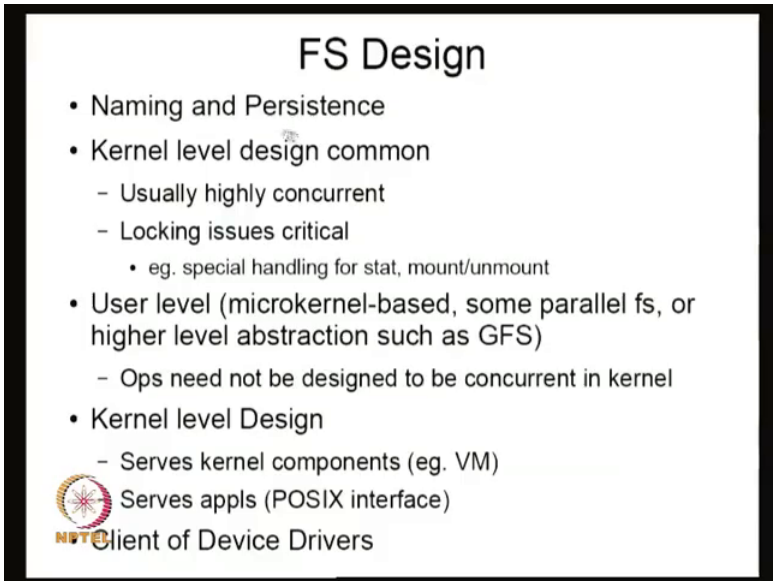
Storage Systems
Dr. K. Gopinath
Department of Computer Science and Engineering
Indian Institute of Science, Bangalore

Storage Filesystem Design
Lecture - 21
Filesystem Design Part 1: High-level design issues, Data Structures, Designing Redundancy

Welcome again to the NPTEL course on Storage Systems. Today we will discuss some high level aspects of a file system design and in the next class we will look at a specified file system. So, let us try to see what are the issues when you are trying to design a transistor.

I think, as most of you know a file system has two important aspects.

(Refer Slide Time: 00:56)



One is called as the naming aspect and the second is what is called the persistence aspect or storage point. Because, why do you use a file system, you need to able to give a name and say please store it or ready form. So that is why there has to be a name that is why it also have a storage function these two are critical.

Now, this naming can be of many types for example, the hierarchical file system was first introduced in Multix, this came about in mid 1960s. Multix was the first which were systematically developed is notion of a hierarchical file system. And if you look at

extremely simple file systems they may not have this hierarchy aspect in it. If you look at very old personal computer file systems you will find out that the magnitude what is called a flat file system. And this also can be there even in current generation file systems, because they are used for a specific purpose. For example, if you store boot images it may be that you do not need a hierarchical file system there.

And this is important because when you are booting I need to be able to look at the distracters and be able to locate the booting image; if it is a reasonably competitive file system then the booting software also has to know about all these things. So, it makes sense for booting images to be kept in a particular file system which has steady infrastructure than other ones.

Of course, nowadays these kinds of issues are not that serious because tuple the booting software like GAB etcetera, they all have complexity. Nowadays, for example, some of this systems like grout, they can even handle not only file system details they can also handle what is called voluminised details that is you are providing a type of abstract storage which is compost a multiple disk, and then there is two levels of abstraction that you have to work with. The files down abstractions about how you store the thing in a file system, it is also an abstraction of how when your multiple devices are combined together to provide a larger device how this blocks themselves are combined.

So, there are substitutions were the naming part can be done a slightly simpler manner, but the current tradition is traditional file systems have hierarchical models. And as we have looked at earlier it also tells some other context this hierarchical model does not work very well, which we are talking about a parallel file system does not work very well because a parallel consists wants to do things in parallel. If your name consists of multiple components then the definition sequential; if it is things are sequential then there is a again issue of loss of parallelism or means you cannot really scale up easily.

So, there also you will find that some people play around with, this part of it thinks like different manner. Or if you look at other kinds of good scale kind of designs again the same sequentiality is only a problem, you want hatch things in one short and give it do look up somehow very quick. You do not want to keep on iterating over each of the components of a hierarchical file, so that is why this also has changed quite a bit, so that are various designs possible here.

Now, about persistence also there are quite a few possibilities. Normally, we assume that we have a disk, which towards the files, but you can also have it in memory or other devices also; for example, flash or a feature something called storage class memories. So, persistence can be let us say double memory, but because it is volatile you need something called battery backed memory; if you are able to provide battery backed memory then you can make it persistent. Of course, you have depend on the battery being let us say really doing its job. So, you can make it persistent memory or you can use a disk or other kinds of designs, each of them has its own implications for the files in design.

For example often times you will find unique scaled systems something called as slash temp files. A slash temp for files is basically the something that is some files are being stored in memory, and basically you are not trying to store it in just because the temporary files even in case the machine crashes and we lose things does not matter. And because it is in memory, it is fast. And this is why this temporary files is useful is because nowadays memories are somewhat cheap compared to old times about 20 years back or more. So, it is we have fairly large memories and it can be everything in memory especially those who know the does not that do not need to be persisted. After the process visited then that kind even also is quite obvious, so that is why you will find many temporary file system that are used.

So, there is a lot of possibilities here I just mentioned that the conventional thing is to store it in disk, and there are also people will do tape file systems. So, those things are slightly more let us say specialised, but you can do tape file systems also. And another classic thing is that of a CD ROM file system, and it is written once and read multiple times and everything is structured, because it is a lonely kind of medium those file systems I have studied different characteristic that what we need for develop file system because now write involved. So, those kinds of designs also have slightly different design let us say prospective is to take into account.

So, what I was trying to say was that there are lot of dimensions signatures possibilities. Normally, in a file system design, it is usually they use a kernel based design is very common. Basically because trying to put the kernel, so the file system also (Refer Time: 08:13) the kernel introduces lots of transitions in and out of the kernel. And you need a strong service reasonably awful, it is not something which is a luxury, it is an need quite

often. Machine boots, when you want to do anything to read data, input data, output data whatever it is a very reasonably common thing. You try to be everything in memory as far as possible because that is where the speed comes from, but we have to interact with IO devices reasonably often.

So, many people have tried to take the file system and keep it outside the kernel, but generally it has not been widely popular or widely used. For example, minix 3 they have a reasonably file system, but if you studied carefully you will find that it is slow, the numbers of times it goes back and forth between user level and kernel level is quite dramatic, it is quite substantial. And you need to do lots of thing to make those kind of a file systems fast, so that is how generally most people use a kernel based design, and usually this have to be highly concurrent.

Why do you have to be highly concurrent because if you are dealing with some slow and devices above, but bad luck you are behind that slow operation, then you are really stuck and these things take a long time. Because they are as you well know the speed differential between disk and memory is a factor of three or four or five or else a magnitude. So, if you are slightly let us say if you take your design of bit slightly like this, it is possible that you might get behind a very slowed operation which is so slow.

That finally, it gives your performance the way, which is unacceptable that is why it is always important to make your design as highly concurrent as possible. Because you do not know when that slow operation is going to get interlinked with operation that goes a ahead of it you do not when it is going to happen is it going to happen that yes that whenever it happens it is going to be problematic.

For the same reason the locking issues are also critical and why is it critical because if it so happens that a slow operation has kept the lock then you are again stuck. So, you want to make sure that you do not you ensure that multiple locks are available so that the lock only those things that are necessary for you not lock at a high level or gross level you want to make sure that does not happen. And there are other some special issues also. For example, there is a common thing called stat what is stat doing it gives you the file system state status whether the file exists etcetera those kinds of operations, it tells you something about a status of file. And it turns out if you look at your scripts, lot of the

times the interact with the company systems to lot of scripts, and these scripts all the time are checking whether some particular file exists or not.

For example, suppose I am being v m often times there is a configuration file called vmirc dot vimrc. So, first thing that that program does is checks whether it is presenting your local directory, your home directory. And if it is not there who knows it may be therein slash some place in the file system hierarchy which is common for all the users. So, it might to some slash users slash beam model wherever it goes somewhere and checks whether there is a system wide vmrc. So, even before you have started looking at we had started it is already done two look ups checking within your own directory and then looking up in some other part of the file system hierarchy. Whether there is a configuration file for vi which is appropriate for the whole system, which is used in case the user has not specified it.

Now, invariably turns out if your libraries for example, if you are searching multiple places, not one place multiple places, it can be as in 5 to 10 places. So, what we are talking about is when the application starts, it is going to do lot of stats, it is going to do whether this file exists that file exists is it open all kind of things. Now, if you want to make your system reasonable quick you have to do this thing quite frequently, it quite fast if you cannot do it does not work out. And actually sometimes you will notice that some file systems they take some special paths to make this fast; sometimes they do not even take a lock before they look up this stat file.

There is normally when you have to look at an object you will take a lock make sure it is nobody change the status of it state of that particular file then only you will look at it. But this turns out that this locking order is too much, so sometimes now people many file system do is they are optimised in a particular way they will basically say I am going to check whether that file exists or not without taking a lock may be. It seems dangerous, but it also that there are some special based things are done, so that you can handle calls like stat which come quite frequently.

Same thing similar to that there are issues like mount and un-mount. For example, if you are traversing NFS kind of file system, it turns out that you are going to walk on the set of file systems mounted. And normally what it should do is it should take a lock on the set of file systems mounted, and typically we keep doing these kind of things it turns out

there is some amount of processing required to do the get the lock, and it finally, these things slow it down. And if you are planning to make your file system scaled to multiple course name machines each of these things which are not parallel will essentially they do some performance.

So, what you need to actually take a lock on the list of file systems mounted because you are afraid that somebody could be unmounting, but normally what happens is the unmount operations extremely in frequent. So, what you can do is you can say that since unmount is happening very rarely, but lot of the times people are accessing various file system, for example, NFS handle has been given, you need to go on seeing which particular NFS file system is being accessed. So, we have to actually work the set of file systems.

So, it turns out that more often there are reads than let us say right kind of things like unmount. So, because of this you can optimize this system also, often times what happens is that the wait is down you essentially assume that it can walk on the very first place without taking a lock and then whenever you want to do unmount, you take special care to see that nobody is actively accessing it means. So, it will now responsibility of the unmount code to do some extensive checking before actually does it the idea is to shift the burden the very frequent read operations to the slight moving frequent operations which is slight more costly operation, but done in similar.

So, you will find that most file system remains worry about these things systematically they think about these things a lot because if you end up by as I mentioned already behind a slow operation your performance can become arbitrarily bad. So, that is why this the reason why often the kernel designs are favored for most file systems. Firstly because you have to mix slow and fast operations and you need to have concurrency and need to proper locking and always things are of better control at user at kernel level. Of course, this sequential is not really true for some other kinds of systems for example, parallel file systems which are not let us say mutilator intensive, they are doing lots of big writes, then it comes out that you actually ideally want to avoid data going in and out of the kernel. Because when you write something typically we are through the kernel, you get the buffers.

Now, pushing your data through the buffer from user level to the kernel buffer some kernel buffer to go outside to the disk what I call, it turns out to be also expensive. So, in some other context a kernel level design is not appropriate. And it often tells the user level design is often appropriate. And this also is similar to the situation in networking also. In networking, there are protocols including the kernel, there are some protocols including the user level.

Now, if you are doing user level networking it is often appropriate to do user level file system in the same way, so that you try to avoid going through the kernel. The kernel is essentially set up the connections will setup the you know validate parameters etcetera or check authentic equations etcetera, but finally when it is done it tries to directly talk to the end devices. So, those things are also possible, but normal design is based on the kernel, level design.

So, the other possibility of course, as I just was talking about user level this is often time in micro kernel based systems like minix 3, for example, I finished already parallel file systems find user levels designs useful because you can always sending it all the data through the kernel; that means, that can multiple cortex. Or we already seen for example, in Google file systems it uses another native file system ext 3 and so this particulate file system can be at user level this is kernel abstraction, but it is essentially done being done at the user level.

So, in some of these user levels file systems depending on the design or simplicity often times concurrencies may or may not be seriously stressed. It is up to the design. For example, in a parallel file system, if there is a very long file very large file let us say multiple gigabytes, then it is fairly easy to ensure that that 1 gigabyte file or 10 gigabyte file is broken up into so many pieces and each piece done independently by one node. And that node need for in concurrency because somebody at the higher level has taken care of how it is being accessed.

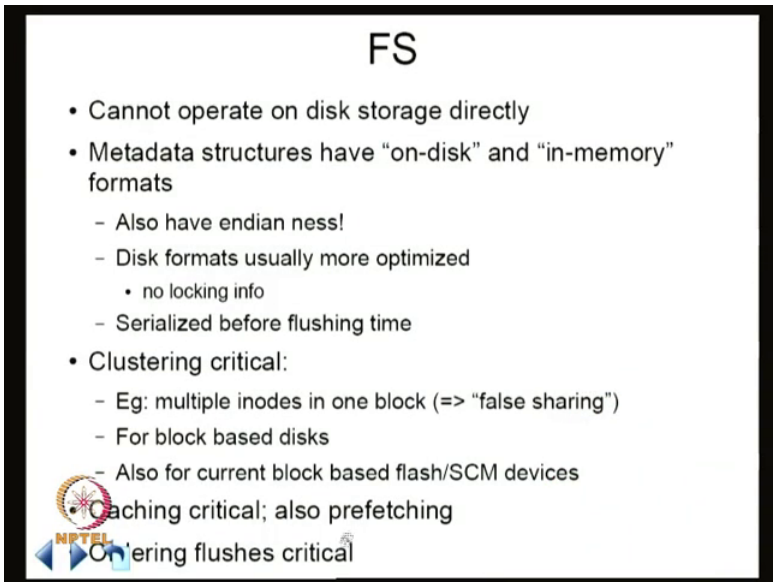
So, in that kind of situations you might not really worry about concurrency because that concurrency is being handled at a slightly different level somebody is managing how each of the pieces is being accessed. So, for that reason it may be simpler. If you have a user level design, because you do not try to be fairly concurrent, the design can be slightly simpler.

Now one thing about the kernel level design is that it has to it has got multiple clients, for example, it shows kernel components like the volume manager I am sorry I made a mistake, it can like the virtual memory manager. And it also has to serve applications the process into this. Now, this introduces lots of other additional complexities let us try to see what this is with respect to virtual memory. Now, a file system it depends on some services from the kernel, and it provides certain services to the kernel also like for example, here it is providing a service to the virtual memory manager.

Why is that because if you are using virtual memory, it may be that you are swapping happens to your file system. It may because swapping can happen to a block device where no file system exists block device or it can also be swapping onto a file system that is your backing store for pages could be in a file system. That means, if you are using a virtual memory manager, it may actually need a services of the file system to be able to pull out the corresponding pages. So that means, that the VM actually will depend upon the file system capabilities at the same time the file system itself needs the VM capabilities because the file system is actually a manipulating memory and all memory locations are virtual that means, that they are in the control of the virtual memory manager.

So, there is a certain interesting circularity here, where the file system depends on the virtual memory management because all the addresses at ever talks about all virtual. At the same time, it is providing a service to virtual memory manager, so there can be interesting circularities and this have to be taken care of that is why the design of these kernel level designs in a virtual memory based operating system it is going to be far more complicated. At the same time, the kernel level design has to certainly handle the POSIX interface. And always think about file system doing they have the user device driver framework.

(Refer Slide Time: 23:31)



Now, one thing about the file system is that it cannot operate on disk storage directly because disk storage is typically block disk you can directly manipulate it you have to get portions of it into memory. And then manipulate it there that is why you always have what are called on disk structures and in memory structures in core formats or on disk formats. And because of this also they also have an endianness, let us begin the analytic endian how you store it, it matters. So, be careful with this also. Now, disk formats usually are also more optimized, because in the past at least disk storage was expensive. So, people will ensure that you store in exactly what you have to what has to be stored.

Of course, nowadays this issue is not that serious, but still people do worry about having to store lots of things where probably not necessary, they avoid quite avoid as much as possible. So, I think this is similar for example, in the case of processes, it something called new area and also slash pros structures right, so pros structures. And it was this particular distinction was need let us also more at a time memory was a big issue, so that is why you noticed that whatever you can avoid storing, you like to optimize it away.

So, the similar is also happening here, we try to avoid unnecessarily storing things then disk may not be stored out and of course, the usual things are that you try to avoid you can get to store in locking information etcetera, so those things are all out. And most of the times in memory structures there are parameters they all have to be serialized before they are flushed to disk. So, any inter connections between various components have to

be serialized, have to be essentially made into disk characteristics before it is sent out that is why you need to this is special step taken before it is flushed disk.

So, what you do is often times you take all data that has to be flushed to disk and reconstruct it in a new buffer and that part is actually flushed out. Another thing about a file system is that clustering is very critical. Normally, a block is chosen disk has a block is chosen for by certain criteria that is typically most files let say about 10 kilo bytes or 15 kilo bytes whatever then you would like to keep a block size which is some more connected with this particular frequently occurring files size. And most of all in the past 4 kilo bytes or 1 kilo byte has been chosen blocks.

Nowadays with the increasing use of music files and what not probably multiple mega bytes could be good size, but generally in a systems perspective 4 kilo bytes, 8 kilo bytes, 1 kilo byte these are the common sizes. Now, if that is the case if a block size is held to be 4 kilo bytes then you also need to store things like inodes. What is inode? Inode is basically the metadata of a particular file.

And typically an inode can be about 128 bytes or 56 bytes or in some advanced file systems 14 bytes that means, that you cannot store when inode in one block is because of block is much bigger 8 kilo bytes or 4 kilo bytes. So, that means, that if your are trying to flush inodes to disk, you want to cluster multiple inodes into one single 4 kilo byte for example block before sending it out.

Now, this can introduce what is called false sharing. Why is that happen, it may happen that since you have put multiple unrelated things in one block, what was once upon a time complete unrelated now gets accidentally related because unrelated things are getting put together. So, when I am flushing it, I am probably sending probably eight inodes or four inodes to disk at the same time. Now, if there is any requirements for ordering of this flushing, now this extra unrelated or false sharing kind of relationships that crop up that can reduce certain circularities also.

So, basically you may have situation where A depends on B that is a is A inode B is the data and you have C and D, C is an inode and D is a data. Now, I can put A and C together and then now it may be that I need to flush B and D in a particular order with respect to A or C - the inodes. Now, once I put A and C together the inodes together that

is a new dependency that has cropped up which was not there before that might be the wrong kind of dependency, and you have to worry about this when you cluster it.

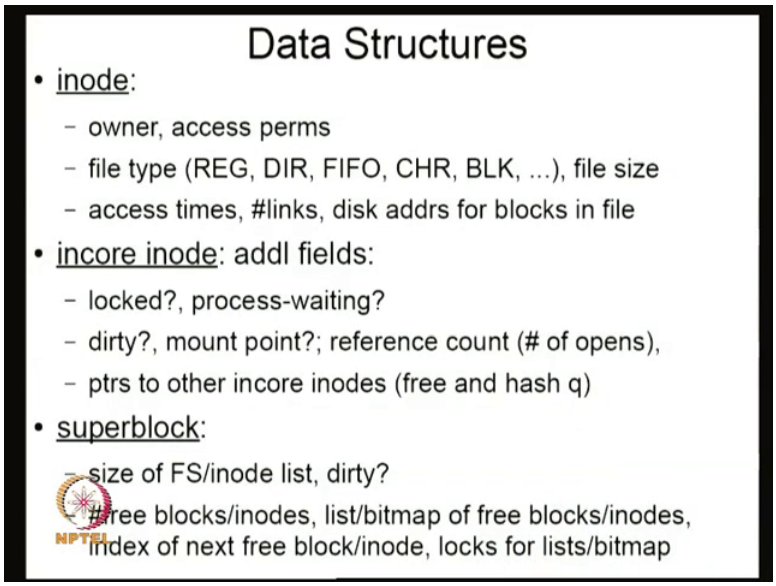
So, you also of course, caching this critical you also need to do what is called prefetching that is caching is case where you request something and once you have picked up your the data it keep it around in case hoping it is useful. Whereas, prefetching you actually predict that something is going to be useful and get it before you want it is asked. So, it turns out all these things are related clustering, caching, prefetching, ordering all these are quite.

However, related and these are going to be there in the future also not just current situations, current now we are usually using disks, but they will also be appropriate for other types of newer designs like for example, based on flash or what is called storage class memories. Because there also it turns out some of the same similar issues crop up, unless this SCM devices are available at byte level byte or board level. There are some of these designs have gone to crop up this will be accessible at the board level for example, that is you can write and read at the level 32 bits right. Now, there are those designs probably some of these issues we come less important because this is looking more like memory now not like disk blocks. Even there it turns out that even if we have board level access, read and write, for board levelling purposes you may want to read and write at higher level of similarities.

So, again all these issues is to be characterised, it is not something which is going to probably disappear. Of course, if there is no board levelling issues and you have byte level access, all these (Refer Time: 31:36) we are talking about is not going to be that critical that was the case sometime in the past. For example, you may have core memory, core memory did not have either these issues, it was persistent it has byte or board level access and there was no board levelling required. So, this was in early sixties, those designs if those kind of designs crop up again of course, we have to we can throw most some of these issues that (Refer Time: 32:04).

So, again then there I am talking about is because the file system design depends very much on the devices k you have to think about designs before you start doing anything. Again we talked about disk formats and memory formats.

(Refer Slide Time: 32:25)



I just give you simple example of some of the kinds of structures. You have an inode which is under disk for example, but if you design it needs to keep track of information of who wants it that is why it has to be part of that disk information, also access permissions, the owner access. The file type there are various types of files regular files is the ones from most common we have worked with normally used directories. There are sometimes FIFO files that is for example, in a types they can also have a name in the file system hierarchy and typically it is there in they kept in the slash temp directory you can have temp files also. Character files and block files these are basically for raw for direct access without a file system mediating it.

So, you need to say what kind of file it is, you also want to say what kind of file size it is. Now, this file size also is a slightly tricky business, because it turns out a file size can have different sizes, different values at the same time. For example, there could be a disk file size, there could also be a in memory file size, also there could be an another intermediate file size while something is going on, let me explain what I mean. Suppose, you have a file and you try to extend it, but you want to write let us say to make it more graphic let us say you want to extend it by 1 gigabyte, there is a file we can extend it. How do you extend the file by upending it, now we cannot write 1 gigabyte, 1 terabyte just like that will take time it will take some time.

Now, there is a notion of the file size that is there in the disk. now we are starting to write it now you not complete the write. So, you cannot immediately update as you are writing it right you cannot keep on updating the disk files, because that is going to be cached in memory. The inode that metadata will be cached in memory and that will be flushed to disk at some suitable intros that means, that what is there on the disk the files is different from what is there on the what is being as your operating on it.

Not only that as you are writing it the kernel keeps track of what the you might be getting certain information about that file, let us say you are talking out a email file, it may be that as email is coming through a slow network, it is being upended to your file. If you look at email files in the past at least all their email would be as a single file. Of course, now a days with what is called the mailed in format every file is a separate file sorry every message is a separate file, but in the early past for example, inbox format you would find that all your mail would be a actually a single file, a single file. And what is going to happen as you get mail you will the particular mail application is extending a file is doing a upend.

Now, this upend is how is it happening, it is happening because the network packets are coming in as soon as coming in it actually attaching it. Now, it may be that the files that the data you are getting does not fit of single full block or memory page. So, it is going to be partial that means, that you do not have any your files as for example, there is a particular size that as it is happening as it is unallocated, because now we have unallocated 4 kilo bytes disk block for it as allocated. So, that is what is there on the what is called tentative size of the file. So, there are three files is as here, one is as it is exact to the amount that has been upended to it, one as has been extended, but it might have some junk in it; at the end of it and as it is exists in the disk because I have not updated it.

So, there are I am just giving you three sizes, but actually there can be one or two more also depending on the circumstances. So, we have to make sure that these files are taken care of properly. If you do not get this right because of the concurrent context you can essentially make some mistakes, actually it happened in some file systems I will talk about it later. So, one thing I just want to mention is that there can be multiple files in this at the same time some in the disk some on memory. And you have to ensure that these things are taken care of properly.

Inode also might have access times. Now, there are two types of access times one could be incore one could be inode disk. It is basically (Refer Time: 37:52) of the last term we accessed it one of the last time we modified it or created it that has persist also because you often times look up the information. But once it is in memory once you are accessing a file you could be accessing that multiple times you have to update it as you are accessing it also as a changing also little bit, and that also keeps changing while it is in memory. And we will be flushing it to disk every so often; that means, that the access time as exists on the disk is going to be slightly different from what is there in memory there is going to be some kind of time like.

Of course, you can attempt to do write through, all the time any time anything happens changes you keep on writing, but write through designs are extremely slow, nobody actually likes to tolerate such slow file systems that is why, this there is going to be a some disparity between access times also. So, file size is there multiple possibilities access time also multiple possibilities. Again the number of links you might have hard links and soft links. So, you might want to keep track of this part if you have multiple when you have hard links.

In addition you need to have disk addresses for blocks in file because, this is we are talking about disk inode. So, it has to keep track of where it is going to find actual disk addresses for blocks that also has to be stored in the inode, all these in a attribute. Whereas, in the case of incore inode, you need additional fields in addition to everything that is here are slightly different versions of these things, you also need to know whether it is for example, the inode is locked that means, that only one part you can access it, the other parties can this was synchronization process. Or is a process waiting for this for example, it may turn out that I tried to access a particular file, and then as I am accessing it somebody also wants the same file. We want to able to say that once it my request has not handled, you want to let somebody else request actually handled as soon as the data comes in. So, you can actually incorporate things like or file that multiple processes can be waiting for the same particular file. So, you need to get back with that also.

You also need to keep track of whether it is dirty, what is its mount point, what often times what happens Is that you can have a file system which can be drafted on to another tree structure of the parent file system. I can always; that means that my file system is subsidiary file system can be made to lie on top if another tree like tree structure of a

bigger file system. So, I want to know where exactly I am attached to the parent file system, So that is a mount point where I need to also keep track about it and that is necessary for if I am doing enough files also I need to do that.

I also need to know how many processes are open this particular file. Again this is strictly this is not anything to do with a disk is only connected with process which are using this particular file of a files has been opened, so I need to keep track of this also. And of course, you need to do another thing like pointers to other incore inodes. So, this cache is also we have to keep track of.

Similarly, super block also we need to keep track of size of file system size of the inode list, how many inodes have been allocated what is the; if you want through all the inodes then I need to keep track of those. Whether the super block is dirty again when whenever you allocate or deallocate something, you are going to update in memory structure correspond to those things.

The question is how often we keep on flushing it to disk. So, you do it every single allocation, deallocation that means it is going to be invariably slow. So, you need to somehow ensure that you have the rights of that information which is safer or in spite of crashes and (Refer Time: 42:24) delicate business. So, you have to find the way in which in case the crash, you can recover in the safe manner that something that has to be done. I think basic reason is that you have to keep track of what you allocate and deallocated, and this has to be reasonably accurate in spite of crashes, but you are not going to be able to flush your disk every single time you allocate deallocate. If you flush your disk every single time you allocate deallocate it is your file system is invariably slow.

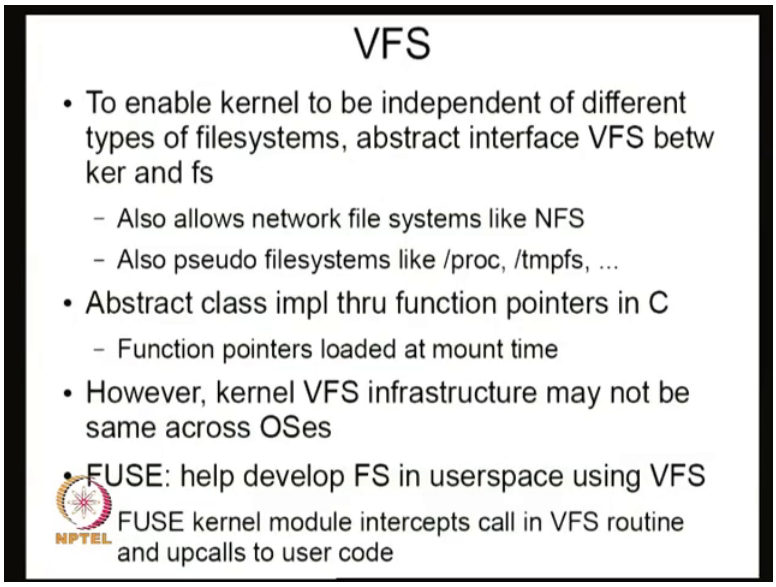
So, we have to somehow figure out the way in which you in spite of if there is a crash you come out in a safe way that is how we have to do. So, and that is not trivial and some of the most complications most complicated things that people do in advanced file system is how to do it correctly you want to be fast, at the same time not give a window opportunity where you essentially lose a file. So, there is lot of tricks people play here and it is still the complicated tricks which unfortunately will be difficult to go through here.

So, basically you need to keep track of number of tree blocks, the inodes, the list and bitmap of free blocks, it could be either list or a bitmap, a free blocks and inodes, index

of next two block and inode locks etcetera all these things have been kept. All these things keep on changing every single allocation, deallocation. You can see the magnitude of the problem if you really want to keep all these things current on the disk. And only if it on the disk you actually survive so that means that any time you do any of these things if you can use a log then your log is going to help you to keep track of what has changed. But the log has to be persistent that means, that if you are unable to make the log persist and crashes etcetera, they are again you lose it. But the prior thing is to do it in such a way that you recover in a way which is consistent, so that you lose only mostly switched off not things unrelated to the recent activity.

Again as I told you there is lot of complexities here and there is one thing which I will see if I can look at it next class some of the aspects here.

(Refer Slide Time: 45:53)



Now, the other things that we have to is to as I mentioned a file system is both a consumer as well as the provider. It provides some services to the kernel for the virtual memories of system for example that itself requires some services in the kernel. Now, as you saw already there are sufficiently complicated things going on. So, a kernel typically would like to see that it does not have to know all these complexities. If you can isolate a complexities then there is some chance that a kernel actually work. The way it is done is using something called a virtual file system model. It is a abstract interface that will have kernel on the file system.

And the good thing about this is that such an abstract interface allows things like network file systems also what are called pseudo file systems like slash, proc, temp files etcetera. So, for that reason virtual file systems are quite commonly used UNIX based systems. And basically each what we have is a virtual file system essentially is an abstract class, that you concertize for each particular file system through function pointers. At the time of mounting these function pointers are basically installed with and they are connected to the real corresponding file system calls for that particular file systems.

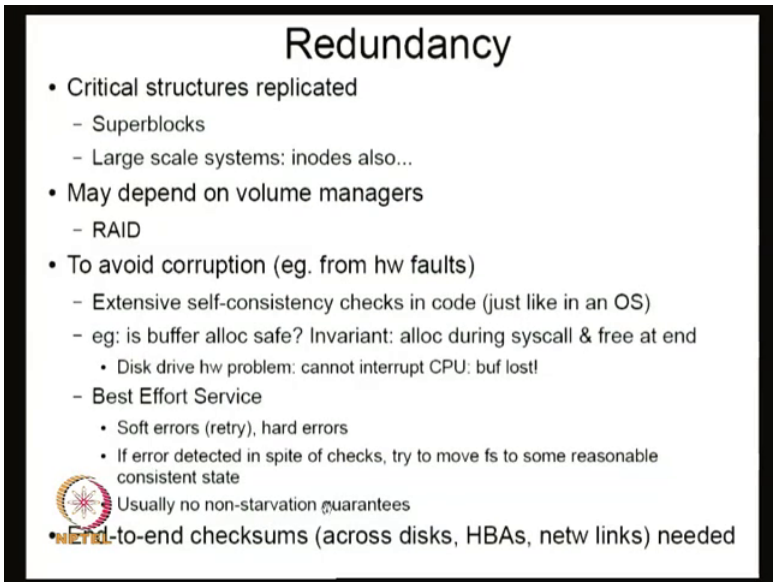
Now, only problem with this kind of model is that the kernel VFS infrastructure may not be same across OSes they could be different. For example, the one which is available for solar is somewhat different from what is available for Linux; that means, that you cannot really use the same kind of abstract interface between Linux and Linux kernel and a (Refer Time: 47:31) file system relative go on think a little somewhere. So, that unfortunate that is the way it is.

There is now slightly different model something called fuse basically to avoid all these dependency of the kernels, it is possible for you to have a kernel module which implements the fuse model. And by doing that what you can do is we can ensure that we can develop the file system user scales, but using the VFS which essentially calls these fuse kernel module routines, but in turn basically makes up calls to user code.

So, basic idea is if you want to avoid some of these dependencies, what you can do is you have a regular file system with the regular meaning other states unique standard accesses of the files like saying slashes slash p whatever. And then what happens is that it gets into the kernel it goes the VFS routine, the VFS routine will actually get it will make some calls to the fuse kernel module and this actually makes the up calls. And then you implement a corresponding file system routines user space.

So, the basic idea is that once you have installed this fuse based file system, the VFS routine calls actually will go to the fuse kernel module routines which actually reflect it back to user space. And then you know have complete control that user space to use whatever without worrying about what infrastructure the kernel is providing. You can now only solve the standard like c calls you can call these things, so it become that much simpler. So, lots of people are using these kinds of modules now.

(Refer Slide Time: 49:56)



One thing about this particular with file system is that you need to have redundancy. Most of the critical structures are replicated, it can be super blocks. And also in many large scale systems example if you are talking about large scale para file systems which has multiple terabytes of storage for example. It turns out that if you do not replicate your inodes, you can in a trouble also, because you need to actually replicate even things like inodes multiple two or three copies you should have. Because it turns out that at a particular size of the file system, the chances of your corruption of your data structures also starts increasing. And once the minute you have the inodes is that also gets corrupted then your file system is essentially useless.

I have some small experience with this because (Refer Time: 50:57) we developed a parallel file system and we did not because it was meant only for scratch space does not because on a parallel file system often times, you use the storage for what is called out of core computation. Because memory is not big enough, you want to dump some partially computed results on the storage as a temporary thing and then read it later when you read it. So, this particular design what we designed what we had did not replicate inodes and we found that every two or three days your file system will get into some corrupt state and the investigation showed us that inodes were getting corrupted.

So, it will lasts for about 2 or 3 days after (Refer Time: 51:45). And of course, we did not have a very large file system that time, but probably 10 of giga bytes, but nowadays a

terabyte this will happen every hour, every hour or more. So, it is not going to be. So, we have to actually replicate inodes also if you look at our current desktop file systems, we do not replicate inodes, we usually replicate super blocks for example, the BHD file system replicates super blocks, it get it does not replicate inodes. We are depending on the fact that you have a smaller system that usually thinks are ok.

Other thing possibilities are these are the file system now itself. We can also look at the level one level below what is called the volume managers. For example, if you have rate consistent you do random c at the block level, and it is independent, it is not known to the file system, the file system is set obvious to this part. So, this also is a fairly well developed module in industry. And the only problem with that is the that the file system and the volume manager have to really bit developed together; otherwise there can be some let us say redundancy in terms of operation or in terms of inefficiencies in terms of operation.

So, for example, if you use something like raid file it has got what is called a small write problem. What is a small write problem, you will find that when you are doing raid file you are writing multiple strikes, five strikes. So, if you write all the five strikes then the parity is computed in one short, because you are writing all the five strikes, you compute the parity and write it. Whereas, if you are writing not the full all the five strikes, you also writing one strike it results in what is called the read modify write cycle that means, that the block size should correspond to the one which does not result in a read modify write cycle. So, somebody has to choose the size of the block size in such a way that read modify write cycle you do not have. If this is not done then the performance will be disastrous. So, it has to be some understanding between each of the components file systems in volume managers.

So, again we need redundancy for as I mentioned, we have to avoid corruption because that could hardware faults. So, one thing about a file system is that it has to do extensive self-consistency checks in code and this is quite of course, true for also operating system code also. But this is more desperate because if you do not get this write, your data is getting destroyed is getting corrupted, because finally, people worry about data that is what you have. So, whereas, if there is error in process, it just crashes and dies and you know that computation did not get through.

Whereas, if you do not have it in a file system your data itself is going to be suspect, everything is going to be suspect, because both are important, but usually the users worry about individual stuff. If something happens to the process and has to be killed, you can read that one more time. Whereas, if you lose data there may not be any person agreeing on it, so that is why they has to be it has to be very careful with that.

Also for example, if you are doing some operations the file system, it assumes certain invariance. For example, a file system call might allocate some buffers during the system call and once it is done with operation, it frees it. It is typical of write then there will be write system call you allocate the buffers and then you free it when you write system call and idea is that usually it stays out the buffer stays back. So, that its some kind of caching also goes out, but it is freed at the end, so that it can be reuse by anybody else. Now, the problem could be that you write it, but turns out that the disk driver, there is a problem the disk hardware for example, have a problem when it never responds back saying its finished writing it that means, that the buffer now is now freed.

The serious problem disk drive is file, but I am trying to say is that the file system assumes that invariant, which can be falsified by the hardware malfunction. So, we have to actually to worry about the fact that you might have written your code correctly, but it is not enough because other parties are actually violating what you are assuming. So, we have to check for things which also not usually check for things which may be let us say which could be happening because not because you made mistakes, but because somebody is also doing something for you. So, that is something which has to be worried about.

So, typically for this reason file systems often give you best effort service. You retry in case you can recover from it. You cannot then you try to move the file system to some reasonable consistent state. And usually you do not give any non-starvation guarantees this is very difficult to give non-starvation guarantees. So, these qualitative services will to will talk about later this is not usually given it is very difficult to control all these things.

If you really want to be careful you need to do what is called end to end checksums across all the components, across disk, host bus adapters, network links. And this turns out to be again nontrivial because in addition to these you also have to do it across

memory also, because often times there are a buffer cache involved. So, again you have to worry about how these checksums are across memory and that may not be standardized across operating systems. So, it is not going to be easy unless all the part is across all these devices actually agree on a common format before we can do it.

We will continue in the next class regarding some particular aspects of (Refer Time: 58:24).

Thank you.