

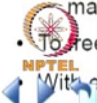
Storage Systems
Dr. K. Gopinath
Department of Computer Science and Engineering
Indian Institute of Science, Bangalore

Storage Filesystem Design
Lecture – 24
Filesystem Design_Part 4: Semantics, ZFS

(Refer Slide Time: 00:39)

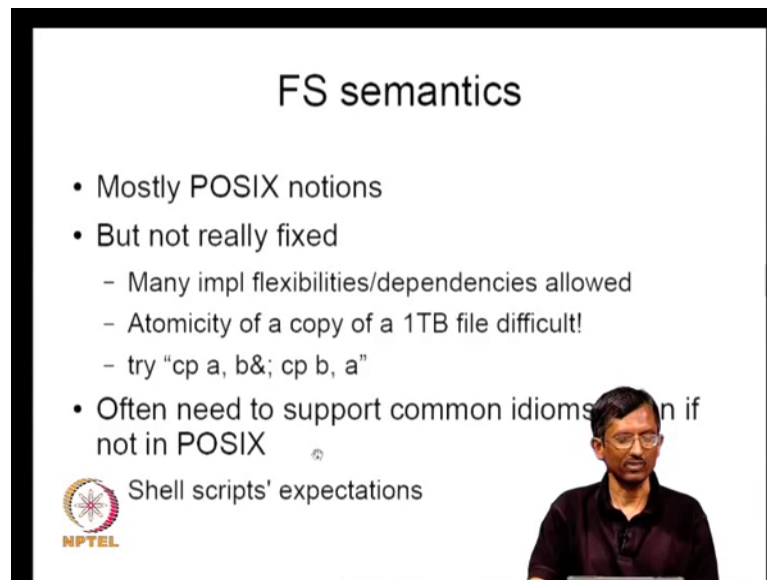
General Issues

- Single threaded kernels: interleaved execution of getblk with interrupt handler or with brelse
 - May need to block interrupts in general
- Multi-threaded kernels: in addition, with other concurrent fs ops
- Avoid deadlocks at all costs
 - use lock ordering where possible!
 - Eg: in rename, have to lock source and target dir entries
 - (eg, zfs) lock dir with smallest object id first, or if it's a tie, the lexically first
 - where not possible to order, drop locks as necessary
- marshal resources and be conservative to avoid deadlocks
- To freeze and thaw fs, need infrastructure
- With errors/triggers, have to store continuation and restore it



Welcome again to the NPTEL course on Storage Systems. So, we have been looking at summations in (Refer Time: 00:29) design. And we briefly looked at some issues general issues of file systems last time.

(Refer Slide Time: 00:44)



The slide is titled "FS semantics" and contains the following text:

- Mostly POSIX notions
- But not really fixed
 - Many impl flexibilities/dependencies allowed
 - Atomicity of a copy of a 1TB file difficult!
 - try "cp a, b&; cp b, a"
- Often need to support common idioms that are not in POSIX

Below the text, there is an NPTEL logo on the left and a small image of a man in a dark shirt on the right. The text "Shell scripts' expectations" is partially visible below the main list.

So, let us get slightly more into slightly tricky aspect of file system that of semantics. I think as you may be aware file systems have grown, but collate organically. What means that there is no specification that is given and then you go and design something for this specification. So, for a period of time various notions came around and then finally, the some kind of standardization they rapid and POSIX, so that is applicable to most UNIXes and because that is a wide variety of UNIXes and there was of need for applications to run on different types of file systems.

So, for example, of a semantics that most of you are familiar with is the deleted, but open file, this is slightly non-standard or peculiar thing about UNIX; other operating systems need not have such semantics. And this kind of understandings have been codified into POSIX, but POSIX is basically not really does not nail down all the details, the design committee of POSIX, they to prevent to tie down too many details which will prevent good implementations right.

They purposefully leave many flexibilities and other dependencies they just leave it, they keep it in that is true for any specifications. The idea of specifications there are some ground rules everybody should follow, but (Refer Time: 02:35) is left to engineering of the implementers to be better than other people. So, there is a competition between various parties who have come together to study standard. So, because of this the semantics is not really completely fixed, and people always creatively try to exploit some

feature and gives them some competitive advantage. So, in terms of or sometimes it become a excepted idiom and finally, somebody else has to comes later might have to observe that idiom that is a standard way. And this is true for not only here, but in almost every part of operating system design.

I will give you one simple example it terms out that some of you might heard about MEM copy and MEM move right, there are two different types of things one of them handles overlapping memory regions, one does not. And one of those MEM copy file example, if you give overlapping regions it says the behaviours undefined. So, it turns out that recently because then a lot of we will say discussion on some kernel many group (Refer Time: 03:56) group that some major application like slash for example has used MEM copy instead of MEM move, because it turns out that it breaks the semantics has broken with respect to in some implemented architecture. For example, (Refer Time: 04:12). And the basic issue is what is to be done about it, do you go and (Refer Time: 04:19), do you tell the application guy to fix this codes etcetera, because there are various possibilities.

Now, thing is that to get somebody to fix a code, it may not be under your control. And basically what happens is that you are learning implementations will not be able to do flash let us say that means, that in the past at least not now 99 percent of all let us say web pages dependent on flash. So, if this particular application does not run, you may be correct you actually may be correct. It may be that flash guy that Adobe person he used the wrong call and it works an almost all platform except (Refer Time: 05:05) let us say. But whatever it is it turns out that that is not a problem on a widely what say used software platform like flash it is on work on a (Refer Time: 05:19) architectures.

Now, what is to be done? Some people will say go and get Adobe will fix it, some people will say we do not have any control over it, you go and change it in (Refer Time: 05:32). You define MEM copy to be MEM move that is what the problem. You take a small hit in terms of performance, but all your flash programs work and called end users are happy. So, sometime this kind of arguments are made because finally, the question is about users. We worry about users or do you want everybody to curse a Linux saying my flash is in work and then they go and start using Windows or MAC OS or whatever. The question is what do you do about discontinuation because you do not have complete control.

So, in terms of the same situation it happens in file system also. If your file system design though correct, and a some moral, if it breaks all control scripts then people run away from the file system and say something is wrong with your file system and go on do something else, go on start looking at something else. So, there is a certain given taking this business, it is a bit unfortunate even if you write you have to change something, so that everybody is happy sometimes it happens. So, these are factor of file.

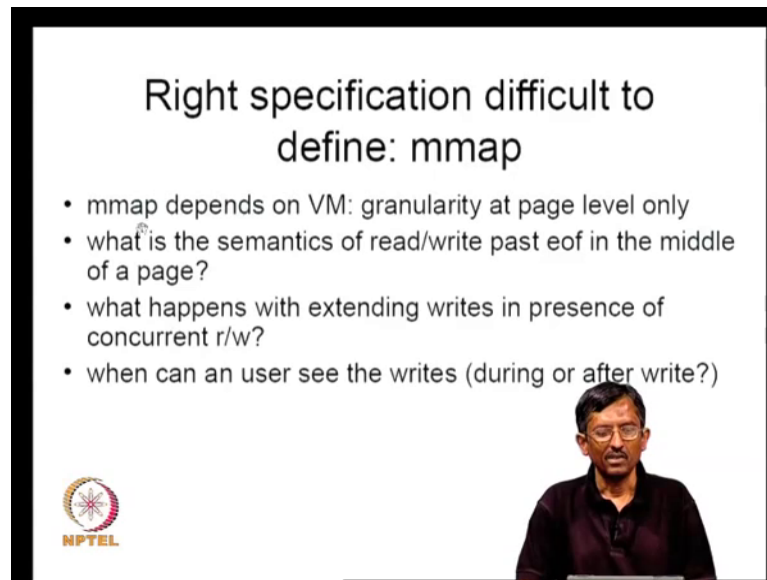
In commercial world, it is often happen the same thing except that in the commercial world you talk to the most important customer or most important few customers take care of the reason, take care of their problems (Refer Time: 06:53) you just drop, so that is from saying that semantics is not really fixed typically. So, it is a bit of a difficult business to be because you are now having to support so many applications which are written by lots of diverse set of people and then a file system can run of so many operating systems also right with this device and kernel dependencies. So, you have to get it working across both this two kinds of platforms over which you do not have any control. So, it becomes slightly difficult.

So, in addition to this that is also this issue that in principle something are inherently difficult to do for example, copying a 1 tera byte file is not that uncommon nowadays people do it, copying 10s of giga bytes is very common, it is nothing unusual. Even one tera byte, but this one unless we have a let us say huge amount of memory, there is no way to give you transaction semantics takes place absolutely no way. Basically, what it means is that either you it can be done, but you have to essentially use locking extensively all over the place, you cannot allow anybody else to come in between while you are copying it. So, all cancel your file system essentially that hold directly for example, will have to be force doing that time. And 1 tera byte copy is going to be it can be multiply days, it can happen, it will be that slow because normally you do not have you think about it you calculate it. So, thing is you cannot really do this it can be atomicity for really stubborn about it, but typically it is hopeless business.

So, for the same reason, we will find that there are various things that you can do which give you very unusual results. One thing I will discuss it soon. We can try some seemingly nonsensical thing like copy a comma b and copy b comma a, this is legitimate as per (Refer Time: 08:58) you can talk about. What you get is completely can surprise you, and different platforms will have different kinds of. So, we will we will look at this

soon. So, as I mentioned earlier we need to support common idioms even if not in POSIX. Basically we have to listen to your customers or your users, if you keep going against them we lose that is it.

(Refer Slide Time: 09:28)



Right specification difficult to define: mmap

- mmap depends on VM: granularity at page level only
- what is the semantics of read/write past eof in the middle of a page?
- what happens with extending writes in presence of concurrent r/w?
- when can an user see the writes (during or after write?)

NPTEL

The slide features a black border and a small inset image of a man with glasses in the bottom right corner. The NPTEL logo is in the bottom left corner.

So, let us try to see get an example good example of this. Mmap is one of the most intercate things that one of the most difficult things to for file system to support. Basically because once you map it you lose complete control over that map result until unless you unmap it, because you get any control over how that map part of the file is used. Only way you can get any control is if it falls in it, then it has to come back to the file system that is only way does it, but one thing what mmap is that the protections or etcetera all those things granularity at the page level only because mmap depends on the VM sub system.

The VM sub system as notion of pages and pages for example, the protections or there is other things are all connected with 4 k or the page size only. So, similarly for map file and let say it is 10 k, so you have pages of 4 k and there will be 3 pages involved now, 4 k, 4 k, 4 k and after 10 k that will be something which is not specified what is going to be there. Typically it zeroed, but in case you write there, the VM cannot help you to catch the page, it cannot really fault a dead point because let say validity of the page is or 4 k granularity. Therefore, even if you write a 10 k plus 1 nobody is going to notice.

So, now it turns out that for this reason, you can actually do read write passing the file in the middle of webpage. Suppose, we have end a file and I can go and write half end of file and mmap is supposed to be let say cannot extend it, but only the read write interface suppose to go extend it. So, even if I write it, it can give you depending on your operating system which might give you different undefined behaviour it cannot really tell you what is going to happen.


Similarly, you can also have after the 10 k, somebody can extend it by read write interface one is a m map interface somebody is coming through read write interface we will extend it the question is what is visible to users is not clear there is also a the last byte specified. So, again the issue is basically about when you writing or a when you are doing a write, do you expose a whole thing that is being written after then it is over or can you do you want to give you it intermediate access. Now, that easy and sensible thing to do of and use to say I will do only after I finish the write, but if it is very long write to push for that particular we are looking at it might be difficult for some users. Like for example, I talk to about 1 tera byte write right. If it insist the write do it after everything is shown after everything is done its, so let say it takes multiple days right.

It may be that I need to see something in between its one can always manufactures scenarios all that can so that is why various file system do different things, there is no commonality in this. And generally the people who worked with application developers, they try to figure out what their main concerns are, and they try to make sure that their concerns are handled decently where that is usually what happens. And because of this things you will find very peculiar problems also, we can just highlighting one problems I I came across.

(Refer Slide Time: 13:12)

File Corruption Prob

- Files rapidly accessed on a WinNT file server,
® intermittent data corruption! Sep96
 esp news server data files & on MP systems
- appln performing a write-extend of a file
- cache manager read ahead thread on current last page (part of larger read); write blocked
- mem manager wakes up write & zeroes last page beyond curr file size & writes new data into page; read also zeroes later!



On a Windows NT file server some 10 years, 5 years back, but this kind of problems are there in other file systems also not necessary windows, it could be then any particular file system. And we see the kind of this all that I we can see from the bug report. We do not have a source called Windows NT, therefore, we cannot see exactly what is a problem, but this is the description given on the Microsoft page. What is the thing it says files rapidly accessed on Windows NT file server intermittent data corruption, but is if you access it too many people access in too fast too many request are coming in then in some corruption. And especially new server data files on mp systems. Now, we can see why this should be the case new server data files means basically the somebody is extending the message. As I mentioned earlier in many mailing systems, all the mail is put together in single file, where a new mail comes in you extend it.

So, I cannot understand it completely what this is saying because I cannot understand the bug just by looking at the description given in that bug report. Now, one can guess that some lock was not taken. So, what exactly is happening application performing a write extend of a file that means, a new mail messages come in, and somebody is putting the next message into that huge mail file, single mail file, somebody is doing a write extend. Cache manager read ahead thread on current last page part of larger read write blocked, basically there is a cache manager read; that means, somebody is caching it in memory.


On current last page; that means, that you could have been at 10 k in some extend instead of taking of course, a much bigger file, but for time being let us called 10 k. You are at 8 k and somebody is copying doing a read ahead of the next page. So, the only two pages that are the only stuff that is valid is 2 k, but it is a page. So, when you are reading it, you have to have some notion of the current length of the file and somebody is extending it also. So, as the person is reading it should have one notion of a file size and at that time you should have a different size of notion of a file size, and they should keep it consistent.

Now, what is being mentioned here is that while the cache manager reading it, the write gets blocked, because this somebody is doing write extend. And of course, this is slightly mystery, I cannot understand it completely, but basically it says memory manager wakes up write. So, the read ahead is going on the write is blocked, now the write is now unblocked at zeroes last page beyond current file size, and writes new data into page; for some reason read also zeroes at later.

So, there is some locking problem here, I am not very clear what it is. One can guess that the let say the reader should have got the more current version of the size of the file from the writer and that done some something of the somehow it is not getting done. And there is some locking issue I think that what get the problem. But this kind of problem of quite common basically there is no semantics also here exactly what is a meaning that we have transcribe to when somebody is extending it while somebody is reading it.

(Refer Slide Time: 16:53)

Record and file locking: cp a, b&; cp b, a

- one rwlock: deadlock! but with rwlock + glock: OK
 - cp a, b: maps a & then write into b from mapping of a
 - mmap not atomic but r/w "atomic": need rwlock (shared/excl)
 - mmap'ed pages may fault: need to call VOP_GETPAGE
 - Holes in file: allocates atomic; need a lock (glock)
 - Interlocking betw truncate ops and getpage ops: truncate has to prevent getpage from bringing in pages; use glock
 - mmaping: rwlock a; map; rwunlock a;
 - reading: rwlock b; uiomove (getpage a); ...
 - cp b, a:
 - rwlock b; map; rwunlock b;
 - rwlock a; uiomove (getpage b); ...
-  If both glock and rwlock the same lock? deadlock!

So, let us look at a specific example, which b have some control over, because we can. Suppose, I do this thing as I mention I am want to look at this slightly crazy example copy a comma b then copy b comma a both are (Refer Time: 17:03) at the same time. Now, we can see that if I that is only a single lock for a particular file, as I mentioned to you because of what is called file control locking ffc and tl locking, you need to have lock on ranges. A database for example, can lock 1-gigabyte part of a file or a in applications also can do similar things.

So, we just have a single lock then what will happen, you will get into deadlock because there is a lock for a, this lock for b, let say a gets a lock and b gets a lock. Then you try to get a lock for b or a then it will stuck, usual cross in terms of locking let us say orders. So, of course, one can say the following we can say that this an application problem. You can say whoever is a fool has tried to do this thing, it is also it that is one way to think about it.

So, any ways what you are saying is do not do bad stuff then I will do it correctly, but this could be there are some situations where this attitude is probably not a good one. What happens if a or b is an important file which essentially is required for the health file system there are situations where this kind of things can be important. And it turns out that any way you cannot implement file systems of the single lock. In addition to the

application visible lock that (Refer Time: 18:56) lock, you need to also have other locks in the kernel for managing the write.

So, it turns out that one way to do avoid this is by using other locks possible inside the kernel and then trying to do, and see if can do something interesting with it. So, it turns out that there is a lock which is called a getpage lock in many UNIX based file systems which also can be used as a way to avoid the deadlock. So, let see you how it happens. Now, when you do a copy a comma b, what normally happens in many recent UNIX based systems is that a is mapped to kernel memory, and then when you do the copy, you copy from kernel memory to user memory, and that is how you that stuff gets into b that is how it is done.

So, let us look at how it is done. So, the first step is map a into some reasonable place and then you write into b from mapping of a that is taken one important thing is mmap is not atomic, but read write is atomic. This is another feature of some other semantics that people have fixed now that means that if why is mmap is not atomic it basically map it if there are concurrent people you have access to the same file they have the permissions anybody can writing into that map region without any coordination. That means, that there is no reason anything should be atomic that is why could be one guy could be writing some portion of the file partly here and partly there, somebody else can be some other party can also do partly here and partly there. Therefore, it may not be giving that either this or that, you can these kind of things may not be guaranteed.

So, what you do is you basically take lock on a first, map it, and then you unlock it like basically the way this is done. You first take a lock on a, so that no other concurrent maps take place at the same time. This purely is to make sure where you are the only person mapping it. Because if somebody else concurrently mapping it then the system will force it to wait and then they will look at the previously mapped things and then provide that information to the new guy who is trying to concurrently map it. So, you take the lock to ensure that you are the only party trying to map it, then you map it and then you unlock it. So that it is now the mapping has gone through concurrently sorry in a the mapping has gone through in a mutually in a synchronous manner, but the file a for example can be accessed by anybody who has appropriate permissions.

So, now what do you try to read from a and then copy it to b now for b we do here rwlock on b etcetera this is the lock that is visible to this application this not a kernel only visible lock. So, rwlock b is you take the lock on b ,and then now as it is copying it what will happen is that since it has been mapped to memory either it is already cache in memory or it will not at cache in memory, so it will suffer a page fault. If it suffers a page fault then you have to do what is called a getpage correspond to that, and then you do what is called uiomove, uiomove is what it form you basically you copy it into from kernel to user area you are moving it that is what happens here.

So, basically now it getpage is what is finally, again entry into the file system this is what is again entry into the file system. So, copy a comma b is basically rwlock a map rwunlock a and then for reading b a to b rwlock b uiomove getpage a etcetera, and then it will continue till the copy is done then you actually rwunlock b at the end of it. Copy b comma a is also is something similar to like this right. So, we will see now that depending on how these things in interlink, and how this things in implement by different file systems you get different behaviours.

For example, right now we will say copy a comma b, since you have done a lock on a only for mapping purposes, after the map is done you do not have a lock; that means, somebody else can actually modify in principle. So, those things are essentially followed from the way it has been implemented. And the reason why people do this mapping is because they find that if you do not do this mapping there can be multiple copies of the same data both in buffer cache as well as in virtual memory it will lose extra overhead and it also introduces not more memory pressure and also can introduce consistent issues. As I think as I recollect in the previous class I mentioned that in Linux for example, up to Linux 2.0 and two point (Refer Time: 24:55), it turned out that write we have to do write for two places.


First have to go and check in the buffer, and then also check if that same thing also is somehow in the virtual memory somewhere, and you have to write to that also that is every write actually is now can be two writes, one to buffer cache and one to virtual memory. So, to avoid those kind of overheads, you have try to combine them together this is what is called integrated design and. So, the mapping is what is preferred to avoid those kind of issues, but once you have this mapping you have variety of possibilities in terms of what can be happened during the copy.

So, I just want to point out that this copy a comma b become is basically it is actually composed of multiple components, and how this interacts decides the output what you get. So, again this get page for example, you get a page every 4 kilo byte blocks typically page size. So, the copying happens at atomicity of 4 kilo bytes. So, while you are getting the page from a suppose you do not do read ahead, then you will get some 4 kilo byte, copy it, and then you will again get stuck the next time. And then while your process sleeping somebody else can come in and look at the other things all those things can also happen. So, that how that gives you some idea that the semantics of some of this file systems are slightly nebulous, it is not really that clear. So, people really want to get their semantics clear right they need to take extra care, they have to be very careful what they have gone to do. It is not going to be come just like that.

(Refer Slide Time: 27:09)

Locking issues

- a thread locks a resource and calls a lower-level routine on that resource; lower-level routine may also be called by others without locking resource
- deadlock possible if lower-level routine does not know if resource locked
 - unless params passed informing low-level routine about locked resources *but* this is non-modular!
- recursive locks can avoid deadlocks: need owner ID
 - functions deal with their own locking reqs: clean, modular i/f; do not need to worry about what locks callers own

 Ex: ufs_write handles both writes to files/dirs

- files: unlocked vnode passed from file tbl entry to ufs_write
- dirs: vnode passed locked f DNLC/pathname traversal func

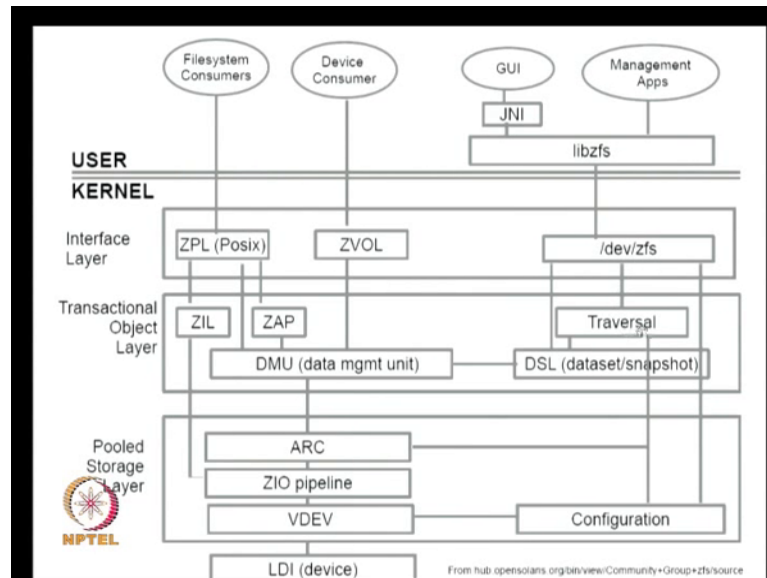
And so similarly there are other issues in when we design a file systems, for example, you have many multiple execution threads. So, for example, some there could be some kernel threads, there could be an applications which has comes into kernel, and does the system call various possibilities are there. So, it turns out that often times there are multiple entries through different paths to the same thing. For example, you can lock a resource, and you call a lower-level routine on that resource; and this lower-level routine may also be called by others without locking the resource.

So, the deadlock is possible if lower-level routine does not know if resource locked. So, the only way you can avoid this problems is some parameters have passed in forming the lower-level routine about lock resources, but this is from soft engineering point of it, it is a problems. Because that means that normally the lower-level routine should not do anything about upper level routine, they are sort of written in by different parties or different they want to keep it a modular right.

If you the lower-level routine has know everything about all the parameters has to be known from what are the top level guys doing, it becomes a problems it become non modular. So, that is why people have sometimes or provide called recursive locks that means, that somehow you keep information about who has locked it, because as I said multiple entries are possible, and we keep track of it and therefore we can probably disambiguate the situation. So, but it turns out that most kernels do not provide the recursive locks. So, these are the kind of problems you have to handle in large scale system.

The thing is that you can say why do not people provide a recursive locks? The standard answer could be that if you provide recursive locks, then you are adding extra code in a critical part of the locking things, and you are making everybody slow. So, this may not be available. And you often see the same kernel going through different designs where somebody in the past has provided kernel locks in the recursive locks, the next iteration of the thing somebody has decided that it is too expensive and taken it out, it has actually happened and some operating systems I know. So, a file system also has this issue.

(Refer Slide Time: 29:47)




And you can see this is quite not uncommon, because this is an example of a reasonable modern file system the ZFS. You will see that there are multiple ways of doing the same thing. For example, I can come from like this right, I can also come like this right. So, I do not need to come through some of these things, you can directly come like this. So, sometimes, I can some of these things for example, I can come like this or I can come like this. So, multiple paths by which I can get one of these guys. So, given the multiplicity of paths, sometimes the design becomes that much harder because you do not know exactly how it is coming in, you need to keep some infrastructure to know which path you came through and then only you can sort of make sense of what is going on.

(Refer Slide Time: 30:45)

ZFS

- Integrated file system + logical volume manager
 - zfs and zpool: ZPL (ZFS POSIX Layer), DMU (Data Management Unit), and SPA (Storage Pool Allocator)
 - design for fast, reliable storage using cheap, commodity disks
 - copy-on-write transactional object model
 - blocks containing active data never overwritten in place
 - instead, a new block allocated + written, then any metadata blocks referencing it are similarly read, reallocated, and written.
 - To reduce overhead, multiple updates grouped into transaction groups, and an intent log used when synch write semantics required



So, let me take a look at let us start taking a look at some reasonably new file system. This is came about let us say started we used about 5, 6, 7 years back and design by Sun, now part of Oracle. But previously I looked at what I mentioned previously was older 1980s kind of model file system, this one is a more recent one. It is an integrated file system plus logical volume manager. So, it is a regular file system plus logical volume manager.

What does it mean, what is a volume manager, it basically helps you to think of a pool of storage on different disks etcetera in an integrated manner, so that you can add up particular disk or remove a particular disk. And then somehow as long as its making sense is still can the files the ZFS systems can give you a coherent view of the file system on whatever is available disks, whether it is something is added to something is taken out is still have a reasonable view without too much work on the part of the user. Whereas, if you had a non integrated system then you need to have two interfaces one which you go on talk to the volume another first, and we tell that volume manager, I am adding a particular disk. Then you will have to go and tell the file system that I says that something has changed now please change something because I added something. And this some of things are slight tricky because if you want to add a volume right, and then some activity is going on the file system, we have to now to be able to tell the file system, please be able to use this new disk that are added.

We have to essentially get the concurrent activities in the file system to either stop or to get this particular state, where they are all in some reasonable consistent state, then I can finally make the new disk that has been added available to the file system. File system has to now recognize it. Once it recognizes it then it can again continue from there. So, you need some notions of by which you can what I discuss before we can freeze a file system, add the thing, then we can (Refer Time: 33:22) the file system.

So, as I mentioned something like surgery right, you basically anathematize the file system, add some extra limber whatever it is and then we say now you go ahead you right something of that kind has to be done. So, in a integrated file system logical volume manager, this is done slightly more simpler way, because the file system knows about the volume manager. There are two distinct interfaces you have to actually use a single unmanufactured of commands two manage the file system.

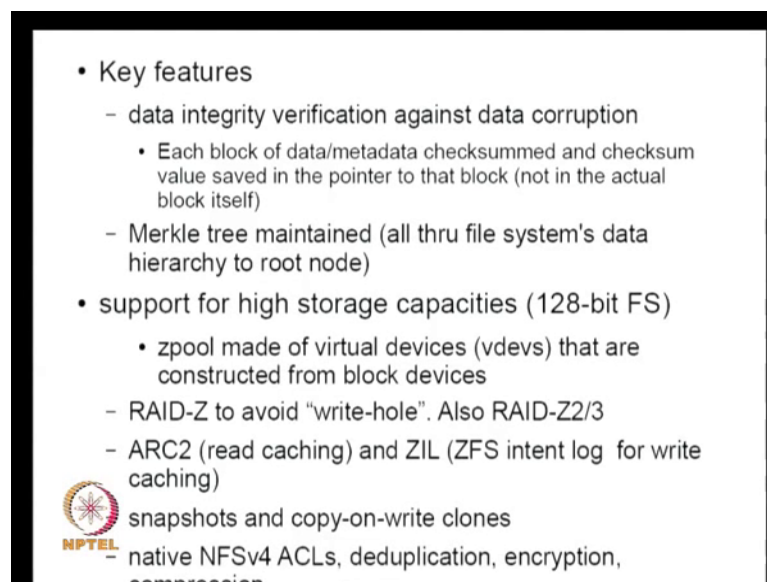
So, basically the file system is the part ZFS, the volume as typically part zpool this two together are constitute the file system. There is a POSIX layer that is a ZFS posix layer ZPL. There is data management unit, which is basically the part which gives you the transaction of semantics to the whole system. And there is a storage pool allocator this is based which actually deals with a the block device unit cell. So, and it actually puts them together, you can request resources from it, all those things are done in this part.

The ZFS the idea has been to see if you can design fast on reliable storage using cheap commodity disks that is the a goal of this system. And basically they use a copy and write transactional object model extensively. Again they similar to some other models for example, I mentioned before Netapp has something called write anywhere file system they also have a similar copy and write transaction model somewhere similar. And because similarity here there has been some legal cases between Son and Netapp also for the same reason.

What is the interesting about this is that blocks containing active data never overwritten in place, they are always using you are going to do copy and write. Instead, a new block is allocated and written and then any meta blocks metadata blocks referencing it, because some data has changed, the metadata block also have to get changed. They get the original stock is picked up and then reallocated and written with a new information because now it is pointing to the new block that has been allocated for the copy like that.

So, because you want to have a transactional model, you also want to reduce overhead, all this multiple updates are grouped into transaction groups, we have an intent log and then which is if you want to you basically make the changes to intent log. And then which is if you want some synchronous write semantics, then you will flush it to disk immediately. Whereas, if you are do not mind some synchronous models, then the updated data structures in memory they will start being written to their home locations for a period of time. And then once all the home locations for the corresponding transaction is completed then you release space on the lock was this is basically what happens.

(Refer Slide Time: 36:55)



So, one of the key features of this particular system is that data integrity verification against data corruption. So, at every stage when they are doing writes or when they have to read especially when they have to read, they check some check they have some checksum and they actually check it whether it is corresponding. Each data block of data metadata checksummed and checksum values saved in the pointer in the block not in the actual block itself. Again the key checksum separate from the block, it is not nearby because if its nearby if there is any disk error it might affect both of them. So, the way they have done it is that the checksum is always slightly some distance away from the block itself.

And the keyboard is called as a Merkle tree basically what it means is that you have a tree of checksum of checksum of checksum. So, it depending on the data structures for example, the file has got multiple blocks you do a checksum on each block and then all the checksum of each of the block are again checksummed and then that becomes the checksum of the file. Now, we have multiple files and directory we have a checksum of all the files, and then any other data structures that are directly skipping for its own management, all those data structures are checksum and final that becomes checksum of the directory. Again the directory is also part of other this directory is part of some other higher level directories again you do checksum of all this. So, you have basically what is a tree of checksum this is what is called a Merkle tree, and all they have obtain the root node.

So, if it terms out that there is the checksum do not match then we have to figure out where in the we have to work down the tree figure out which part of it actually else got a problem. So, because here this Merkle tree can easily figure it out, because if there is a problem one side of the tree other side is then that is how the tree the Merkle tree of the top will what has been read and the corresponding will be same whereas in the other situation it would not be the case. So, again we have to go down one by one.

So, in this particular systems, there is it was design for here 128 bit file system. So, it has very high storage capacities. And the pool of storage in zpool is made of virtual devices and that are constructed from block devices because this means what it means is that I can have different types of technologies available at the same time. For example, I can have some of it on a flash kind of device or I can have it on a disk kind of device, they are all virtual devices and then I can combine together the way I want it.

Also use because I mentioned to you earlier this is a integrated file system and volume manager. So, this they have specific type of volume management they do what is called RAID-Z, and I explain what this is they try to avoid certain problems inherent in RAID 5 and I will discuss this soon. They also in addition to RAID-Z, they also have RAID Z 2 and RAID 3. Here it is tolerant to one disk failure Z 2 is tolerant to two disk failures and Z 3 is for three disk failures some failures.

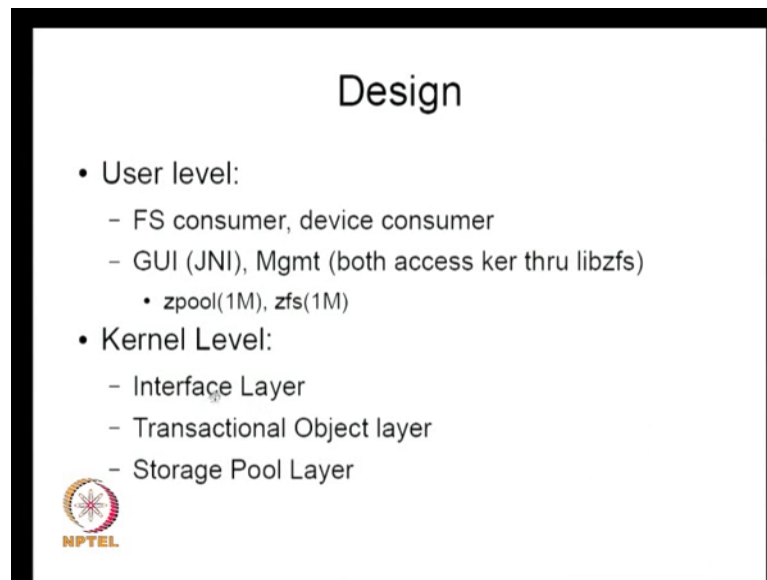
There is some extensive caching going on in something called adaptive read cache basically often times you will find that file system load has certain locality. So, (Refer

Time: 40:36) have to sort of works out very well, but there are some other situation where locality is not good, there are some streaming reads or writes going on. So, the locality issue is not good. So, ARC2 is basically tries to figure out how much to how much of the caching should happen with respect to things which have locality and how much it for streaming.

So, for example, at any particular time, there is lot of streaming going on. This ARC2 cache will discover this fact and start giving more space for streaming workload if that is what is dominant. Whereas if the LRU kind of node is predominant, it will be more space for LRU; so it some kind of adaptive system that they have. And as I mentioned they have a intent log and they have some other features this are all quite common in advance file systems. So, ZFS is no different it provides snapshots like this you can take a particular view of file system at any point time of. And then you can have multiple snapshots, so you can also what is called copy on write clones that the basically you update something you will get a new copy. And then you can actually let say you can even have what is called writeable clone that is you can it can essentially keep one complete fork of a particular system that is been modified.

You have the original file system, then you have a slightly modified file system and then you can start writing on that thing and have a completely let say a branch of the original file system those kinds of things also can be done. And it provides because NFSv4 before provides access control leads which is not there in regular UNIX, they also provide this. And the encryption, decryption also are provided as part of the file system itself. And a new thing that is there is deduplication that is if it turns out that multiple files share some common data instead of copying in so many times keeping so many copies of written disk you keep a pointer to one exact one copy. So, those things also can be done.

(Refer Slide Time: 42:50)



So, let just look at the design. So, it has got a user level and kernel level. There are multiple types of consumers of user level, and then there is some structure to the kernel level. For example, a standard the most important user of course, is the file system consumer that is the standard its interfaces like you except files systems are provide for example, the read write interfaces or m map interfaces all those things. There is also a device consumer that is many of the structures in that system you want to have it exported as a device, so that you can play around with it. For example, if there are some memory structures for example or you may want to manipulate it through a slash type interfaces. I think some way similar to kernel memory for example, if somebody wants to manipulate the kernel memory, for example, for doing debugging. If a operating system provides you slash (Refer Time: 43:57), then throughout trusted debugging routine you can go and add break points and remove break points. And that you get by looking at kernel memory as a device, and then opening the device and then seeking to the place where you want to add the break point for example, and you update at that point.

So, while the kernel is going on you can actually go and manipulate the some specific area of the kernel if possible. So, similarly device consumer you export different parts of the file systems space, file system devices or structures for example, you can export it as a device. They also have a GUI based model I think I will skip this part. And there is also a management interface. One important thing of about any complex system is that we

have to provide some management applications. For example, in a standard UNIX kind of system, one management application FSCK right in something bad happens we need to provide a way of recovering from it. Another management interfaces is make FS, we want to be able to make a file system. Also mount a file system unmount all these things are management to do this.

So, you have a management interfaces and it turns out that the two main ones in ZFS are zpool and ZFS. And zpool basically handles all the things relating to creation of pools of disk, we have a lot of block devices, and you want to create virtual devices and then you add them and remove them though, zpool provides you the capability for managing those things. And ZFS is for creating file systems and let say mounting file systems all this kind of things. So, that is what ZFS is. At a kernel level, there are multiple angles one is the interfaces layer example POSIX, we had to provide some we have to translate let say the capabilities, so that the POSIX interfaces is available at the user level that. So, that is an example interfaces layer.

Transactional object layer basically want ensure that if you have to update multiple sub objects, how do you ensure that higher all of it happens or only some other happens, you want to give that kind of semantics that is done by this. And storage pool layer is basically the one which varies about how to get these virtual devices etcetera, how to put them together, how to if interested in reliability, can you say that there as to be three copies of this particular let say virtual device compared to two copies of some other things. So, all that reliability and stitching multiple types of devices into one single common thing look all those things are done.

Again let just briefly look at all the things that happen here. So, this is user level; this is a kernel level. So, all the regular applications which use the file system interface they come from here. As I mentioned the device consumer is a party who wants any of these structures whatever is coming from here, whatever that this path can be read. We want to see it has a device then the ZVOL basically gives you the device interfaces. So, for example, it may be that you need to write a specific application which wants to look at the log itself we want to recover from the log do something we want to do the application level.

So, this ZVOL will actually manufacture for you a device by which you can actually write a program here which actually can look at the lock structures on the disk and then do something with it. As I mentioned management applications there is a libzfs and basically it in turn uses slash dev ZFS as a way to talk about all the let us say based on manipulating some other data structures that are provided by the file system. So, let us say what is libzfs, it could be things like suppose I want to manipulate certain pools for example, there are multiple pools of storage and then there libzfs gives you the capability for talking of dev ZFS, how to create a pool, how to delete a pool. If there is a pool that has got some error for example, how to remove for example, the part which are bag basically because it can be composed of multiple parts. So, have a virtual device it composed of multiple block devices and one part of the block device is gone back. So, I can actually take this out all this kind of things.

And similarly if it is talking about file systems then it has to basically all the mounting, unmounting all those kind of things happens from here. Now, better the user and kernel they all have to understand the data structures. For example if I am writing the again in the traditional file systems if I am doing the fsck file transfer, if I am doing a fsck there is an on-disk data structure. And on-disk data structure it has to be used for your fsck application to be able to pass it that means, that there is some there is some set of data structures that are common between the user and kernel.

So, these things have to be included in both the kernel code as well as in the user application code that of the identical. So, some of those things are exported through like there included in many other programs that are doing this for libzfs. Now, there is also something called I told you this is the POSIX interfaces and that is where all the linear operations get done. Here as I mentioned to you earlier there is typical UNIX file systems have a file system independent layer and they manipulate something called vnodes virtual nodes. Whereas the physically realised nodes are called I nodes in ZFS they called z nodes. And basically what happens is that by the time you come here you already gone through the VFS layer and it is finally come to this point. And to do some of this file system operations, it has to use the intent log so for the logging etcetera.

In addition this is for incrementing directories, which is something called ZPA, basically this is a ZFS attribute process basically it is a metadata how to anything to do metadata for example, for directories is being handled here. And now there is also this I mentioned

data management unit, this is the one which actually is involved in the transactional semantics providing the transactional semantics.

And because you can provide snapshots some clones, again all this come from this side the management application because you can because this system administrator sits the terminal basically types and saying create a clone for this that comes the management libzfs. So, all this clones all this things are manage through this part. And this part requires the transactional layer transactional part of the system, so that you can keep different as I mentioned this is for copy and write transactional model. So, anytime you do as a snapshot etcetera, you need to have copy and write and so that actually interacts like this transactional kind of system.

You can also have traversal what is this traversal basically it is a way in which you can look at different types of clones and snapshots. There is a hierarchy wants all this thing, because I start with the file system, I have a what do copy and write I get one version of file system a snapshot even that finally, also can snapshot it. So, there is some kind of a tree of snapshot etcetera, and there is lot of sharing going on between various versions of it. So, you have to traverse these particular tree typically out where the block actually is. It may be the block what you looking for could be at the top most level then because you recently updated that one or it could be that it is there in the lower-level snapshot layer or it may actually the present in the original file system itself. So, you have to do that traversal on the files. So, I can look and start seeing why what this is basically slash dev ZFS gives you the ability to talk about all this different types of snapshots and clones and other things because you need to have a name to each of these things. So, you provide that naming etcetera through slash dev ZFS.

This is a cache again this is the part which is concerning closer to the in the older systems we talk of buffer cache this is closer to the buffer cache part of it that that is why the caches out here. And this is the part which actually splits a large IO into smaller parts. And then since it is basically it is smaller parts if it actually tries to see all of the parts are complete those cache etcetera actually waits for sub task to finish or else can nothing happens here.

So, VDEV is basically a virtual device. So, it is basically what you need to call as solar device because it can be using RAID etcetera right, you can construct a logical device or

a virtual device composed of physical device right. And then this is what they called layer device interfaces this is a final thing that finally talk to actual device. So, this is a some are the kernel infrastructure the device driver infrastructure which is used a way to talk to the file systems. We will also notice as a configuration here which goes through all these things, this configuration goes to VDEV, goes to ARC, goes traversal etcetera.

So, you can see the failure complex infrastructure works out here. So, it will it is nontrivial to get most of these things work correctly given all kind of concurrency given the demands of user with respect to interface demand which are changing. Work period of time, device is the changing where work time all kinds of computations are there that is why designing a file system is a reasonable complicated business. It is not you do not see new file systems even other any other day.

For example, when we talk about architectures right we will see that people can come up with new architects quite fast, but somehow the same level at which the speed at which architectures come up, you do not find it here, because in the case of architectures, even though we have the similar problems, but the instructions sets sort of isolates the complexities. Of course, there also there is a small (Refer Time: 56:51) we talk about things like memory models or etcetera, there also there are issues there also, but this one is bit more tricky because all the applications are out here.

We have to satisfy all the applications and the application is written by large numbers of people. Whereas hardly any guys usually directly manipulate instruction sets. So, in a sense the disk device is more complex is because the number of applications that you have to satisfy that means, it is quite difficulty. I will continue a bit on this next class.