failure (exception: RAID 0)
- ➜ There is a range of standard configurations
  - numbered 0 to 6
  - various redundancy and distribution arrangements

Slide 28

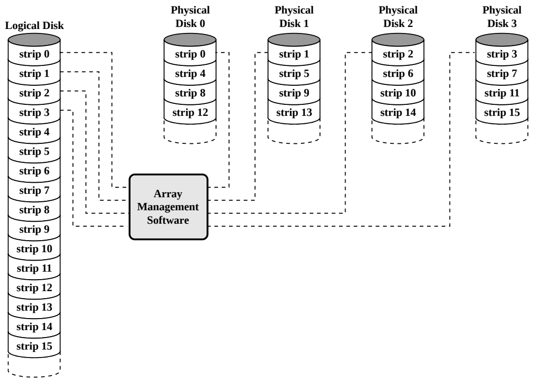
RAID 0 (striped, non-redundant):



- ➜ controller translates single request into separate requests to single disks
- ➜ requests can be processed in parallel
- ➜ simple, works well for large requests
- ➜ does not improve on reliability, no redundancy

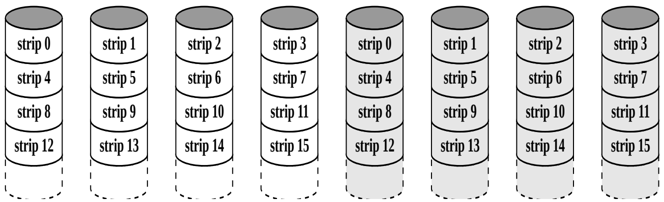
Slide 29

Data mapping for RAID 0:



Slide 30

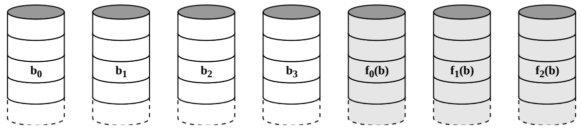
RAID 1 (mirrored, $2\times$ redundancy):



- ➜ duplicates all disks
- ➜ write: each request is written twice
- ➜ read: can be read from either disk

Slide 31

RAID 2 (redundancy through Hamming code):



- ➜ strips are very small (single byte or word)
- ➜ error correction code across corresponding bit positions
- ➜ for n disks, $log_2 n$ redundancy
- ➜ expensive
- ➜ high data rate, but only single request

Slide 32

Error Correction and Redundancy

Just keeping two copies doesn't necessarily help to correct the error:

Example:

Original	Copy
1 0 0 1 0 1 1 1	1 0 0 1 0 0 1 1

?

1 0 0 1 0 1 1 1 1 0 0 1 0 0 1 1

- ➜ it is not clear if the error occured in the copy or the original
- ➜ no error correction possible

Slide 33

Hamming Distance between two bit-strings: The number of bits in which they differ.

Slide 34

- ➜ One-bit error detection could be achieved much cheaper (parity bit)
- ➜ How much redundancy is necessary for a one bit error correction?

Slide 35

HAMMING CODE

For every four bits of data, three parity bits

① Parity (3,5,7)
② Parity (3,6,7)
③ Data
④ Parity (5,6,7)
⑤ Data
⑥ Data
⑦ Data

Slide 36

7	6	5	4	3	2	1	
D		D		D			Parity bit 1
D	D			D			Parity bit 2
D	D	D					Parity bit 4
1	1	0	0	1	1	0	Data

Slide 37

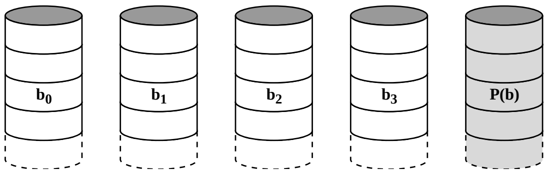
7	6	5	4	3	2	1	
1	1	0	0	1	1	0	Data
0	1	0	0	1	1	0	Corrupted Data
D		D		D			1 : bit 1 not ok
D	D			D			1: bit 2 not ok
D	D	D					1: bit 4 not ok

Error Correction:

- ➜ bit 111 (ie, 7) is corrupted
- ➜ two bit errors can be detected, but not corrected

Slide 38

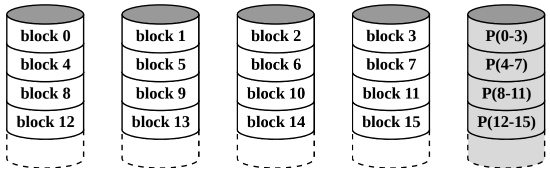
RAID 3 (bit-interleaved parity):



- ➜ strips are very small (single byte or word)
- ➜ simple parity bit based redundancy
- ➜ error detection
- ➜ partial error correction (if offender is known)

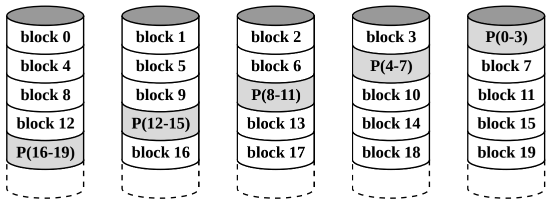
Slide 39

RAID 4 (block-level parity):



Slide 40

RAID 5 (block-level distributed parity):



Slide 41

RAID 6 (dual redundancy):