Real-Time Systems
Professor: Durga Prasad Mohapatra
Department of Computer Science and Engineering
National Institute of Technology Rourkela
Lecture 40: Unix as a Real-Time Operating System (Contd.)

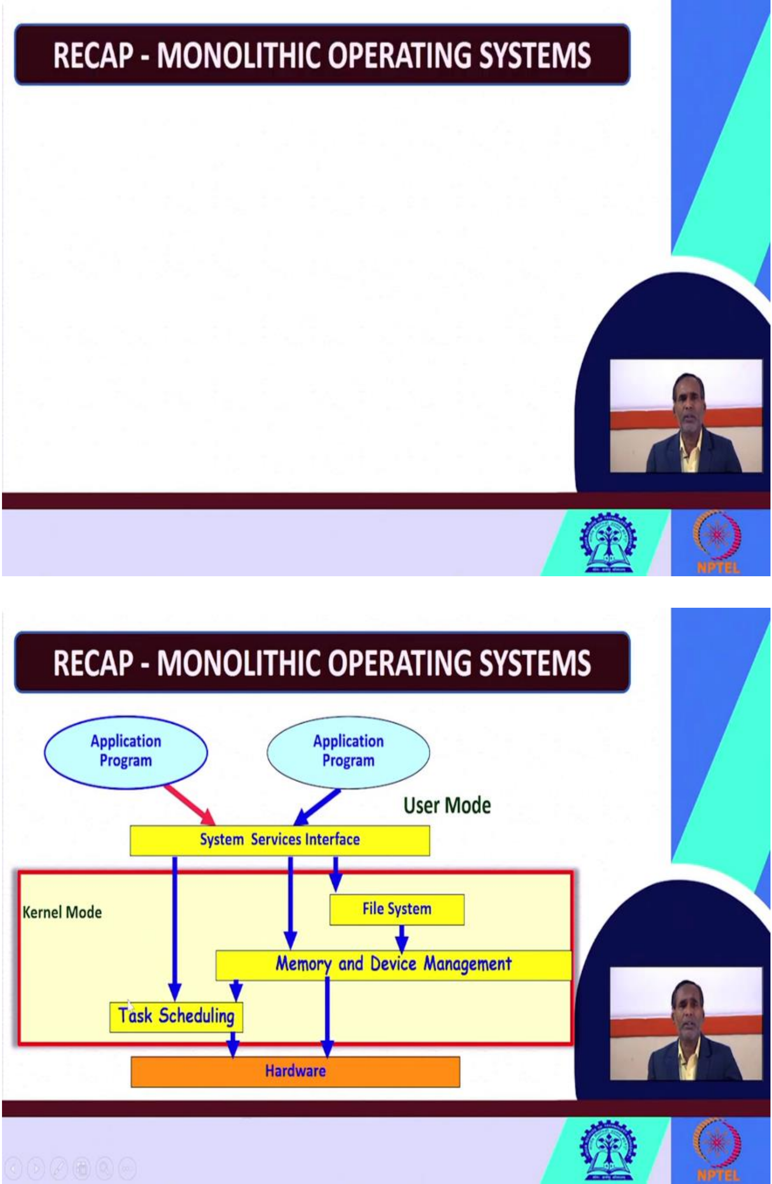
Good morning to all of you. Now, let us start where we have left in the last class. In the last class we were discussing about the basics of Unix. Now let us see, whether Unix, can it be used as a real-time operating system? Can we use Unix for real-time applications?

(Refer Slide Time: 00:37)

We will discuss about these monolithic operating systems, microkernel, we will just quickly review it. Non-preemptive kernel and this dynamic priority levels in Unix, and what are the deficiency of Unix to be considered as a real-time operating system. We will use the non-

preemptable interrupts, bast priority, jitter, device drivers etc. So, let us quick look at what we have discussed in the last class.

(Refer Slide Time: 01:02)

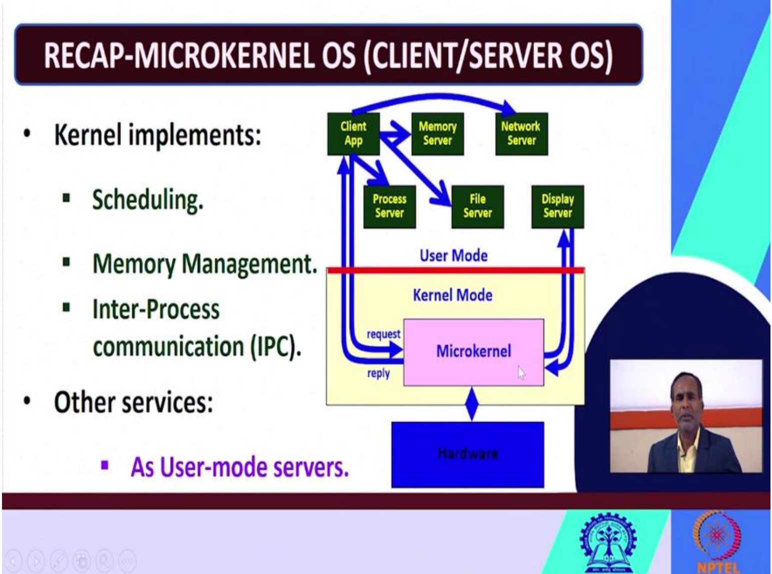


I have already told you, in monolithic operating systems, there are two modes, kernel mode and user mode. All the important activities or almost all of the activities or almost all of the tasks, they are processed, they are executed at the kernel level or in the kernel mode, such as, file system, memory management, device management, memory and device management, task scheduling, inter-process communication everything they are considered in kernel mode.

Then in user mode, you will look only write the application programs. These application programs in user mode, they can interact with the different services of the kernel mode through

system service interfaces. And the kernel mode in turn, it can what interact with the hardware. This is how the monolithic operating systems works. This is we have already discussed in the last class.

(Refer Slide Time: 02:00)



Next, we will look at quickly about this microkernel operating system or which is otherwise known as client server operating system. Here, I have a lead told you the kernel is very much tiny, it contains only the very important tasks, such as, scheduling, memory process, memory management and inter-process communication.

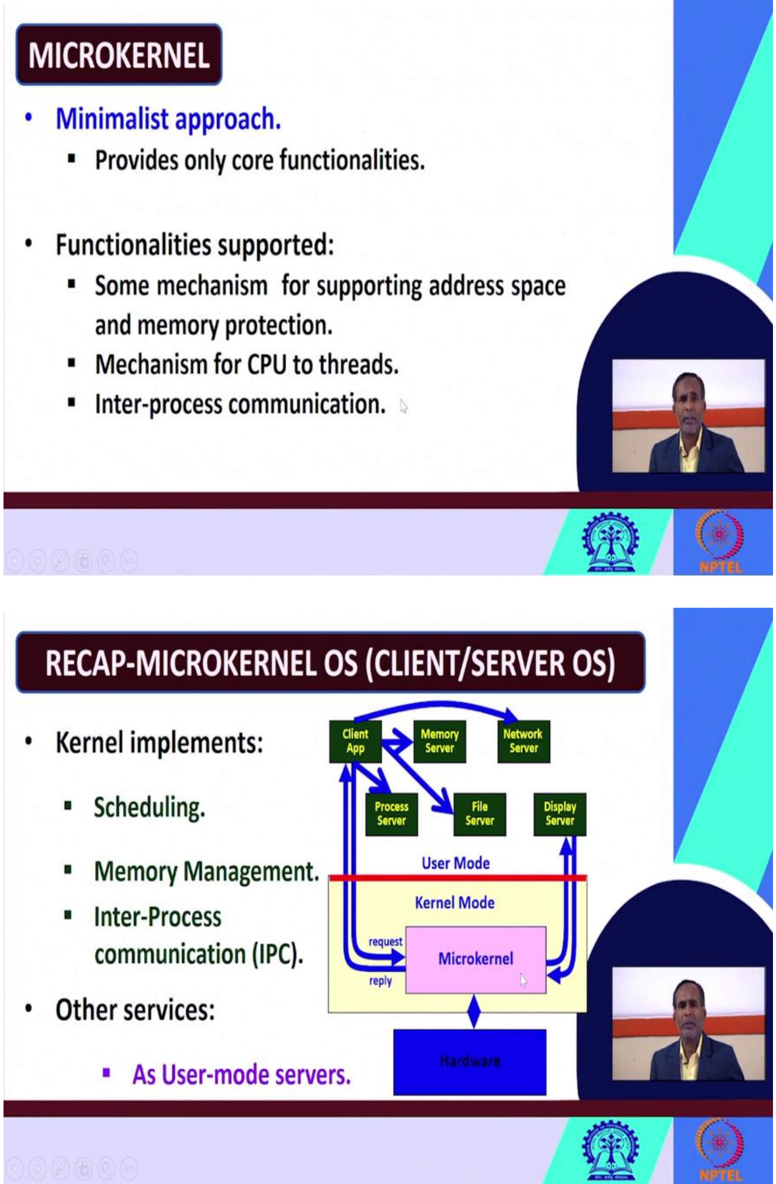
All other services they are performed as user mode servers, they are performed as, all other services accepting this they are executed as user mode servers. So, this also we have discussed in the last class. This thing I have already told you. So, you can see that, this microkernel is very tiny. This is the kernel mode the microkernel is tiny.

All the important activities almost all the other activities like your memory management and network, networking aspects, process management, file management, they are all performed at the user level or user mode, they are developed at the user model servers. And only the important what processes like scheduling, memory management, IPC they are performed in the kernel mode. And this microkernel interacts with the hardware.

So, in this way this client apps, it can send for a service as a request, it can send a request for getting any service and in turn microkernel will reply to that. Similarly, while you are displaying, display server it can send any request to the microkernel and then in response to

the microkernel will reply to that. This is how the microkernel operating system or the client server operating system it works.

(Refer Slide Time: 03:40)

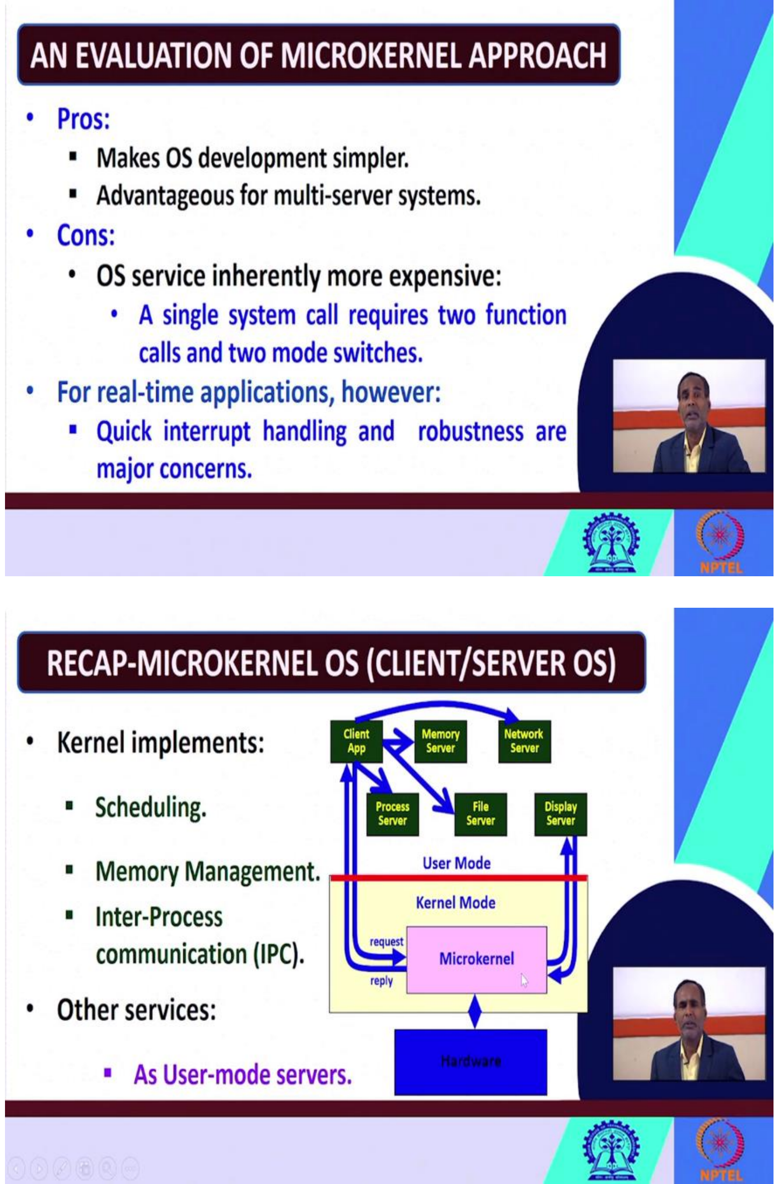


This I have already told you that, microkernel it uses a minimalistic approach, that means, it provides you only the core functionalities through this microkernel, all other services are provided as user mode servers, hence the size of the kernel is very small that is why it is portable, and I maintenance and the extension are quite easy.

The functionalities supported in case of your microkernel are as follows. It supports have some mechanism for supporting the address space and memory protection, they are provided. Then

mechanism for CPU to threads that he also provided. And what other we supported about IPC inter-process communication is also supported in this microkernel.

(Refer Slide Time: 04:29)



Now let us quickly, look at an evolution of the microkernel approach. What are the advantages and disadvantages. Advantage is that I have already told you. Its size is small, hence, it can be easily portable, it is easy to maintain, it is easy to extend. It makes the operating system development much simpler and it is advantageous for multi-server systems.

If you are going to develop some multi server systems, it is very much advantageous because you will see, we can represent all of these things in user mode using different server. So, it is

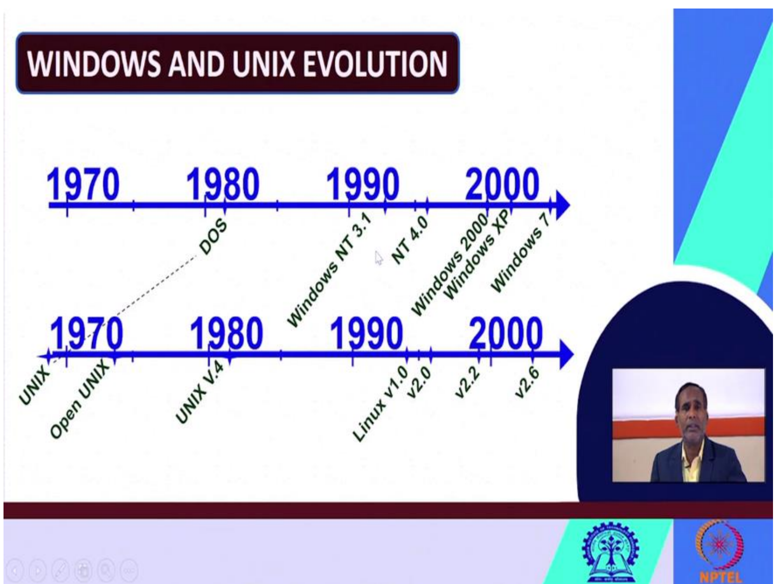
very much easier for advantages for multi-server systems. There are several drawbacks, such as, operating system service inherently more expensive.

To get the operating service OS service, inherently, it will be more expensive because a single system call it requires two function calls and two mode switches. You just see operating system service inherently more expensive. Why? Because when you are performing any single system call, it requires two function calls and two mode switches.

What are they? See, from the client app, you are making any system call. First, you have to send a request, then next time the microkernel will send you the reply. So, two function calls you have to make. Similarly, two more changes. So, once it will be changed from user to kernel and again, kernel to user.

So, a single system call requires two function calls, and two more switches. For real-time applications, however, quick interrupt handling and robustness are major concerns. Is it not it? For any real-time we require quick interrupt handling and robustness. These are two important constraints we require. So, these are the advantages and disadvantages of the microkernel approach.

(Refer Slide Time: 06:13)



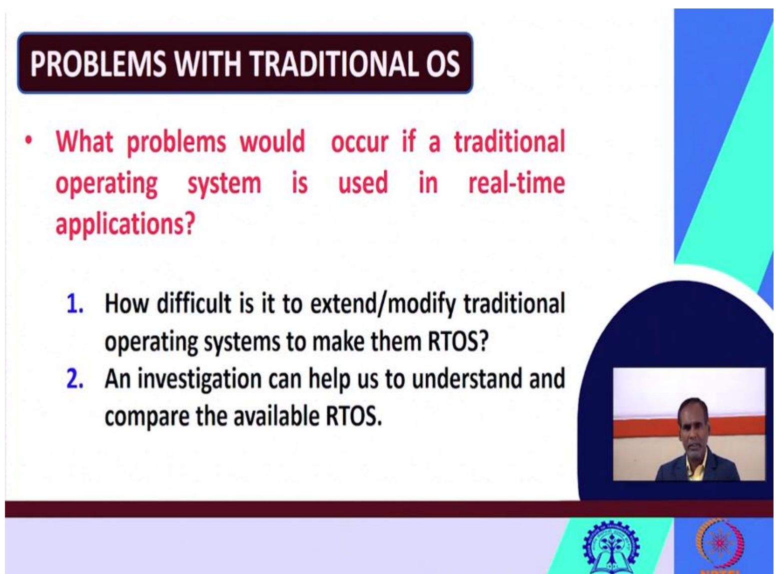
With this now let us start about we will first see, whether Unix is suitable for real-time applications or not. So, if not, what are the reasons. Before that, let us quickly look at the evolution of windows and Unix operating systems. So, this is partially what evolution I have

shown. UNIX was developed in the year of 1970, and then later on Open UNIX was made free, when UNIX was made as an open-source software.

In 1980 both the DOS was disk operating system was developed and in UNIX the UNIX verson 4 it came. Then later on what happened in 90 what happened, Windows NT 3.1 came. And here maybe in 92 or 3 something Linux version 1.0 came then version 2.0 came. And just before 2000 this version 2.2 came, and then, after 2002 then version 2.6 came. This is how this UNIX has evolved. Similarly, how DOS has evolved?

In 1980 DOS was invented, then in 1990 there Windows NT 3.1 yes, in 19 almost I think 90 or so towards that Windows NT 3.1 develop then NT 4.4 and the windows 2000 came. Then after 2000 the Windows XP came then Windows 7 and then Windows 10 those things, you know. So, this is how, the windows and UNIX these operating systema have been evolved. This shows the evolution of Windows and UNIX operating systems.

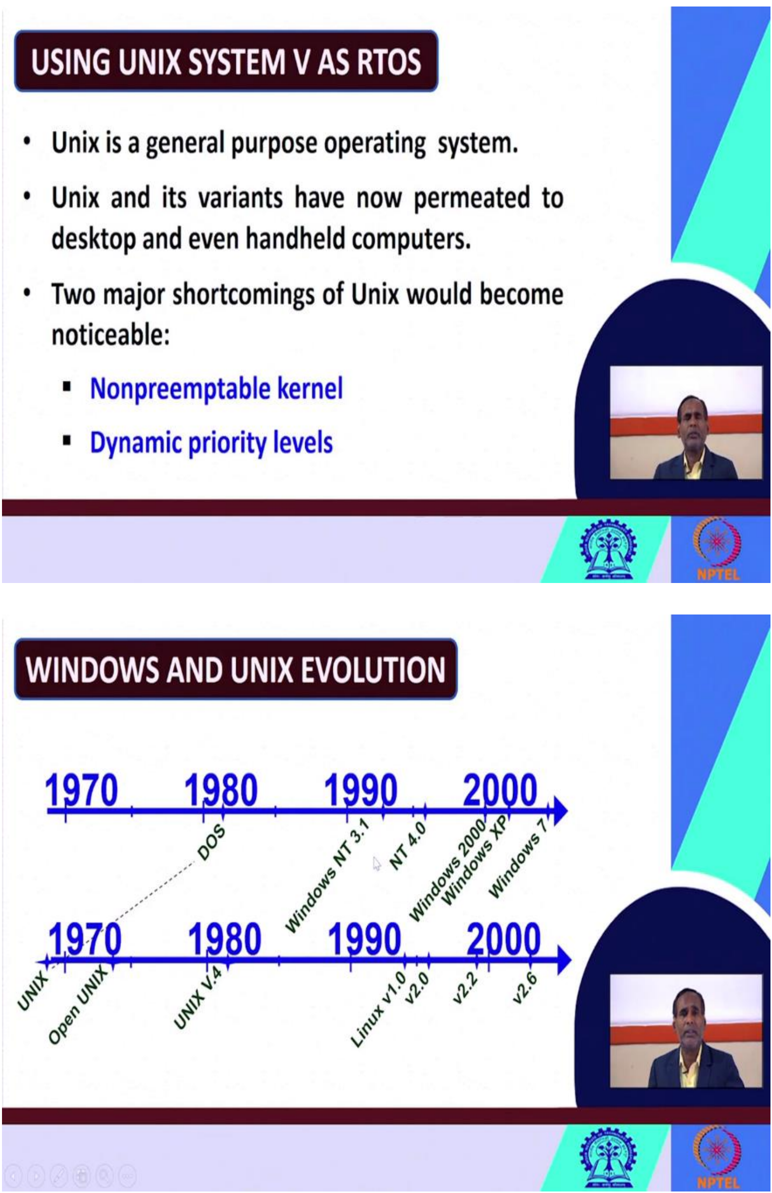
(Refer Slide Time: 07:59)



Now, let us go to the, our previous original topic. What are the problems with traditional operating system? Whether OS or whether UNIX operating system can be used as a real-time operating system? Let us first see, what problems would occur if a traditional operating system is used in real-time applications.

Here you will see, how much difficult it is, to extend or modify the traditional operating systems like Unix to make them real-time operating systems. So, this investigation will help us to understand and compare the commercially available real-time operating systems.

(Refer Slide Time: 08:39)

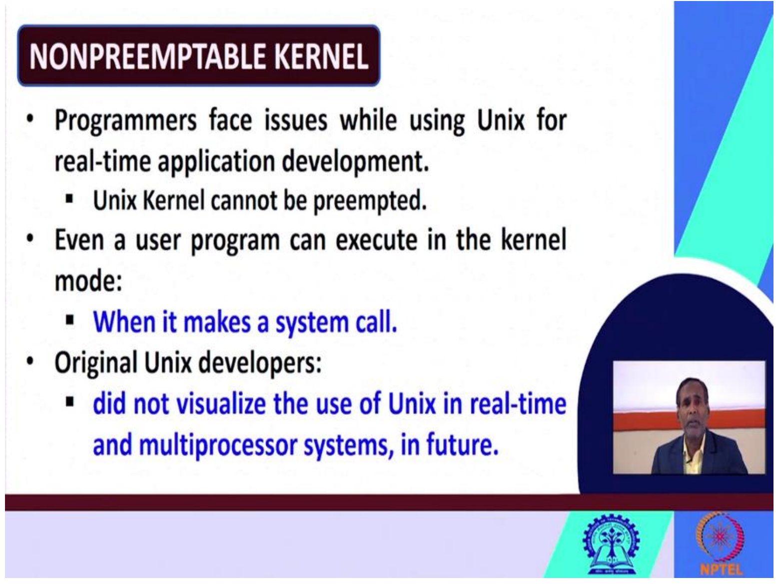


So, first see about whether you can use this Unix system V, as an RTOS, units based on a system view, and it has come Unix system V, something here, version 4 and then version 5 like this. So, we will see those things. We will see using Unix system V, as an operating system. You know, that the Unix is a general purpose operating system, we have already seen earlier.

The Unix and its other variants, the other versions they have now permitted to desktop computers and even to handheld computers like tabs etc. Actually, why we cannot directly use Unix as real-time operating systems, it is because of two major shortcomings. What are the two major shortcomings? The two major shortcomings of Unix to be used as real-time operating system they would become noticeable. These two shortcomings are as follows.

One, it is having a non-preemptable kernel, that means, the kernel it cannot be preempted, and two, it does not to have static priority levels. It does not use static priority levels. Instead, it uses dynamic priority levels. The priority is changing dynamically. So, it does not have static priority levels. So, these two shortcomings due to the presence of these two shortcomings Unix system is normally, it is, it cannot be used as a real-time operating system.

(Refer Slide Time: 10:09)

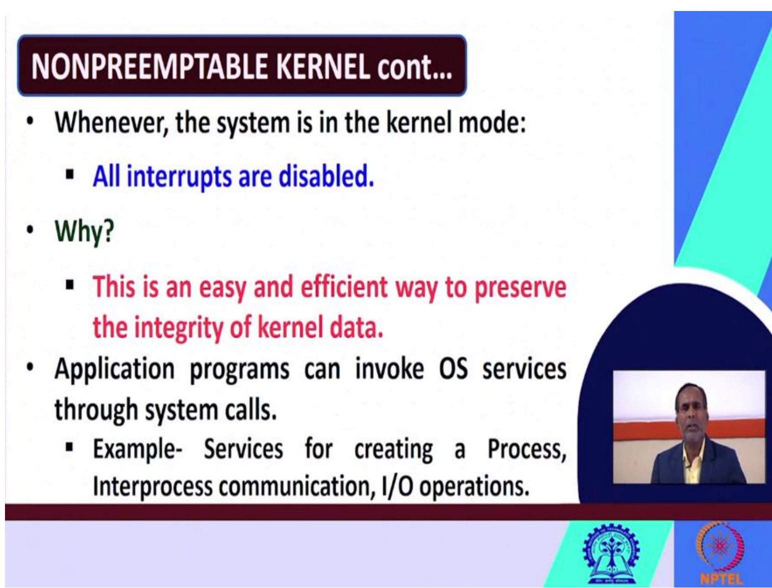


Let us say, the first short comings first. This is non-preemptive kernel, non-preemptable kernel. So, I have only told you Unix as a non-preemptable kernel What does it mean? That means, the kernel, it cannot be preempted. So, programmers they face various issues while using Unix for the real-time application development because of the fact that Unix kernel cannot be preempted. Then what will happen?

So, even a user program it can execute in the kernel mode. So, even if a simple user program it can execute in the kernel mode when it makes a system call. So, whenever a user program it makes a system call it can execute in the kernel mode. So, then why the original Unix developers have done this thing? Why they have made it non-preemptable?

The original Unix developers they cannot visualize the Unix of Unix in real-time applications or in multiprocessor systems in future. Somehow they cannot visualize it, and they have made the Unix kernel non-preemptable.

(Refer Slide Time: 11:17)

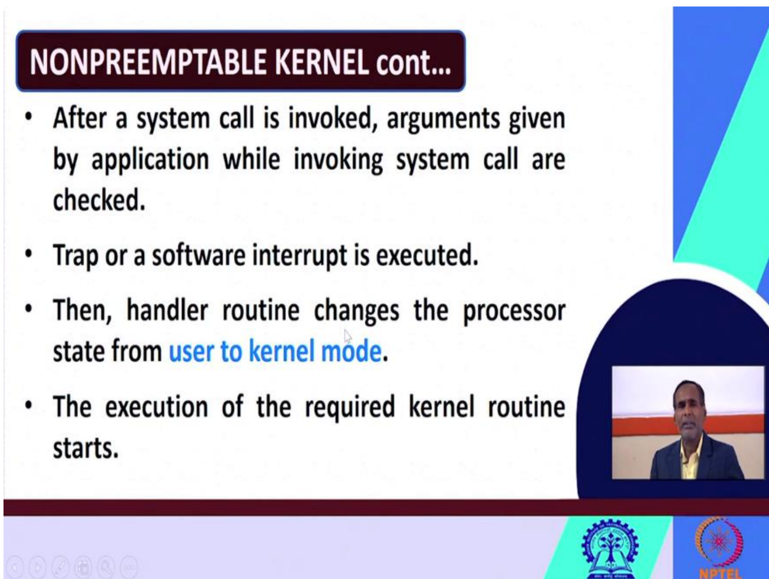


Whenever the system is in the kernel mode, what happens, automatically all interrupts are disabled that is why it is not preemptable So, why the Unix operating system is non-preemptable? So, whenever the system is in kernel mode, automatically all the interrupts are disabled. Why? So, because, why the Unix developers they have done, they have made it non-preemptable? Why they have made all the interrupts disabled?

Because they thought that this is the easiest way, this is the efficient way to preserve the integrity of kernel data. In order to preserve the integrity of the kernel data, they thought, this is easy and efficient way that is why they have led all interrupts visible and the kernel is non-preemptable. The application programs, they can invoke different operating systems services through system cost, that we know, is it not it?

The application programs, they can invoke the different operating system services through system calls. For example, services for creating a process, and services for interprocess communication, I/O operations, these are different services, and the application program it can invoke the operating system services through different system calls.

(Refer Slide Time: 12:32)

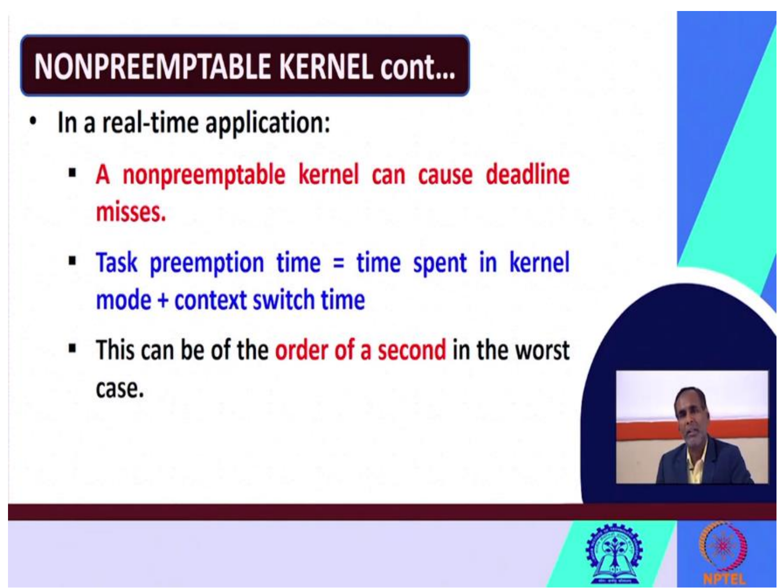


Now, after a system is invoked, arguments given by the application while invoking the system called are checked. So, after a system call is invoked, what happens, the arguments given by the application while invoking this system calls the are checked. Now, the trap or a software interrupt is executed. So, after the system calls in invoked, so, what we do? The arguments given by the application or reject, trap or a software interrupt is executed.

So, a software interrupts is executed. Then the handler routine changes the processor state from user to kernel mode because you are using what the system call. So, then, when, once a software interrupt executed then the handler what it does, the handler routine changes the mode, changes the state from what state to what state, from which mode to which mode, the handler routine it changes the processor from user to kernel mode.

Because initially, the system call was invoked then the software interrupt is executed. Then the handler routine it changes the process state from user mode to kernel mode. The execution of the required kernel routine it starts. So, after changing the processor state from user to kernel mode then the execution of the required kernel routine, it starts.

(Refer Slide Time: 13:57)



In a real-time application what happens a non-printable canal can cause a deadline misses. I am discussing, why the traditional Unix system, Unix operating system, it cannot or it is not suitable for real-time applications. This is the important reason. I have already told you, the Unix has a non-preemptable kernel. And in a real-time application when you are using a non-preemptable kernel a non-preemptive kernel it can cause the critical tasks to miss the deadline.

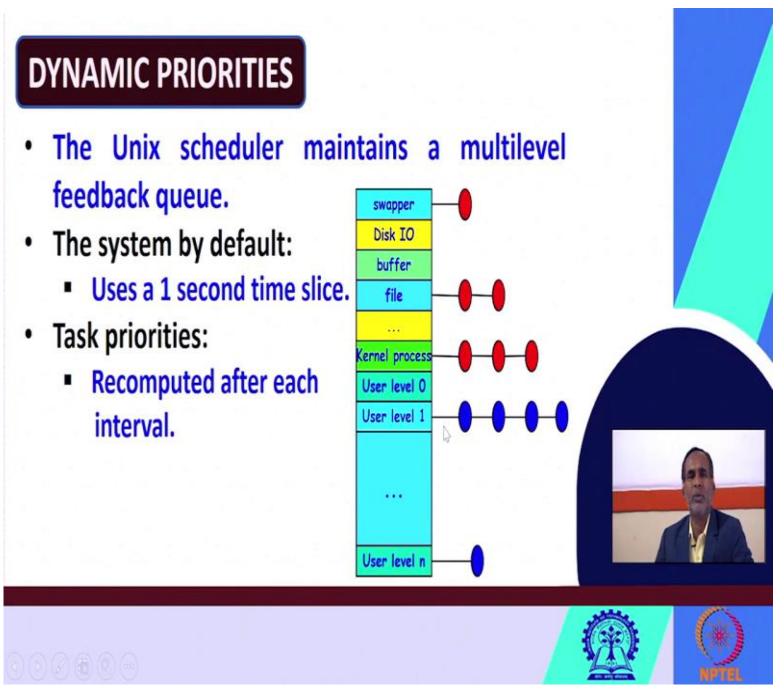
And how you compute the task preemption time? Task preemption time is equal to time spent in the kernel plus the context switch time. And these preemption time maybe, of the order of a second in the worst case. See, it can be at the order of a second, but you know, the critical task, the critical real-time tasks, they are of the order of some few milliseconds.

And if this maybe the user applications they are running and they are of the order of seconds, and you cannot preempt, then what will happen? Obviously, these critical tasks which are of the order of some millisecond, they will definitely miss the deadlines. So, that is why, due to the presence of the non. Had it been the case it could have been preempted you can easily run the critical tasks.

So, since now, you cannot preempt whatever tasks is running even if a user application, so you cannot preempt because it is a non-preemptable kernel. So, due to that what will happen? The critical tasks it has to wait, which duration is of the? Which deadline is of some or of the millisecond?

So, there is a heavy chance that the critical tasks may miss the deadline because these what applications which is currently running they can be order of some second, and the duration of the deadline of the critical tasks may be of some of the milliseconds. And hence, while it will wait for the task that is currently running, since it cannot be preempted, and each order is of its duration is of the order of seconds so by that time, the critical task may miss the deadline because its deadline may be of the order of some milliseconds.

(Refer Slide Time: 16:16)



So, then, we will come to this next problem that is the dynamic priorities. So, what will happen here? The Unix scheduler normally maintains a multi-level feedback queue. So, the unique scheduler, what does, how it maintains the different processes? The Unix scheduler maintains a multi-level pediatric queue, and it stores the processes they are in. The system by default uses a one second time slides.

So, the system by default it uses, it uses the different time slices, and the time slices are up for what, magnitude one second. The system by default uses a one second time slice. These task priorities they are re-computed after each interval. So, the priorities in Unix what is happening, the priorities of the different tasks they are re-computed after each interval. So, after each interval the task priority is re-computed.

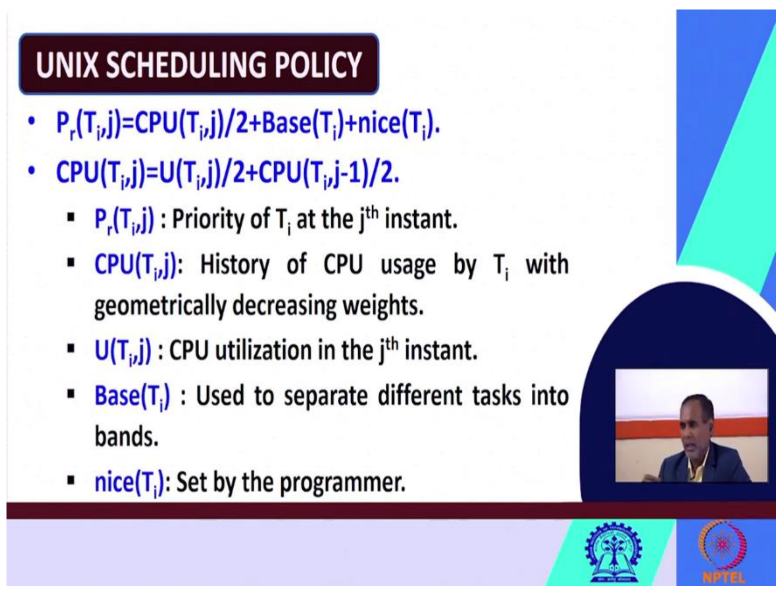
So, as I have already told you here, this diagram. Well, this one I have earlier shown in the previous class. Here, there as you can see, two modes are there. So, one is this user mode

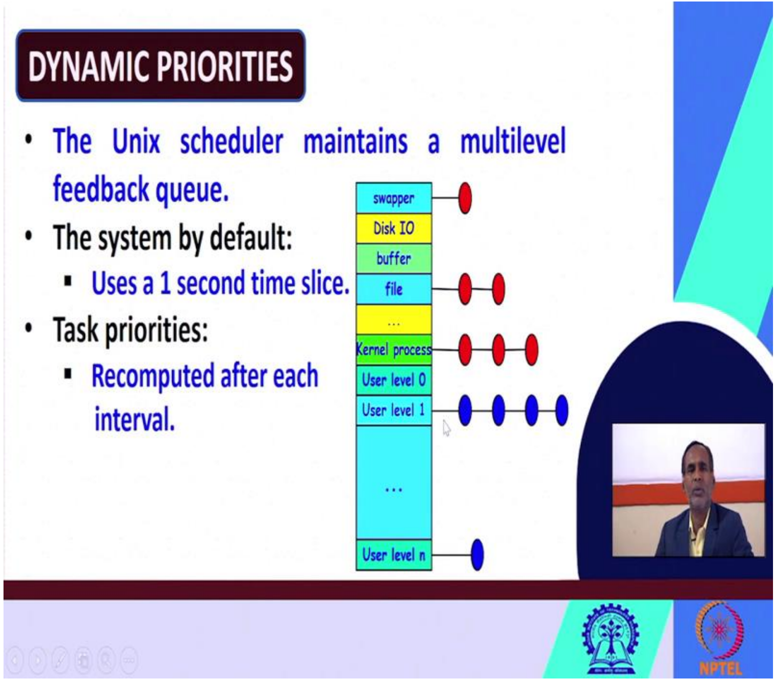
another is the kernel mode. And the priorities we give in user mode are what? For the kernel mod the priorities are 50, 0 to 49, and for the cardinal mod is rest 78, that means, 50 to 127.

So, these priorities, that means, for kernels the priority is 0 to 49 and the priority levels for the user mode are 51 to 127 that is 78 numbers. These priorities, they are recomputed after each interval. These are the processes, they are stored in a multilevel feedback queue. Actually, there must be a feedback from these and again it will come to hear.

Again, from this these are the processors. Again, from the last one, a feedback path will come to the first one. In this way, the Unix scheduler maintains a multi-level feedback queue. You can see the period from the book.

(Refer Slide Time: 18:05)





Now, let us say, how scheduling policy works? So, as I have already told you here, that the task priorities are recomputed after each interval. Please recall in the last class, I have already told you, the task priorities are recomputed by using an CPU usage factor. So, that now I will show you, how the task priorities are computed after each interval by using a CPU usage factor. Let us see, how we do it.

So, the task priorities are recomputed after each interval using the following equation. The priority of task Ti after the jth instant is computed as CPU(Ti,j)/2 +Base(Ti) + nice(Ti). So, what is CPU(Ti,j) refers or it is the history of the CPU usage by task Ti with geometrically decreasing weights.

So, recursively you can find out this also here. And what is UT, what you can say U(Ti,j). So, and first we have seen that priority of Ti, j = CPU(Ti,j)/2 +Base(Ti) + nice(Ti)., where Pr(Ti,j) = priority of task Ti at the jth instant. CPU(Ti,j) is the history of CPU usage by task Ti, with a geometrical decreasing weights, the weighs are geometrical decreasing. And what is base, base (Ti) it is used to separate different task into bands.

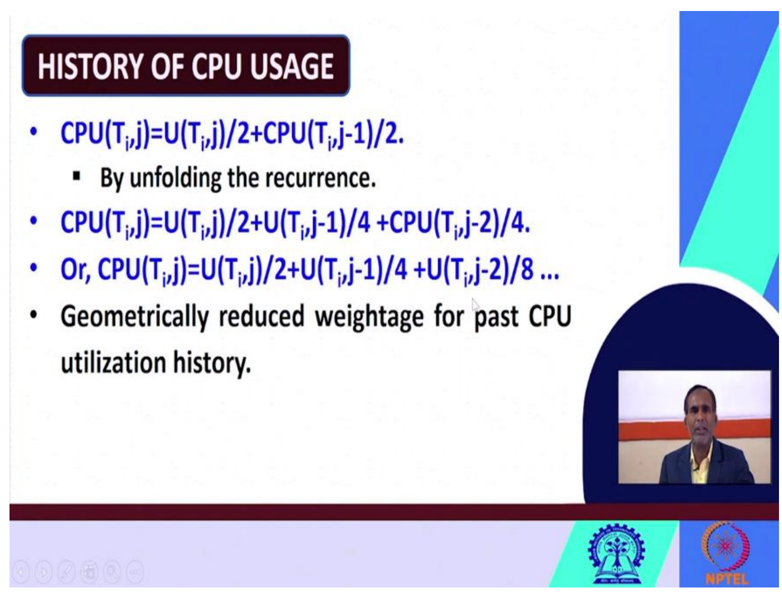
So, you say, these are the different tasks, they are separated into different bands. So, base(Ti), is used to separate different tasks into different bands. And what is nice(Ti)? So, nice(Ti) normally it is set by the programmer. And then you can say that, I am recursively evaluating these terms, CPU(Ti,j).

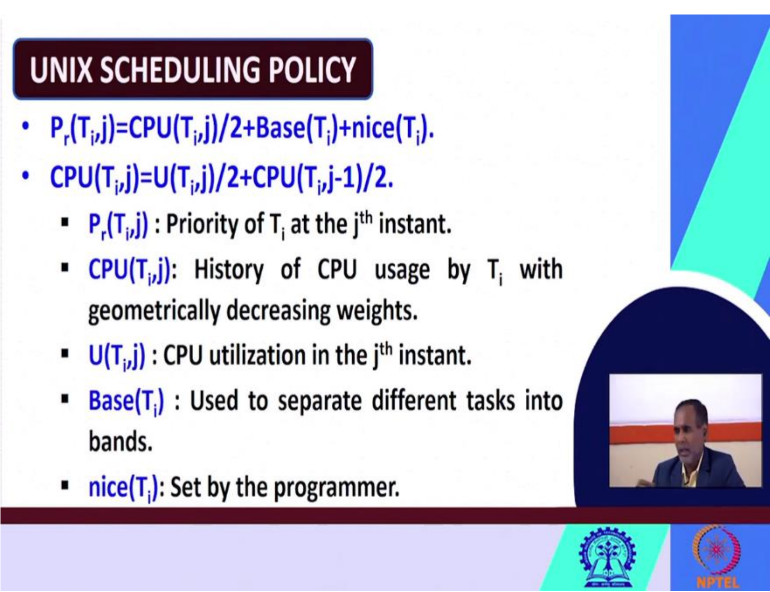
So, CPU(Ti,j) it can be computed as U(Ti,j)/2+ CPU(Ti,j-1)/ 2. So, recursively I am finding out the value of CPU(Ti,j). So, what is U(Ti,j) here, U(Ti,j)as its name suggests this is the CPU utilization, when, in the jth instance. So, CPU(Ti,j), it represents CPU utilization of task Ti in the jth instant.

So, in this way, the scheduling policy works, the tasks by that is they are recomputed after each interval by using this formula on that Pr(Ti,j) that is priority = CPU(Ti,j)/ 2 + base(Ti) + nice(Ti) for Pr(Ti,j) is this priority of Ti are the jth instant CPU(Ti,j) is the history of the CPU usage by Ti with geometrical decreasing weights, and U(Ti,j) you have already seen, this is the CPU utilization, when, of the task Ti in the jth instant, and base Ti, it is used to separate the different tasks into bands are shown in here.

You will see, that different tasks, they are separated into different bands, and a nice(Ti), it is set by the programmer. So, this is how the priority is computed for the different tasks after each interval. Using the factor, what, CPU utilization.

(Refer Slide Time: 21:38)



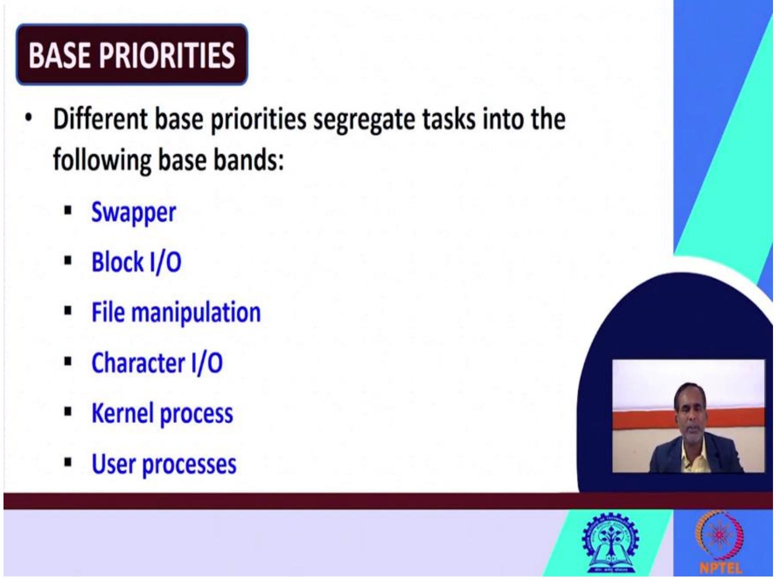


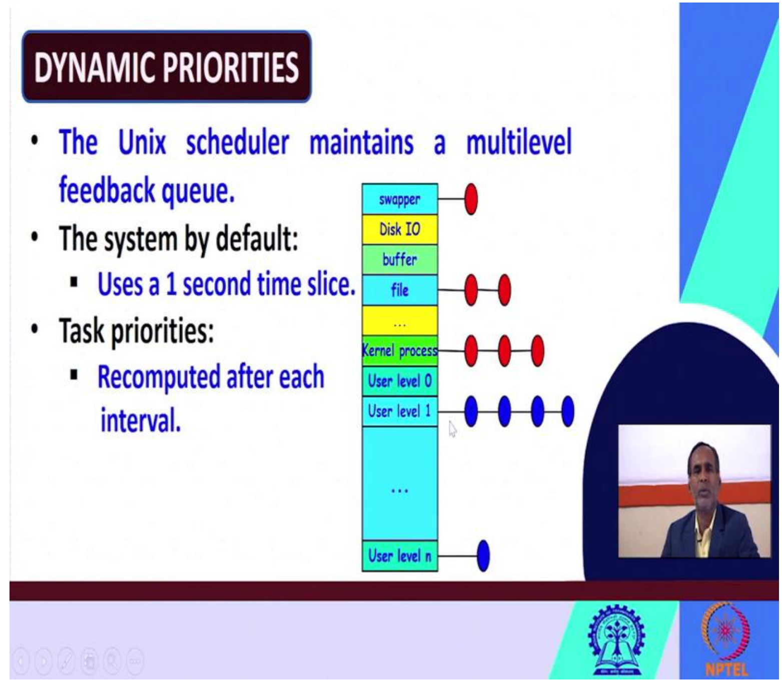
Now, history of CPU usage, Now, it can be computed, I have already told you, this factor CPU(Ti,j) you can recursively find out. So, CPU(Ti,j) can be found out by using this equation, which already we have shown here. This is equal to U(Ti,j)/2 + CPU(Ti,j- 1) / 2. Similarly, this CPU(Ti,j) you can put the value here like U(Tij) / 2 + this, evaluate this one or the expand the CPU(Ti,j -1) / 2 this will be now CPU(Ti,j-1)/ 4 + CPU(Ti, j – 2) / 4 or again you can again expand it.

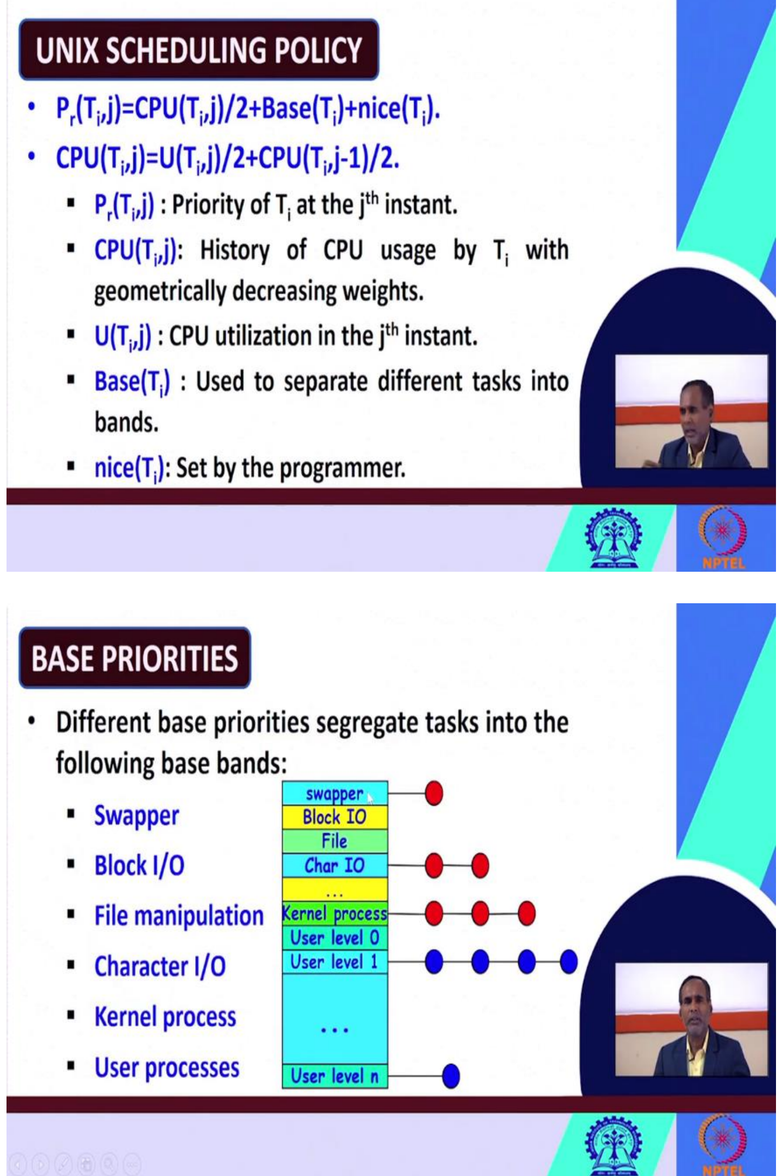
So, this will be equal to. So, just recursively expand this. So, the first term is remaining same, second term also remaining same, then the third term you can divide that CPU plus U have what (Tj, j – 2) / 8 plus like this. So, this will be just recursively evaluate. So, you can see that the geometrical reduced weightage for past CPU utilization history.

So, here, it is shown the geometrically reduced U weightage, for what, for the past CPU utilization history. So, using this equation you can compute the, you can recursively compute the priority of task Ti at the jth instant periodically. So, the task priorities are recomputed after each interval using this equation. So, this is happening. So, the priority is recomputed in this what Unix operating system using this equation.

(Refer Slide Time: 23:17)



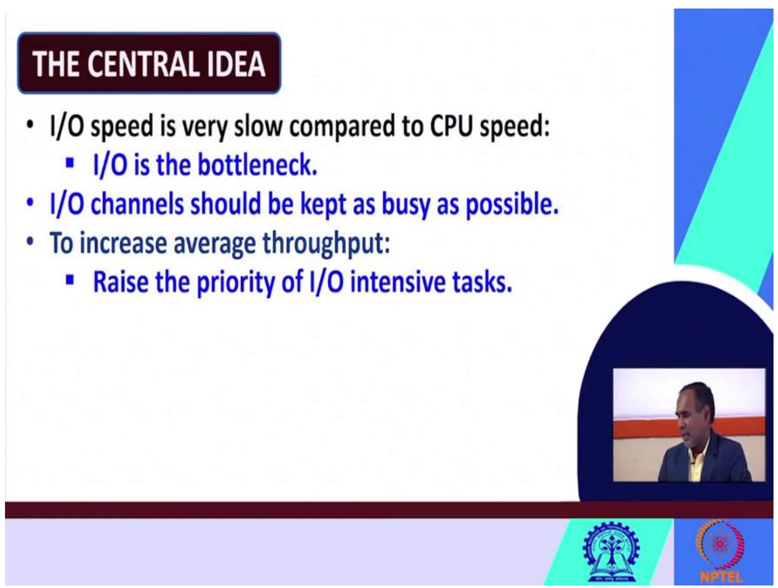




Now, what are the best priorities? I have already told you that the different base priorities, the segregate task into the following basebands. This I have already told you, these are the base priorities swapper blocker IO, here we have seen swapper Disk IO, buffer, file etc. So, and this can be how segregated I have already told you this can be segregated by using base(Ti), Base (Ti) is used to separate different tasks into the bands.

This I am telling here. The different base priorities they segregate what, they segregate the tasks into the following best bands. The different base priorities they segregate the tasks into the following basebands. The basebands could be swapper block, IO, file manipulation, character IO, kernel process, user processes, as we have shown in this diagram. So, these are the different basebands. This already I have shown here.

(Refer Slide Time: 24:11)

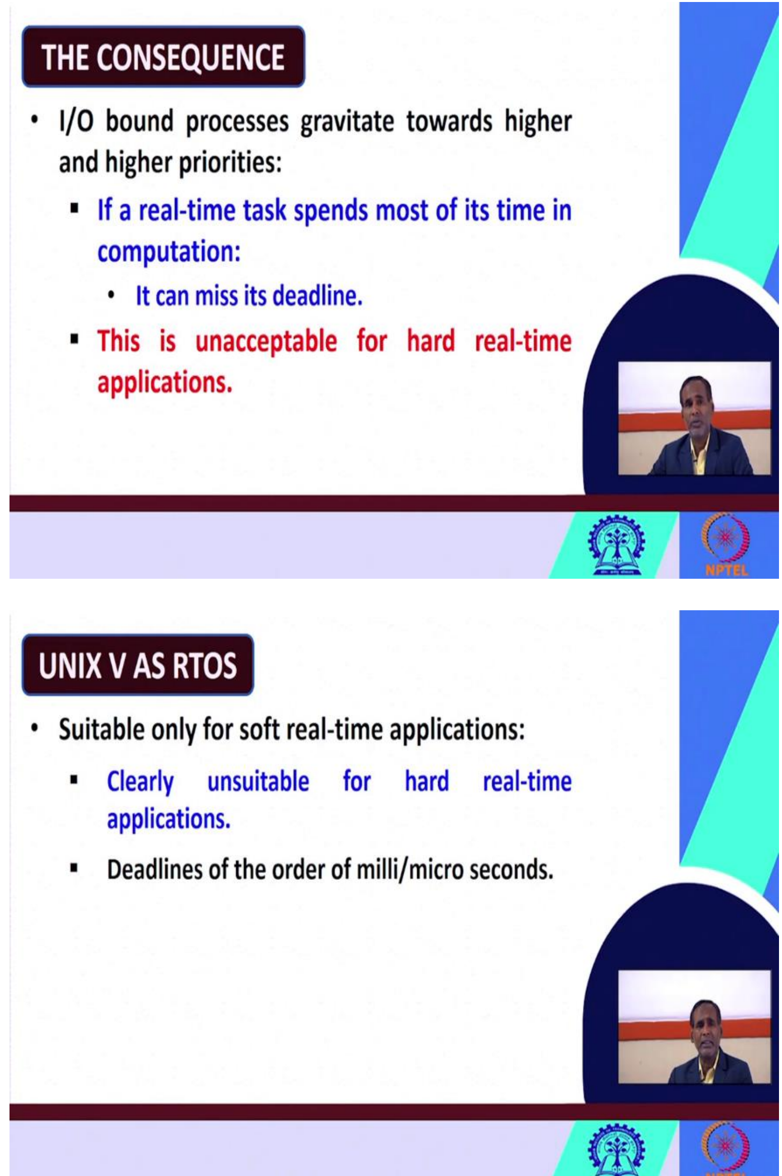


Now, let us see, the central idea. Why there is a dynamic priority changing? Why the priorities are dynamically changing? You know, in you know normally the I/O speed is very much slow compared to the CPU speed, very general knowledge you have known earlier. I/O speed is very much slow in comparison to the CPU speed. So, the I/O operations I/O is the major bottleneck.

So, what should be the objective? The objective should be that, the I/O channel this should be kept as busy as possible, and this will help you to make the system busy. The I/O channel should be kept as busy as possible. To increase the available throughput, you have to, what we have to do? How can we increase the average throughput? Because see, CPU is very fast and I/O is very slow. So, CPU might be waiting several times for I/O.

So, to increase the average throughput, we have to raise the priority of the I/O intensive tasks. So, in order to increase the overall throughput, the average throughput, we have to keep busy the I/O channels. How we can keep busy the I/O channels? We have to raise the priority of the I/O intensive tasks. So, we have to increase, we have to raise the priority of the I/O intensive tasks.

(Refer Slide Time: 25:29)



So, the I/O bound processes the gravitate towards higher and higher priorities. So, normally what we have to do, the I/O bound tasks, the I/O bound processes they gravitate towards what, higher and higher priorities. We have to assign higher and higher priorities to the I/O related jobs. So, if a real-time tasks spends most of its time in competition what it will happen, it can miss this deadline.

That is the objective, why we want to put or we want to assign higher priorities to the I/O bound processes, otherwise, if a real-time tasks, it spends most of its time in computation, it can miss a deadline. So, this is unacceptable for the real-time application. So, in the real time applications, the tasks should not miss its deadline.

So, in order to see that these tasks they are not missing the deadline, and so that is why in the Unix the I/O operations they are given high priority. So, the I/O bound processes the gravitate towards higher and higher percentage. If real-time tasks, if you are using real-time tasks, if real-time tasks spends most of its time in competition, then it can miss its deadline. This is very much unacceptable for hard real-time applications.

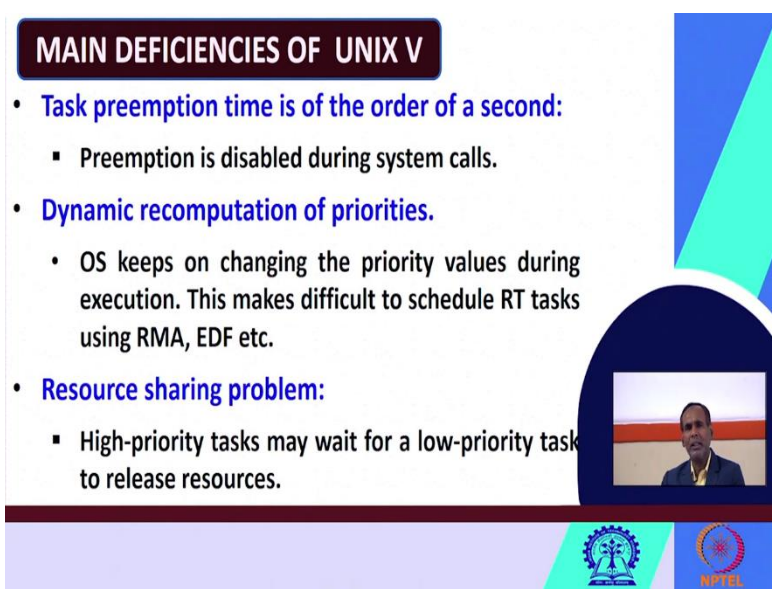
So that is why, we have seen that Unix, it cannot be or it is not suitable for real-time applications, due to these two problems. One, due to the fact that it has a non-preemptable kernel and the dynamic priorities are changing. It has a dynamic priority level. The priorities are not static, the priorities are changing dynamically. So, this Unix version V, Unix V, it can be used RTOS only for the real-time soft real-time applications, not for hard real-time applications.

It may be suitable only for soft real-time applications. It is clearly unsuitable for hard real-time applications. Because if it will be used for hardware applications, the consequence the critical tasks, they will miss their deadlines. So that is why here what it is suitable only for soft real-time applications. It is not suitable for hard real-time applications.

The deadlines are of the order of milli or microseconds. Because you know, in hard real-time applications, the critical tasks their deadlines are of the order of milli or microseconds, which can be easily missed, because these priorities are changing and always it will give the priority to these I/O related jobs, which maybe of the order of some seconds by the time these critical tasks they might miss their deadlines because their deadlines are of the order of milli or microseconds.

So, this is how we have seen that Unix V is suitable only for soft real-time applications, not for hard real-time applications.

(Refer Slide Time: 28:21)



Now let us see, the other differences of Unix V to be used, as a real-time application. So, number 1, the task preemption time is a order of a second. So, in case of this Unix, the task preemption time is the order of a second. The preemption is disabled during system calls, this I have a lead told you because it has a non-preemptable kernel, so the preemption is disabled during system calls.

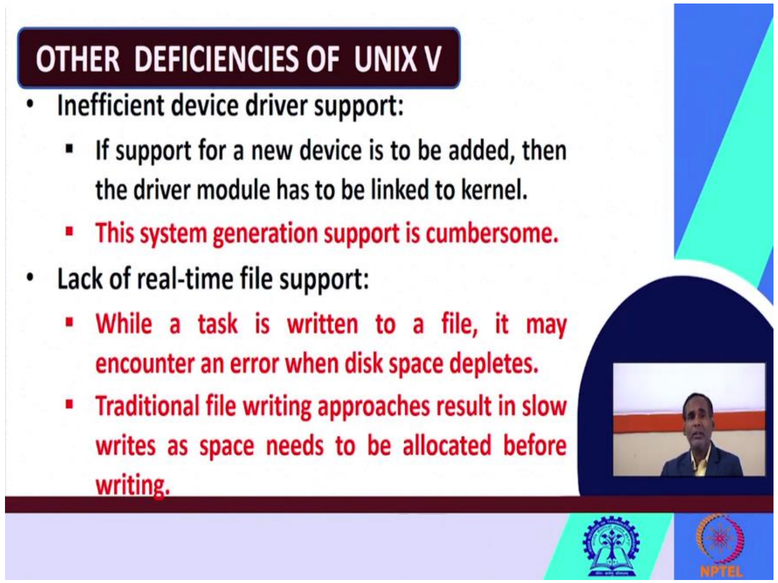
And the task preemption time in the order of a second, while you will see, these critical tasks they in a hard real-time system the critical tasks, their deadline is of the order of sub millisecond or microsecond, so it is heavy chance that the critical tasks will miss their deadlines. Number 2, dynamic re-computation of priorities.

I have already told you, Unix does not have a, Unix does not statically assign the priorities to the tasks, it dynamically assigns the priorities. There is a dynamic re-computation of priorities. The operating system keeps on changing the priority values during the execution, it gradually changes the priority values. And since the priority values are changing due to this fact, this makes difficult to schedule the real-time tasks using RMA or the EDF.

Because you know in RMA on the EDF, it needs that, the task should of what static priorities. Once, the priorities are assigned they should not be changed by the operating system. But in Unix, it is dynamically assigned. So, dynamically the priorities are changing. So, this will make difficult to schedule the real time tasks using our real-time task scheduling algorithms such as RMA and EDF.

So, another resource sharing problem, another resource sharing problem. So, high priority tasks may wait for the low priority tasks to release the resources. So, resource sharing you have already known in earlier classes. So, in Unix it might be possible that some of the high priority tasks they may have to wait for a low-priority tasks to obtain or to release the resources. So, this under problems. So, these are the main deficiencies of Unix V to be used as a real-time operating system.

(Refer Slide Time: 30:29)

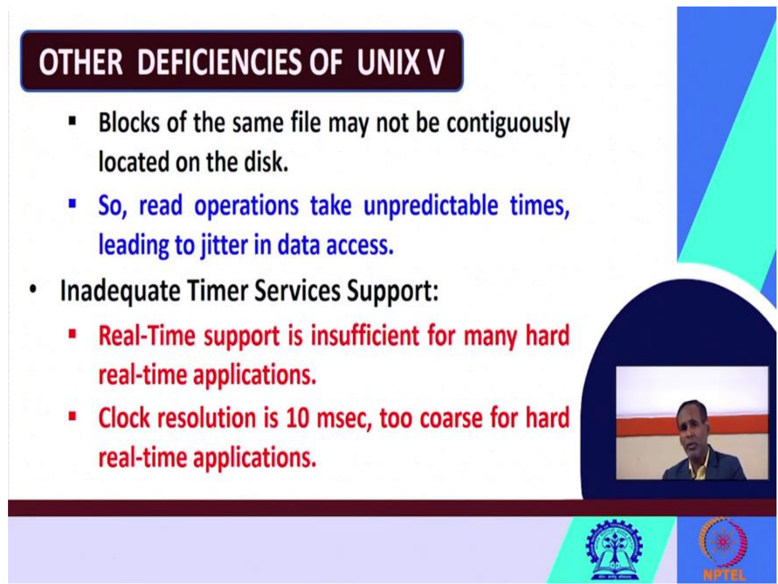


What are the other deficiencies of Unix V? So, the other deficiencies are as follows, like inefficient device driver support. So, it does not have an efficient device drivers support. Unix V has inefficient device driver support. Also if you support for the new device has to be added then what will happen? If support for a new device is to be added, then the driver module has to be linked to the kernel fast.

So, if support for a new device is to be added then first the diver module has to be linked to the canal. So, this system generation support is very much cumbersome. This is another deficiency. Next deficiency is lack of real-time file support. It does not have real-time file support. So, while a task is written to a file, it may encounter an error when disk space delays.

So, while a task in Unix is written to a file, it may encounter an error when the disk space, it depletes. Now, the traditional file writing approaches result in what? The traditional file writing approaches result in slow writes the space needs to be allocated before writing. In the traditional approaches, the space needs to be allocated first before writing. So, this will result in slow writes.

(Refer Slide Time: 31:48)



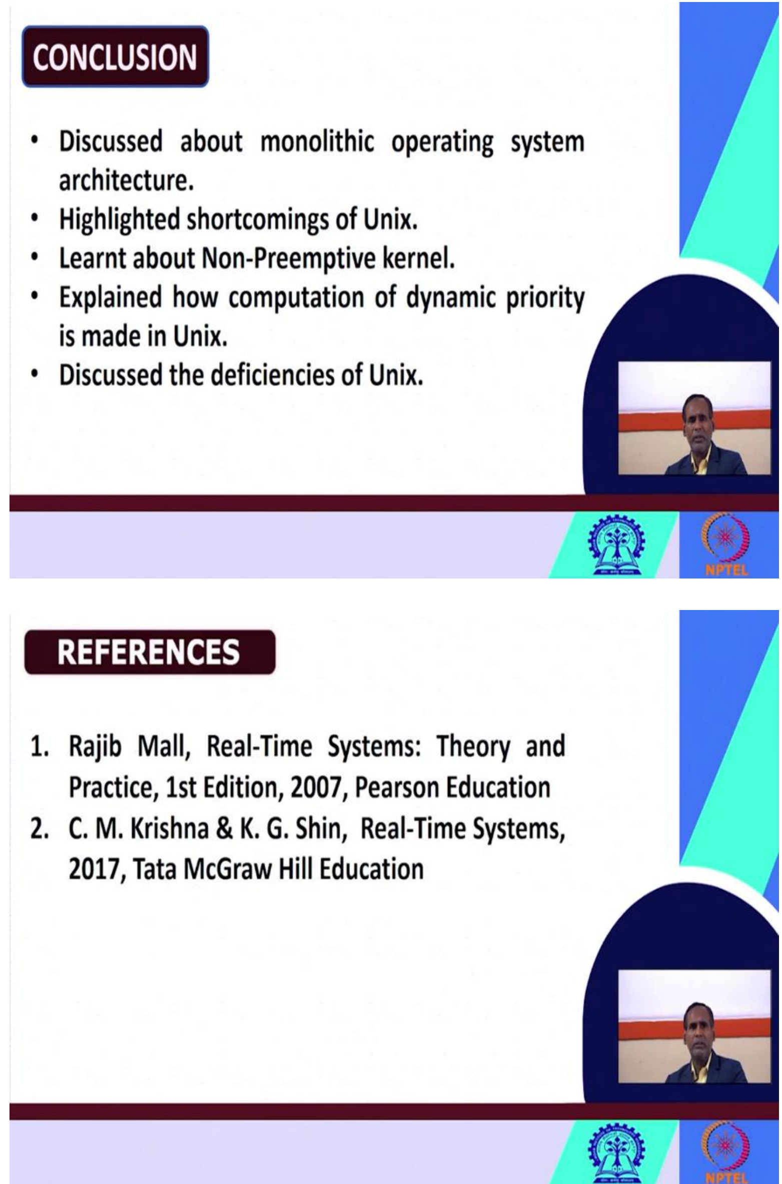
So, the blocks of the same file may not be continuously located on the disk. So, the blocks on the same file in file management and the blocks of the same file may not be contiguously located on the disk. So, the read operations like they may take unpredictable times leading to jitter in data access. So, since the blocks have the same file, they may not be contiguously located on the disks, then what will happen, the read operations they may take unpredictable times, which may lead to jitter in assessing the data.

So, then next deficiencies inadequate time service support. The time service support we have discussed already in the last class. So, Unix V, it has inadequate time services support, the real-time support is insufficient for many hard real-time applications. So, the time services support or the real-time support provided by your Unix V is very much insufficient for several or for many hard real-time applications.

You know that in Unix V the clock resolution maybe of the order of some seconds or 10 sorry, the clock realization is of the order of some 10 millisecond is too coarse for hard real-time applications is it not it. So, in Unix V the clock resolution is of the order of 10 millisecond which is two coarse for real-time applications because in real-time application on the hard real-time tasks may have some milli.

The duration or the duration could be of the deadline maybe of the order of one millisecond, but here the clock resolution is 10 millisecond, maybe too coarse for hard real-time applications. So, these are some of the other deficiencies of Unix V to be used as real-time operating systems.

(Refer Slide Time: 33:37)



So, today, we have discussed about we have recapped the monolithic operating system architecture and the microkernel operating system architecture. We have highlighted the important shortcomings along with some other shortcomings of Unix to be used, as a real-time operating system. We have also learned about the two major shortcoming of Unix, that means, it is having a non-preemptive kernel and it computes and it has a dynamic priority level. It does not have static priority assignment.

We have we explained, how, we have explained mathematically, how the computation of the dynamical priority is made on Unix after each interval. We have also discussed several other deficiencies of Unix such as its poor file management system and etc, that we have also seen. So, these are some of these things we have discussed. Particularly, we have discussed, why

Unix V is not suitable for the old time of applications. We have taken this from this from these two books. Thank you very much.