Silberschatz and Galvin Chapter 11

File System Implementation

File System Implementation: Overview

- Organization
- Allocation
- Free-space management
- Directory implementation

File System Organization

- Secondary storage: disks
  - i/o transfers performed in units of *blocks* (one or more sectors)
  - blocks vary between 32 bytes and 4096 bytes. Generally 512 bytes
- *File system* used to provide structure for the information stored on a disk. Provides for efficient and convenient access
  - how should the file system look to the user (file, attributes, operations, directory structure)
  - how should the file system be mapped onto physical secondary-storage devices (algorithms, data structures)

File System Organization

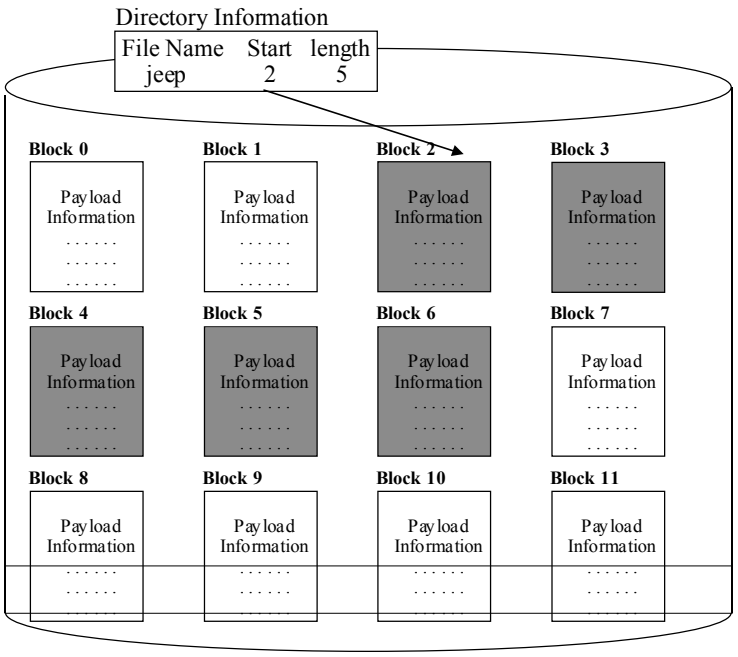
application programs
(open(), file descriptor, uses open-file table)
logical file system
(symbolic file name)
file-organization module
(files, logical blocks, physical blocks)
basic file system
(generic read/write physical block cmds)
I/O control
(device drivers, interrupt handlers)
devices

File System Allocation Methods

- How are blocks associated with a file stored on disk?
  - contiguous
  - linked
  - indexed

Contiguous Allocation

- Each file occupies a set of contiguous addresses on disk
  - a file *n* blocks long occupies addresses *b* through *b+n-1*
  - Sequential access is easy (just remember address of last block accessed and get the next)
  - Direct access also easy (access *b+i* directly)
- Issue: how to find space for new file (instance of the general *dynamic storage-allocation* problem discussed earlier)



Contiguous Allocation

- Finding a section of contiguous free blocks
  - first fit
  - best fit
  - worst fit

Contiguous Allocation

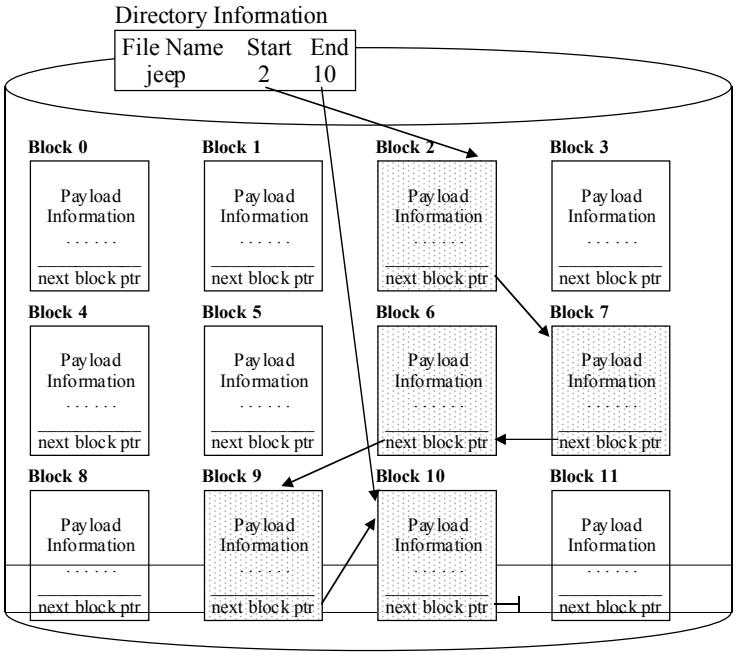
- Problems
  - external fragmentation (e.g., requiring re-packing to reduce external fragmentation--requires down time)
  - determination of needed file size at creation time
    - modification of file implies changing the size. how to handle expansion if no free space adjacent? terminate? relocate? expensive...
    - overestimation of size causes internal fragmentation if extra space left (perhaps for lifetime of file--years)

Contiguous Allocation

- A modification to contiguous allocation also incorporates an extent
- if additional space is needed, the extent is added to the initial allocation.
- Directory contains means to include pointer to the first block of the extent (in addition to the location of the initial allocation and its size)
- Can be generalized to permit multiple extents... See grouping in later discussion of directory implementation

Linked Allocation

- File: a linked list of disk blocks
- Directory: contains pointer to first and last blocks in file
- Initially the directory entry is **nil**
- Requires space for links. If physical block is *n* bytes and disk address takes *x* bytes, then available space in block is *n* - *x*



Linked Allocation

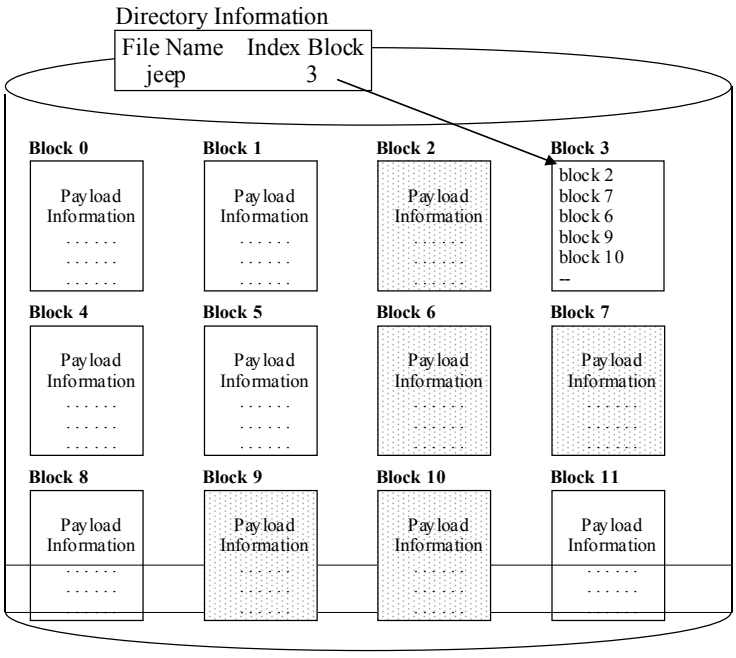
- Advantages:
  - no external fragmentation
  - no compaction needed
  - easy to modify file (insert/delete blocks)
- Disadvantages
  - access method: sequential access only
  - reliability: recovery difficult if pointers lost
  - disk space efficiency: needed space for pointers is "wasted"
    - one option: clusters (collect blocks into multiples and allocate cluster not block). Expense: internal fragmentation

Linked Allocation

- Variation: File Allocation Table (FAT)
  - Section of disk set aside at beginning of partition containing one entry for each disk block
  - Links represented by storing next address in slot in FAT
  - Must cache to keep from having to do two head seeks for each read...
  - Eases implementing random access because location of block can be determined by processing FAT (not traversing disk). But you still have to process the FAT.

Indexed Allocation

- *Index block* brings together file's pointers to disk blocks into one location
- Each file has its own index block
- Index block contains disk addresses for blocks allocated to the file
- Initially all are **nil**
- As blocks are allocated, they are added to the index block



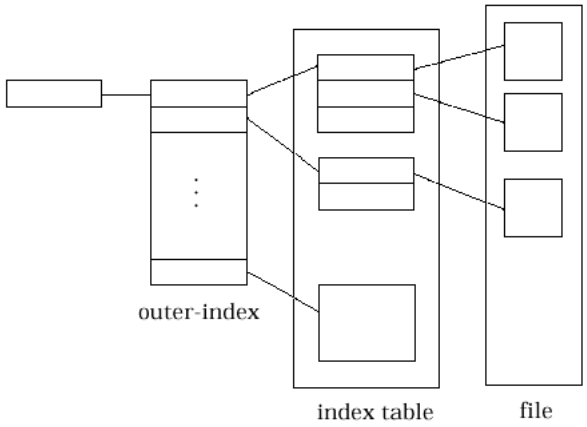
Indexed Allocation

- Advantages
  - no external fragmentation. no compaction required
  - access method: both sequential and random access
  - reliability: lost pointers have limited effect, (not global)
  - easy file modification (only have to rearrange block)
- Problems
  - pointers require disk space. Wasted space in index block for small file (unused indices)
  - size of index block limits size of file unless we develop a modified scheme

Indexed Allocation

- Incorporating multiple index blocks
  - linked indexed allocation
    - Last word in block is either **nil** or a pointer to another index block
  - multi-level indexed allocation
    - two level: index block that points to index blocks. With 2048 byte blocks and 4-byte addresses can get 512 pointers. Two levels is 4,194,304 data blocks or 8.5 gigabytes.
    - multi level: further levels of indirection

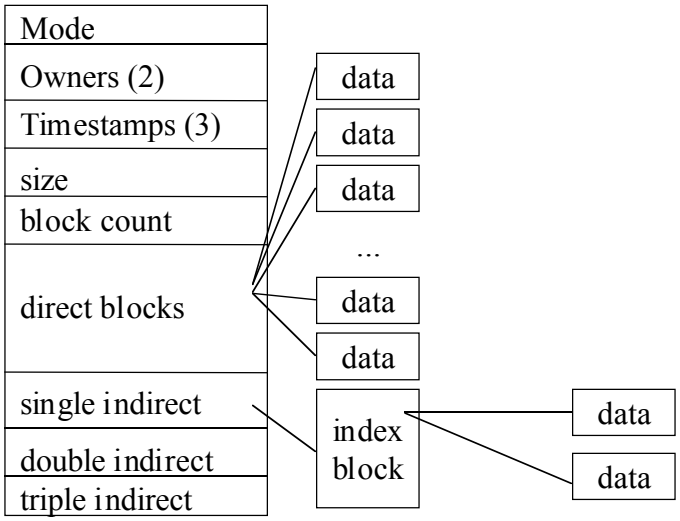
Indexed allocation



Indexed Allocation

- Combined scheme (BSD UNIX)
  - first 15 pointers of index block kept in file's index block (or inode). First 12 of those point to direct blocks (contain data). Next 3 point to indirect blocks. First is a single indirect block (points to index block which points to data blocks). Second is a double indirect block (points to index block which points to index block). Third is triple indirect blocks.

Unix inode



Unix inode

- Small file accessed directly from inode
- Larger files require additional indirections
- Total addressable blocks exceed that addressable by the 4-byte file pointers used by the OS.
- Indexed blocks can be cached in memory but data blocks may be spread all over a partition (requiring seeks for sequential access)

Free-Space Management

- How do you locate unused disk blocks?
- *Free-space list* records all disk blocks that are not allocated to a file or directory (i.e., *free*)

Free-Space List

- Bit Vector: each block represented by a bit which is 1 if free, 0 if allocated
- Simple and efficient to find first free block or *n* consecutive free blocks, often with processor bit-manipulation instructions
- But a large disk requires a large bit vector. For example 1.3 gigabyte with 512K blocks requires 310K of bit vector. (Clustering of blocks helps.)

Free-Space List
Linked List

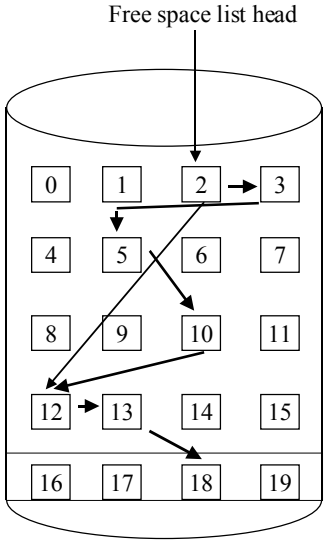
- Link together all free disk blocks keeping a pointer to the first free block in a special location (cached in memory). First block contains pointer to next free block, etc...

Grouping

- Linking all the free blocks
- Using some space on disk.
- Taking less time than linked list.
- Storing n-1 addresses in the first free block.
- The n-th address is used to point to the next "address block"

Block 2: 3, 5, 10, 12

Block 12: 13, 18, -, -



n = the size of block = 4

Free-Space List Grouping

- Store addresses of *n* free blocks in the first free block. The first *n-1* are allocatable. The *n*th contains another list of addresses
- Addresses of a large number of free blocks can be found quickly (when compared to linked lists)

Free-Space List Counting

- Keep address of first free block and the number, *n*, of contiguous blocks that follow the first
- Entry in free-space list is disk address and a count
- Each entry requires more space but there are fewer of them
- Assumption is that contiguous blocks are allocated or freed simultaneously

Counting

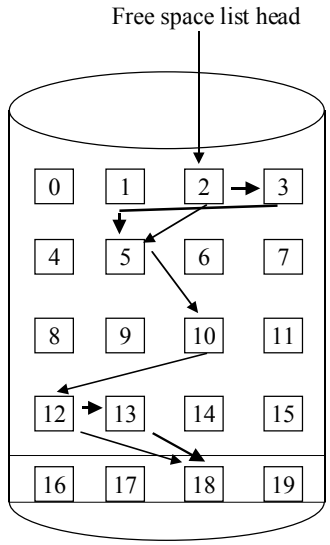
- Linking all the free blocks
- Using some space on disk.
- In each item of the list, store the number of contiguous free blocks from the current one.

Block 2: 2 - - 5

Block 5: 1 - - 10

Block 10: 1 - - 12

Block 12: 2 - - 18



n = the size of block = 4

Directory Implementation

- Linear
  - new file: search for name then if not found add to end
  - delete file: find and release space. Either mark entry "unused" or replace with a valid entry (for example, the last one in the list)
  - search time can be a factor when directory gets large (search times in UNIX directories, for example). Cache, ordered lists, etc., can help.

Directory Implementation

- Hash table
  - both a linear list and also a hash table
  - as with any hash table have to deal with *collisions*

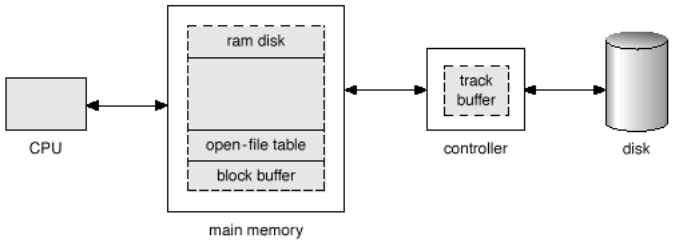
Design Issues in Disk Storage Management

- Block size
- Free space management
- Allocation methods
  - fragmentation, internal and external
  - access methods
  - reliability
  - space efficiency
  - run-time overhead
  - limits on file size
  - file modification limits

Efficiency and performance

- Efficiency dependent on:
  - disk allocation and directory algorithms
  - types of data kept in file's directory entry
- Performance
  - *disk cache* – separate section of main memory for frequently used blocks
  - *free-behind* and *read-ahead* – techniques to optimize sequential access
  - improve PC performance by dedicating section of memory as *virtual disk*, or *RAM disk*

Various disk caching locations



Recovery

- Consistency checker – structure with data blocks on disk, and tries to fix inconsistencies.
- Use system programs to *back up* data from disk to another storage device (floppy disk, magnetic tape).
- Recover lost file or disk by *restoring* data from backup.