Logical disks

Reference: Bach 2.2.1

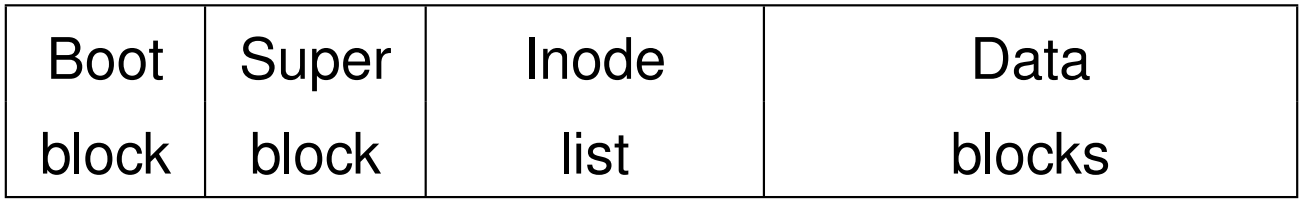
- Physical disk is divided into partitions or *logical* disks
- Logical disk $\equiv$ linear sequence of fixed size, randomly accessible, blocks
  - disk device driver maps underlying physical storage to logical disks
  - large blocks $\Rightarrow$ faster access, more fragmentation
- *File system* $\equiv$ logical disk whose blocks have been arranged suitably so that files may be created and accessed

NOTES:

- Logical disk may also be used for swap partition
- Logical disk may be located on multiple physical disks (e.g. volumes, striping, raid)

File systems

- File $\equiv$ inode (header, or admin. info) + data



- Boot block: usually contains bootstrap code
- Super block: size, # files, free blocks, etc.
- Inode list
  - size fixed when configuring file system
  - contains *all* inodes
- Data blocks: file data
- Files, directories organized into tree-like structure

Super block: summary of file system

Reference: Bach 4.5

- Size of the file system
- Free data blocks: # of free blocks, list of free blocks, pointer to the first free block on free list
- Size of inode list
- Free inodes: (as above)
- Locks for free block list, free inode list
- Dirty flag

Inodes

Reference: Bach 4.1

File attributes:

- Type: regular file, directory, device special file, pipes
- Owner: individual + group
- Permissions: read, write, execute, for owner, group, others
- Access times: last modified, last accessed, last modification of inode
- Number of links
- File size: 1 + highest byte offset written into
- Table of contents: disk addresses of (discontiguous) disk blocks containing the file data

```
-rw-rw-r-   1 mandar mandar    2647   Mar 11 14:58   filesys.tex
```

Table of contents

Reference: Bach 4.2

- File data stored in non-contiguous disk blocks
  - contiguous storage $\Rightarrow$ fragmentation, expansion of files problematic
- File space is allocated one block at a time (i.e. data can be spread throughout file system)
- 10 direct entries: numbers of disk blocks that contains file data
- Single indirect: number of a disk block that contains a list of direct block numbers
- Double indirect, triple indirect
- Processes access data by byte offset; kernel converts byte offset into block no.
- Block no. = 0 $\Rightarrow$ corresponding logical block contains no data
  - no disk space is wasted

Directories (SVR2)

Reference: Bach 4.3

- Data blocks contain a sequence of 16 byte entries
- Each entry = inode number (2 bytes) + null-terminated file names (14 bytes)
- Compulsory entries: current directory (`.`) and parent directory (`..`)
  - for root, parent directory = root
- Inode number = 0 $\Rightarrow$ empty directory entry

Inode cache/table

Reference: Bach 4.1

- List of buffers stored in main memory
- Each buffer contains *in-core* copy of *disk* inode
- **At most one copy of any inode is present in the cache**
- Additional information stored in each buffer:
  - logical device number of file system that contains the file
  - inode number
    - inode list on disk $\equiv$ linear array
    - numbering starts from 1
  - pointers to other in-core inodes
  - status (locked/free/awaited, dirty bit, mount point flag)
  - reference count: # of active uses of the file

Accessing inodes

Input: Inode no. i

Output: Location (Block no., byte offset)

Method:

1. Let $n =$ number of inodes per block
2. Block no. $B = (i - 1)$ div n + starting block of inode list
3. Byte offset $b = ((i - 1) \bmod n) \times$ size of disk inode
4. Return $\langle B, b \rangle$

ifree: freeing inode after file/directory deletion

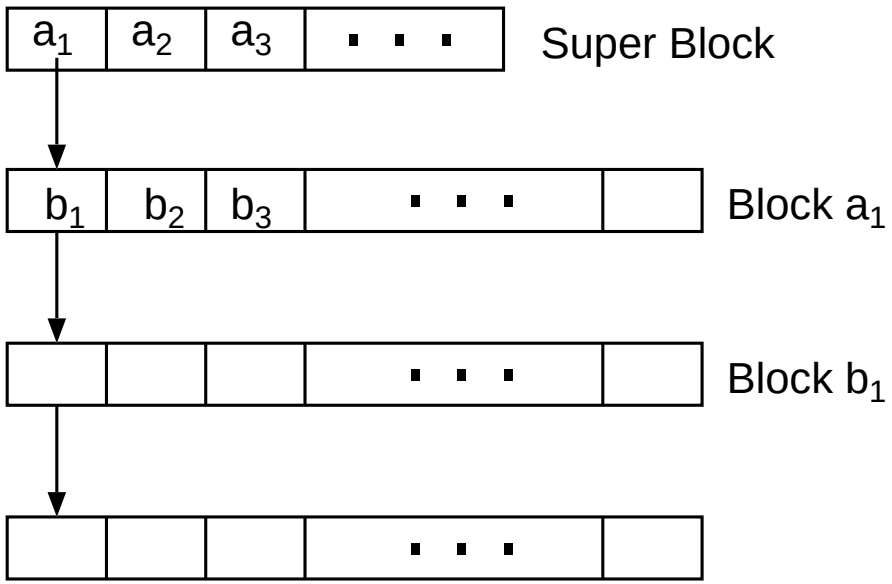
Algorithm:

```
increment file system free inode count;
if (SB is locked) return;
lock SB;
if (inode list is full) {
   if (inode no. < "remembered inode")
      update "remembered inode";
}
else store inode no. in free inode list;
unlock SB;
```

Disk block allocation

Reference: Bach 4.7

- SB contains a list of free disk block nos.
- Initially, `mkfs` organizes all data blocks in a linked list
  - each link (disk block) contains (i) list of free disk block nos., (ii) no. of next block on the list



- Free nodes identifiable by type field; free disk blocks not identifiable by content
- Disk blocks consumed more quickly than inodes
- Disk blocks large enough to contain long list of free block nos.

alloc

Algorithm:

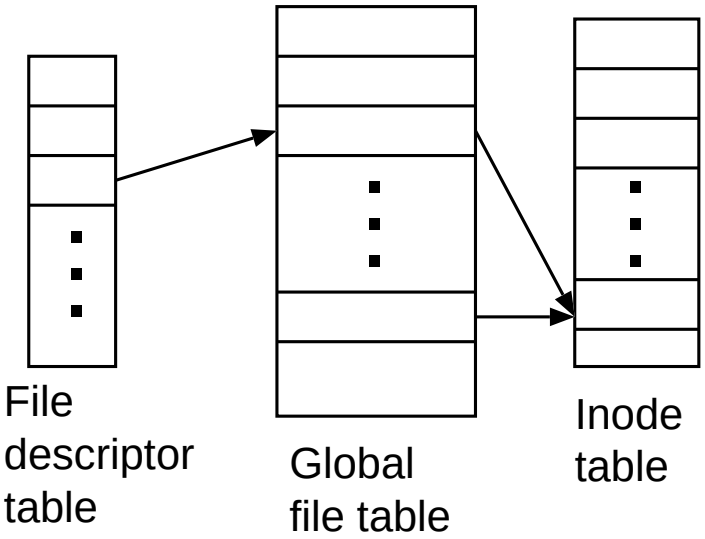
```
while (SB is locked) sleep until SB is free;
remove block from SB free list;
if (last block was removed) {
   lock SB;
   read block just removed;
   copy block nos. into SB list;
   unlock SB; /* wake up other procs */
}
zero block contents;
decrement total count of free blocks;
mark SB modified;
return buffer;
```

free

Algorithm:

```
if (SB list is not full)
   put block on SB list;
if (SB list if full) {
   copy SB list into freed block;
   write block to disk;
   put block no. of freed block into SB list;
}
```

System calls



- File descriptor (per process) – pointers to all open files
- Global file table – mode, offset for each `open`-ed file
- Inode table – memory copy of on-disk inode (only one per file)

- *creat*, *open*, *close*
- *read*, *write*
- *mount*, *umount*

dup

Reference: Bach 5.13

Syntax: `newfd = dup(fd);`

Algorithm:

1. Find *first* free slot in the user fd table.
2. Copy given file descriptor into the free slot.
3. Increment ref. count of corresponding global file table entry.
4. Return descriptor (index) of this slot.

Shell

Reference: Bach 7.8

Algorithm:

1. Parse command line.
2. If command is internal command, call suitable function.
3. If command is external command, fork a child. In child:
   - 3.1 if input / output redirection is required:
     ```
     fd = /* create new file */
     close(stdout);   dup(fd);   close(fd);
     ```
   - 3.2 if pipes are required:
     - 3.2 create a pipe; fork a child
     - 3.2 in child: setup pipes s.t. stdout goes to pipe, exec the first component of command line
       in parent: setup pipes s.t. stdin comes from pipe.
   - 3.3 exec command (or last component of command).
4. If command is run in foreground, wait for child to exit.