

**Operating System Fundamentals**  
**Prof. Santanu Chattopadhyay**  
**Department of Electronics and Electrical Communication Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 59**  
**File System and Secondary Storage (Contd.)**

(Refer Slide Time: 00:29)

**RAID Structure**

- RAID – redundant array of inexpensive disks
- Use of many disks increases the **mean time to failure**.
  - 100 disks with MTF of 100,000 hours.
  - $100,000/100 = 1,000$  hours or 41.66 days.
- Solution is to have data redundancy over the 100 disks.
- **Disk striping**. Splitting the bits (or blocks) across multiple disks
  - **bit-level striping**. The bits of a byte are split across multiple disks.
  - **block-level striping**. The blocks of a file are split across multiple disks.
- For example, if we have an array of eight disks, we write bit "1" of each byte to disk "1". The array of eight disks can be treated as a single disk with sectors that are eight times the normal size and, more important, that have eight times the access rate.

25

In our last class, we were discussing on this mean time to failure, like a disk that we have. So, it has got its lifetime and then it has got some time after which it is expected to fail. So, before that it is expected that the disk will not fail, the probability actually probability of failing is low below before that time.

So, if you have got a large number of disks then if you want the disk to be highly reliable then the cost of that disk will be high because the cost the disk will be expensive in nature. So, if you want to reduce the overall cost then we have to use inexpensive disk and inexpensive disk if we use then this mean time to failure will be low.

So we use a redundant array of such inexpensive disk with the configuration we called RAID structure and then this if we use many disk then the meantime to failure can increase. For example, if I use a say 100 disk with each disk having meantime to failure as 100,000 then you look into possibility of any one disk failing so, this is about 41.66 days so which is definitely not that high. So, what we do is that we distribute the data across the 100 disks so that each disk is not accessed many times in its lifetime.

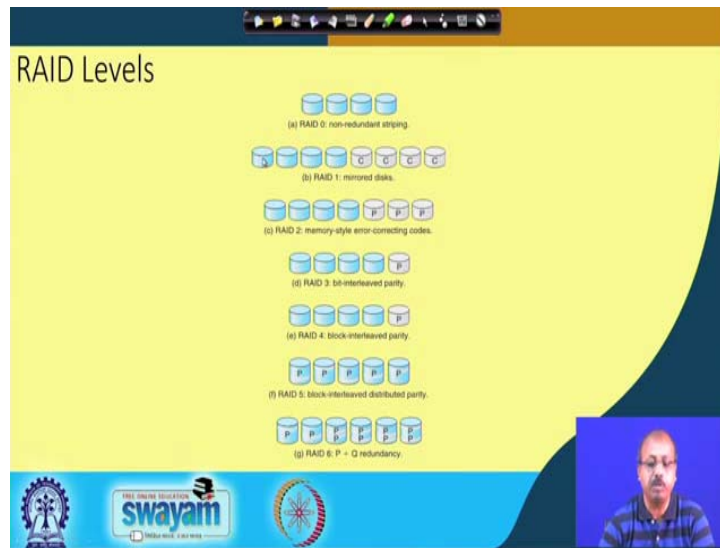
So that way we distribute the data and have data redundancy over the 100 disk and there is a concept called disk striping. So, we will split the bits or blocks of a file across multiple disks. So, we have got bit level striping in which the bits of a byte are split across multiple disk and we have got block level striping in which the blocks of a file are split across multiple disks.

So; that means, if you are accessing one particular block of a file then if in a bit level splitting striping so, you are not accessing that particular disk again and again so, it is distributed over the disk. Similarly, block level striping so, if you are accessing multiple blocks of a file then it is not that for each block it is going to the same disk so that way it is accessing a number of disk and as a result the access is distributed over them so number of disk accesses is low.

For example, if we have an array of eight disk and we write bit I of each byte to disk I. The array of eight disks can be treated as a single disc with sectors that are eight times the normal size because also and the more importantly they have eight times the access rate because; each bit is accessed from a different disk.

So, as a result of course, after getting the content into memory so you need to reorganize the content, but still the access is fast and this after in memory reorganisation maybe simple because that is done by the processor; so that reorganisation is a is done on semiconductor memory so that may be easy, but accessing the disk so that time is reduced by doing this. So, access time is increased eight fold and your sector size as if the sector size has increased eight fold.

(Refer Slide Time: 03:34)



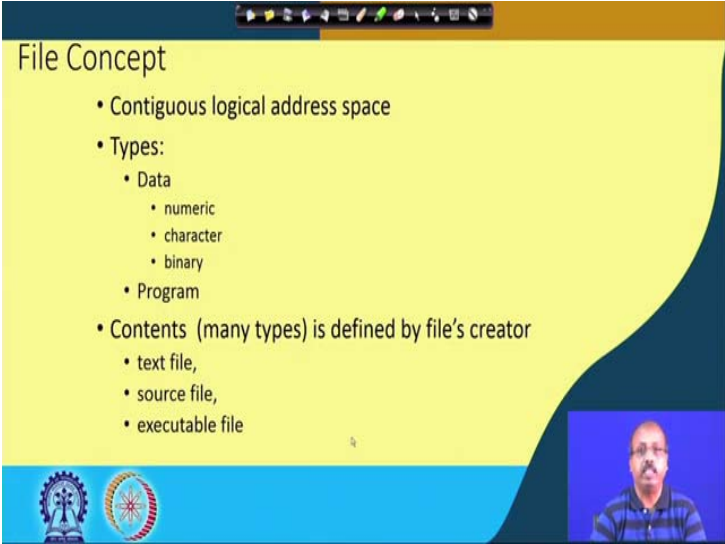
So, this is one RAID structure so, the RAID 0 configuration here we do not have any non-redundant striping. So, there is no redundancy actually the bits are distributed bits or blocks are distributed across the disk. Now, if you have got mirror disk so, basically for this disk so, this is a copy we copy disks that is a mirror. Similarly, this disk so, this is another copy disk so, which is the mirror so, you have got mirror disk.

So, if this disk fails so, we can access from this disk or if there is a high pressure high load on access to disk to this particular disk so, we can distribute the accesses between the copy and itself. So, we have got the mirror disk of course, with the additional responsibility that when this disk blocks are updated so both the original disk and the mirror they have to be updated.

Then we have got this memory style error correcting codes so, we can have some error correcting codes implemented and this parity bits they are stored in this parity disks and then we can just access the content and from this parity bit so, we can try to rectify the error. You can have bit interleaved parity also because here we can so, one base so, this parity bit is stored is computed and it is kept in somewhere intermediate.

So, in this way there are a number of such alternative like we have got this P plus Q redundancy so, this P number of disk are there plus there are some additional disks so which are the for the redundancy. So, here I have got say 4 redundant disks that way.

(Refer Slide Time: 05:11)



**File Concept**

- Contiguous logical address space
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program
- Contents (many types) is defined by file's creator
  - text file,
  - source file,
  - executable file

Next we go to the concept of file. So, file as a logical concept so, as a user looks into a file, it is a contiguous address space where we have got this individual bits of information they are stored. So, if you look into the types of the file, they can they are data files and program files. So, data files they contain data it may be numeric data, character data or binary data and the program files they contain program.

Now, content or the types is defined by the files created. So, you may create a text file you can create a source file you can create a executable file so that way the creator of the file so, knows better like what is the type of the file and different operating systems so, they will allow different types of files, for some operating system, all files are binary files and interpretation is left to the user.

Some operating systems they will demarcate between the types of the files and accordingly certain operations are possible on certain types of files only so, that is there.

(Refer Slide Time: 06:14)

The slide is titled "File Attributes" and lists the following attributes:

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on the device (disk)
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – information kept for creation time, last modification time, and last use time.
  - Useful for data for protection, security, and usage monitoring
- Many variations, including extended file attributes such as file checksum
- Information kept in the directory structure (on disk), which consists of "inode" entries for each of the files in the system.

The slide also features a small video inset in the bottom right corner showing a man speaking, and two circular logos at the bottom left.

So, if you look into the attributes of a file so, we have got the name of the file, the only information kept in human readable form so, the file name. Then identifier so, there is a unique identifier that identifies the file within the system. So, there is a tag which is unique for the entire system so that identifies the file. Type so, if a system supports the different types of files then this type information is necessary.

Location is a pointer to the file location on the device in the disk where exactly is the file located. Then the size of the file what is that is the current size and the protection so, it will control like who can do reading, writing and executing the execution on the files. So, for example, if I have written a program then and compiled it so, I may have the read, write, execute permission for that whereas, for other users so, I may like that they will only be allowed to execute the program do not, but they cannot modify the file so, that way the protection bits may be set.

Time, data and user identification so, these information are kept for the when the file was created, when it was last modified and when it was last used so, these information they are useful like if you want to get the most recent files and things like that, many a time that helps in searching for a particular file. Useful for data protection, security and usage monitoring so, they are very much useful.

Then many variations including extended file attributes such as file checksum and also these additional things are there in some advanced file system concepts so, we have got this thing. Information kept in the directory structure so, almost all the files they are kept

in some directory structure such that the search becomes easy. So, we have got so, each file has got an inode entry. So, this inode is a concept that actually came from the Unix operating system.

So, this inode for a file so, it holds all the necessary information so, this name, identifier, type, location, size etcetera. So, they are all kept in the inode and this directory listing so, it contains a pointer to the inode so, from there it is actually informing we can come to the appropriate entries for the file.

(Refer Slide Time: 08:32)

The slide is titled "File Operations" and lists the following operations:

- Create
- Write – at **write pointer** location
- Read – at **read pointer** location
- Reposition within file - **seek**
- Delete
- Truncate
- **Open( $F_i$ )** – search the directory structure on disk for inode entry  $F_i$ , and move the content of the entry to memory
- **Close ( $F_i$ )** – move the content of inode entry  $F_i$  in memory to directory structure on disk.

A diagram on the right shows a vertical rectangle representing a file. Two horizontal arrows point to the left and right edges of the rectangle. The top arrow is labeled "WP" (Write Pointer) and the bottom arrow is labeled "RP" (Read Pointer). The bottom edge of the rectangle is labeled "0".

The slide also features a small video inset in the bottom right corner showing a man speaking, and a Windows taskbar at the bottom with various application icons.

So, what are the operations that we can do on a file so, we can create a file, we can write in into the file so, at some write position so, we can there is a write pointer and there is a read pointer so, that tells that so, if this is a file if this is a file so, at there is a write pointer so, which tells that if I do a next write operation so, it will modify this location. Similarly, there is a read pointer maybe somewhere here and if I do a read operation then it will be reading from this location.

So, this read write pointers they can be controlled by programs by various system calls. So, they, but they are actually identify the position where the next read or write operation will be done. Now, so, we can reposition this read write head read write pointers within the file by the seek operation. So, seek is another operation so, this read write so, these are two operations and this seek is another operation by which we can put the, we can take the file pointer to a specific position within the file.

So, this is particularly useful you want to if you want to selectively modify the records in a file so, you can move this read write pointers by some positions in terms of number of records to put it at a particular position so, you to access a particular record.

Then there is a delete operation by which we can delete the file or you can truncate the file so that some of the data or the file is not deleted, but its content is lost so, you want to clear the file basically that is the truncate operation. So, we can open a file so, for opening a file so, there is a most of the operating system it will support one open system call.

This open F i so, it will search the directory structure on disk for the inode entry F i and move the content of the entry to memory. So, it will get the it will search out that it will search out the inode entry and then the inode entry will be returned to the calling process and then the close F i so, it will move the content of inode entry F i in memory to directory structure on disk.

So, that way the from so, since the file has been modified so, after that we want to close the file so, accordingly the inode structure on the disk so, that gets modified.

(Refer Slide Time: 10:51)

**Open Files**

Several pieces of data are needed to manage open files:

- **Open-file table:** Keeps information (inode data) about all the open files
- **File pointer :** A pointer to last read/write location, per process that has the file open
- **File-open count:** A counter of the number of times a file is open – to allow removal of data from open-file table when last processes closes it
- **Disk location of the file:** Many file operations require the system to modify data within the file. The information needed to locate the file on disk is kept in memory so that the system does not have to read it from disk for each operation.
- **Access rights:** Per-process access mode information

The slide includes a hand-drawn diagram of a file structure with two sections labeled  $P_1$  and  $P_2$ , and a word 'edit' written above it. A small video inset of a speaker is visible in the bottom right corner.

There are several pieces of data that are needed to manage open files. So, there is system wise there is one file which is called open-file table so, keeps information about all the open files in the system. So, maybe at some point of time, if there are say 10 different

users who are running their programs so, they might have opened files and this open files. so they are kept in the open-file table.

Then there is a file pointer, a pointer to the last read write location per process that that has the what can happen is that on the multiple processes they can do read write operation on the same file. So, maybe that one process is doing a read write operation here, another process is doing a read write operation here.

So, apparently it seems it is a bit impossible, but it is very much possible because what can happen is that maybe there are two programs P 1 and P 2 and both of them so, they are say doing the editing job and in doing this editing job so, they are accessing this P 1 and P 2 both of them are accessing the piece of code which is the editor.

Now, editor code so, it is divided into number of pages maybe the process P 1 so, it is actually using this part of the code where it is doing some normal text typing and maybe for search and replace so, this is a portion of the code for that, maybe the process P 2 it is doing a search and replace on some edit document so that way it is using a different part of the code.

So, the file is same so, there is a system wise there is only open one open file, but the read write pointers are process specific. So, every process has got one read head one read pointer and one write pointer for all the files that it needs to access. So, for the same file it may be used by several processes and each of them have got different file pointer.

Then the file-open count so, it is a counter of the number of times a file is open. So, so that so, as I was telling that one file may be opened by several processes now one of those processes if it decides to close the file, it should not happen that the file should close for everybody. So, there should be a count like how many processes have opened the file and only when that count becomes zero so, then only this the file should be closed formally otherwise only the this read write pointer for the for the process should be removed.

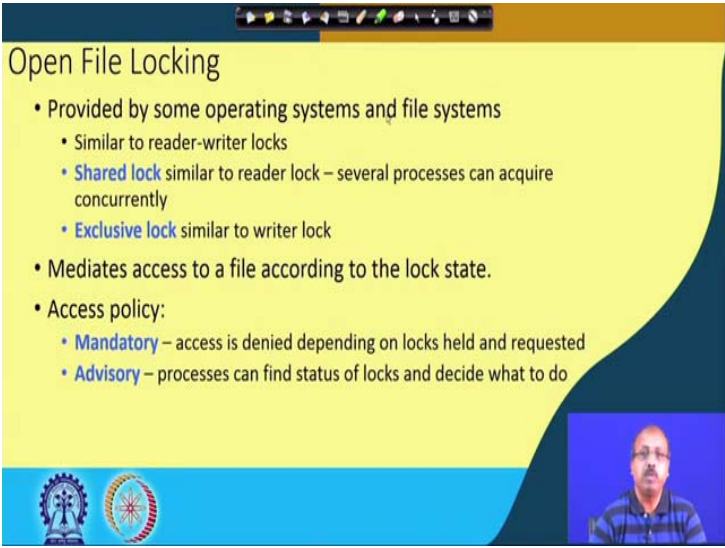
Then the disk location of the file so, many file operations require the system to modify data within the file and the information needed to locate the file on the disk is kept on memory so that the system does not have to read it from the disk for each operation. So, where exactly is the file located in the disk so, this sector number and all so, they should



be remembered and this information is kept in the memory so that so that the process need not be accessing the inode again and again to come to inode in the disk file. So, the disk copy of the inode to come to the location of the file.

So, that way it is kept in memory and from there it accesses. So, access right so, per process access per process access mode information. So, a process what type of permission it has for modifying a file so that is process specific and that way there will be access rights so that has to be remembered per process.

(Refer Slide Time: 14:34)



The slide is titled "Open File Locking" and contains the following text:

- Provided by some operating systems and file systems
  - Similar to reader-writer locks
  - **Shared lock** similar to reader lock – several processes can acquire concurrently
  - **Exclusive lock** similar to writer lock
- Mediates access to a file according to the lock state.
- Access policy:
  - **Mandatory** – access is denied depending on locks held and requested
  - **Advisory** – processes can find status of locks and decide what to do

In the bottom right corner of the slide, there is a small video inset showing a man with glasses and a beard speaking. At the bottom left of the slide, there are two circular logos: one with a gear and a person, and another with a sun-like symbol.

Then other type of information that also required, sometimes you have to provide some sort of locking so, provided by some operating systems and file systems. So, where one process wants to access a file and that access has to be made exclusive so, no other process should be allowed to access it.

So, access may be shared or access maybe exclusive just like this reader writer locks that we have so, similarly at the file level I can have this I may need this reader writer this type of access locks, this is particularly true when you have got say the database files. So, there are multiple processes that wants to modify the database file for example, maybe in a banking system we have got the file database files corresponding to the accounts of users now there maybe two transactions on the same account.

So, they are trying to modify the same account and as we have seen that this can lead to some inconsistency in the database. So, before doing operation so, there may be some lock set on the file itself so that when one transaction is accessing the file other transaction is not allowed. And, if the if both of them are trying to modify it or at least one of them is trying to modify it, but it may so happen that both of them they just want to read the content so that is the so, they in that case, we have got the shared lock.

So, we can have shared lock or we can have exclusive lock. So, this mediates access to a file according to the lock states so, this operating system should do this depending upon the file locking state so, it will find out whether it is possible or not. Access policy so, one is mandatory so, access is denied depending on the locks held and requested and advisory processes can find status of locks and decide what to do. So, in one case, we are we are not allowing the processes to access the file if the lock is set exclusive.

In another case so, this locks are set, but it is up to the programs to find out or the processes to find out what it should do so, after looking into the lock status so, it can take a decision that I will be modifying the file or I will not be modifying the file at this point of time responsibility is left to the process.

(Refer Slide Time: 16:54)

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

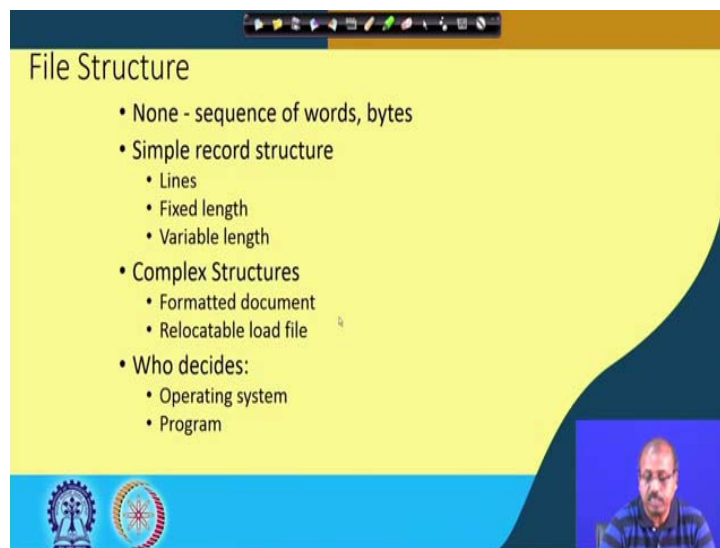
So, there are some file types and name and extensions like this is a executable file so, normally if they are given the extension exe, com, bin a sometimes no extension is given so, function is the ready to run machine language program so, they are there. Then the

object files so, the extensions are obj and o so, they are basically the compiled versions of files source programs machine language program and they are not yet linked actually so, after linking you get the executable file.

Then the source code files so, just to have an idea like what language is there what are the language in which the program is written. So, some compilers they will take that the file name should extension should be C for the C program C c for the C plus plus program like that.

There are different types of files and some extensions are given so that it is easily understandable like what is the type of the file and what can we do with the particular file. So, this is more of I should say human understanding because ultimately the software that will be accessing the file so, if it does not find it in proper format then it will not be able to do anything with that. So, in that case, the software can determine that this file is not in the proper format, but anyway so, this is more for user understanding.

(Refer Slide Time: 18:13)



Now, if you consider the structure of a file so, one is that you do not have any structure some operating system particularly like Unix they follow this particular structure.

So, it is just a sequence of bytes so that is that is how this structure is made so, the entire file is made. We can have simple record structure so, we can every line is a record so, particularly for the text file it may so happen that we take every line as a record.

Then we can have some fixed length record for example, if we are keeping the student information and every record has got student roll number, name and certain characters for address and all, all these character fields if they have got some fixed length when we have got a fixed length record structure or it can be a variable length structure also, we may say that this address part it can vary it can have 80 to 120 characters so that way it can be a variable length.

So, you can have this variable record structure also or we can have more complex structures like formatted document like this particularly this word processor when they keep this information so, they keep some formatting information with each of the text that it stores.

So, the formatting whether it is bold face, whether it is underlined, whether it is italics or not so like that it keeps all these information. Then we have got more complex files, relocatable load files so, these every operating system it will tell like what is the format in which the load file should be there so that it can be loaded by the loader so, that way it is that structure is again complex one.

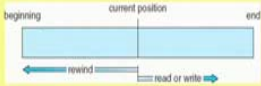
Now, who decides like what should be the file structure one is the operating system will tell for the system files what should be the structure. For example, this executable file the formats operating system should tell this is the thing and some user programs like say word processing software so that can tell what should be the structure of the file that it processes. So, may be decided by the operating system, may be decided by some other programs.

(Refer Slide Time: 20:18)


## Access Methods

Sequential Access

- General structure



- Operations:
  - `read_next ()` – reads the next portion of the file and automatically advances a file pointer.
  - `write_next ()` – append to the end of the file and advances to the end of the newly written material (the new end of file).
  - `reset` – back to the beginning of the file.



If you look into the access methods then this if this is the current position then the current read write pointer is here then the read or write operation, the pointer will go this way and if we do a rewind so, rewind will take you back to the beginning of the file. So, this is the sequential access so, if you are at this position. So, you will be accessing the next byte only for in this way ok.

So, this read next so, it reads the next portion of the file and automatically advances the file pointer so that is the read next operation. The similarly the right next operation so, append to the end of the file so, if the current end is here then it will be appended to the end of the file and the file pointer. So then advances to the end of the newly written material or it can be reset so, you rewind to the beginning of the file.


So, these are some of the operations that are allowed on a file. Now, you may find that looking into the particular operating system or particular programming language so, it will be telling you like what are the operations that we can do on a file.

(Refer Slide Time: 21:25)

## Access Methods

Direct Access

- File is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order.
- File is viewed as a numbered sequence of blocks or records. For example, can read block 14, then read block 53, and then write block 7.
- Operations:
  - **read(n)** – reads relative block number n.
  - **write(n)** – writes relative block number n.
- Relative block numbers allow OS to decide where file should be placed



So, that was the sequential access so, you are doing this access in a sequential fashion. There is a direct access so; direct access means that we can go to a particular record rapidly. So, sequential access if you want to go to record number 100, you have to start at record 1 then you have to go on reading till you come to record number 100, but that takes time. So, this direct access method so, it can be useful to come to a particular record directly.

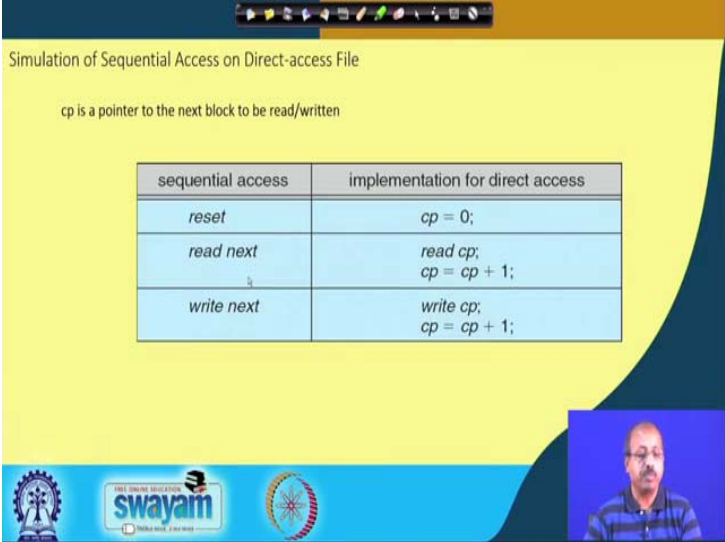
So, if the file is made up of fixed length logical records that will allow programs to read or write records rapidly because if the length of the record is fixed then by telling that I want to go to record number 100. So, I can find out the offset of the record from the beginning of the file very easily and it then it can be accessed there.

So, file is viewed as a numbered sequence of blocks or records for example, we can read block 14 then block 50 read block 53 then write block 7. So, this is possible because given a block number, we can immediately figure out what is the corresponding offset and we can put our read write pointer accordingly.

So, operations that are possible on direct access files are read n so, reads relative block number n so, current position from current position, it will go to block n and it will read that one. Then write n, it writes relative block number n with respect to the current block it will go to block n and write it there and relative block numbers, they will allow operating system to decide where the file should be placed.

So, so, it will be it will find out where exactly the blocks should be there and accordingly it will be putting the block there.

(Refer Slide Time: 23:10)



The slide is titled "Simulation of Sequential Access on Direct-access File". Below the title, it states "cp is a pointer to the next block to be read/written". A table compares sequential access operations with their implementation for direct access. The table has two columns: "sequential access" and "implementation for direct access". The rows are: "reset" (implementation: `cp = 0;`), "read next" (implementation: `read cp;` and `cp = cp + 1;`), and "write next" (implementation: `write cp;` and `cp = cp + 1;`). At the bottom of the slide, there are logos for "swayam" and other educational institutions, along with a small video inset of a man speaking.

sequential access	implementation for direct access
<i>reset</i>	<code>cp = 0;</code>
<i>read next</i>	<code>read cp;</code> <code>cp = cp + 1;</code>
<i>write next</i>	<code>write cp;</code> <code>cp = cp + 1;</code>

Then simulation of sequential access on a direct-access file so, basically if you have got direct access you have to tell what is the next block that you want to access, but for sequential access, it is always going to the next file next block ok. So, this is so, reset will take this pointer current pointer to location 0. Then read next so, it will do it will read the current pointer and then this current pointer will be implemented by one so that way you can simulate this read next behaviour.

Similarly, this write next behaviour so, it will write the, it will read the sorry it is a mistake so, it should be read. So, it will read the current pointer position then this current pointer position will be updated to the next position then it will be doing the operation.

(Refer Slide Time: 24:01)

**Other Access Methods**

- Can be built on top of the base methods
- Generally -- involve creation of an **index** for the file
- Keep index in memory for fast determination of location of data to be operated on (consider UPC code plus record of data about that item)
- If too large, keep index (in memory) of the main index (on disk)
- IBM indexed sequential-access method (ISAM)
  - Small master index, points to disk blocks of secondary index
  - File kept sorted on a defined key
  - All done by the OS
- VMS operating system provides index and relative files as another example (see next slide)

Other access methods they can be built on top of these base methods. So, generally we can we can create some index files so, normally if we want to search for a particular file so, it is difficult to tell the block number that we are trying to access. So, if you are looking for a particular record for example, a student record with a particular roll number then it is you have to search this entire file to figure out like which record has got that roll number, for sequential access you have to do it like this.

Direct access is difficult because for direct access I have to tell the record number that I want to access. An intermediary is to have an index for the file. So, we keep index in memory for first determination of location of data to be operated on. For example this some code plus the UPC code maybe some record number or something a some so, this is the this UPC code is basically some field that is that is that is unique for the record.

So, and plus record of data about that item so that way we can we can have it, if it is too large we keep the we keep index we cannot keep index in main memory so, we just keep the main part in the memory and the main in and the remaining part, we keep on the disk. So, this IBM indexed sequential access method or ISAM so, is it is one of the very important indexing mechanism so, they have got small master index that points to disk blocks that contains the secondary index. So, at the top level so, we have got this index that index so that index block so, it has got the pointers that that points to the next level of blocks.

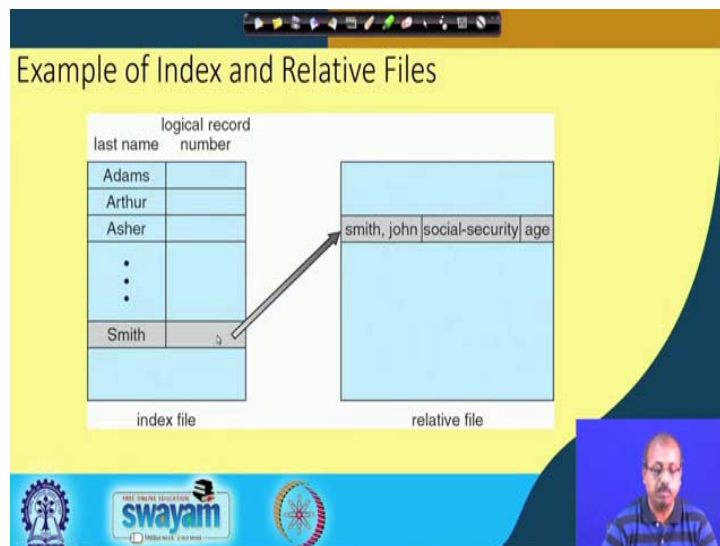


So, they are also indices to actual physical block so, these are the physical block. So, the first index points to this block, second index points to this block so like that. Similarly, so, so, these by looking into the so, this top level index is in the main memory, second level index is in the disk and the at third level we have got the actual disk. So, by doing this we can have very large file search so, if you want to search for a particular record so, first you look into this index to figure out like which second level index will have that particular record so that way we come to this one.

Again, you search on to this and then you come here. So, for searching on to this again you can understand that this block has to be copied onto main memory because we cannot search directly on disk, but that way you are just if there are multiple such second level disks second level indices available. So, we are just copying one of those files onto main memory one of the blocks into main memory and getting the doing the search operation.

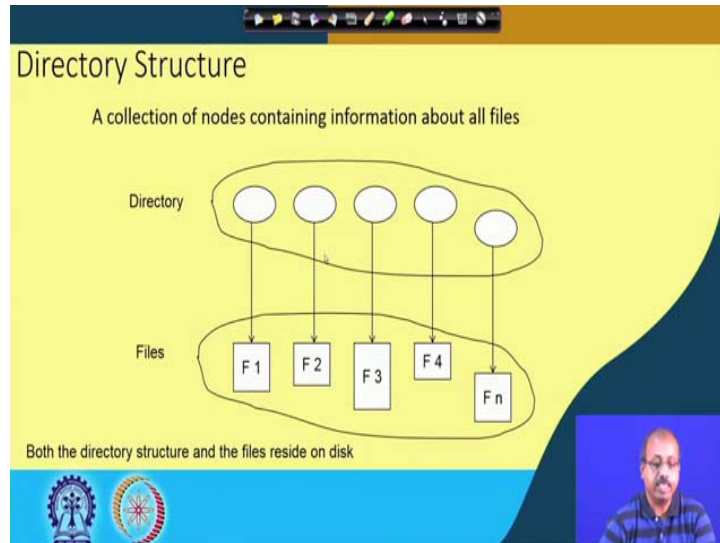
So, we have got small master index that points the disk blocks of secondary index and file is kept sorted on a defined key and all done by the operating systems so, user will not understand like what is going on. So, VMS operating system, it provides the index and relative files as another example. So, we will see that in the next slide.

(Refer Slide Time: 27:09)



So, this is a relative file so, this from the index file there is a pointer and then there is a relative position like within this where exactly it is located so that is determined by the relative position of the record.

(Refer Slide Time: 27:24)



Next, we will look into the directory structure. So, directory is actually the way in which they the information kept about the about the file. So, we keep a note like how the files will be there in the in the disk. So, here you see that these are the actual files that we have in the disk and directory is nothing, but another file. So, here this it has got these indexes or you can say the addresses where this file is actually located. So, this file may be starting at say sector number 100, this file may be at sector number 200, this may be at 500 so, like that.

So, this a values are kept in this directory so, whenever you are searching for a particular file, you do not need to look into this individual file blocks or you can just copy the directory into the main memory and start searching in the directory and directory containing information only about the files it is the not the actual file so, directory is a small sized one and then we can search for the file in the directory.

So, we will continue with the directory structure in the next class.