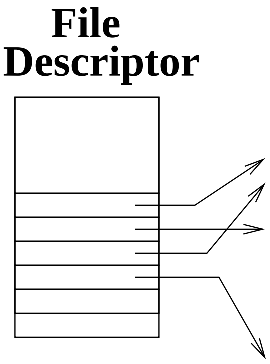
How do we find and organize files on the disk? (continued)

Indexed files:

- OS keeps an array of block pointers for each file.
- The user or OS must declare the maximum length of the file when it is created.
- OS allocates an array to hold the pointers to all the blocks when it creates the file, but allocates the blocks only on demand.
- OS fills in the pointers as it allocates blocks.



Advantages:

- Not much wasted space.
- Both sequential and random accesses are easy.

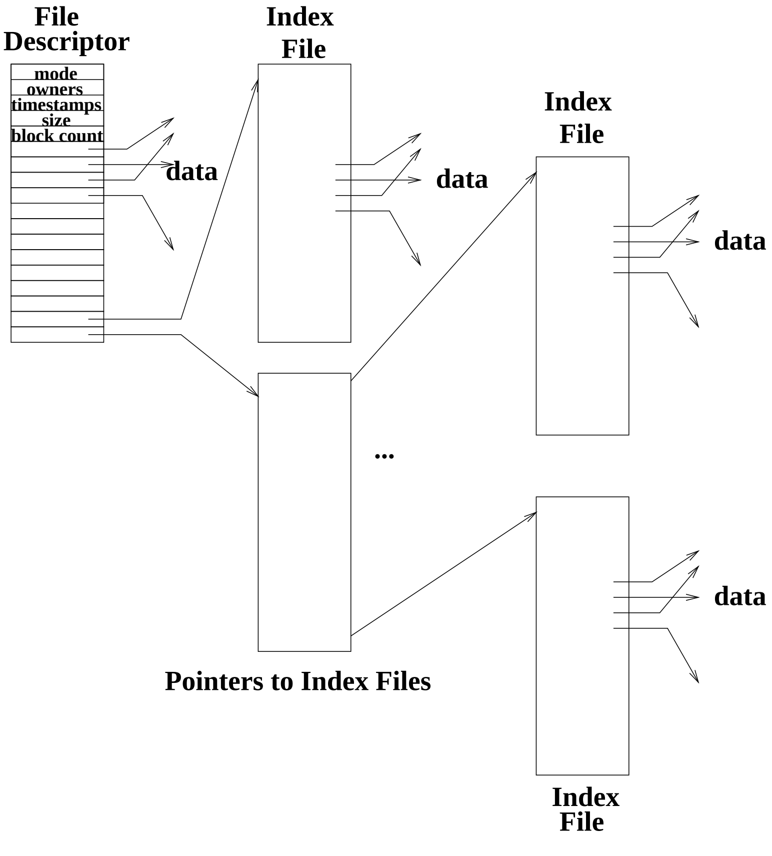
Disadvantages:

- Sets a maximum file size.
- Lots of seeks because data are not contiguous.

How do we find and organize files on the disk?

Multilevel indexed files: BSD UNIX 4.3

- Each file descriptor contains 14 block pointers.
- First 12 pointers point to data blocks.
- 13^{th} pointer points to a block of 1024 pointers to 1024 more data blocks. (One indirection.)
- 14^{th} pointer points to a block of pointers to indirect blocks. (Two indirections.)



Multilevel indexed files: BSD UNIX 4.3

Advantages:

- Simple to implement
- Supports incremental file growth
- Small files?

Disadvantages:

- Indirect access is inefficient for random access to very large files.
- Lots of seeks because data are not contiguous.

- Is the file size bounded?
- What could the OS do to get more contiguous access and fewer seeks?

Free-Space Management

- Need a free-space list to keep track of which disk blocks are free (much as we need a free-space list for main memory)
- Need to be able to find free space quickly and release space quickly.
- A common implementation is a *bitmap* data structure.
- A bitmap is a long array of bits, with one bit for each block on the disk.
- If the bit is 1, the block is free. If the bit is 0, the block is allocated.
- Can quickly determine if any page in the next 32 is free, by comparing the word to 0. If it is 0, all the pages are in use. Otherwise, you can use bit operations to find an empty block.

  110000100100011111110...

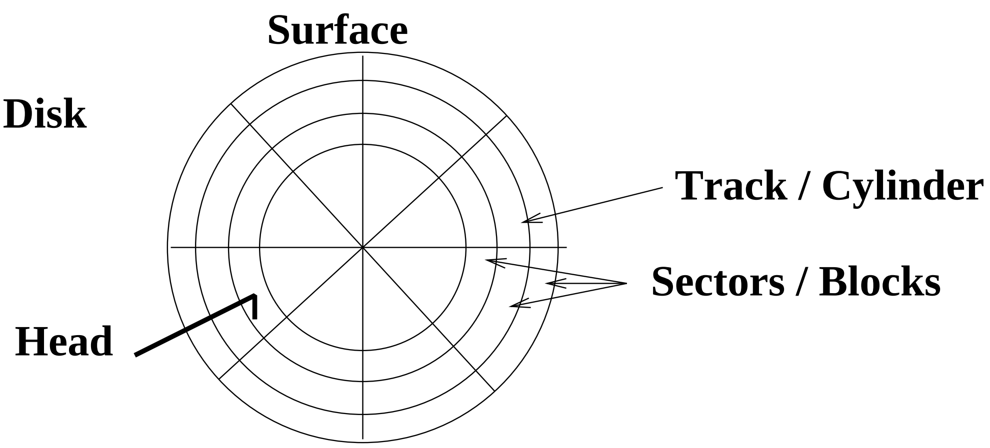
- Marking a block as freed is simple since the block number can be used to index into the bitmap to set a single bit.

Free-Space Management

- **Problem:** Bitmap might be too big to keep in memory for a large disk. A 2 GB disk with 512 byte sectors requires a bitmap with 4,000,000 entries (500,000 bytes).
- If most of the disk is in use, it will be expensive to find free blocks with a bitmap.
- An alternative implementation is to link together the free blocks. The head of the list is cached in kernel memory. Each block contains a pointer to the next free block.
  - How expensive is it to allocate a block?
  - How expensive is it to free a block?
  - How expensive is it to allocate consecutive blocks?

Summary of file and free space organization on disk

- Many of the concerns and implementations of file system implementations are similar to those of virtual memory implementations.
  - Contiguous allocation is simple, but suffers from external fragmentation, the need for compaction, and the need to move files as they grow.
  - Indexed allocation is very similar to page tables. A table maps from logical file blocks to physical disk blocks.
  - Free space can be managed using a bitmap or a linked list.



To read or write a disk block:

Seek: (latency) position head over a track/cylinder. The seek time depends on how fast the hardware moves the arm.

Rotational delay: (latency) time for the sector to rotate underneath the head. Rotational delay depends upon how fast the disk spins.

Transfer time: (bandwidth) time to move the bytes from the disk to memory.

Disk I/O Time = seek + rotational delay + transfer.

Typical Disk Parameters

Disk Capacity	36.4 GB
platters per pack	10
tracks per surface	11540
sectors per track	215
bytes per sector	732
revolutions per minute	7200
average seek time	7.5 ms
average rotational latency	4.17 ms
buffer to host burst transfer rate	200 MB/sec
buffer size	4 MB
size	3.5 inches

Access Time

Key: to get the quickest disk response time, we must minimize seek time and rotational latency:

- Make disks smaller
- Spin disks faster
- Schedule disk operations to minimize head movement
- Lay out data on disk so that related data are on nearby tracks.
- Place commonly-used files where on the disk?
- We should also pick our sector size carefully:
  - If the sector size is too small, we will have a low transfer rate because we will need to perform more seeks for the same amount of data.
  - If our sector size is too large, we will have lots of internal fragmentation.

Disk Head Scheduling

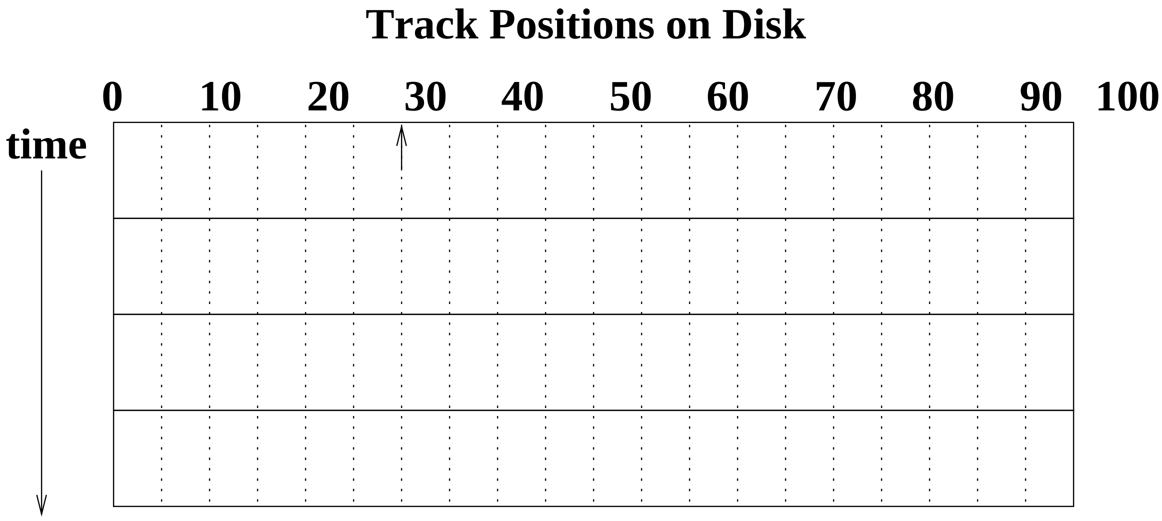
Idea: Permute the order of disk requests from the order that they arrive from the users to an order that reduces the length and number of seeks.

1. First-come, first-served (FCFS)
2. Shortest seek time first (SSTF)
3. SCAN algorithm (0 to 100, 100 to 0, 0 to 100, ...). If there is no request between current position and the extreme (0 or N), we don't have to seek there.
4. C-SCAN circular scan algorithm (0 to 100, 0 to 100, ...)

FCFS Disk Head Scheduling

Example requests: 65, 40, 18, 78

1. FCFS - service the requests in the order that they come in

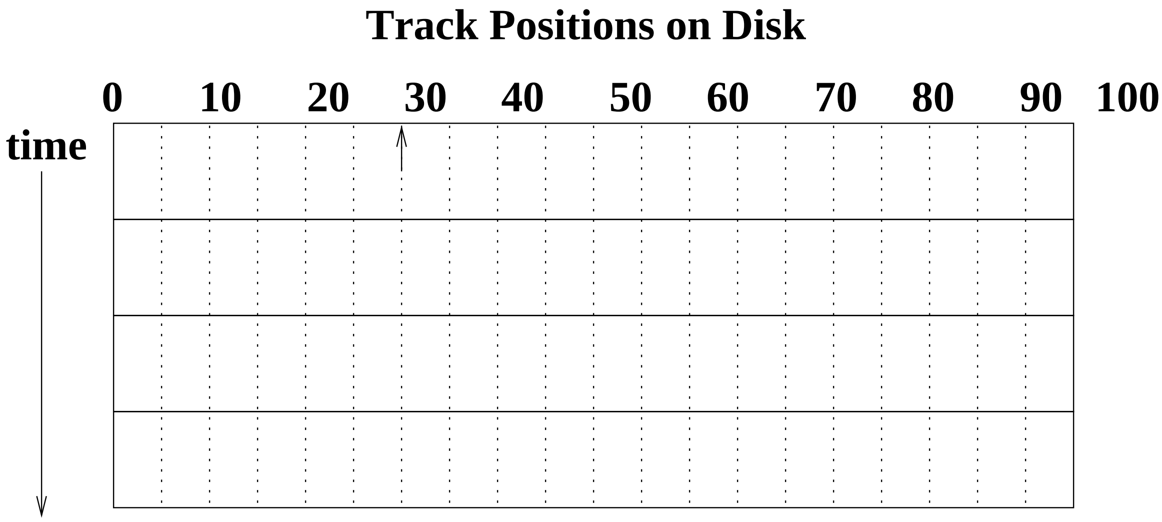


- Order of seeks:
- Distance of seeks:

- When would you expect this algorithm to work well?

Disk Head Scheduling

2. SSTF - always go to the next closest request

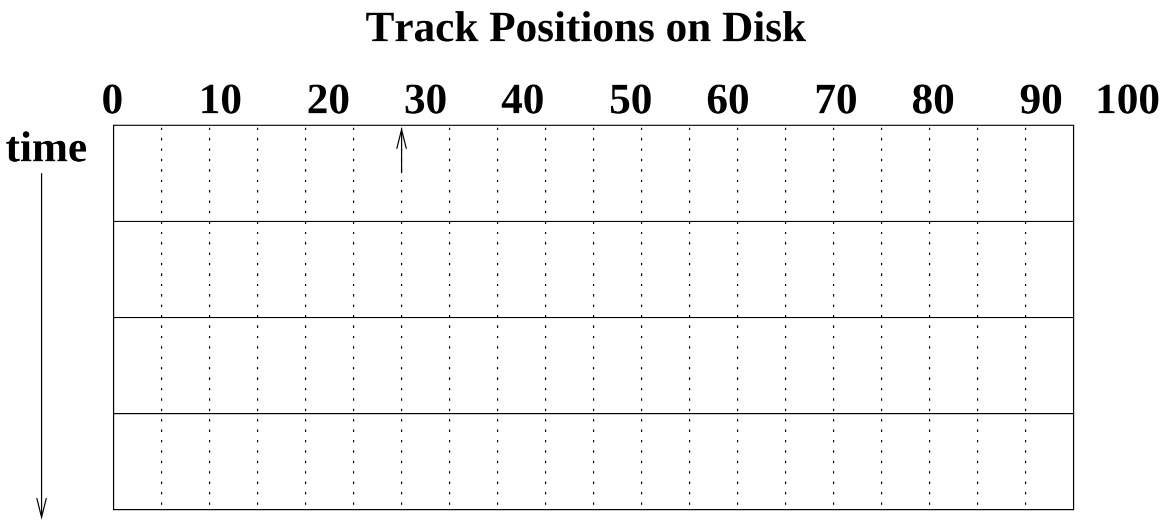


- Order of seeks:
- Distance of seeks:

- Can implement this approach by keeping a doubly linked sorted list of requests.
- Is this efficient enough?
- Is it optimal?
- Problems?

Disk Head Scheduling

3. SCAN - head moves back and forth across the disk (0 to 100, 100 to 0, 0 to 100, ...), servicing requests as it passes them



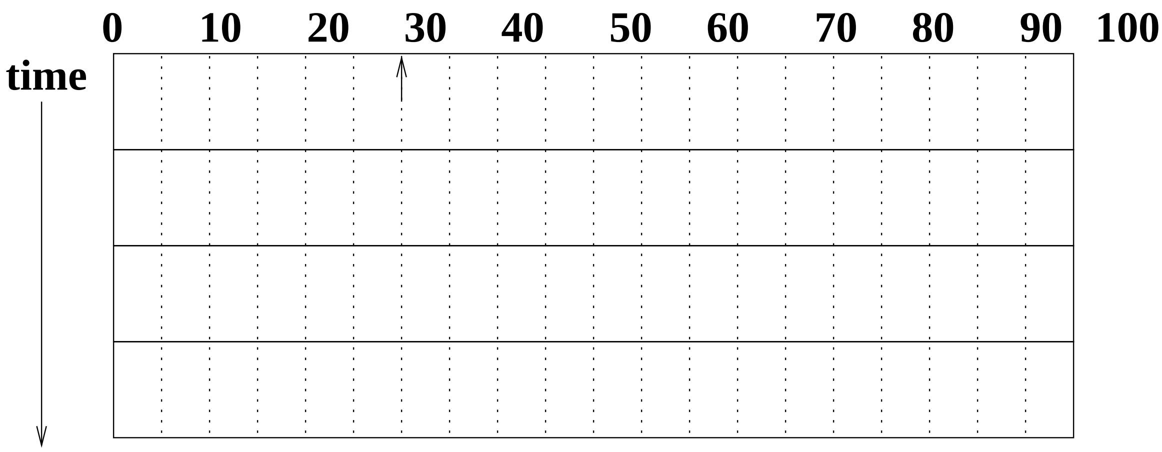
- Order of seeks, assuming the head is currently moving to lower numbered blocks:
- Distance of seeks:

- Requires a sorted list of requests.
- Simple optimization does not go all the way to the edge of the disk each time, but just as far as the last request.

Disk Head Scheduling

4. C-SCAN - circular scan algorithm (0 to 100, 0 to 100, ...)

Track Positions on Disk



- Order of seeks:
- Distance of seeks:

- More uniform wait times for requests. Why?

Other Ways to Improve Disk Performance

Disk Interleaving

Problem: Contiguous allocation of files on disk blocks only makes sense if the OS can react to one disk response and issue the next disk command before the disk spins past the next block.

Idea: Interleaving - Don't allocate blocks that are physically contiguous, but those that are temporally contiguous relative to the speed with which a second disk request can be received and the rotational speed of the disk. Might use every second or third block.

Read Ahead

Idea: read blocks from the disk ahead of user's request and place in buffer on disk controller.

Goal: reduce the number of seeks - read blocks that will probably be used while you have them under the head.

We considered pre-fetching virtual pages into physical memory, but decided that was difficult to do well since the future is difficult to predict. Is disk read-ahead any better?

Summary

- Disks are slow devices relative to CPUs.
- For most OS features, we are very concerned about efficiency.
- For I/O systems, and disk, in particular, it is worthwhile to complicate and slow down the OS if we can gain improvement in I/O times.