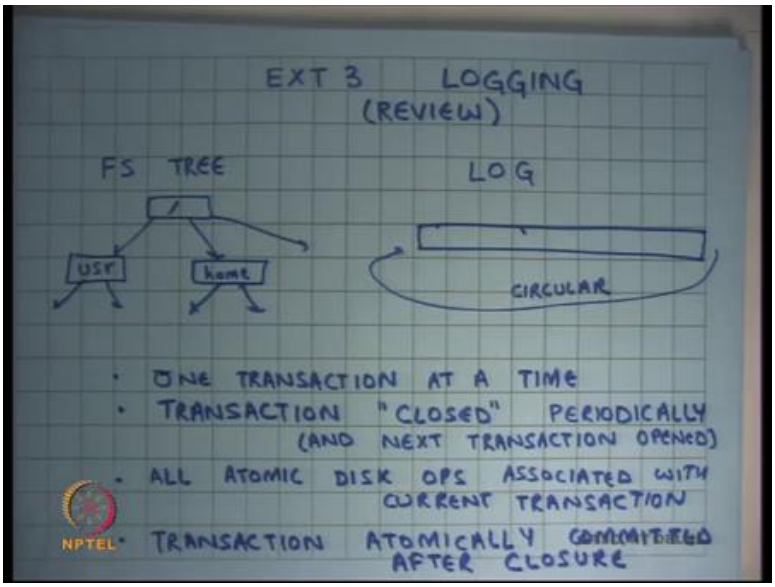
Operating Systems
Prof. Sorav Bansal
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

Lecture – 36
Protection and Security

Welcome to Operating Systems, lecture 36, right. So, so far we had been looking at logging as a way of maintaining, ensuring that the system can recover after crash, in other words ensuring that the disk does not get so corrupted that you cannot even recover your data out of it, right.

So, we have the first thing we had looked at was ordering where we said that we are going to order the disk operations, as they go to disk in a certain way, so that we do not lose data. We may have space leaks, temporary space leaks, but they can be fixed whenever we. But there were problems with that the problem was that recovery was really slow and the other problem was inconsistency was still possible.

(Refer Slide Time: 01:01)



So, logging was other way where basically we the file system consisted of a tree which is your regular file system tree and a circular log. You will have one transaction at a time. So, I am talking about the ext3 file system. You will have one transaction open at a time which you will close periodically. So, let us say you close the transaction every 5 seconds for example. And all the atomic disk operations that are started at any time will

be associated; will be tagged with the current transaction with the current open transactions.

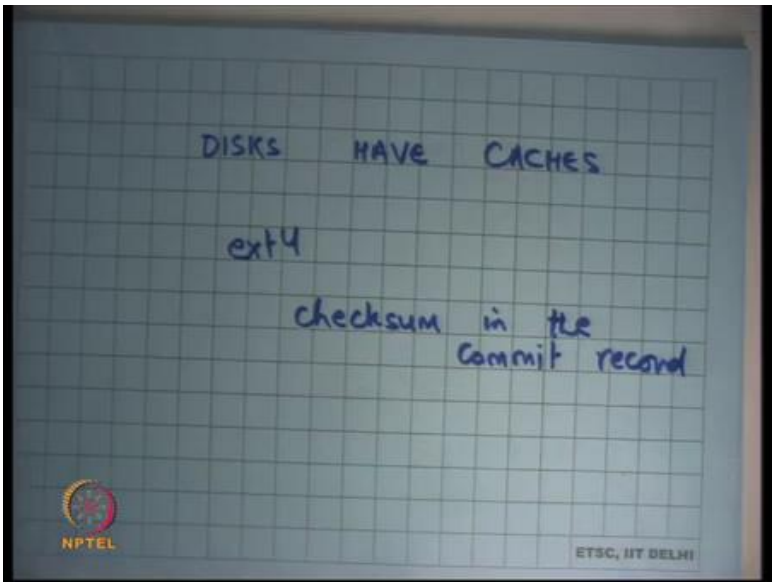
The atomic operation will either belong completely to one transaction or not belong to the transaction at all. It is not possible for an atomic operation to be half in one transaction and half in another transaction. Once you close the transaction you also open the next transaction. So, all atomic operations that start after you close the transaction will belong to the next transaction. All atomic operations that started before you close the transaction irrespective of whether they have completed or not will belong to the previous transaction. If they have not completed the transaction will wait for those operations to finish before it starts committing them to disk; alright.

So, all atomic disk operations associated with the current transaction and transaction is atomically committed after closure. And as we saw last time commit, committing the transaction, before you start the commit of a transaction you have to wait for all the atomic operations to finish number 1. Number 2, you also want to make sure that no atomic operations of the second transaction or the next transaction has started before all the operations of the previous transactions are finished, otherwise inconsistencies can result and we had seen some inconsistency last time that can happen, ok.

And we also said you know even though, so in this case we log old data blocks atomic commit of the transaction is implemented by the right of a special commit record at the end of the log and assuming that this right of the commit record is atomic you are basically sure that either that transaction will be atomic atomically committed or it will not be committed at all; in which if it is not committed means nothing happened if it committed it means everything happened. So, there is no intermediate state. So, there cannot be any inconsistency assuming that commit is atomic.

So, we had been so far assuming that the commit has atomic; we have been so far assuming that the disk behaves exactly like the way we expected to be here. So, for example, we assume that if we tell the disk to write to a particular sector it has written to it before it actually comes back to us. But the truth in fact, is that the disks have caches, right.

(Refer Slide Time: 03:43)



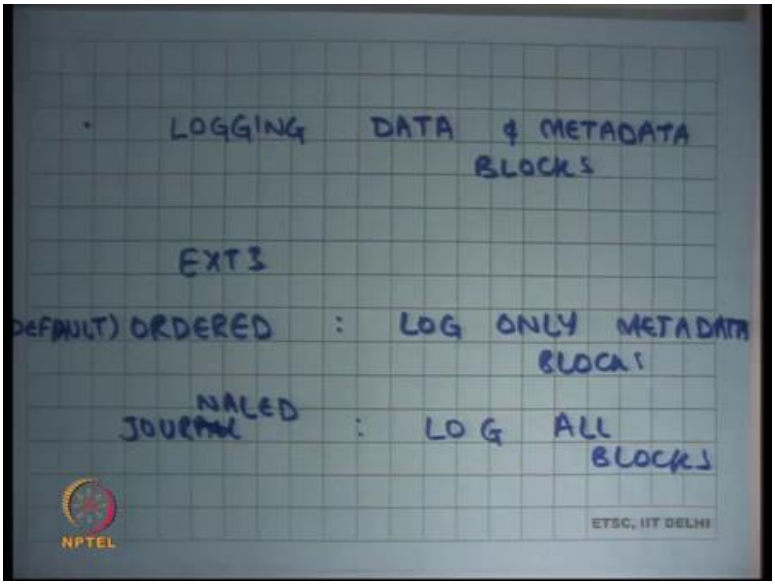
So, in the real-world disks are not really truthful, right not necessarily truthful. You know you can imagine that this manufactured by different devices different manufacturers and for better performance these manufacturers are putting special as you know volatile caches which means you know they are either using technologies, the same technology that we are using for DRAM for your main memory into your disk. And so, even though the disk has said that it has finished it may be doing some optimization at the back end where it may not have written to the magnetic platter, right to the actual magnetic platter.

So, if something like that happens then you know all your reasoning about correctness actually goes for a toss, right. Because if you know if you ask the disk write the log and wait for the disk to respond and then you ask the disk to write the commit record. And, the disk instead internally has cached the first write has not actually committed to the magnetic platter, and it has it commits the commit record before the actual contents of the log then when you recover you will see that there is a committed transaction, but the transactions contents may actually be junk, alright.

So, these kind of things are possible and so, ext3 does not deal with these kind of problems, but the next file system ext4 introduces checksums in the commit record, right. So, the idea is that the record, the commit record will have a checksum or some kind of a one-way hash of the contents of that transaction, right. So, when you are

recovering you can check that the checksum actually matches the contents of the transaction and if not then you can say that you know something fishy has happened and you should not be actually doing that transaction, ok. So, that is one way of dealing with these kind of issues, alright, ok. So, as we have discussed logging, we were logging whole data blocks both data and metadata blocks, right.

(Refer Slide Time: 06:05)



So, if I were to look at the file system and I was to look at all the blocks in the disk, I could divide that blocks into either their data blocks which means they are blocks that hold the contents of the files that the user is actually writing to. And there are other blocks which are which I can call the metadata blocks.

Metadata blocks are blocks like the super block, the inode blocks, the free bitmap blocks, the directory the blocks that contain the contents of the directory even they are metadata blocks, right. So, all the blocks that are associated with the semantics of the file system are metadata blocks. All the blocks that are not associated with the semantics of the file system are data blocks, right.

So, for example, the contents that blocks that are stored in the contents of a file have nothing to do with the semantics of the file system, right. So, they are metadata blocks and everything else will be metadata blocks. And so far, as we have discussed the logging file system, we are basically saying that we log everything, we logged both data

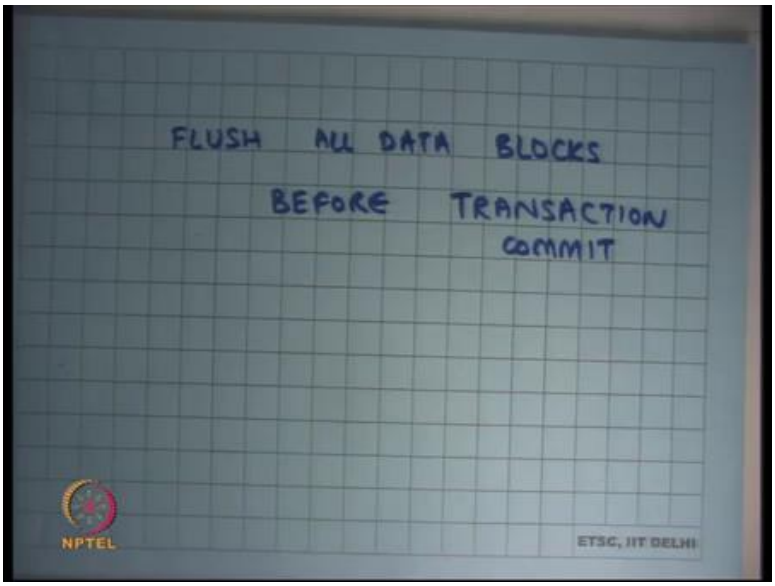
blocks and metadata blocks. But you know notice that the logging data blocks is not necessary for making sure that your file system remains consistent, alright.

So, and the other thing is that logging all data blocks may be expensive, if you can avoid logging data blocks and only log metadata blocks and you know do not. So, you avoid the 2x writes on the data blocks you only do 2x writes on the metadata blocks that is likely to be more efficient than logging everything, right. So, that is actually the default mode. So, ext3 as 3 two modes, ordered mode which says log only metadata blocks and journaled mode, journaled we had said everything, and the ordered mode is the default mode.

So, most of us most people use the ordered mode. And the order mode you only do this logging or atomicity maintenance for metadata blocks, but you need to be careful because imagine if the inode has been committed, but the data block that the inode is pointing to has not been committed to the disk then you will have a dangling pointer in your inode, right.

So, in the ordered mode the invariant that is followed is that you will flush or write to disk all the data blocks belonging to the transaction before committing the transaction which contains the log of the metadata blocks, right. So, in the ordered mode you basically ensure that all the data blocks are written to disk before you commit the current transaction that is all. So, instead of logging the data blocks you just make sure that there is an ordering relationship between the data blocks and the metadata blocks. And so, the ordering relationship is something like this.

(Refer Slide Time: 09:33)



Flush all data blocks before commit transaction commit. So, you flush all the data blocks before you do a transaction commit. Notice that am in this ext in this ordered mode of journaled ext3 I am using both ideas from the logging idea and ideas from the ordering method, right. I am saying, I will flush all the data blocks before I commit the transaction. So, let us say let us see what happens.

If I flush all the data blocks and I have not yet committed the transaction and there is a failure what can happen. Well, you have just updated some data blocks on the disk, but you know you have not really updated the inode corresponding to those data blocks. So, there is there is really no problem. So, there is no inodes will probably be point into the old data blocks or some data blocks have been updated some have not been updated, but the file system remains consistent.

The contents of the file may have changed partially, but the file system itself remains consistent, right. So, it is possible that some know the user wrote 10 blocks only 5 blocks have been written to disk on a crash and 5 have not, but the file system itself is consistent, right. So, that is important thing.

So, the difference between the ordered mode and the journaled mode and the ext3 is ordered mode is not concerned a concerned about atomicity of full operations atom, ordered mode is just concerned about atomicity of all the metadata operations to ensure that the file system remains consistent. It does not care about the user side or the users

data. So, users really left to his own on his own devices to basically make sure that his logic remains correct, right. So, that is one thing, ok.

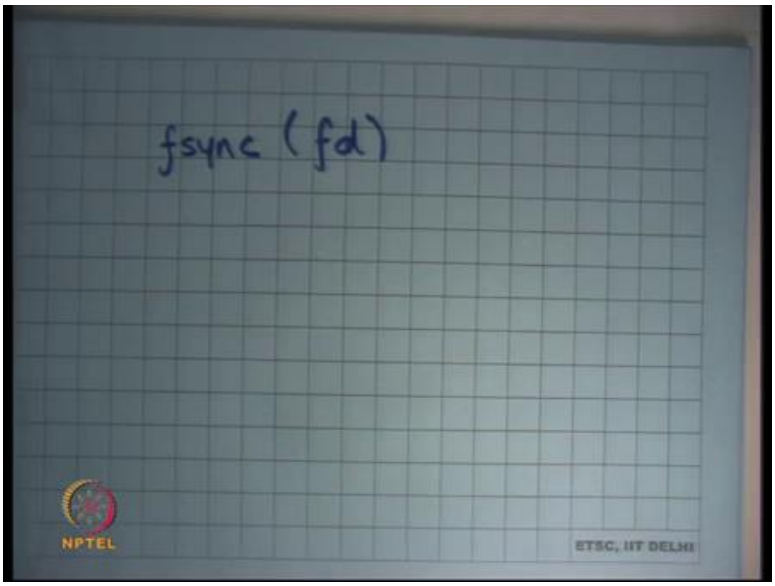
So, now, let us talk about how does the user. So, firstly, you know I hope we are all convinced that using this method we can be sure that the file system remains consistent. There are a few things one needs to be careful about you know, a few detail details about the ext3 ordered mode. For example, you know if you let us say delete a block you cannot use it in a free list.

So, before, so you cannot recycle blocks before the transaction commit because otherwise there can be inconsistencies. So, there are some details that I am skipping over this over, but basically you know at the high level its basically saying that you will flush the data blocks and you will you know and you will not provide any consistency on the data blocks, but you will make sure that all the metadata blocks remain consistency, ok.

So, how does the user, I mean this gives no guarantee to the user. The user is doing something how does he know whether you know what he has done has been written to disk or not, right. So, for example, I have given you an example of an ATM machine let us say your ATM program was running on top of an operating system and the operating the ATM program said you know I want I want to make this transaction and he debits my account with some money and gives me the cache.

But the debit has not actually been reflected on disk; so, what is the device that the user has to make sure that the thing has, has been committed to disk.

(Refer Slide Time: 13:03)



And so, there is a system called fsync, right or you know a different operating systems can have different versions of this. And you can say you know I want to flush there is the contents the data blocks of this file to the to the disk, right. How will the file system implement something like fsync? How will ext 3 implement fsync let us say? In the journaled mode.

So, there are two modes in ext3 there is a journaled mode and there is an ordered mode. So, let us say in the journaled mode if the user calls fsync what should the operating system do before it returns from the system calls. It should commit the current transaction, it should commit the current transaction and that is it, does it need to wait for the transaction to get applied to the file system. Tree, before it returns from fsync.

Student: Yes.

How many say yes? 1, 2. How many say no? 2. Everybody else is undecided, ok. None of the above, alright. So, an answer is you do not need to wait for it to apply, right commit is enough. Recall that the semantics of the file system is that you know, anything that is been committed is as good as done. It is just a matter of you know apply syncing the state to the tree, now that is just something that and that is you know you can consider it as an optimization. So, that the next time you are looking through the tree you do not have to go to the log to look at the latest contents you know you can just do it

from the from the tree itself, ok. So, as soon as you commit the transaction it is as good as it is done on the disk and it will persist across powerfulness, right.

So, when you do an fsync the operating system just in the journaled mode just commits the current ongoing transaction and so, closes the current transaction and commits it before it returns.

Student: Sir, in case of recovery likes the power failure occurred in this stage after fsync, sir we should want that the tree should be write?

So, let us say if there is a power failure after fsync and you know you recover then you the question is should not you should not you desire, should not you require that the tree is correct well not really, right. I mean at recovery time you will just you know you know that there is a log. So, the tree could be in.

Student: already seen committed, right.

So, the log is showing committed, but the log has not been freed. Yes. Recall that the ext 3 file system first commits the log, after the commit it starts applying the log to the file system tree. If there is a failure in the middle or at the beginning of this application, it is ok, you know you have not freed the transaction you have just committed the transaction you will free it after the transaction has been applied completely to the file system disk, at that point you will free the transaction. So, in the journaled mode it is enough to basically commit the transaction and then return.

So, but you know an fsync is a relatively slow calls as you can imagine because an fsync cannot return without actually making a disk access and that is that is expected, right, unless of course, there were no dirty blocks in your block cache in your buffer cache. On the other hand, most other operating system most other system calls which is accessing the file system do not actually need to go to the disk because we are using it right back cache for the buffer, right. So, the fsync is a device for the user to basically make sure that that things are getting flushed to disk.

So, what happens if the user is using fsync too often? System becomes really slow you are write back cache becomes completely ineffective, you basically come you know reserve you basically look start looking like right through cache, you know let us say he

is doing fsync after every system call, it is basically like saying that every system call becomes one transaction. So, this system becomes really slow, right. And it becomes even slower for something like a transaction from, something like a logging system like ext3, right ext3 logging was actually heavily relying on the fact that you are batching lots of writes. They are doing 2x writes, but because you are batching them the performance is not really 2x is lower, right.

But if you if you as a user is actually calling lots of fsync you are actually 2 x lower, you are writing it to the you are writing small transactions to the log and then you are applying them to the system to the tree, ok. Let us see what about the ordered mode. So, if I am using the ext 3 ordered mode and if the user calls fsync what do I need to do? I need to flush all the data blocks and I need to commit the transaction which contains all the metadata box, right, that is all, ok.

So, what are the some types of applications that require fsync? Why that require the data beyond the disk before you know some you do something? And most of these; so, such applications are usually what are called transactional applications or transactions, where you know, and an ATM transaction is an example of that, or a bank transaction is an example of that.

You basically want that you know, you basically have committed to the disk before you actually display to the user that your amount has changed or you actually disperse dispense cache to the user etcetera, right. So, you would want something like that. And how are such applications typically implemented? These applications are typically implemented by using what is called a database, right.

So, database basically maintains all this state you know who has how much money etcetera and database is supposed to give you guarantees that you know if you have said something, you have asked the database to do something, it has actually done it on disk it has committed the whatever operations are you done by a commit in a transaction of a database you basically make sure basically mean that the data will persist across power failures.

So, if you are running a database on top of an operating system you know as an application, then the only way the database can provide you those guarantees is by calling the fsync call, right.

So, database is an example, a user a user space database or a database running as an application is an example of an application that makes lots of fsync calls and is likely to be very slow on something like this, right. And that is the reason you would typically implement a database as a standalone system without an operating system below it, right.

So, you would not implement a database on top of an operating system because you have two layers of management of disk. It is been, you know if the disk, if the data the database should optimize itself, so if the database had come direct access to the disk it would have been able to optimize these things much better, right as opposed to sitting on top of a file system like ext3, ok.

On the other hand, most other applications do not care about persistence for transactions or and so, they can run easily, and they do not need to call fsync calls and they perform excellently all on something like ext3.

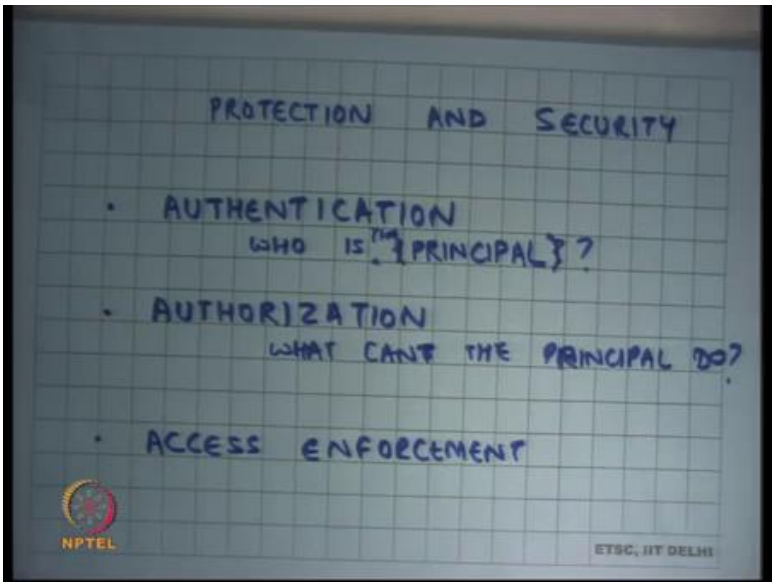
Student: Sir. So, the DBMS has the, it does not has those privilege levels to actually access the disk directly?

Does the DBMS have the privilege levels to access the disk directly? Well, I mean, so I said there are two ways to implement a database management system DBMS, one is you implement it as an application on top of an operating system in which case it cannot access anything. The other thing way to implement a DBMS is to implement it as a kernel itself or within the kernel, right in which case it has all the privileges, right.

And so, what I am saying is you know here is a very good example why you would want to implement it in the kernel as opposed to doing it at the application level because you do it at the application level, you have to go through the operating system interfaces. And, the operating system is playing tricks underneath and then you want certain guarantees out of it and so, the total performance sum of performance is really not what you would like it to be.

On the other hand, if the database had complete direct access to raw disk it would have been able to optimize these things much better, ok. So, that is you know I will finish. I will wrap up the topic on file systems with this discussion and moving on to the next topic protection and security, right.

(Refer Slide Time: 21:17)



So, we all know that we need you know we need lots of security or protection guarantees from an operating system. Your files on the shared infrastructure like the shared lab that we have should not be readable by other people, should not be writable by other people and so on. And so, you have a notion of these are you know these are my files, these are your files, you also have firstly, you have notion of identities. So, they are different identities each of us has a different identity in the system, right.

So, protection and security are consistent, is made up of 3 things authentication which is a method of saying who is who, right. So, our login ids or you know some other names basically say that who is who is behind this action, ok. So, we associate we have some notion of entities and we basically have some way of ensuring that this person is behind this action or this entity is behind this action. This entity is also called principle, alright.

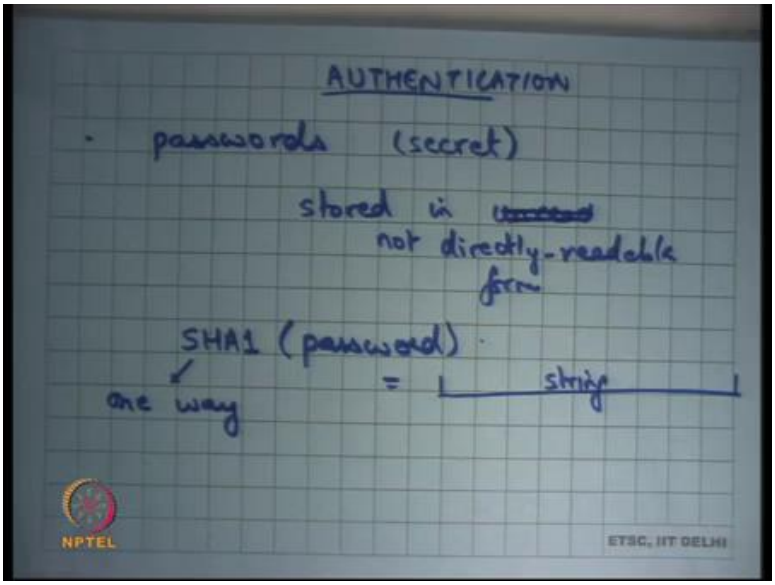
So, authentication identifies the responsible principles behind each action. Then there is authorization. So, this says who is the principal behind; let us just say this authorization is what can the principal do, right. So, basically says that user xyz can access files a b c in read mode and access files d e f in write mode and so on, right. User route can access all files in a b c in read write mode for example. So, these are all examples of who can do what, right. So, that is authorization, who is authorized to do what actions.

And, then these access enforce enforcement. So, you have a you have a method to do authentication, who is behind this action. You have a method to do you have a method to

do authorization, what can this principle do and then there is an access enforcement which ensures that only legal things are possible which means only things that a principle is allowed to do is as possible, right. So, these are these are these are the 3 sort of aspects of prediction and security.

And, if there is a flaw in any one of these then there is a security attack. So, let us say if there is a flaw in the authorization and you can impersonate as me, then you can do anything you will you like. If you can make you can add a flaw to the authorization table and say you know I can do a something that you are not supposed to do then there is a problem. And, thirdly even if you have all these things correctly, but if the access enforcement is not correct then you know there is a problem, ok.

(Refer Slide Time: 24:47)



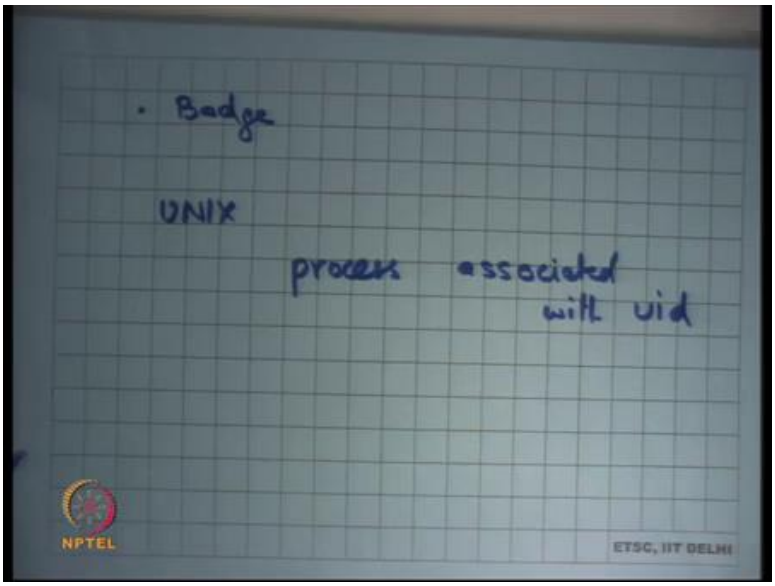
So, let us look at you know how these things are done. Authentication you know let us say passwords. You all know what our passwords, they are relatively weak way of establishing user identity usually stored. So firstly, they are secret and stored in unreadable form or not directly readable form, right. So, the way passwords work is that you store, so let us say I have a password on a system then you will use a one-way hash.

Let us say there is a there is a famous for there is the popular one-way hash called SHA 1. And you will apply the SHA 1 my password and you will get a large string, ok. And this is a string that you will store on your file system, ok. And this function is what is called a one-way function.

So, by one way it basically means that it is very easy to get this string from this password but given the string it is very hard to recover the password to know what the password is, ok. So, it is very hard to recover. So, even if you have the string you will not be able to you know guess the password and so, even if you have the string you will never be able to log in to the system because you will not be able to guess the password. So, that is typically how you will store these things.

The best attack you know assuming that you know our cryptographical algorithms are correct, our axioms in the science in computational theory are correct, then the only way to guess or to break this password based authentication is through brute force attacks where you basically go through all possible passwords up to a certain length and you basically check and so, this and so to the way to make this secure is to make sure that your passwords are long enough, so that brute force attacks are relatively ineffective, right. So, that is basically that is pretty much mostly how authorization is done. Alternate form of authorization is badges, right.

(Refer Slide Time: 27:19)



So, everybody has a badge a badge could be something you know a card that you are using to authorize yourself or it could be a USB, USB device that you have to plug in etcetera. It is the advantage of something like that it does not have to be kept secret should not be forceable or copiable, but you know if its stolen then at least the owner

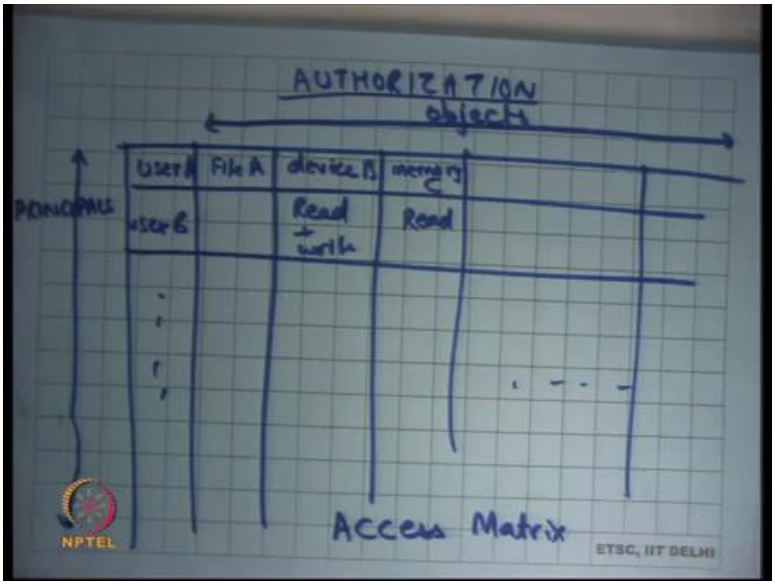
know that its stolen and can register a complaint etcetera and or change the change the authentication mechanism.

If you are using badges you know there is an interesting paradox that the badges should be easy to make, cheap to make so that you can distribute lots of badges to lots of people, but should be very hard to duplicate badges. So, it should not be possible that if you have a badge and I just get access to it for few for some time I should not be able to duplicate it.

So, you know there should be some technology there that ensures that is not possible to duplicate it and yet it should be cheap to sort of manufacture, right. So, once authentication is complete you basically all the actions that are performed are basically done with that principles id. What does this mean in the context of an operating system? It basically means that all the processes run with that user ID, right

So, basically on UNIX every process associated with uid, right. So, recall that process control block that we had for every process. So, apart from all the other Meta information and other information that you will have is basically who is the user id or the particular process, right and that is; so, whichever process is running you always know who is the uid of that particular process.

(Refer Slide Time: 28:59)



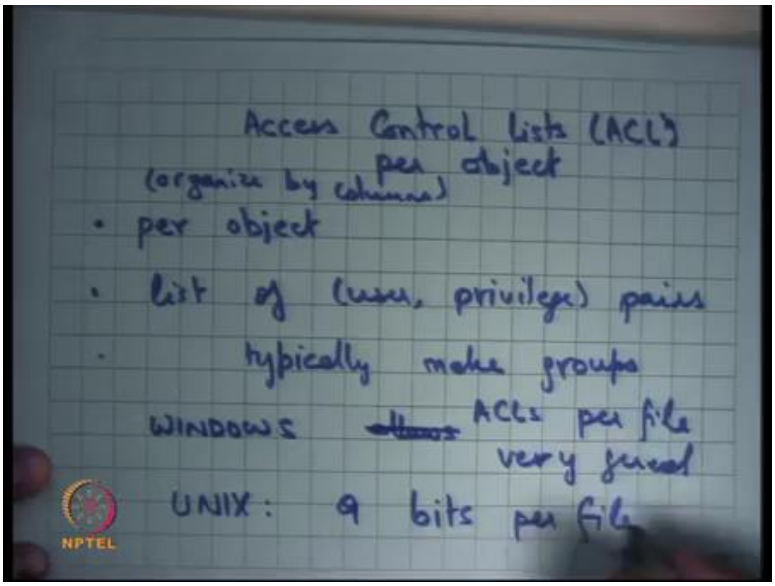
So, let us see authorization. So, who is authorized to do what? First let us look at the you know what does authorization mean in the more general sense. Authorization is nothing but a matrix, where let us say you have one row of principles. So, these are principles. Let us say this is user A, user B and so on and on the columns, you basically have objects.

So, for example, may file A, device B, memory region C and so on, right. These are all examples of objects that you have, and this authorization table basically says that user B can access memory region C or not, right and in what way. So, let us say read or read plus write etcetera, right. So, that is basically not a on a logical level that is what an authorization matrix looks like and this is also called an access matrix.

So, this is a logical representation of who can access what, but different things different authorizations are implemented. So, this is the logical authorization mechanism, but different authorizations or different authorization for different types of objects are implemented in different ways. For example, authorization for memory is implemented using virtual memory subsystems like page tables and segmentation, right.

So, you are whether you can access this particular region of memory or not is basically authorized, the authorization is implemented using some kind of a page table. Authorization for a file is implemented in a different way, ok. So, we are going to discuss that very soon. So, in practice you do not store the full access matrix there are two ways to store the access matrix either you store what are called access control lists Or ACL per object.

(Refer Slide Time: 31:17)

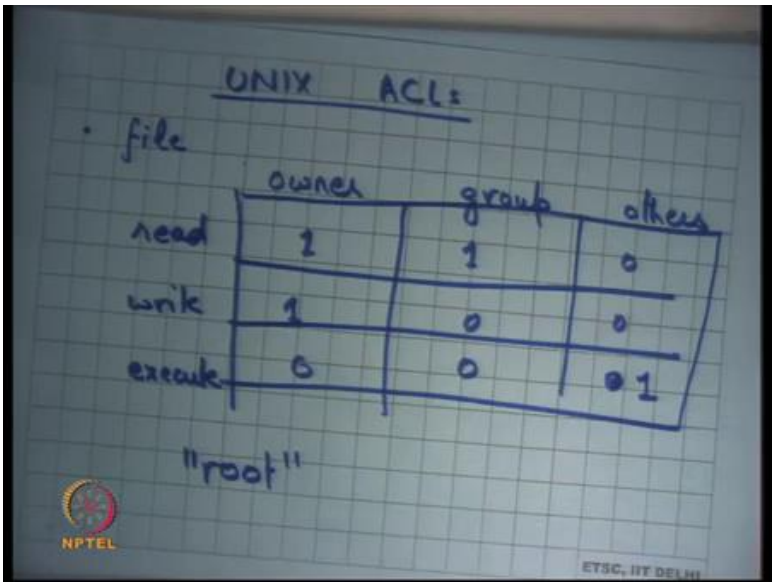


So, here you basically saying that I will organize things by column. So, who all can access memory C or who all can access file A.? I am going to have with file A I am going to have a list of all the users who are allowed to access it and in what way are they allowed to access it, whether read or write or execute or whatever else, ok. So, that is an access control list. So, per object organized by columns, per object, list of user privilege pairs, alright.

So, let us now this list can become large. So, typically, typically make groups and indicate that this resource is accessed by this group. So, you organize users into some groups, and you say that this file is accessible by this group, so all the users in that group can access it without those trouble etcetera. So, let us look at it in practice Windows allows, so Windows implements ACLs per file very general. So, on Windows you can go to you know the properties of a file and specify a long list of which user can do what, right.

So, here is an example on reading system that allows very general ACLs ah. Something like UNIX has 9 bits per file, right and we are going to see what these 9 bits are.

(Refer Slide Time: 33:25)



So, UNIX ACLs; so, firstly, it divides, so let us say, so I am talking about a file it divides it into it divides all principles into 3 types owner, group and others, and read, write execute, right. So, this becomes, so there are 9 bits there could be 0 or 1. So, that is basically the UNIX ACL. Every file is associated with one owner and one group, right.

So, every file on the UNIX file system will say you know the owner is user A and the group is group B and you know the creator of that file can or the owner of the file can choose which group this particular file belongs to or is owned by. And then you can have ACLs of this type which basically say that this particular user is allowed to and so, the owner can do these things to this for example, the owner can read and write, but not execute this file you may want to say that.

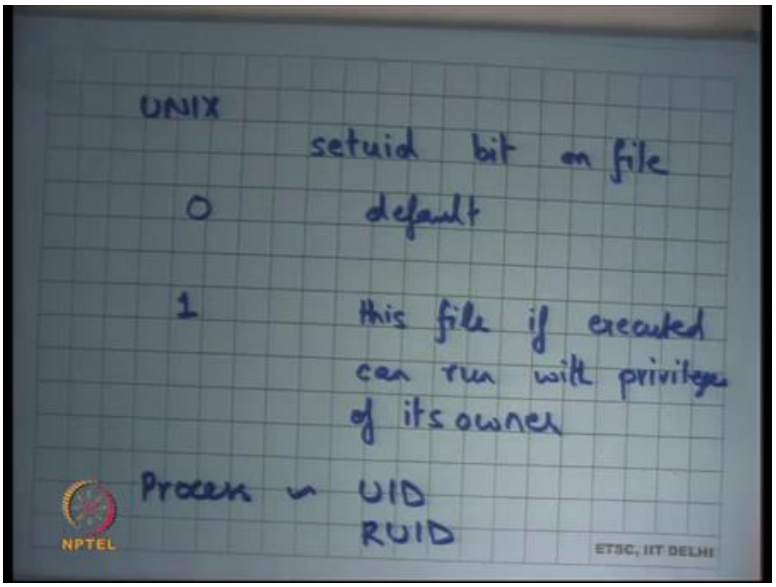
The group can read the file, but not write to it and others everybody else should not be able to either read or write. So, let us say here is an example. This is 1, this is 1, this is 0, this is 1, 0, 0 and this is 0, 0, 0, right that is 9 bits to ensure to basically implement ACLs. So, clearly this has less, there is less general generality than a complete list that you had on windows.

There is another thing about UNIX, there is a special user called root and root user is allowed to do everything to every file, right. So, for the root of these this access matrix is not these 9 bits are immaterial; the root user can do anything to any file for example. Yes.

Student: Sir, as in the group the group permission should be a subset of owner permissions?

Should the group permissions be a subset of owner permissions? No, no. See, it is a full 9 bits, so you can do whatever you like, ok. So, that is basically you know that makes it. So, the advantage of doing this is that you have a bound on the size of information meta information that you have you know you will store you can store this information, you can hope to store this information in the inode of the file as opposed to an unbounded list of user privilege pairs which will be very difficult to maintain, alright.

(Refer Slide Time: 36:11)



Apart from this, so the UNIX also provides an interesting thing which is a setuid bit on the file, right. This basically says so, with, so by default at 0, so you know it is the default behavior whatever you expect, but let us say this file is 1 then this file is executed can run with privileges of its owner, ok. So, every file also has one bit which is called the set uid bit.

If this bit is set then you can run when if some if let us say I have a file in my home my directory, but I have set the set uid bit to 1 and I am basically allowing anybody else to run this program and with my privileges. So, for example, this program if run, so recall that I said that any process that you that user x invokes will run with the privileges of user x, but if this process is exiting a file that had the set uid bit set or 1 then this program can also execute with the privileges of the owner, ok.

So, why is this required? Let us first understand what is the application, why do you think this is this kind of thing is required.

Student: (Refer Time: 37:47) this file is executable file is leads to execute some other executable files.

If this executable file leads to execute some other executable files so.

Student: It is like it should have net privileges of the.

Well, I mean let us say all the other ones are also executable by others, so you know then there is no problem. Why do you need this? Are you; I think all of you are using an executive call set uid you know through the course of this project of this course? Can you think of it?

Student: (Refer Time: 38:25).

Ha.

Student: (Refer Time: 38:27).

Yeah, all the scripts that you are using on policy are actually set uid files, right. So, if you want to submit your assignment, so what I have done is I have a script which is living in my, I have a file which is living in my home directory whose owner is you know whose owner is me. And, but when you run it you can actually you have the privileges that process has the privileges to write into my home directory, ok. For example, it copies files from your home directory into my home directory to make the submission.

So, how can the process that you are invoking has the privileges to write to my home directory? Right. I said all the processes that you invoke will run with your privileges and you do not have privilege to write to my home directory, right. So, the way to do that is basically to make to basically allow this file to be executed with my privileges, right. So, that is what the set uid bit allows you to do. The set uid bit allows the owner to say that this particular logic you know I have tested this particular logic and this is safe and I allow other people to run this particular this piece of logic with my permission, so that they can access resources that I could access. Yes.

Student: Sir, is that similar to just saying that for example, part of ux is my privilege then I will set the others privileges also to (Refer Time: 39:47). So, is not that similar that my owner privileges are also what yeah those?

Is not it similar to saying that you know I have an execute bit set to a file? Now, what is a different between a set uid bit; I am just saying that this file is executable by others.

Student: (Refer Time: 40:05).

Right. If I just say that this file is executable by you will be able to execute it, but this process that will execute out of this you know out of your invocation will execute with your privileges not with my privileges and so, it will never be able to touch my files or write to my directories, right. So, set uid allows you to not just execute the file, but also execute it with the privileges of the owner of that file which also means that if there is a bug in this program then you can trick this program into you know doing things that I would not want you to do for example, right, ok.

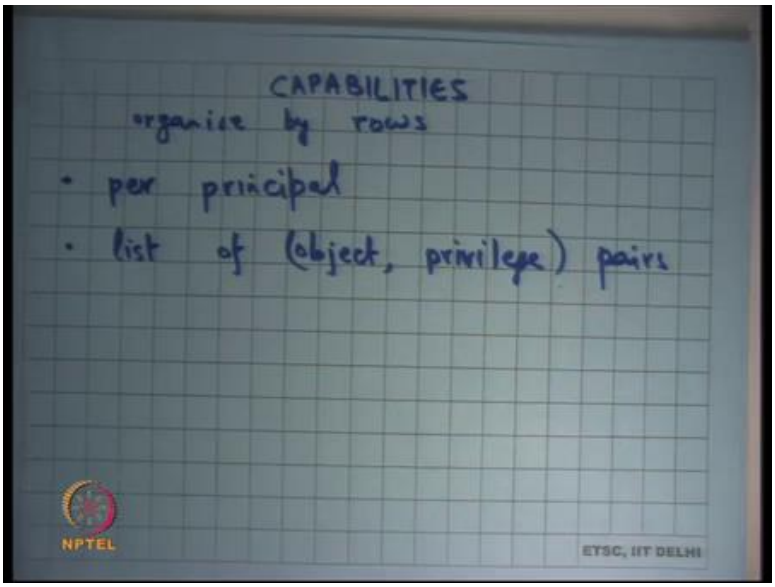
So, also this you know typically when you do these things for example, the submit script that we have will not just is not as capable of writing to my directory, it is also capable of reading from your directory irrespective of what permissions you have for your directory. So, if your if your permissions for your directory are saying that only I can read to this directory, yet this particular process that is you know that is that can run as my privileges can actually access your particular directory. So, this process can either can switch between your privileges and my privileges, right.

So, how does how is this done? Well, I said every process has associated with a uid which says you know who owns this. Actually, every processes two identifiers, one is called the uid and another is called the real uid, right. And so, a process can switch between its uid and its real uid at will, right.

So, they are using system calls, it can basically say that now I want to run with the privileges of my real uid and then I can say I want to now run with the privilege of uid. By default of the set uid bit of a process is not set then you then real uid and uid are equal to your uid who has invoking it, but if its if set then you know real uid will be your uid and the uid would be let us say the owners uid and, so the process has flexibility to switch between the two.

So, when I want to read from when this process wants to read from your directory it should switch to your uid, when he wants to switch to my directory, he wants to he should switch to my uid and so, right. So, that basically allows him to do this, ok, right.

(Refer Slide Time: 42:49)



The other way to implement access control are capabilities, what is called capabilities. So, we looked at access control lists. The other way to do access control is called capabilities where you organized by rows, right. So, where basically you organized by rows, the access matrix; so, per principal, list of you know list of object privilege pair let us say. So, user x can access all these files or user x is allowed to access. So, you know you store this information in per user as supposed to per file that the other way to implement this, ok.

So, let us see what are some examples of capabilities that we know about. Well, here is an example file descriptors. So, every process has the set of file descriptors that basically say that these are the files that I can access, or I have already opened. So, I can you know I can access it in xyz way. And so basically a process can be thought of as a principle and all these file or these pointers to file blocks or structures that that represent open files can be thought of as objects and you basically storing file descriptor table that is your capability list, what are the capabilities that this process has.

So, one some advantages of capabilities are that let us say I spawn a new process. So, it is very easy to delegate. So, if I want, I have some capabilities I want to give you those

capabilities I just give you my table and you know you have the same level of capabilities. For example, forking the process involves giving my capabilities to my child process, alright. So, that is that similarly you know that is a memory management, so virtual memory. So, you basically say that you know what all areas of memory are you allowed to access this is kept per process. It is a you know it is a capability list per process.

In general, the advantage of capabilities is that it is very easy to delegate. So, you can you know giving, if I have some capability, I can give you the same capability by just giving you access to the same table basically, right. The other thing is capabilities are also a little more secure in the sense that only if I have a capability can, I even name the system, right, name the object, right. So, capabilities are also used as a way of naming the object. For example, if I you know if I have an, so for example, virtual memory, so I can name this particular location because I have a capability of accessing that particular location.

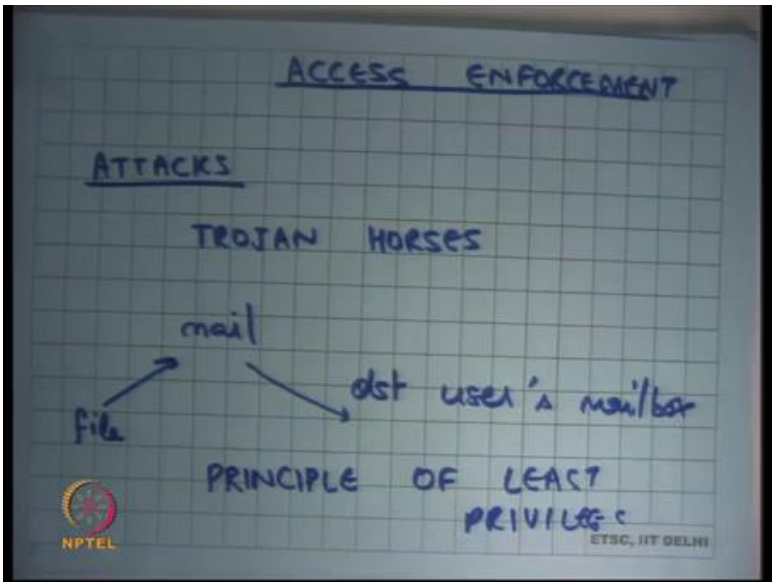
Another process who does not have the capability of accessing this location will not even have the name on in or the way to access that location. So, in general you know capabilities allow private namespaces, so basically you have a namespace for this process or namespace for this user and a name space for that user and this user does not even know that those files exist you know all those resources or those objects exist which are in your capability list. Whereas, if you have access control lists the entire namespace is public, ok.

So, capability systems discard its visibility, namespaces are private by default and so, in there they also thought to be more secure than access control list in that sense and there have been experimental systems that have tried to implement use capabilities for file system instead of access control lists, but you know there they tend to be a little clumsy for file systems whether I used for other things like you know virtual memory systems etcetera. They are also used in distributed systems a lot and capabilities can are also implemented using cryptographic means, right. So, for example, a capability could involve a signed key or a signed certificate.

So, let us say I got a you know if I have if I want to authenticate to somebody I can use or I can I can use a certificate from sub; so, if I want to prove my authorization to access

a resource I can show a certificate for somebody else to show say that I can access this particular resource, right. So, that certificate, so crypto capabilities can also be implemented using cryptographic means basically, ok. Finally, enforcing access and for a kernel, so what does access enforcement mean.

(Refer Slide Time: 47:25)



So, we have seen ways of saying that here is, here is how you authenticate. You authenticate using passwords or badges or some manual biometrics or whatever. Then you have some way of authorizing which includes access control lists or capabilities access control lists or capabilities and you know depending on. And, there is a there is a trade off on how much space you want to take to represent this stuff and how much generality you want, and we have seen some an interesting example of UNIX versus windows. And then you have some way of ensuring that these things are obeyed, right.

So, which this basically means that some part of the system should be responsible for enforcing this and this part of the system should have total power on everything, ok. So, there has to be some part of the system which has total power and its basically and this part of the system is checking and saying yes or no to actual controls and for an operating system this part of the system is the OS kernel basically, which can which can which has total control and it is going to do authentication and authorization.

So, let us look at some attacks. So, why do you know let us say what are the what are some kinds of attacks that happened, what does what does the security attack mean. So,

we keep hearing that you know xyz, there is xyz vulnerability there is this attack and that attack, what do these things really mean. Basically, an attack means that a user has firstly, it basically means that a principal has been able to do something that he was not authorized to do and the most common way of doing an attack is basically what a Trojan horses, right.

Trojan horses are basically, so from derived from the mythical tale. A Trojan horse basically means that a useful program that was supposed to do something legal and meaningful is subverted to do and had privileges of, had higher privileges than what you have is subverted to do something that it was not supposed to do, right. So, that is what a Trojan horse mean. So, let us see where all, what are some ways of implementing Trojan horses.

Well, let us you know let us take extreme case, if you could potentially subvert the kernel you know that is the ultimate Trojan horse that you can have because that has the ultimate power and if you can subvert that the access control enforcement engine itself then you have basically done, you can you can bypass everything else. Similarly, you know if you could subvert program that has a set uid bit set then you can get control over that particular programs resources.

Many programs you know it especially in the older days, many programs in a systems run with set uid bit set and owner as root, ok. So, which basically means that this program is going to execute as the root owner, root can will execute with root privileges, but anybody can invoke it, right. So, let us you know let me give you some example. So, let us say you know there is a program called mail, right that I can give some file to it and it will copy it to the destination user, so mailbox, alright.

So, let us say there is a program that I provide in my system that is called mail that says, basically I you know that takes an argument which is some text or some file which contains some data and I say I want to mail this program to this particular principle and what this program is going to do is copy that that data to that person’s mailbox. So, let us say this is the program. So, this particular program in the older days would run using root privileges because this program, so this program will be a program which will be owned by the root user and we will have the set uid bit set because this program requires the capability.

Because, I can name anybody in my in the in my destination in the may recipient field and so, this you program should have the power to write to anybody's file mailbox. And so, this program for that reason needs to run with the root privileges, at least for some time and so, and with set uid bit set. So, if I could subvert such a program then I can basically get do anything that the root user could do.

And there are many types of attacks which are not the subject of this course that that can allow, so the many kinds of common bugs in programs that they were very common or you know 10 years back not so common today, so it is harder and harder to find bugs and programs that can be subverted, it was very common to do that 10 years back. And if you could if you could identify the bug you could exploit it to actually be as the program to behave in a certain way which the programmer did not expect it to be there.

In general, you know when you have programs of this type this you should follow what is called principle of least privilege which basically means that if there is a program that is running. So, if you have a large program let us say let us take the example of the mail program then it should try to run with the; so, it has this mail program has a set uid bit set. So, it can either run with root privileges or run with my privileges.

So, the program should run with the least privilege at any time. So, least privilege will be required at any time. So, for all the operations that do not require root privileges at that time the privilege the program should be running with my privileges. So, basically what that means is all the code that executes executed during that time if there is a bug in that code then there is no problem you know or you know actually there are fewer problems, as supposed to if you run all the time with the root privileges that is it, ok, alright.

So, with that let us finish. And we are going to discuss another topic next time which is called scheduling.