In this lecture we are going to spend time discussing Homework 3 & Homework 4.

(a) Homework-3: Simulation of a file system

- We are not going to allocate blocks on the disk.
- Methods that are implemented:
  
  **Directories**
  - mkdir
  - rmdir
  - readdir

  **Files**
  - mknod
  - delete
  - append
  - read

- The file system is just a structure of

  - **Directories**

  | name | ptr |

  - **home** ➔ **joe** ➔ **file.txt** ➔ **len**

  - blocks
Files: It is a set of blocks each of 512 bytes.

- One of the challenges is to implement the read operation.
  - Read is supposed to handle the empty space at the end of the block.
  - Also handle the data i.e., read only specific parts depending upon the offset & length specified.

So, we express the entire thing as a single loop.

```c
read (offset, len)
    block = current block
    while offset > 512
        block = next block
        offset -= 512
    to_copy = min (file len - offset, len requested)
    while (to_copy > 0)
        512 - offset
        this_copy = min (512, to_copy)
        read from block this_copy bytes starting @ offset
        block = next block
        to_copy -= 512
        offset = 0
    ```
(b) **Homework - 4 Overview**

We will use the FUSE (File system in User space) toolkit.

We write the FUSE application here.

**Normal Application**

- **Fuse**
- **Lib**
- **Fuse**

10MB

Regular file

(Stored on disk & we are going to look at it as array of blocks)

The main operations to be implemented in the assignment are:

```c
struct fuse_operations myops = {
    .get_attr = get_attr, /* MAN lstat */: reads a directory entry for a particular file
    .mknod
    .mkdir
    .unlink
    .rename
    .open: check to see if a file is there.
    .truncate: chop the file back to a certain length
    .read
    .write
    .stats
```
After doing the above we will then call

maint()
{
    fuse_main(.....,my_ops)
}

- Parts
  - Mounting:
    - Superblock:
      - It tells us how long the block size is and how long the FAT is. From N, we know where the root directory is.

    In Homework 4, we need to develop a generic routine that will let us read from a file.

The directory structure defined for it is

<table>
<thead>
<tr>
<th>Type bit</th>
<th>D/E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start Address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Each directory entry has all the information)

linked list of blocks
We use the `MAN` command in Unix to read the manual pages.

- **Explanation of some of the operations in Homework 4:**

- **Open operation:**

  ```c
  int open(const char *path, fuse_file_info *fi)
  path = "/dir1/filename" (example of a path)
  if lookup path = NOT FOUND
      ERROR
  if permissions fail or is_directory
      ERROR
  otherwise
      return OK
  ```

The parsing of paths is similar to homework 3. It follows the UNIX convention i.e. if the operation is successful it returns a 0 or positive number. In case of an error, it returns a negative number.

[Diagram of file system structure]

Each filesystem is ignorant of other file systems in the kernel.
- mknod operation: creates a file, allocate a block. Start block will be zero.

- Unlink operation: free up blocks allocated for a file.

Free space management is quite simple.

Every block contains a next pointer which points to the block next in line. It belongs to the same file.

EOF = End of file

The FAT contains entries for each block. While allocating blocks for a file, we set the free bit & go to the next available free block. For the last block of the file, EOF is set.
- For the following operations, we will use the data structure given by UNIX.

  - `getattr`: struct static *st*

    ```
    mode_t st_mode
    st_nlink = 1
    st_uid = user id
    st_gid = group id
    st_mode
    st_size in bytes
    st_blocks
    st_atime, st_mtime and st_ctime
    ```

    Note: To find information about the above, use `lstat` (int fd, struct stat *), `fstat` (file).

- To find the status of the file system, `statfs`

  ```
  f_blocks: blocksize (1024)
  f_blocks: total # of blocks - use superblock & FAT
  f_bfree: free blocks
  f_files: total files
  f_ffree: free files (look through FAT & see how many are free)
  f_name len: 443
  ```
the flow can be given as

- **Startup**
  - Traverse Dir Table
  - Open Disk Image
  - Read / write Blocks
  - Allocate / Free Blocks
  - Give it to System I/O

**Recommendations for homework 4**

1) Although you want to encapsulate the details, you do not want to read & cache in memory.

2) The implementation should have block I/O's for every block in memory.

3) Start with a minimal implementation and build on your structure.

4) Add some optimization, determine how many block I/O's occur for the different operations & improve on them.