CIS 452 - Operating Systems Concepts

Nathan Bowman

Images taken from Silberschatz book

Linked Allocation

Files are broken into blocks and stored on disk

How blocks are stored on disk is up to OS and invisible to user

Simplest mapping is to allocate contiguous set of blocks, but external fragmentation can be major problem

Linked allocation -- file stored as linked list of disk blocks

Completely solves issue of external fragmentation

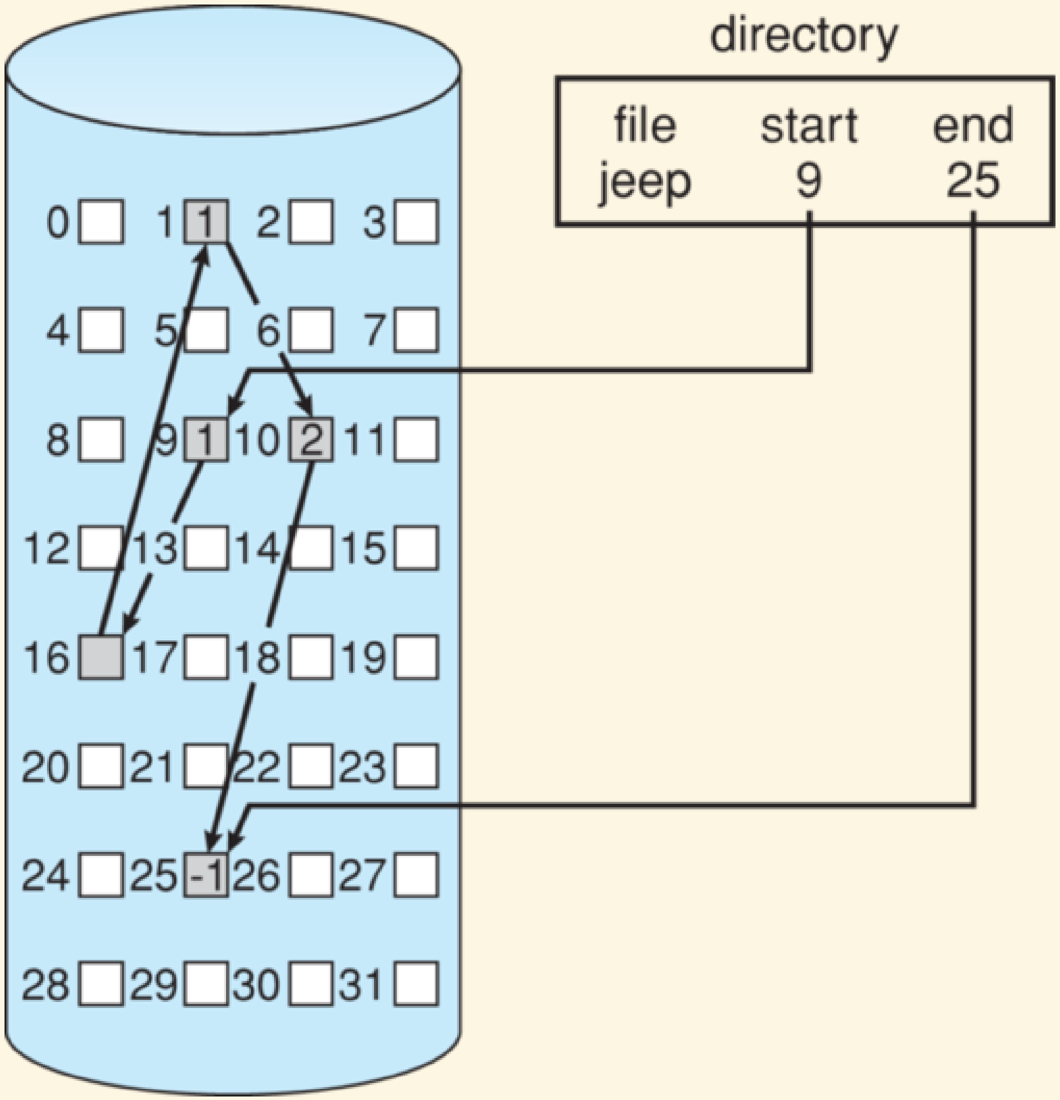
Blocks can be allocated from anywhere on disk

File can grow arbitrarily large as long as disk has free blocks (no compaction necessary)

Directory entry stores pointer to first and last blocks

Blocks contain

- data, and
- pointer to next block


directory
file
start
end
jeep
9
25
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
1
1
2
-1

One downside to linked allocation is storage of pointers

For example, with 512 byte blocks and 4 byte pointers, only 508 bytes per block free to store file data

Bigger problem is same one always seen with linked list -- linear access time

Contiguous allocation was efficient for both sequential and direct access to file

Direct access to location in file with linked list is expensive

Cannot go directly to specific block -- must follow links until desired block found

Essentially must simulate direct access using sequential access

Each pointer access is *disk read*, which is extremely slow

Alleviate both problems (storage overhead and excess disk reads) somewhat using clusters

Cluster -- collection of multiple blocks

Allocate files in clusters instead of blocks

One pointer per cluster instead of per block means lower storage overhead and fewer links to follow

Essentially similar to increasing block size

Downside to larger allocation units (clusters instead of blocks) is increased internal fragmentation

Half-empty cluster more wasteful than half-empty block

Linked list can reduce reliability of system

If pointer anywhere along the line corrupted, remainder of file not accessible

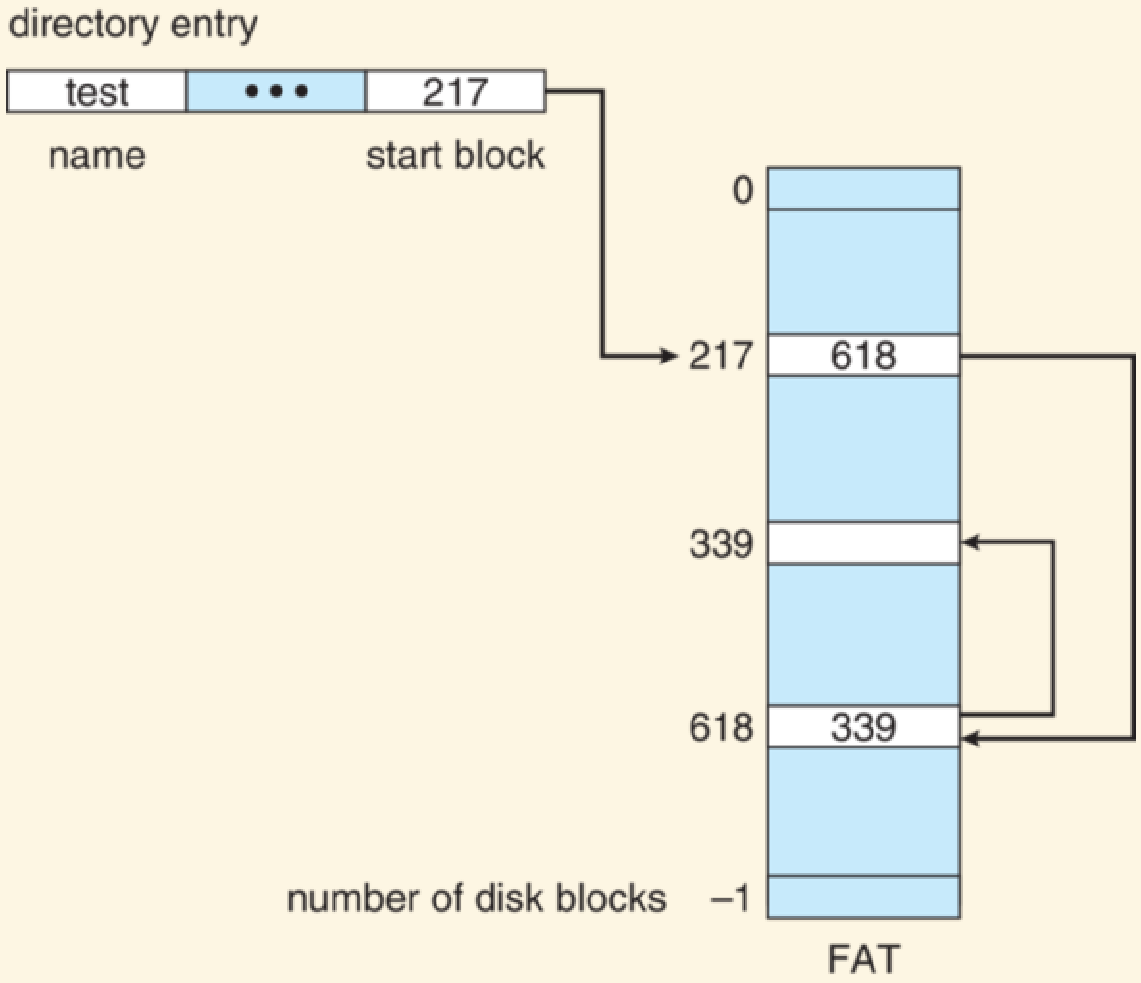
Can somewhat alleviate this using extra storage, such as doubly-linked list, but overhead increased

Variation on linked allocation is file-allocation table (FAT)

Rather than storing pointers in each block, store all pointers for all files together in single table

Table has one entry per block -- entry location (index in table) corresponds to block number

Value of `i`th entry is location of block/table entry where next part of file resides


directory entry
test
• • •
217
name
start block
0
217
618
339
618
339
number of disk blocks
–1
FAT

Functions similar to linked list, but all links reside within table

Directory entry stores location of first block

To find next block, go to corresponding table entry and read value, which is address of next block

End-of-file indicated by special value

Big question is: why?

Have we decreased amount of space used to store pointers?

Have we decreased number of disk reads to implement direct access to file?

Have we decreased amount of space used to store pointers?

No. Each block still has pointer of same size as before

Have we decreased number of disk reads to implement direct access to file?

No. Still need to follow chain of links to reach specific block -- links are simply stored closer together in one table. But, also, yes.

FAT table generally able to be held in memory

Following links becomes series of memory reads rather than disk reads

Memory access orders of magnitude faster than disk access

Note that table must still be *stored* on disk, or we lose all files when power goes out

Memory is acting as cache for disk