Operating Systems

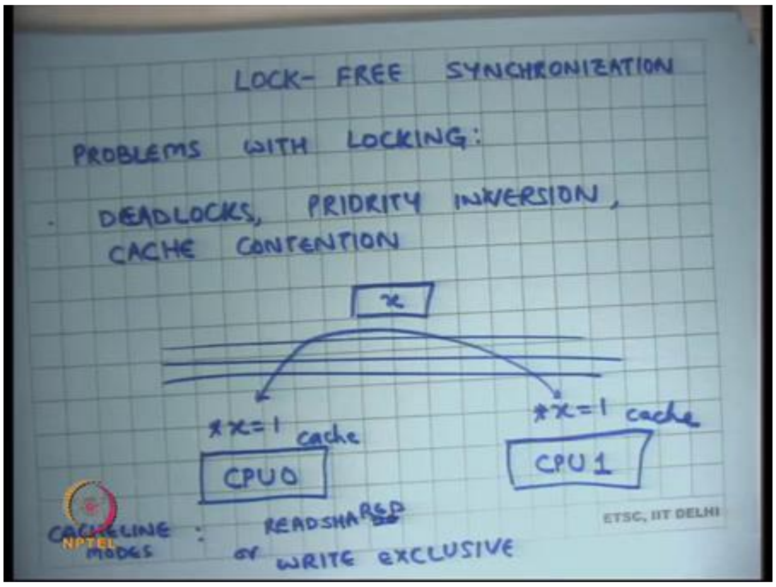
Prof. Sorav Bansal
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

Lecture - 39
Microkernel, Exokernel, Multikernel

Let us start. So, a cache line can be held in read shared mode across multiple CPUs or write exclusive mode. So, the idea is if you if the CPUs are only accessing the location in read mode, then it can exist in multiple caches at the same time. If it is being accessed in write mode, then it has to exist in one cache at any time right.

And if some other CPU accesses it while this is cached in some other CPUs cache, then the cache line has to be brought from that CPU to my cache, that CPUs cache to my cache right. So, that is basically write exclusive mode. And so, what happens is if there are two CPUs which are accessing the same location in write mode, then there is a lot of cache line bouncing that is going on.

(Refer Slide Time: 01:16)


LOCK- FREE SYNCHRONIZATION

PROBLEMS WITH LOCKING:

- DEADLOCKS, PRIORITY INVERSION, CACHE CONTENTION

CACHE LINE MODES : READ SHARED or WRITE EXCLUSIVE

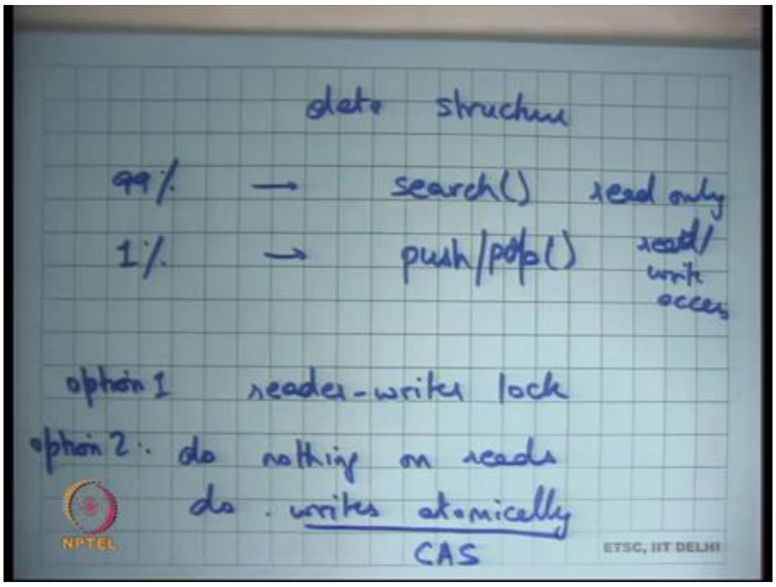
And this cache line bouncing is not limited by the speed of the CPU, it is limited by the speed of the bus right; at what rate can the bus do these transactions of cache line bouncing. And so, what can happen is that most of the time the CPU is just idling waiting for the cache line to come to it.

On the other hand if all these accesses were cache hits, then the CPU could have executed at full frequency and could have been you know up to 10 times faster depending on what the workload is, but you know a cache hit versus the cache miss can have the up to a 10x 10 x penalty ok.

So, what would have been a better thing to do? A better thing to do would have been if I could organize my code such that most of my code is read accesses and so, all the shared variables will get cached in read shared mode in all the caches and all the CPUs will execute at full speed right.

We also said that because of the cache line bouncing problem, it can so happen that your code with two processors or two threads is actually slower than the code that could that you know that was just running with one thread because you know you are not using your caches effectively ok.

(Refer Slide Time: 02:28)



data structure

99% → search() read only

1% → push/pop() read/write access

option 1 reader-writer lock

option 2: do nothing on reads

do writes atomically

CAS

NPTEL

ETSC, IIT DELHI

So, we said let us take a data structure and let us say you know 99 percent of times you are accessing you calling search on it which is a read only access and then you know 1 percent, you are calling let us say push and pop right which has read write access.

Ideally you would have wanted that your search operation executes at full speed and by full speed, I basically mean that your data structure should get cached in the local caches of each CPU and read mode; you know that is search is executing most of the time and so, all these CPUs should execute at full speed. However, you need to make sure that you know search is properly synchronized with write operations like push and pop and so, you know one way to do that is basically have a reader writer lock.

If you have a reader writer lock, then each time you call search you have to set the state of the reader writer lock to locked and that becomes a write operation. And so now, you are not bouncing on the cache lines that hold the data structure, but you would be bouncing the cache lines that hold the lock right. And so, that is not a; that is not you know that is not a particularly good performance and it is possible that the performance actually worse than what it could have been on a single processing.

So, the better thing would have been to you know do nothing on reads. Hey this is option 1 let us say and option 2 is to do nothing on reads and do writes atomically all right. So, what does this mean? Reads do not need to do any synchronization. So, read the code for the search procedure remains the same, you do not use any locks for that. But in your right in your push and pop you will ensure that this update operation happens in one shot atomically right.

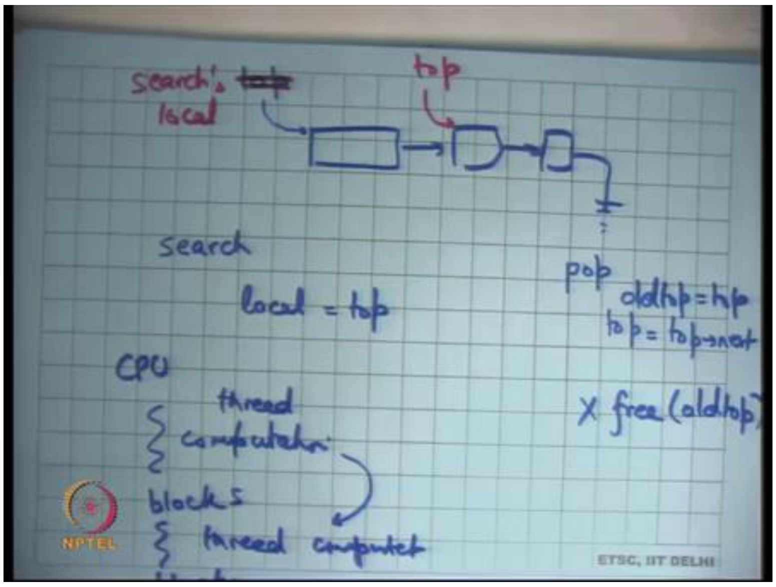
So, the read operation or any other write operation cannot see the update operation half done all right. So, what this will ensure is that if there was concurrent reads and writes, then either the read could have started before the update or the write could have started after the update, but the read could not have been I have started in the middle of the update with the update is atomic right.

If you could do that, then your reads execute at full speed which is 99 percent of your time and your writes execute you know roughly at the same speed, it does not matter; it is anyways 1 percent of the time ok. So, how did we do this? How do you do writes atomically? Well we use this compare and swap instruction on the hardware that allows you to atomically update 4 bytes right. And we said this is not always possible, it is possible when your update operation involves update of 4 bytes.

So, the way you do it is you first in local memory construct you know you first copy the data structure copy some parts of the data structure locally, compute on the local copy and then you use the compare and swap instruction to update a data structure in one shot right. And so, it is also called read copy update, you read the data structure or you read some part of the data structure let us say you read the top pointer or the head pointer, you manipulated it you change head to head you knows head to something else if you are doing push or if you are doing pop, you change head to head dot next in whatever case.

And then you use one instruction to atomically swap head and head dot next right and so, that it is read, copy and then you atomically update ok.

(Refer Slide Time: 06:30)



So, we were looking at this last time and we said let us say here is my data structure right and let us say this is my top and a then somebody called search and he took a reference to top. So, he said local is equal to top right. So, local is some local variable that search is holding; it could be a register for example, just allocated in resistor.

And simultaneously, let us say there was a pop operation and so, pop dot top is equal to top dot next. So, what will happen is that top will now point here, but search will hold a reference to the old top ok, but we say it is because even though search executed after pop has happened so, search will still see a well formed list and so, it is as though search happened before the push about the pop. The only issue is that you should not be reusing this memory right.

So, after you have after a thread has popped the first element in the stack, it will probably want to free this memory right. So, after so somewhere here, it will want to say free you know whatever the old top was. So, let us say old top is equal top. So, you free old top. Now there is where the problem is right. You cannot just free the old top anymore because you do not know whether there could be a concurrent search that is holding a reference to this old top; question is how when can I be sure that I can free it? And the answer is really I can never be sure in general right.

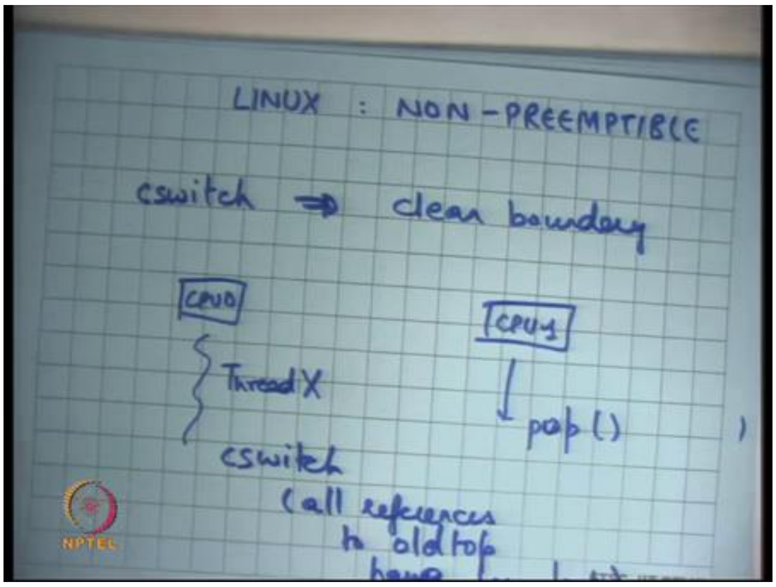
I do not know how long this other thread that called search is going to take to actually start dereferencing it right. Search is going to has a copy of local, but probably you know as soon as it is executes; the next 10 instructions, it will probably lose this reference. But we do not know when it will execute these 10 instructions and we do not have any idea of who all are holding this local this reference to the old top.

So, because this pattern was very common in the kernel where you have a data structure that is being read a lot and updated only very occasionally maybe once in hour or once in days or something, but it is being read a lot; it was very important to optimize something like this ok. And so, the solution that was proposed was to look at the structure of the operating system. Operating system goes through cyclical activity all right.

So, what happens is it does some computation and then blocks. If I look at a every CPU it runs on thread, then blocks let the thread computation blocks, thread computation blocks. If I could be sure that before blocking, all local references to any shared data structures have been given up that would be helpful to me right. If I could be sure that across this blocking, I could not be holding a reference a local reference in my stack or register to something that is global, it would have solved the problem for me right.

Because what I would have done is I would have waited for all CPUs to reach this block and once the CPU has reached a block, I can be sure that you know it is not holding a reference to old top right. All future accesses to the stack will start at new top, it is only the old ones that could have old top. But if it has reached the block, then I can be sure that it is not holding a reference to old top and this is true for the Linux kernel.

(Refer Slide Time: 11:12)



LINUX : NON-PREEMPTIBLE

cswitch ⇒ clean boundary

CPU0

Thread X

cswitch

(all references to old top have

CPU1

pop()

So, let us say. So, Linux kernel is non-preemptible which basically means that if a thread is executing within the kernel and there is a timer interrupt or any other interrupt, it does not cause a switch of it does not cause a context switch across threads right, it waits for the thread to you know yield before it actually does a context switch which basically means that if there is a context switch from thread one to thread two implies clean boundary. Which basically means whatever was the operation that the thread had to do inside the kernel that has been completed before the context which has happened, you cannot context switch in the middle of an operation all right.

So, what this also means that if there is a context switch, then I could not be holding a reference a stale reference to a shared data structure in my local variable across a context switch right. In other words, the search procedure could have would have either finished completely before the context switch or would be started after the context switch it cannot happen that I am in the middle of the search procedure and a context which happens ok.

Given this information, I can devise a strategy I can say that if there is a push or a pop, I do this atomic update right I do this atomic update and I wait to free a location till there is a context switch on all the other CPUs.

And there is a context switch on all the other CPUs, I can be sure no other CPU is holding a reference to this location. I can also be sure that any thread that is not running currently on any CPU, definitely it does not hold a reference to old top because if a thread is actually blocked, then you know it cannot be in the middle of the search it has to be either after the search or before the search.

So, this period where you wait is called the grace period and it depends on the system how you implement the grace period on something like the Linux kernel. You could wait for all the other CPUs to do a context switch before you can be sure that I can free old top after that all right. So, the algorithm is that push or pop or any update are going to atomically update the data structure in a read copy update manner and then I am going to wait for all the other CPUs to call a context switch.

And once they have called the context switch, I can be sure that I can safely free this old top. How do I wait for all the other CPUs? Yes question.

Student: wait till this processed because like one CPU context which is, and the other normal process can also use the same barrier.

Ok.

Student: There is one context switches that all (Refer Time: 14:35) all contexts which use.

Let me just try to understand what you are saying. So, you are saying that let us say there is CPU 0 and there is CPU 1 right. This one says push well let us say this one says pop and he has not freed the location. So, question is when can he free. So, let us say there was some other thread that is running on the CPU; let us say this is Thread X. So, my first assertion is the only other thread that could be holding a reference to old top is Thread X, there is no other thread that could be holding a reference to old top ok.

Because if there is a thread that is swapped out that is not currently running that is definitely not hold that is either you know finish with search and or I it did not call search at all and if it call the search when it comes back again it is going to restart you know starting from the new top. So, the only other thread that could be holding a reference to old top is Thread X.

Now if you can generalize that if the another CPUs and another threads that could be holding a reference of thread top. Question is till when can these threads hold a reference to old top? So, my suggestion is that because Linux kernel is non preemptable; if there is a context switch on all these other CPUs, then I am sure that after the context switch nobody can be holding a reference to old top, make sense right. So, if there is a context switch here, then I am sure at this point all references to old top have been lost.

Student: Sir.

Yes.

Student: Is that one of the CPU take very long time to context switch and then it might slow down the rest of the thing?

So here is an interesting thing, it is possible that one of the CPUs takes a very long time to context switch and so, it slows down everything else does it really slow down everything else? What am I waiting for context switch what am I waiting to do before all the other contexts which is happen?

Student: Free.

It is just a free. So, if it takes a long time for all the other CPUs to context switch then it is a just means that that location lingers are on for a long relatively long time, but that is ok. Assuming there is a lot of space there is a lot of memory, you are just holding on to a location for a longer than it was needed. Assuming memory is plentiful that is an extremely useful trade off to have.

Student: Which let us say the change that we made is going to be used for CPU one has call pop when normally change that is been made will be used by the later process and till the free command is called probably you cannot start.

So here is the question if one thread is called pop, he has made some change so, can the other threads use that change before the free has been called?

Student: No, it is not.

Why not?

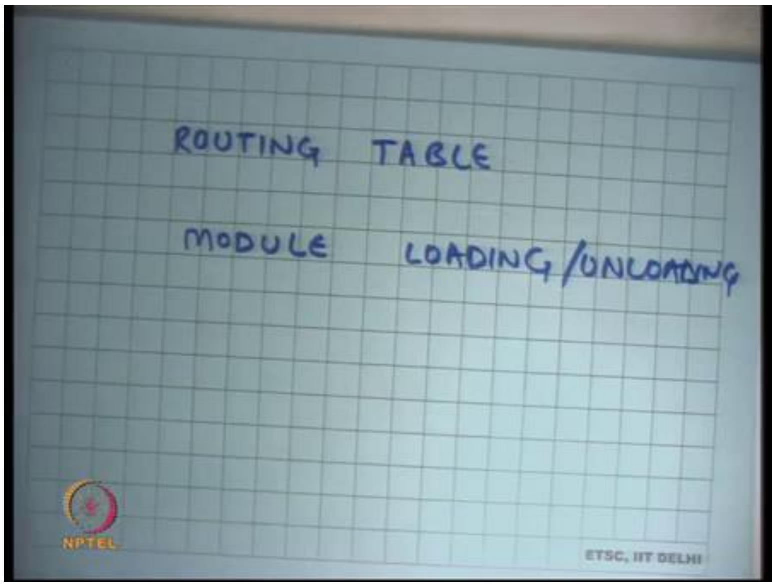
Student: Since because they have started before the change has been made and so, it may cause look at the ... if the ... benefit (Refer Time: 17:59) maybe loss.

I do not see why. So, my suggestion is that if see as far as the pop is concerned it has atomically up made the update, it is just a matter of freeing the location and freeing can be delayed arbitrarily. Assuming I had infinite memory, I could I would say you know I do not even need to care about all this. Assuming I had infinite memory, I would just, you know just execute cast to implement pop and I would not even worry about freeing and I could be sure that so on.

So, it is in a way RCU is a way of trading space for time, you know you are using extra space for longer, but that way you basically make sure that your reads are very fast you do not need any synchronization on the read path all right. So, it is a very interesting idea and it is very practical and useful and let me tell you a few examples there is actually very useful in the Linux kernel. So, firstly, notice that in this RCU approach I use some property of the software in this case I use the property of the Linux kernel that it is non preemptable.

So, I need to know this kind of information to be able to and you know implement this kind of thing. If I could not make any suggestions about the software in which this RCU was implemented, it has it will been not possible to implement the scheme.

(Refer Slide Time: 19:24)



ROUTING TABLE

MODULE LOADING/UNLOADING

So, what is the property that I am looking for? I am looking for some cyclical activity and I need some point in that cyclical activity where can be sure that is a point where you know all old references have been lost local references will be lost right. So, that is good enough for me. So, typically you will implement things like RCU in the kernel where you can make these assertions.

RCU at the user level is much you know much more complicated I do not know I cannot just implement RCU as a library, like I can implement pthread locks and other things right. Pthread locks make no assumptions about the application behavior RCU makes assumptions about the application behavior. So, you know there are user level libraries that implement RCU, but you are would tell them what is what the psychical activity is and when is it safe to actually you know release references or free things in a safe manner ok.

But let us look at the Linux kernel where RCU is used a lot and where is it used? It is used for example, in the routing table. So, routing table is an example of data structure that has a very high rate of lookups and each time a packet comes you look up, the routing table to figure out where it should be sent and assuming you have multiple network interfaces to this machine and it is acting as a router.

So, there is a routing table very occasionally do you update the routing table, but you need to make sure that you know these updates to the routing table and the lookups to the routing table need to be synchronized with each other. If I had used read write logs to synchronize this, then I would have severely penalized my look up path which needs to be very fast because of a cache line bouncing problem.

On the other hand if I use RCU you know that makes my lookups full speed and my updates are relatively slow in the sense that I cannot reclaim memory immediately, otherwise updates are as fast as they could be; it is just the free that becomes low ok.

Similarly, Linux has what is called modules right. So, what are modules? Modules are fragments of code and data that can be loaded into the kernel at runtime. So, you can compile something and then you can load it at runtime and there are some interfaces it is fine. So, it can sort of start running in kernels space. So, those are called modules for example, you would run device drivers as modules.

It is you have a new device you basically attach it, your kernel does not have device driver for this you, you implement your driver as a module and you load it in the module will though the difference between a module and a process is that a module executes in privilege mode in kernel space you can do everything that the kernel can do that is all.

So, once again module modules are an example they are you will probably be doing lots of module read accesses where you say you know there is let us say some module table that looks up the whether it has this function or not and you are going to call that function. So, there are lots of read accesses and occasionally you are going to do module load and unload right. So, unload and load are very very occasional operations with respect to module accesses read accesses.

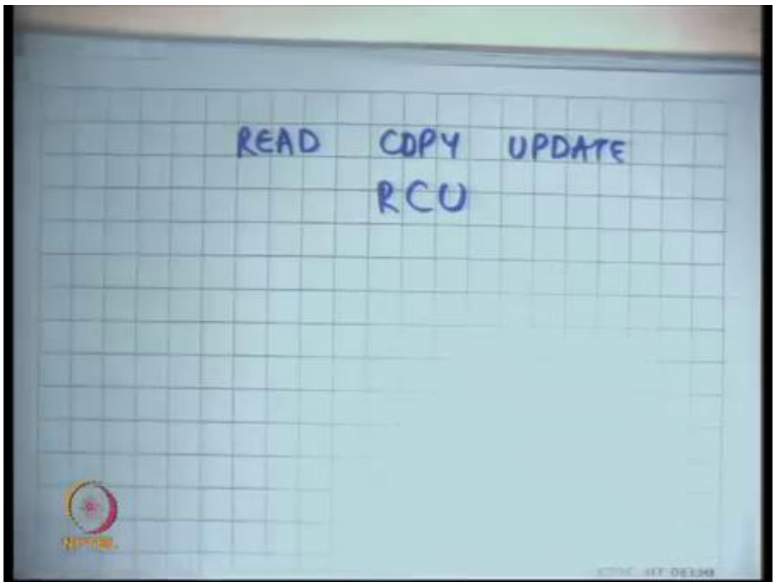
And so, here is another example where you could use RCU to execute your read accesses at full speed and yet synchronize correctly with your load and unload. Let us think about XV6 where does it make sense perhaps to use RCU instead of locks and if I was to use it, then what could I have how would I have implemented it?

Here is my here is one suggestion. Let us say you know there is some buffer cache entries that are accessed too much in read only mode and not so much in read write mode. So, let us say you know let us leave that question what where it can be used it can be used anywhere where you know there is a lot of read access and I am very few right access; let us talk about how you would implement it ok. So, everything else is I mean it is basically saying you use cache instead of raw logs you do not do anything when you are in read path.

The question is how do you decide what is the cyclical activity all right. So, in case of XV6 if you remember, you can probably say something like this that anytime a thread. So, first again same just like before if a thread is context switched out, you can probably say that it is not holding any references to share data structures because there was context which doubt it has voluntarily called the yield function right either because of the timer interrupt or something else. And you could probably I mean you could organize things in such a way it will never called yield without actually you know coming out of any access is of the shared data.

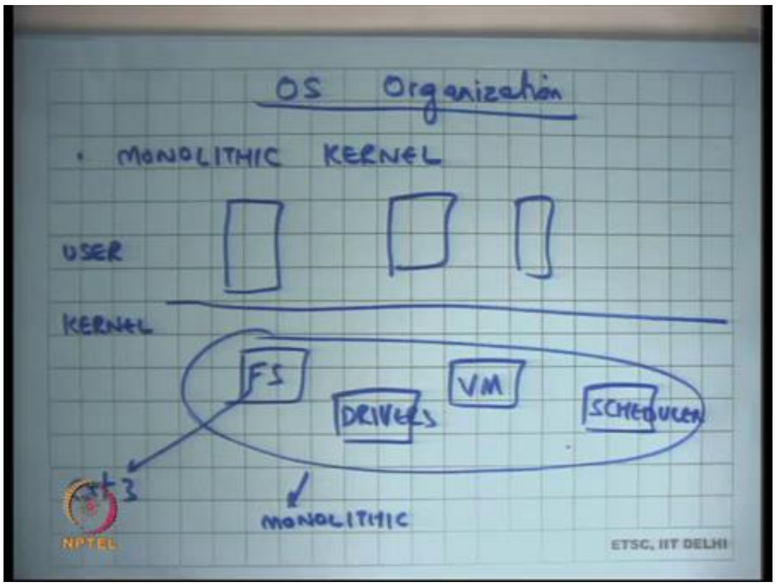
And then you can say that any time and yield internally would call let us say wait and so, you could you know synchronize that wait boundary. So, you could say a CPU goes through some computation and then wait and then some computation and then wait and so, any time the CPU calls wait you can be sure that is not holding any local references to shared data. Also, you could say if a thread if the CPU ever goes to user mode at that point, I can be sure that it is not holding any shared references to user data shared data local references of the shared data; good right. So, I have discussed read copy update.

(Refer Slide Time: 25:10)


READ COPY UPDATE
RCU

So, that was the it is commonly called read copy update or RCU for short all right.

(Refer Slide Time: 25:26)



With that I am going to move on to my next topic which is OS organization. So, far we had been looking at the Unix model of an OS and this model also called a monolithic kernel.

Student: Mono.

Why is it called a monolithic kernel? Well it looks something like this, there is let us say kernel space and there is user space and you know their user processes is which have separated at this face, but the kernel is one giant blob single address space and within this single address space, there are all kinds of module like file system, virtual memory, scheduler, drivers and so on ok.

So, this is one big sort of one big blob and that is why it is called monolithic ok. This is this by far one of the most popular models of an operating the organizations of an operating system at least till they stop in server systems and that is the model that you for example, used in modern operating systems like Linux windows BSD etcetera ok.

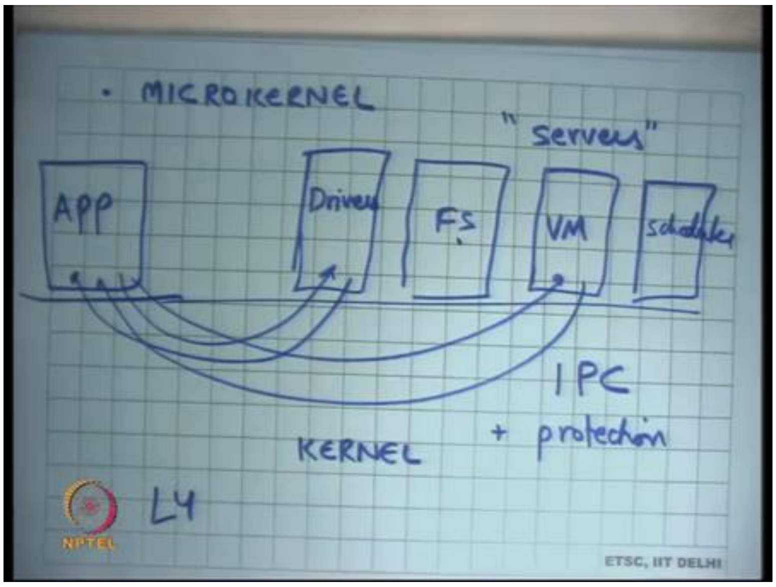
What are some bad things about this kind of organization? Firstly, because they are all sharing one address space and one protection domain; if there was a bug in a device driver, it can actually bring down all the other things right. If there was a security hole in your driver device driver, he could use it to look at your files he could use it to look at your virtual memory subsystem. So, there is very weak isolation between all these different components and so, it can you know gradually become really large this whole monolithic kernel can become really large especially the device driver because there is so many devices they become you know they become a problem.

The other issue is that performance has to be tuned you know you have to work for with an assumption that no so, there has to be some file system here right. So, let us say the file system that we discussed is ext 3. So, how do I choose whether ext 3 is my right file system or whether something else is the right file system? Well my choice will depend on what kind of application they are going to run on this file system wait.

So, I have to commit to a type of file system, or I mean of implementation of type system file system very early. I have to guess what is going to be run on this operating system at the time of OS design and that is not a you know it would have been better if a user had complete flexibility to say I am going to run a database application. So, ext3 is not the best file system that to have.

And so, you know here is my file system that you can plug into your operating system. So, you know there would be more sort of customization to applications in terms of performance and even functionality. If, but that is not possible in this monolithic model the best you can do is support a (Refer Time: 28:50) few different file systems like you know ext3 and tfs or something, but that is not; that is not flexible enough ok.

(Refer Slide Time: 29:05)



So, there is another model called the micro kernel which was made popular in the mid nineties and the idea here is that you would implement all these services file system virtual memory subsystem, scheduler, truck, drivers and anything else that the operating system provides. I will just save separate servers in separate address spaces ok.

And then there will be other applications that is running, and the applications could talk to these modules like this. The interface for talking to these modules or do these servers is what is called inter process communication we know about this IPC Inter Process Communication right and so, what the kernel is kernel becomes really thin all these different servers that were present is part of the kernel in the monolithic model are no longer part of the kernel.

So, kernel becomes really thin. All the kernel needs to do is implement abstractions that allow inter process communication. For example, they can just implement pipes and be done with it ok. And then there are all these different modules that are running all these different servers. So, that is a micro kernel. Of course, you know you could have privilege levels for example, you could say that the file system server can access your disk while the regular application cannot directly access the disk. The virtual memory server can access the physical memory, but you know other things cannot access.

So, you can implement. So, the kernel is basically IPC plus protection you know who can do what basically access control that is all. So, the nice thing about this is you know it gets rid of both the problem that I mentioned in the monolithic kernel. If there is a bug in your VM in your device driver, the only thing that goes wrong is a device driver; all your other parts of the kernel remain completely isolated because these are completely executing a different address spaces.

More importantly if I want to run a database application, I can you know choose my file system by just supplying code and you know doing some level of authentication to say that this code is trustable in the terms of access, you know letting it access your to my disk.

And so, you know I can use it. It sorts of also you know fits in with the principle of least privilege that we talked about I only need to give privilege to this particular server to be able to access the disk and nothing else. For example, it cannot directly access physical memory for example, ok. So, these are good advantages. What is the problem with this? Why is it not so popular?

Student: Slow.

It is slow right. If you know the application needs to talk to the virtual memory serves system, it needs to go from user to kernel, kernel to user, then user to kernel again and then back right. So, they are least poor user kernel crossings that you going to make as opposed two in the monolithic kernel case right. So, it is slower and there were lots of techniques that were proposed to make to implement fast IPC.

So, special interfaces were developed and the kernel and there are you know they micro kernels like L4 that implement fast IPC, but it is not; the idea has not caught on as much. It is caught on in specialized domains like embedded systems where protection is very important where reliability is very important etcetera where you could, where you could do this.

Student: Sir, if the virtual or the VM system was basically trying to provide an address spaces to other processes, but now it is itself a process.

Ok.

Student: So, as a how relate as an relate, we need to do bootstrapping.

So, the VM process.

Student: (Refer Time: 33:20).

Or the VM server was supposed to provide address spaces to other processes, now it itself is a process. So, it does not need to be some bootstrapping? Yes, I mean you know you could imagine that this some amount of bootstrapping that is going on and this VM process has a special address space you know which is equal to the size of the to the you know what maybe it is equal to the identity mapping between virtual address space and the physical address space. And then it has special privileges where it can actually make mappings and change the page tables of all the other processes right.

So, that is just protection and access control and you know this process is allowed to change the page table of that process. So, that is basically you know, but you know the nice thing is that different servers. So, I can come to my the operating system and say that has my application and I know that the best possible the best cache replacement algorithm for this application is not allow LRU, it is let us say MRU right.

So, I can just change my VM subsystem you know I can just plug in play and use MRU instead of LRU that is it right similarly I can use instead of logging I can use ordering or whatever you know. So, this plenty of choices what my cache replacement policy, what is my buffer cache replacement policy etcetera, what is my prefetching degree etcetera.

So, it can all tune it and completely configurable. On the other hand, if you use something like a monolithic kernel like Linux, the developer of Linux has pre committed you to a certain algorithms which is which may not be the right thing for you.

Student: Monolithic kernels used to have the (Refer Time: 34:54) all new drivers and so, can be the we are going to can be exchange into.

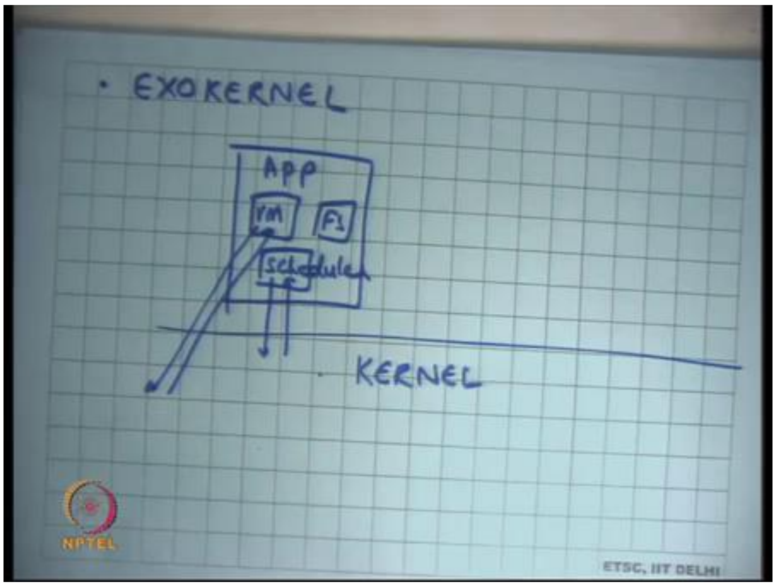
That is a good question. So, in monolithic kernels, there is an ability to install new drivers; for example, and the this ability to install new drivers is what is called through loadable modules. So, in modern kernels you have the notion of loadable modules that is what I was talking about. Loadable modules are nothing, but blobs of code and data that can be loaded in kernel space and they execute with the privileges of the kernel all right and cannot you just use the loadable module idea to plug and play other parts of the kernel like file system virtual memory subsystem and so on.

Firstly, it does not take care of the problem of protection yet right, even the loadable module has identical privileges to everything else. So, there is a bug in your loadable module, it collapse entire system and if you allow arbitrary things to be loaded loadable then you know you that that is the you know you are increasing your surface area of attack so, protection is not handled. And the other thing is that the loadable module interface needs to be very carefully designed. So, that it is rich enough to support all these different subsystems right.

As for example, you know at one extreme that is loadable module could just be behaving like the server in your microkernel where it is just you know the only way to talk to it is IPC, at another extreme you know the interface is so rich that you can actually you know directly you will make function calls to it etcetera instead of doing IPC.

And so, you know choosing that; so, in general while loadable modules have been used for drivers etcetera, they actually also used for file systems really. So, in some sense you know loadable module is somehow somewhere halfway between microkernel and a full monolithic kernel. It is really is giving you some advantages of micro kernels in some sense, but you know there is a protection problem that still sort of does not get solved ok; that is microkernel.

(Refer Slide Time: 37:15)



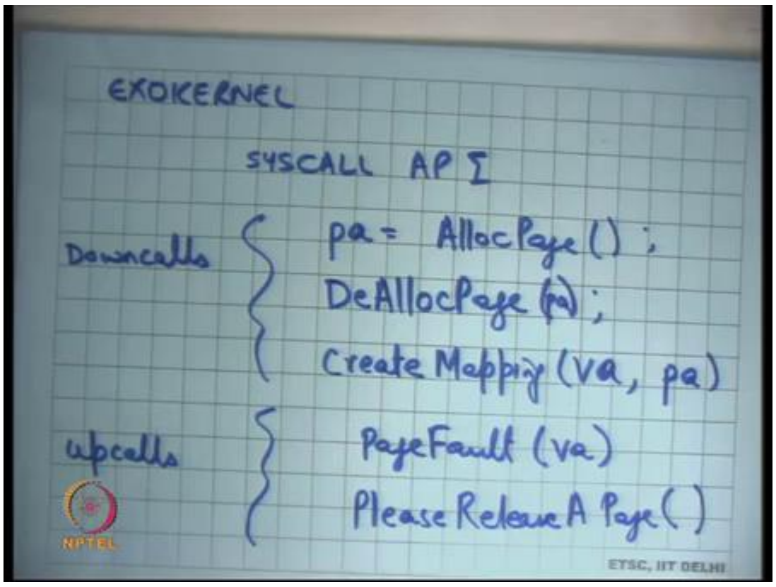
There was all two alternate organization called an EXOKERNEL that was proposed somewhere in late nineties. An Exokernel was also a microkernel except that it tried to move much most of the functionality of the operating system at the application level. So, here the idea is now let us say this is my kernel and this is my application earlier my VM and file system were living inside my kernel instead of that let all these subsystems live inside the application scheduler right ok.

But you know how can they; how can they do that? You provide interfaces such that these different modules can actually function right. So, you basically moving the layer of abstraction one level below ok. So, instead of a exposing a file system to the application, you will expose a raw disk to the application and let the application implement its own file system on top of it or instead of exposing an address space to the application, you expose physical memory and page table to the application.

And let the application choose what mappings it wants and what cache will got them it wants. Instead of saying that I am going to you have a scheduler, you have a virtual CPU that is uninterruptible. You export the CPU as it is and you tell the application, there is a timer interrupt you need to choose what you want to do you know you need to relinquish CPU.

And let the application decide what it want schedule and what when it wants relinquish CPU right. So, let me use a concrete examples to show this.

(Refer Slide Time: 39:25)



EXOKERNEL

SYSCALL API

Downcalls
- pa = AllocPage();
- DeAllocPage(pa);
- CreateMapping(va, pa)

upcalls
- PageFault(va)
- PleaseReleaseAPage()

NPTEL
ETSC, IIT DELHI

So, let us say I am using Exokernel example. My system call interface SYSCALL API looks something like this: physical at page is equal to AllocPage then there is DeAllocPage and then there is CreateMapping from virtual address to physical address DeAllocPages you know it takes an argument physical address. These are the down calls I am going to explain what down calls mean and then there are some up calls; up calls are PageFault on virtual address and PleaseReleaseAPage.

So, what I am doing is I am exposing API is to the application to allocate a physical page. So, the application is full control on physical memory or not full, but at least you know better control over physical memory you can say, I need one more physical page. So, it knows exactly what sets current physical memory footprint as opposed to your monolithic kernel where the application was completely oblivious of you know what physical memory footprint I actually have.

All the monolithic kernel application sees an address space. It is the operating system that is playing tricks under the carpet in implementing the data space sometimes it is mapping a virtual page to a physical page, sometimes it is mapping it to a memory mapped file and other times it is mapping it to a swap space right.

So, instead of that give full visibility to the application and let it say I want to allocate a physical page, you know do not play of tricks under the carpet at all; give it the control. And so, it can also say I want to DeAllocate this physical page and you could control completely tightly how much physical memory footprint it has and then it can create a mapping between it is virtual address to it is physical address. So, example if it wants to create multiple mappings to the same physical address aliasing, it is free to do that.

So, these are the down calls. By down calls I mean these are system calls that the application can request from the operating system and then there are some up calls; up calls are something like signals in Unix with the operating system can tell the application or ask the application to do something.

So, one other application, up calls is a page fault all right. So, a PageFault up call basically means that the application tried to access a virtual address that it is not currently mapped. So, it has not created a mapping yet for this particular virtual address. So, instead of killing the process or instead of doing things under the carpet itself, it just tells the application that look there is a page for that is occurred.

So, it converts the hardware exception into an up-call signal to the application. The application sees there is a; there is a; there is a page fault on this particular virtual address and that is because I have not created a mapping for it and because that is because I am implementing my own virtual memory subsystem.

So, what it may want to do is allocate a page, create a mapping and then re execute that instruction return from that signal right. So, it is giving full control to the application it can choose which page it wants to replace right as opposed to the operating system choosing on it is behalf, yes question.

Student: If it is not does not it calls lot of (Refer Time: 43:20) like preference the since this since my as is call India is now will be (Refer Time: 43:26) yeah beginning time if there will be (Refer Time: 43:28).

So, what about protection? Good question, what is what about protection? What are some bad things that can happen? Well what about how do I prevent an application from just doing alloc page; alloc page; alloc page till it completely exhaust physical memory? Well I can implement protection at that level I can say that you know this application if the I can implement quotas for every application and I can say that I would not give more than these many physical pages to this particular application.

So, it is you know it is perfectly fine AllocPage can you turn minus 1 to say that you know I did not I did not allocate a page for you and you deal with what with it yourself. So, that is your problem right. So, you can implement protection even at this level now right.

Student: Sir.

What about, question.

Student: Sir is actually to go (Refer Time: 44:14) each of them is dividing a different algorithm for VM. Now there is going to be a problem as in suppose one of them like if there is no consistency in the allocation (Refer Time: 44:29).

So, question is if each process implements it is own cache replacement algorithm, would not there be a problem? I do not see a problem right. Why is there a problem? You are just saying that what I have done is I have basically decided I will give this pool of physical pages to you that pool of physical pages to you, you can choose how you want to use this physical pool of pages. It is your decision; you can optimize based on your application which pages to bring in physical space and which pages to keep in swap space that is your decision all right.

So, multiple algorithms cache replacement algorithms can work simultaneously tune to your respective application. Let me give you a concrete example let us say I am a database and I am running the database as a user level application. A database will implement some kind of a cache for all the disk blocks right and so, this if this disk block cache is implemented in the virtual address space which it will be then you know the operating system has no visibility that this is actually cache pages.

And so, it will treat these pages are regular pages and it will keep swapping them in and out of the swap space, but that is a really poor thing to do because the database was using them those memory virtual address space pages as cache slots. And now the operating system ends moving cache slots and you know doing really hard work to maintain consistency of those cache or correct behavior of the correct contains of those cache blocks and moving them in and out of swap memory.

Instead if I had told the database server that I need some memory back from you the database would have known that these are my cache blocks and I can throw them at any time is. They just cache copies, I do not need to preserve their contents because the contains are already present in my disk blocks right. So, that way you know there is a you can save lots of extra copying between your database disk and your swap disk ok.

Now let us talk about the protection bit a little more. So, what is the other thing that can happen. So, one thing is you know an application just calling allocate page allocate page etcetera. Well it is an application is running happily it has lots of pages and then there are lots of other application that start simultaneously, and they also need pages.

So, now, at this point the operating system would want to take pages away from this already running application. In the monolithic kernel it was very uncivilized because I would just you know take pages from you and you would not even know and we said that is not always a very good thing right because the application you could you know application knows which pages to give. In this case, you will make an up call saying please lease a page. So, here it is a different thing you know being more civil you are saying please a page right.

Now, the application is going to decide which page to release. Application knows exactly how many pages it has and etcetera and it is running it some algorithm to say it is going to. But what if the application does not honor your please right you just say is you know the he takes you for a ride he takes the kernel for a ride. So, that is a protection problem I want a page for other applications. I am requesting this guy for a page, but he is not giving it to me. So, the solution to that is you know.

Student: Kill.

Kill or you could say you first you know you add one more up call here ForceRelease; ForceRelease the page right or you know just do whatever you were doing in you know monolithic kernels. You first ask the application please; you know wait for some time and also maintain some history on it is reputation or whatever.

And then you can if it is not doing what it, but you wanted it to do you could probably just take pages away from it. So, you know you could implement protection orthogonally to your common interfaces and so, I get both protection and performance at the same time. I will continue this discussion next time.