0.1 UNIX File Management

- Focus on two types of files
  - Ordinary files (stream of bytes)
  - Directories
- And mostly ignore the others
  - Character devices
  - Block devices
  - Named pipes
  - Sockets
  - Symbolic links
- **UNIX index node (inode)**
  - Each file is represented by an Inode
  - Inode contains all of a file's metadata
    - Access rights, owner,accounting info
    - (partial) block index table of a file
  - Each inode has a unique number (within a partition)
    - System oriented name
    - Try 'ls –i' on Unix (Linux)
  - Directories map file names to inode numbers
    - Map human–oriented to system–oriented names
    - Mapping can be many–to–one; Hard links

```
ozdogan@ozdogan:~/week12$ man ls
.
-i, --inode   print index number of each file
.
ozdogan@ozdogan:~/week12$ ls -i
toplam 128
 901649 drwxr-xr-x 3 ozdogan  ozdogan  4096 2004-05-25 15:00 ./
 885067 drwxr--r-- 5 ozdogan  ozdogan  8192 2004-05-25 14:47 ../
 901651 drwxr-xr-x 2 ozdogan  ozdogan  4096 2004-05-25 14:47 figures/
 901656 -rw-r--r-- 1 ozdogan  ozdogan  1264 2004-05-25 14:59 week12.aux
```

```
 901658 -rw-r--r-- 1 ozdogan  ozdogan  5264 2004-05-25 14:59 week12.dvi
 901655 -rw-r--r-- 1 ozdogan  ozdogan  8654 2004-05-25 14:59 week12.log
 901657 -rw-r--r-- 1 ozdogan  ozdogan    57 2004-05-25 14:59 week12.out
 901659 -rw-r--r-- 1 ozdogan  ozdogan 55968 2004-05-25 15:00 week12.ps
 901652 -rw-r--r-- 1 ozdogan  ozdogan  2153 2004-05-25 14:59 week12.tex
 901654 -rw-r--r-- 1 ozdogan  ozdogan  1939 2004-05-25 14:58 week12.tex~
 901653 -rw-r--r-- 1 ozdogan  ozdogan 13767 2004-05-25 14:47 week12.tex.back
```

A code example for for printing out structure members of files; Try 'structuremembers *' on Unix (Linux)

```
/* structuremembers.c
print structure members of files
st_mode the type and mode of the file
st_ino
st_dev
st_rdev
st_nlink
st_uid
st_gid
st_size
st_atime
st_mtime
st_ctime
*/
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
main(argc,argv)
int argc; char *argv[];
{
struct stat status;
int i;
for(i=1; i < argc; i++)
if(stat (argv[i],&status))
fprintf(stderr,"Cannot stat %s \n",argv[i]);
else
printf("%15s %4.4o\n",argv[i],status.st_mode & 07777);
//printf("%15s %14d\n",argv[i],status.st_ino);
}
```

Internal sructure of week12 inode

```
Inode: 901649   Type: directory    Mode:  0755   Flags: 0x0   Generation: 24
User:  1000   Group:  1000   Size: 4096
File ACL: 0    Directory ACL: 0
Links: 3   Blockcount: 8
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x40b33fde -- Tue May 25 15:45:18 2004
atime: 0x40b34ba7 -- Tue May 25 16:35:35 2004
mtime: 0x40b33fde -- Tue May 25 15:45:18 2004
BLOCKS:
(0):1828886
TOTAL: 1
```

Internal sructure of week12.ps inode

```
Inode: 901659   Type: regular    Mode:  0644   Flags: 0x0   Generation: 2473
User:  1000   Group:  1000   Size: 83309
File ACL: 0    Directory ACL: 0
Links: 1   Blockcount: 176
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x40b34007 -- Tue May 25 15:45:59 2004
atime: 0x40b34016 -- Tue May 25 15:46:14 2004
mtime: 0x40b34007 -- Tue May 25 15:45:59 2004
BLOCKS:
(0-11):7742-7753, (IND):7754, (12-20):7755-7763
TOTAL: 22
```

- Inode Contents (see Fig. 1)
  - Mode
    - Type; Regular file or directory
    - Access mode; rwxrwxrwx
  - Uid; User ID
  - Gid; Group ID
  - atime; Time of last access
  - ctime; Time when file was reference count created
  - mtime; Time when file was last modified
  - Size; Size of the file in bytes

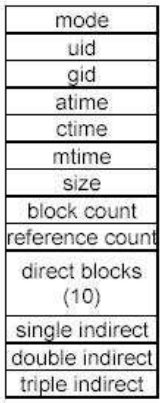

Figure 1: Inode contents.

- Block count; Number of disk blocks used by the file.
- Note that number of blocks can be much less than expected given the file size; Files can be sparsely populated
- Direct Blocks
  - Block numbers of first 10 blocks in the file
  - Most files are small; We can find blocks of file directly from the inode (see Fig. 2left)
- Problem; How do we store files greater than 10 blocks in size? Adding significantly more direct entries in the inode results in many unused entries most of the time.
- *Single Indirect Block*; Block number of a block containing block numbers, (see Fig. 2right) In this case 8
  - Requires two disk access to read; One for the indirect block; one for the target block
  - Max File Size
    - In previous example; 10 direct + 8 indirect = 18 block file
    - A more realistic example; Assume 1Kbyte block size, 4 byte block numbers 10 * 1K + 1K/4 * 1K = 266 Kbytes
  - For large majority of files (¡ 266 K), only one or two accesses required to read any block in file.
- Double Indirect Block; Block number of a block containing block numbers of blocks containing block numbers

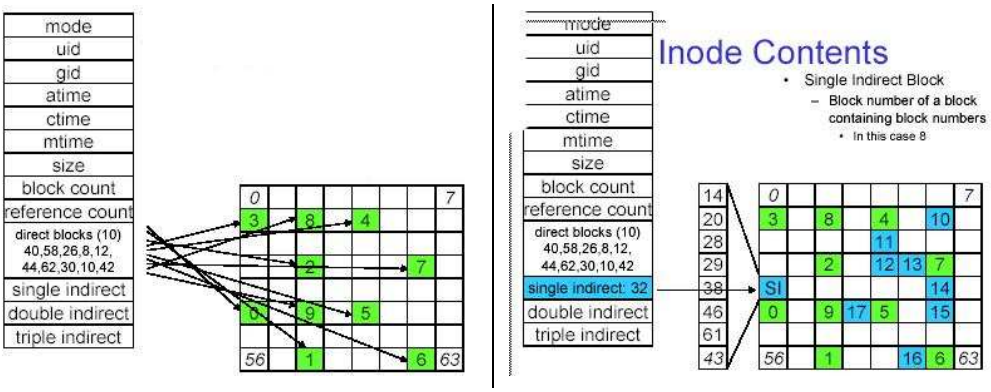

Figure 2: Left: Direct Block. Right: Single Indirect Block

  - Triple Indirect; Block number of a block containing block numbers of blocks containing block numbers of blocks containing block numbers

- Inode Summary
  - The inode contains the on disk data associated with a file
  - Contains mode, owner, and other bookkeeping
  - Efficient random and sequential access via indexed allocation
  - Small files (the majority of files) require only a single access
  - Larger files require progressively more disk accesses for random access; Sequential access is still efficient
  - Can support really large files via increasing levels of indirection
- **Where/How are Inodes Stored**
- **System V Disk Layout (s5fs)** (see Fig. 3Upper)
  - Boot Block; contain code to bootstrap the OS
  - Super Block; Contains attributes of the file system itself; e.g. size, number of inodes, start block of inode array, start of data block area, free inode list, free data block list
  - Inode Array
  - Data blocks
- Some problems with s5fs

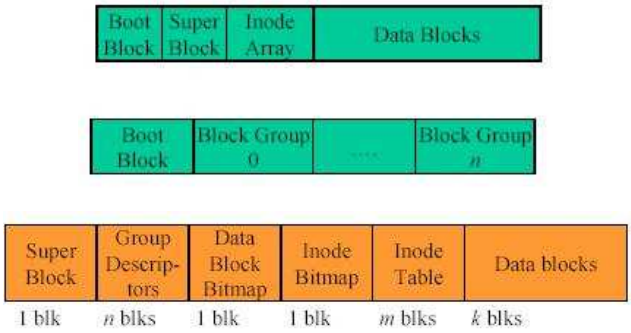

Figure 3: Upper: System V Disk Layout (s5fs). Middle: Layout of an Ext2 Partition. Lower: Layout of a Block Group.

  - Inodes at start of disk; data blocks end. Long seek times; Must read inode before reading data blocks
  - Only one superblock; Corrupt the superblock and entire file system is lost
  - Block allocation suboptimal; Consecutive free block list created at FS format time. Allocation and deallocation eventually randomizes the list resulting the random allocation
  - Inodes allocated randomly; Directory listing resulted in random inode access patterns

- **The Linux Ext2 File System** (see Fig. 3Middle)
  - Second Extended Filesystem; Evolved from Minix filesystem (via "Extended Filesystem")
  - Features
    * Block size (1024, 2048, and 4096) configured as FS creation
    * Preallocated inodes (max number also configured at FS creation)
    * Block groups to increase locality of reference
    * Symbolic links ¡ 60 characters stored within inode
  - Main Problem: unclean unmount → **e2fsck**
    * Ext3fs keeps a journal of (metadata) updates
    * Journal is a file where updated are logged
    * Compatible with ext2fs

- Layout of an Ext2 Partition
  - Disk divided into one or more partitions
  - Partition:
    - Reserved boot block,
    - Collection of equally sized block groups,
    - All block groups have the same structure
- **Layout of a Block Group** (see Fig. 3Lower)
  - Replicated super block and group descriptors; For e2fsck
  - Bitmaps identify used inodes/blocks
  - All block have the same number of data blocks
  - Advantages of this structure:
    - Replication simplifies recovery
    - Proximity of inode tables and data blocks (reduces seek time)