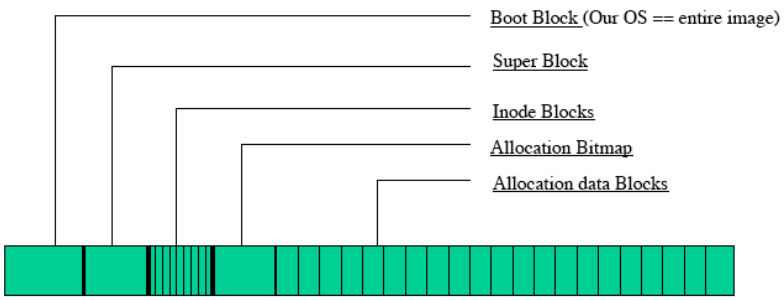
# Project 6: File Systems

## Overview

- Write a UNIX-like hierarchical FS
  - (implement various system calls)
- Completes the OS course
- Final abstraction
  - Expose files and directories instead of raw disk blocks
  - Offers a much more 'familiar' environment

## Global view of Disk Layout




## System calls:

- init(): OS initialization
- mkfs: Formatting
- open: file creation
- close, read, write, lseek: file access
- link, unlink: associate dirEntry to inode (or not)
- mkdir, chdir, rmdir: directory stuff
- stat: information about a file or directory

## fs_init() vs. fs_mkfs()

- What needs to be done where:
  - fs_init() - initialization for whole system
  - fs_mkfs() - initialization for the FS on disk
- Imagine two disks in your system
- When do you:
  - Initialize the file descriptor table?
  - Set bitmaps and inodes to be free?
  - Initialize currentInode (correspond to "/")
  - 'mount' the filesystem?

## Formatting

- mkfs() creates the filesystem.
  - Fill in the superblock structure, write to disk
  - Mark inodes and data blocks to be free
  - Create root directory
- In other words: formats the disk
- fsck() checks integrity of file system
  - Provided

## File Creation / Deletion

- open(): Create file if it does not exist
- link(): Hard link to a file
  - Create a link to an existing file
  - Allows multiple locations in the FS point to the same file on disk
- unlink: Delete a file if link count == 0
  - delete directory entry

## File Access

- read()
- write()
- lseek(): move file pointer on open descriptors
- close()

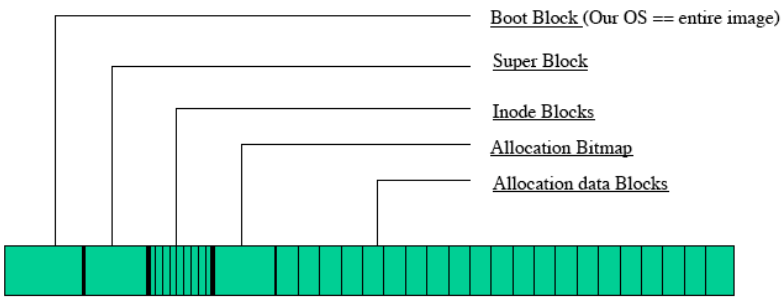
## Directories

- mkdir(): make a directory
  - create an entry in parent directory
  - create two directories: ".", ".."
- rmdir: remove directory
- chdir: change the $CWD
  - For relative path names

## Directories (cont)

- Can be implemented like a file
  - Contains a list of dirEntry structures that contain (filename, inode number) pairs

# Global view of Disk Layout




# Super Block

- Meta data about layout of the disk
  - Magic numbers/name
  - Size of partition/disk
  - Number of inodes
  - Number of data blocks
  - Sectors where inodes or data blocks begin
  - Etc…

# Superblock Structure

```
typedef struct {
    char signature[SIGN_SIZE]; /* Signature */
    int size;                  /* Size of file system in blocks */
    int root_inode;            /* Inode no. of root directory */
    int inode_start;           /* First block for inodes */
    int inode_blocks;          /* Number of inode blocks */
    int bitmap_start;          /* First block for bitmap */
    int bitmap_blocks;         /* Number of blocks used to store the bitmap */
    int alloc_start;           /* First block managed by the allocater */
    int num_blocks;            /* Number of blocks for allocation */
    int free_blocks;           /* Number of free blocks: Note: IGNORE this
since we do not want to update superblock frequently. */
} superblock;
```

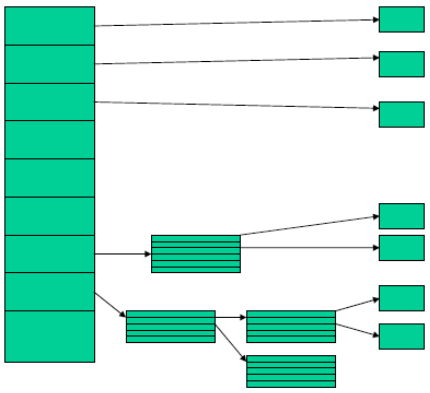
# Inode

- Associates disk blocks with files
- Directory entries point to Inodes
- Structure for book keeping
  - List of blocks in file
  - Type (file or directory)
  - Count of hard links
  - Permission/Owner information

## Structure

```
typedef struct {
    short type;
    char links;
    int size;
    int direct[NUM_DIRECT_BLOCKS];
    int indirect[NUM_INDIRECT_BLOCKS];
} inode;
```

## Inode direct/indirect lookup



## Inode (cont)

- Advantages
  - Simple
  - Fast access for small files
  - Support for large files
  - Support for sparse files

## Allocation Bitmap

- There is a one-to-one mapping
  - Bits in allocation bitmap --> data blocks
- For simplicity use a ByteMap
  - fs_fsck() expects bytes (0 or 1)
  - Document any changes you make to fsck()

## Example: mkdir()

```
int fs_mkdir(char *file_name) {
  if (file_name exists) return ERROR;
  /* allocate data block */
  /* allocate inode */
  /* add directory entries for '.', '..' */
  /* set inode entries appropriately *
  /* update parent directory */
  return SUCCESS
}
```

## Development

- Check Errors
  - Many boundary conditions
  - So, be thorough
- Feel free to add more files
  - Abstraction is your friend
  - Layer between disk and files

# Doing the Assignment

- Most development in Linux
  - Use a file to simulate a disk (make lnxsh)
  - Code is provided (*Fake files)
  - Should be able to move right over to your OS
- Shell supports
  - System calls for File System
  - Commands like “ls”, “cat”, “create”