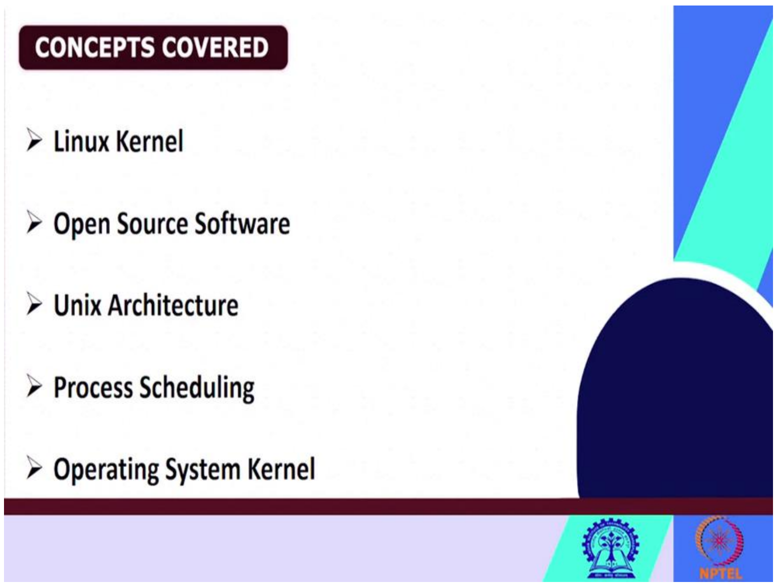
Real-Time Systems

Professor: Durga Prasad Mohapatra
Department of Computer Science and Engineering
National Institute of Technology Rourkela

Lecture 39: Unix as a Real-Time Operating System

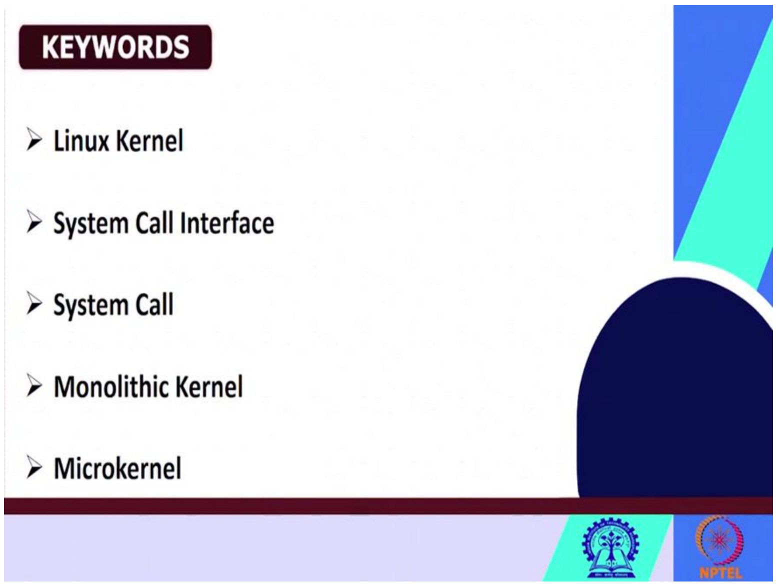
Good morning to all of you. Last class, we have seen about the basic requirements for real-time operating systems. So, today we will discuss about Unix, as a real-time operating system. So, how, whether Unix can be treated as an operating system, real-time operating system or it can be used for real-time applications. So, today we will see that one.

(Refer Slide Time: 00:46)



CONCEPTS COVERED

- Linux Kernel
- Open Source Software
- Unix Architecture
- Process Scheduling
- Operating System Kernel

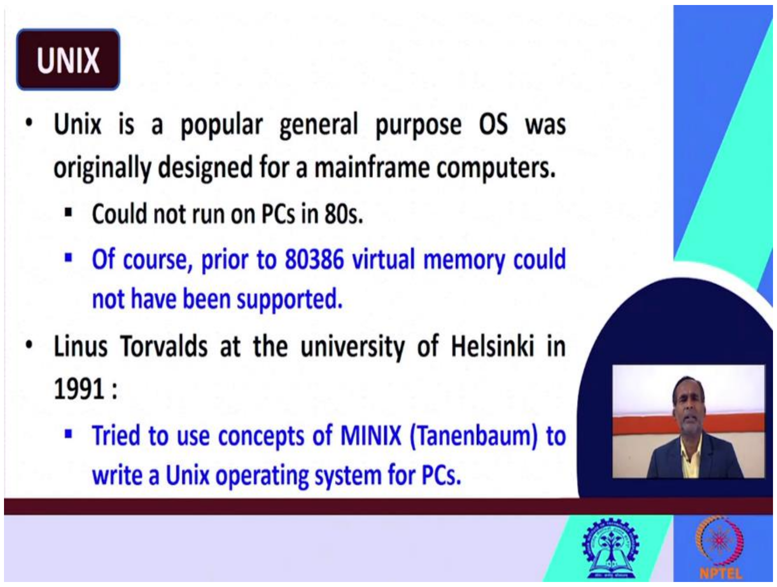


KEYWORDS

- Linux Kernel
- System Call Interface
- System Call
- Monolithic Kernel
- Microkernel

So, we will see about the Linux kernel. What do you mean by open source software or these particularly open source operating systems, the architecture of Unix, processor dealing in Unix and the operating system kernel? So, some of the keywords you will use are like Linux Kernel, System Call Interface, what is your system called, Monolithic Kernel, Microkernel Kernel, those things we will discuss today.

(Refer Slide Time: 01:11)

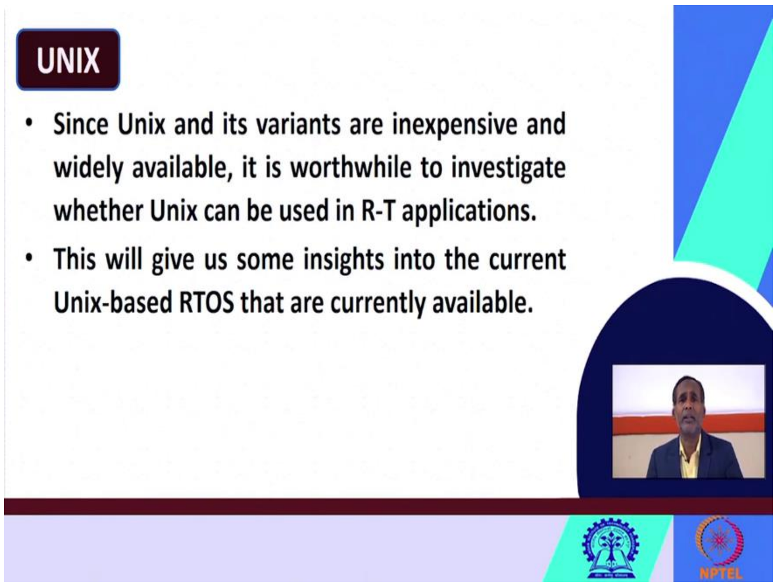

UNIX

- Unix is a popular general purpose OS was originally designed for a mainframe computers.
  - Could not run on PCs in 80s.
  - Of course, prior to 80386 virtual memory could not have been supported.
- Linus Torvalds at the university of Helsinki in 1991 :
  - Tried to use concepts of MINIX (Tanenbaum) to write a Unix operating system for PCs.

Now let's see we will start with this Unix. Why we have chosen this Unix first? See, you know, that Unix is a very popular general-purpose operating system. So, while I have discussed about the spectrum of these real-time operating systems at that time I have told that are the bottom level these general purpose operating systems and the top real-time what operating systems are there.

So, Unix is a very popular general-purpose operating system which has originally designed for mainframe computers, but it could not run on the PCs in 1980s. So, of course, prior to 80386 virtual memory would not have been supported in Unix. So, then what happened, in 1991, Linus Torvalds, so this person at the University of Helsinki, so he has tried to use the concept of MINIX to write the Unix operating system which can be applied or which can work on PCs.

(Refer Slide Time: 02:17)



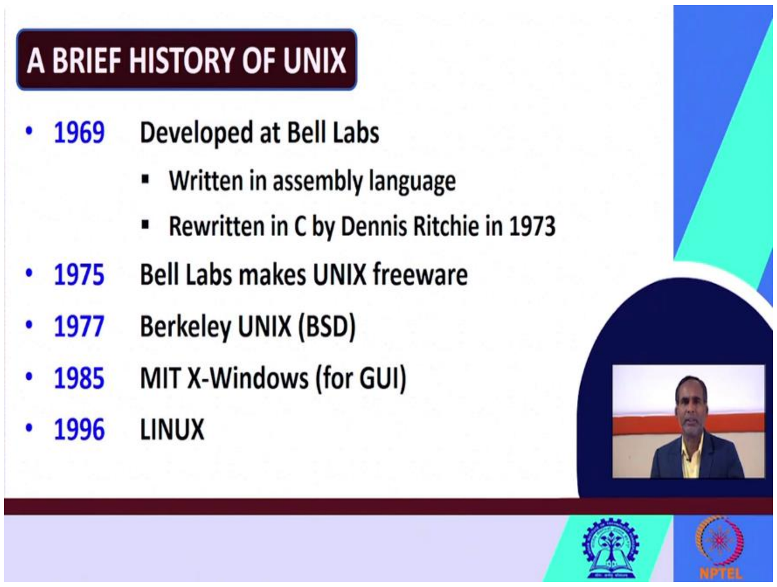
UNIX

- Since Unix and its variants are inexpensive and widely available, it is worthwhile to investigate whether Unix can be used in R-T applications.
- This will give us some insights into the current Unix-based RTOS that are currently available.

So, since this Unix and its other variants, they are quite inexpensive and they are widely available everywhere else, so it is very much worthwhile to investigate whether the Unix operating system can be used in real-time applications. Let us investigate that.

So, this investigation will help us, will give us some insights into the current available what the commercially Unix-based real-time operating systems. So, it will give some insight into those currently available commercial Unix-based real-time operating systems. So, let us start to the brief history of Unix, how Unix words are developed.

(Refer Slide Time: 02:58)



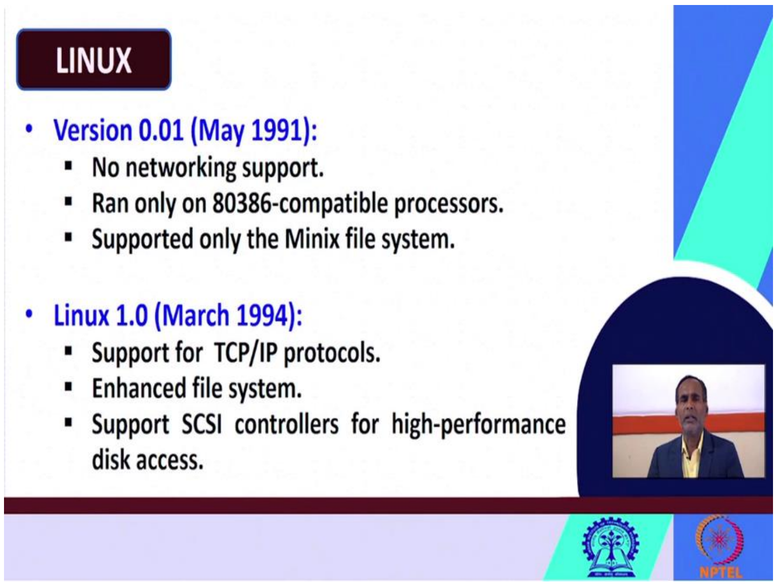
A BRIEF HISTORY OF UNIX

- 1969 Developed at Bell Labs
  - Written in assembly language
  - Rewritten in C by Dennis Ritchie in 1973
- 1975 Bell Labs makes UNIX freeware
- 1977 Berkeley UNIX (BSD)
- 1985 MIT X-Windows (for GUI)
- 1996 LINUX

So, this Unix, you see, it was developed in 1969 at Bell Labs, and it was, at that time, it was written in assembly language and then it was rewritten in C. Is it not it? by Dennis Ritchie in 1973. You have already known, Dennis Ritchie has developed C and simultaneously he has developed this operating system Unix is it not it. So, this operating system Unix it was rewritten in C by Dennis Ritchie. Then 1975 this Bell Lab, it makes this unique such a freeware open-source software.

Then this 1977 and this other variant other version came Berkeley UNIX. And then 85 MIT X-Windows version came. Actually, his X-Windows is normally it is used for graphical user interfaces, those things for developing graphical user interfaces, and then 1996 this Linux came. So, now let us look, talk a little bit about this Linux. So, let us see about this Linux or Linux kernel.

(Refer Slide Time: 04:01)



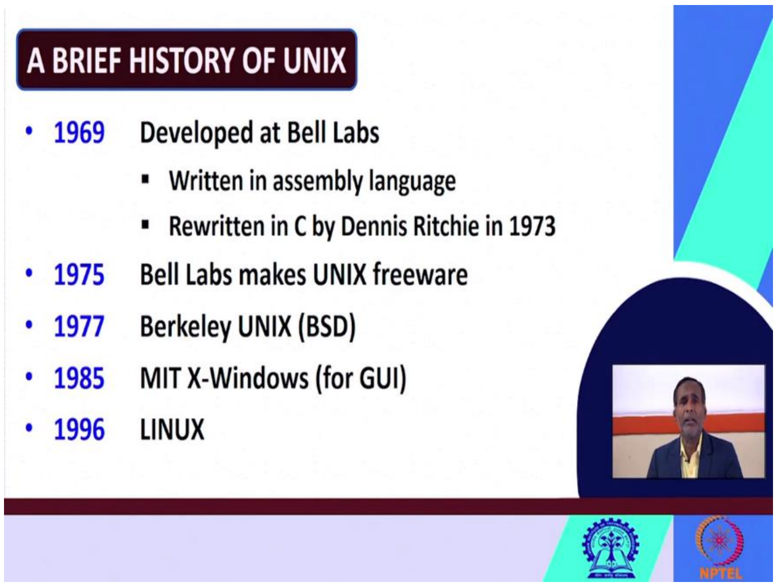
LINUX

- Version 0.01 (May 1991):
  - No networking support.
  - Ran only on 80386-compatible processors.
  - Supported only the Minix file system.
- Linux 1.0 (March 1994):
  - Support for TCP/IP protocols.
  - Enhanced file system.
  - Support SCSI controllers for high-performance disk access.

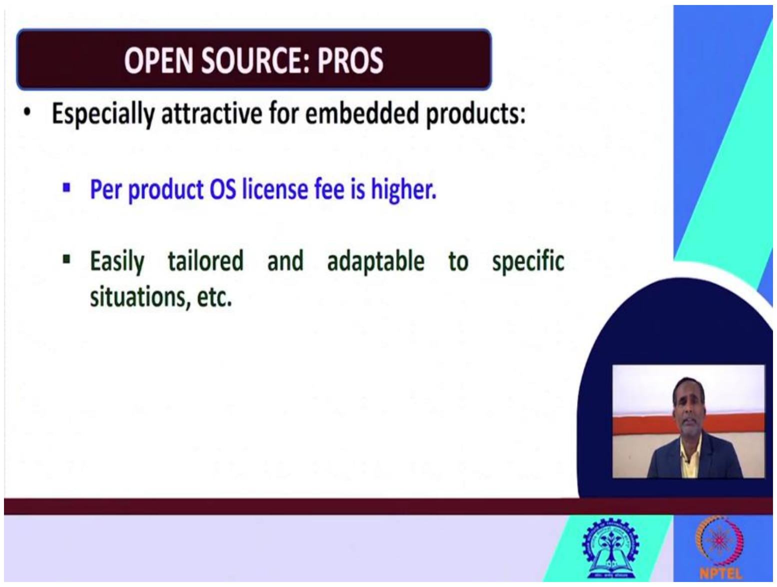
The first version was 0.01. It was developed in May 1991. So, in the version 0.01, there was no networking support. It ran only on 80386 compatible processors, and it is supported only the MINIX file system. No other file system, it supported only the MINIX file system. Then the next version, Linux 1.0, it came in March 1994.

And it supported, there are support for more TCP/IP protocols in this version. There was also enhanced the file system. Because you see in version 0.01 only it supported the MINIX file system because this Linux 1.08 supported enhanced file system. It also support the SCSI controllers. It also support the SCSI controllers for obtaining high-performance disk access. So, this is about a little bit brief introduction or history to Linux.

(Refer Slide Time: 05:02)


A BRIEF HISTORY OF UNIX

- 1969 Developed at Bell Labs
  - Written in assembly language
  - Rewritten in C by Dennis Ritchie in 1973
- 1975 Bell Labs makes UNIX freeware
- 1977 Berkeley UNIX (BSD)
- 1985 MIT X-Windows (for GUI)
- 1996 LINUX

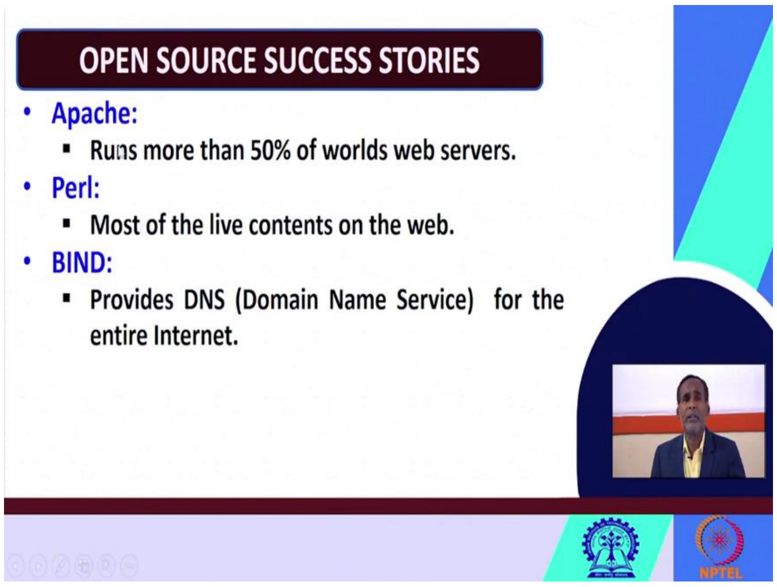

OPEN SOURCE: PROS

- Especially attractive for embedded products:
  - Per product OS license fee is higher.
  - Easily tailored and adaptable to specific situations, etc.

Now, let us see, as I have already told you in 1975 Bell Lab makes Unix as a freeware. So, let us see, what are the advantages of these using these open-source operating systems particularly these open-source, this Unix as a open-source operating system. So, let us see, some of the advantages of the open source, using open-source operating systems.

So, particularly these open-source operating system it was especially very much attractive for embedded products, and per operating system license fee is higher. If we will go product wise, per product operating system license fee, it was higher. So, this open-source software it can be easily tailored, and it can it can be easily adaptable to specific situations to specific environments. So, these are some of the advantages of using open-source softwares, particularly have the open-source operating system.

(Refer Slide Time: 05:54)



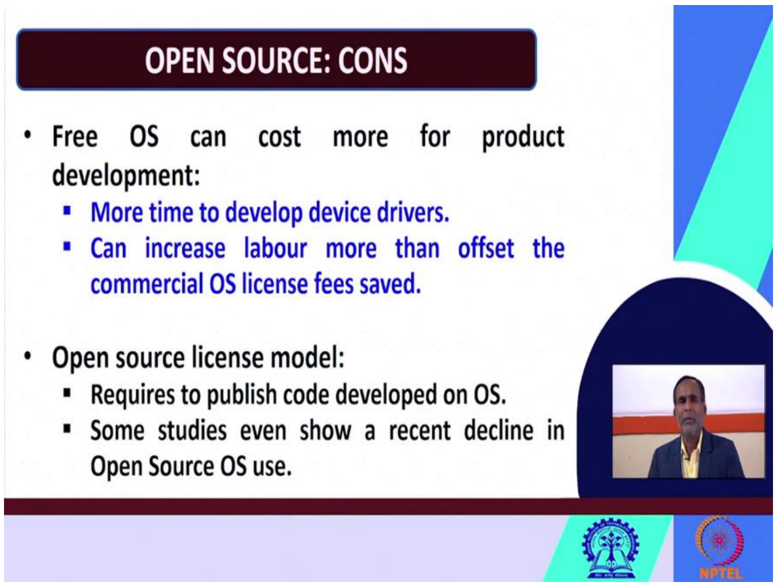
OPEN SOURCE SUCCESS STORIES

- **Apache:**
  - Runs more than 50% of worlds web servers.
- **Perl:**
  - Most of the live contents on the web.
- **BIND:**
  - Provides DNS (Domain Name Service) for the entire Internet.

So, let us see some of the success stories of these open-source softwares. Like you have already known Apache. So, this Apache, it runs more than 50% of the web servers, this is one of the open-source software. Similarly, Perl, another open-source software, which was most of the live contents are on the web. BIND, it also provides it is another open-source software, which provides DNS, DNS stands for dominance name service for the entire internet.

So, several other success stories are there for different open-source software. We will not go into that because we are discussing about the advantages of the open-source, and that is why I have given a little bit of these stories available on the open-source, success of the different software.

(Refer Slide Time: 06:44)



OPEN SOURCE: CONS

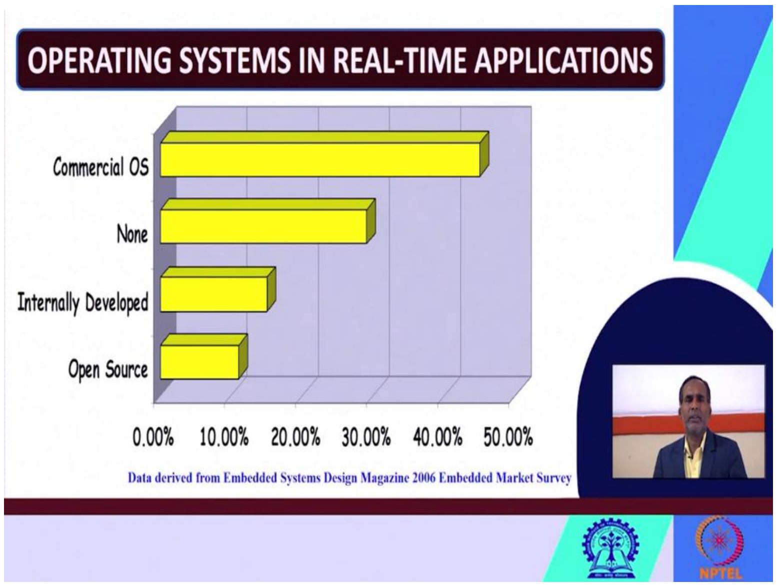
- Free OS can cost more for product development:
  - More time to develop device drivers.
  - Can increase labour more than offset the commercial OS license fees saved.
- Open source license model:
  - Requires to publish code developed on OS.
  - Some studies even show a recent decline in Open Source OS use.

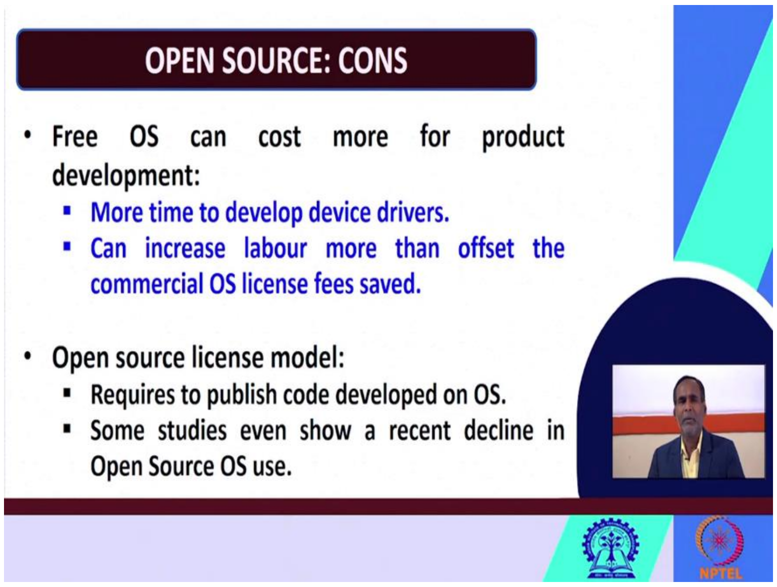
If we will see the limitations side open-source softwares they are having also some limitations like the free operating system that you are getting the free operating system can cost even more for the product development. So, it will cost more for the product development, it will take more time to develop the device drivers, it can increase the labor more than the offset value, the commercial operating system license fees saved.

o, that means, how much money you are saving by using the open-source source of debt, so it can increase the level more than this offset value, more than the offset the commercial operating system license page that you have saved. So, open-source licensing model it requires also to publish the code developed on the OS. Some studies even show that a recent decline in open-source operating system.

So, due to these limitations, so some studies they have shown that the recent decline in use of the open-source OS. We will see some of these statistics now. So how the open-source, what subtracts they are used, how, whether they have there is increase or there is a decrease let us see.

(Refer Slide Time: 08:00)





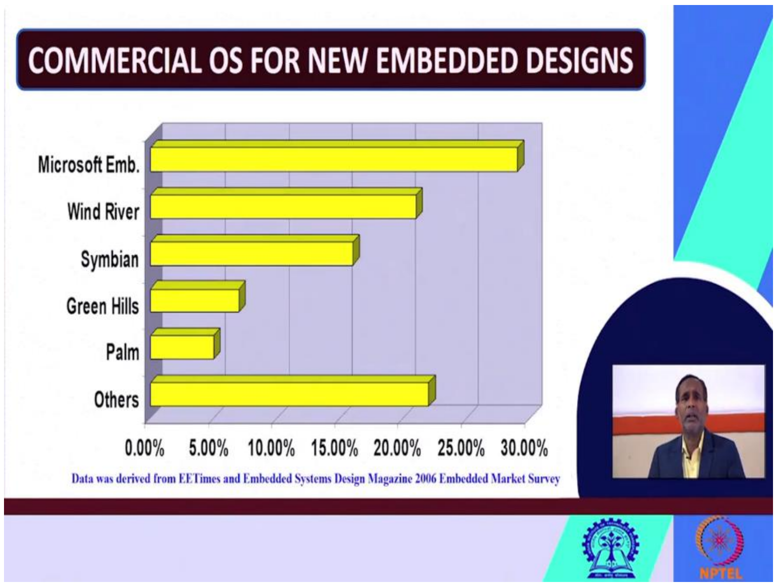
OPEN SOURCE: CONS

- Free OS can cost more for product development:
  - More time to develop device drivers.
  - Can increase labour more than offset the commercial OS license fees saved.
- Open source license model:
  - Requires to publish code developed on OS.
  - Some studies even show a recent decline in Open Source OS use.

If you will see so this is a statistics taken from this data which is derived from the embedded design magazine in 2006. So, you can see that, as I have already told you due to these drawbacks, some of these studies show that a recent decline in the open-source OS use. And you can see that people are preferring using the commercial OS highest maybe more than 40%.

But you see, open-sources softwares they are used say in between of only 10%. So, this shows that due to these drawbacks, these open-source OS they are used to less in comparison to the commercial operating systems.

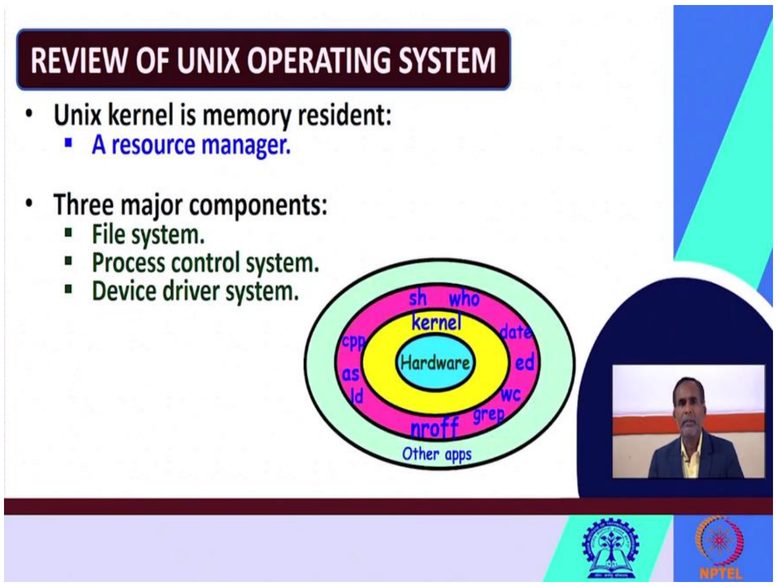
(Refer Slide Time: 08:29)



Now, let us see, some of the commercial operating systems for the new embedded designs. So, there are various products by various operating systems are shown from various companies. Like Microsoft embedded systems, so Wind River, Symbian, Green Hills, Palm and others. So, it shows some of the statistics, and it show that the Microsoft embedded systems it has what more percentage of use more than 25 percentage and Palm is the least less than 5 percent.

So, this is the statistics that is also taken from EETimes and Embedded System Design magazine in 2006. And from a market survey it shows that this Microsoft embedded system it has largely, it has popularly been used in many of the applications which is more than 25 percentage, in comparison to the others.

(Refer Slide Time: 09:29)


REVIEW OF UNIX OPERATING SYSTEM

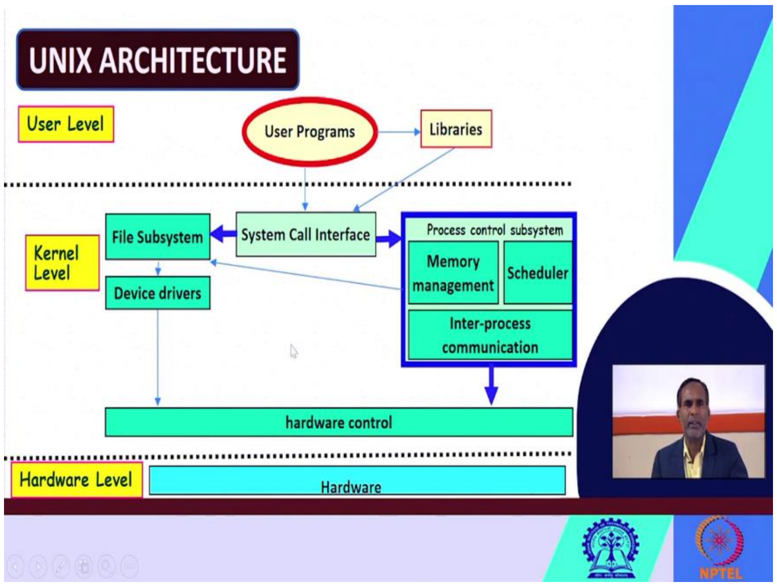
- Unix kernel is memory resident:
  - A resource manager.
- Three major components:
  - File system.
  - Process control system.
  - Device driver system.

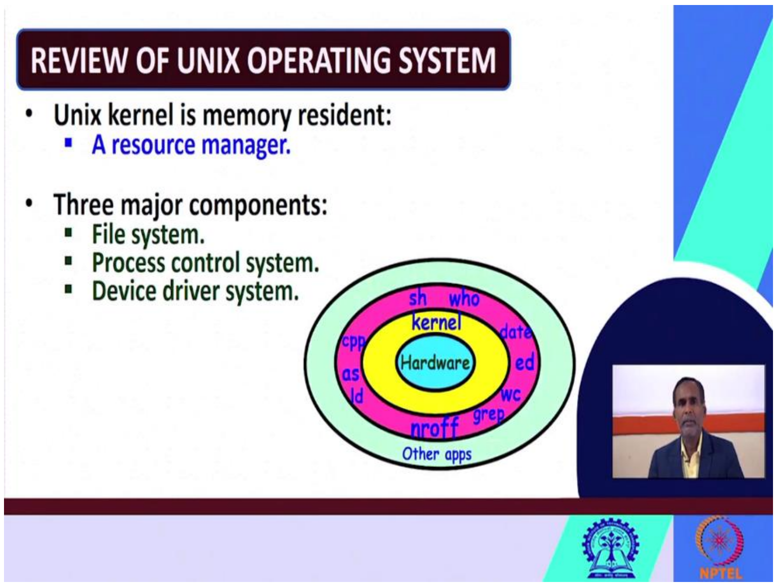
sh
who
kernel
date
cpp
as
ld
Hardware
ed
wc
grep
nroff
Other apps

Now, let us quickly review this traditional Unix operating system. You have already read this operating system Unix in the, your operating system subject previously. I will just quickly review some of the important points, so that will help us in going for the further contents.

Like you know that the Unix kernel is memory resident, and this kernel may work as a resource manager. There are three important components in this Unix kernel. So maybe one is the file system then the process control system and the device driver system, these are the three major components in this Unix kernel.

(Refer Slide Time: 10:09)


UNIX ARCHITECTURE



REVIEW OF UNIX OPERATING SYSTEM

- Unix kernel is memory resident:
  - A resource manager.
- Three major components:
  - File system.
  - Process control system.
  - Device driver system.

Now let us quickly look at the Unix architecture. How does it look like? So, you can see that this UNIX architecture consists of three important levels, the user level, kernel level and the hardware level. In the user level the user programs or the applications they will be there. The users will write down the user programs of the applications. These user programs, they may interact, they may communicate through the kernel level, through what, through the system call interfaces.

And the user programs also may interact or may communicate interface with the libraries, libraries can also want to interact with this kernel level through the system call interface and the user programs they can interact with the file system as well as with the process control system, through what, through the system call interface.

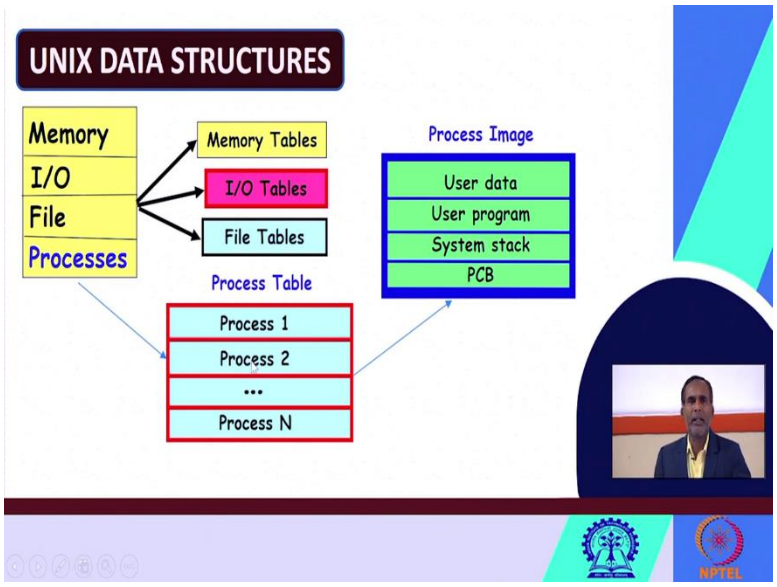
So, in the kernel level, what are the important modules present? The file subsystem, device drivers and the process control system, mainly it carries this memory management. So, this also I have told you earlier that it has three major components Unix kernel has three major components. The file system, process control system and the device driver system.

You can see here, this is the kernel level, here one component is file subsystem, device drivers and the process control system. The user can interact with the file system, as well as, with this process control system through the system called interfaces. Similarly, the user programs can interact with the library and library also can interact with the file system, and this process control system to the system call interface.

The file subsystem can communicate with the device drivers, also the process control system can interact with the hardware control. See, at the bottom level there is a hardware level. One can say it is the hardware level. And here in the kernel level there is a hardware control. So, the device drivers and what process control subsystem they can also interact with the hardware control. At the bottom level the hardware level is there.

Process control system consists of lots of modules, it consists of a memory management sub-module a scheduler sub-module and IPC, Inter Process Communication, these are the sub-models present inside the process control system. So, in this way, yes, in this way the UNIX architecture it looks like and it works in this way.

(Refer Slide Time: 12:43)

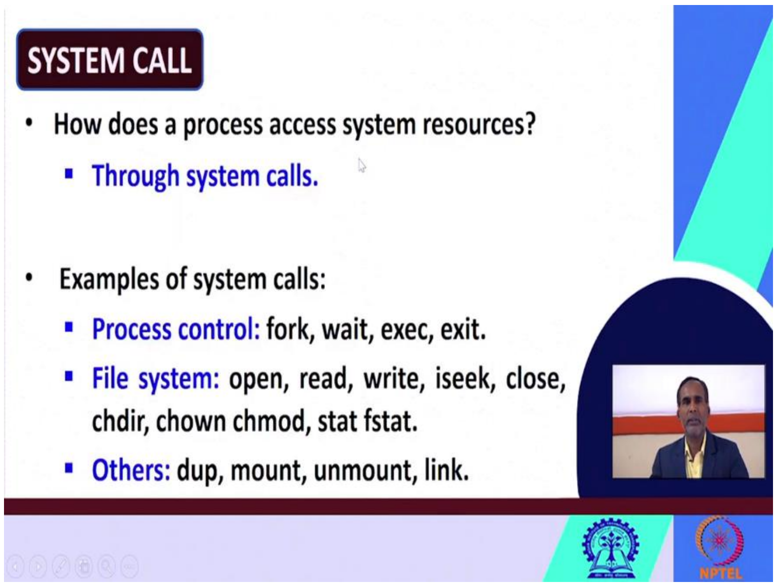


UNIX DATA STRUCTURES

Now, let us quickly look at the Unix data structures. What data structures are used in Unix? You can see, these are the data structures used in Unix like memory, I/O, file and processes. So, for each one there is a corresponding table. Like memory will deal the memory tables, I/O will deal with the i/o tables file will deal with the file table and for each process there is a process table.

Like there are n number of processes, so the process table consists of these n number of processes. And for every process there is a process image, these process image will contain what, the user data, user program, systems stack and the PCB. So, in this way, you can see these are the Unix data structure. So, these are the Unix data structures.

(Refer Slide Time: 13:27)



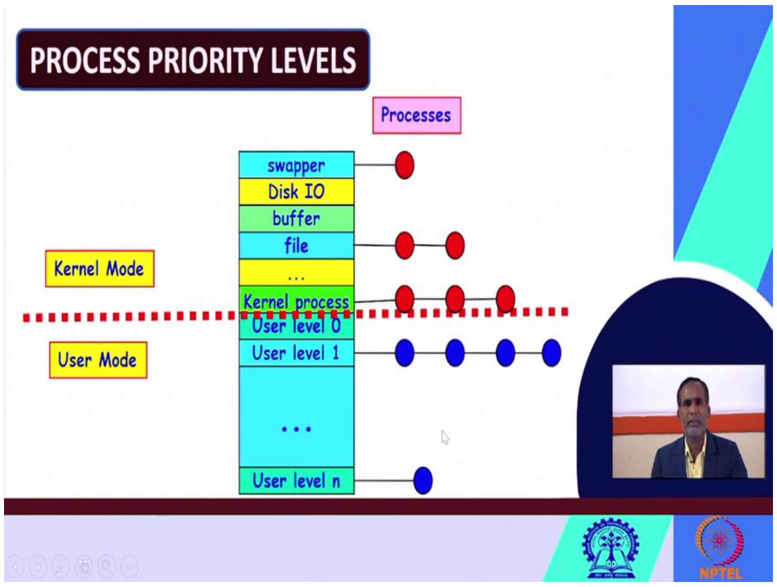
SYSTEM CALL

- How does a process access system resources?
  - Through system calls.
- Examples of system calls:
  - Process control: fork, wait, exec, exit.
  - File system: open, read, write, iseek, close, chdir, chown chmod, stat fstat.
  - Others: dup, mount, unmount, link.

Now let us see, how a system call is performed in Unix. How does your process access? How does your process access the system resources? You know, that is through system calls. So, through system calls a process accesses the system resources. So, what are the examples of system calls? You can see, there are several examples of system calls that you have already known your traditional operating system paper.

Might be for process control you can use the system calls like fork, wait, execute, exit. For file system you can use the system calls such as open, read, write, iseek, close, chdir, chown, chmod, stat fstat like that. And there are several other examples of system calls such as duplicate, mount, unmount, link etc. So, these are some of the examples of system calls. And the process or the process accesses the system resources through only the system calls.

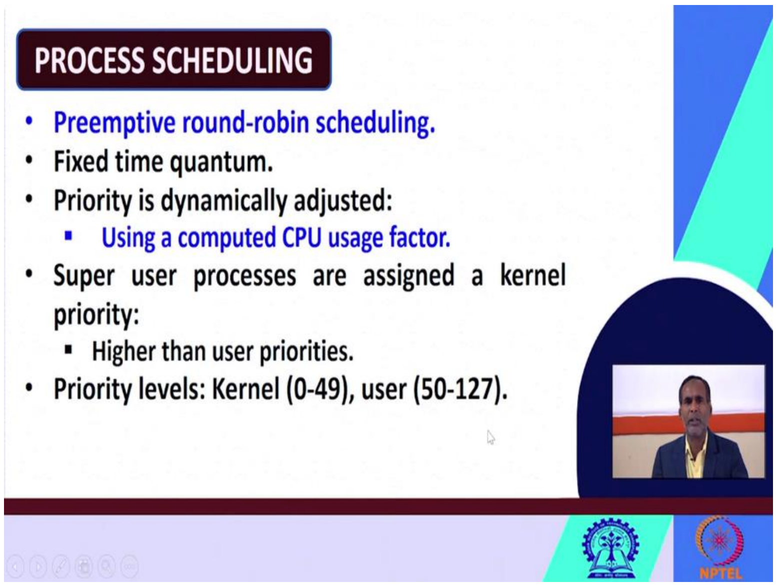
(Refer Slide Time: 14:34)



PROCESS PRIORITY LEVELS

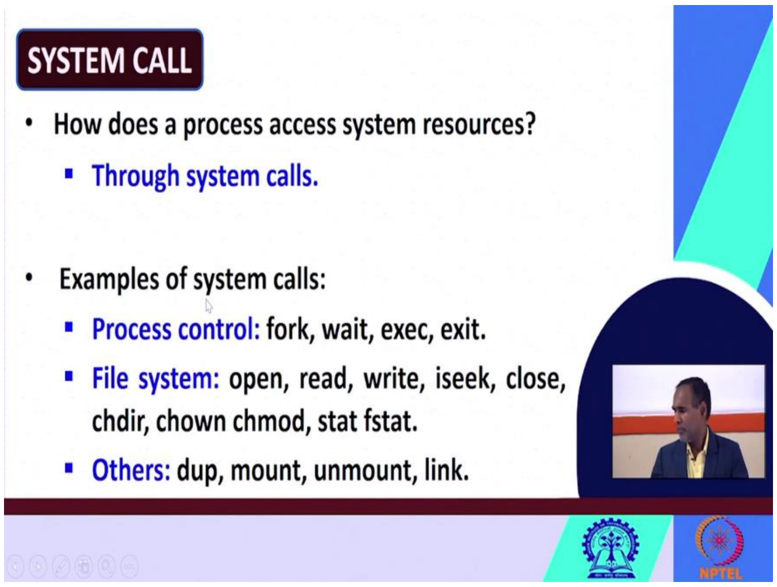
These are about some of the priority levels. So, mainly in the Unix, the priority levels are divided into two types. The priority levels for the kernel mode task, and the priority level for the user mode task. So, based on these priority levels the tasks are divided into different what you can say basebands, these are the some of the basebands, and these are the processes. Let us see, how many, what priority levels are there for each mode.

(Refer Slide Time: 15:04)



PROCESS SCHEDULING

- Preemptive round-robin scheduling.
- Fixed time quantum.
- Priority is dynamically adjusted:
  - Using a computed CPU usage factor.
- Super user processes are assigned a kernel priority:
  - Higher than user priorities.
- Priority levels: Kernel (0-49), user (50-127).



SYSTEM CALL

- How does a process access system resources?
  - Through system calls.
- Examples of system calls:
  - Process control: fork, wait, exec, exit.
  - File system: open, read, write, iseek, close, chdir, chown chmod, stat fstat.
  - Others: dup, mount, unmount, link.

You can see that at the kernel there are 50 priority levels, and those priority levels will vary from 0 to 49. And as user level, the rest. See, total priority levels in Unix is 128, 0 to 127, out of that 50 reserved for kernel and the rest are reserved for the users. So, for, as I have already told you what, kernel how much, 50 and then 128 - 50, that means, about 78. So, 78 are given to what? To the what user?

So, for kernel 50 priorities are reserved, 0 to 49 and rest is 7-8 priorities, that means, 50 to 127 these priorities are reserved for the user mode task. So, in this way, the process priority levels are there. For kernel mode processes, the priority levels are from 0 to 49 for user mode processes, the priority levels are from 51 to 127.

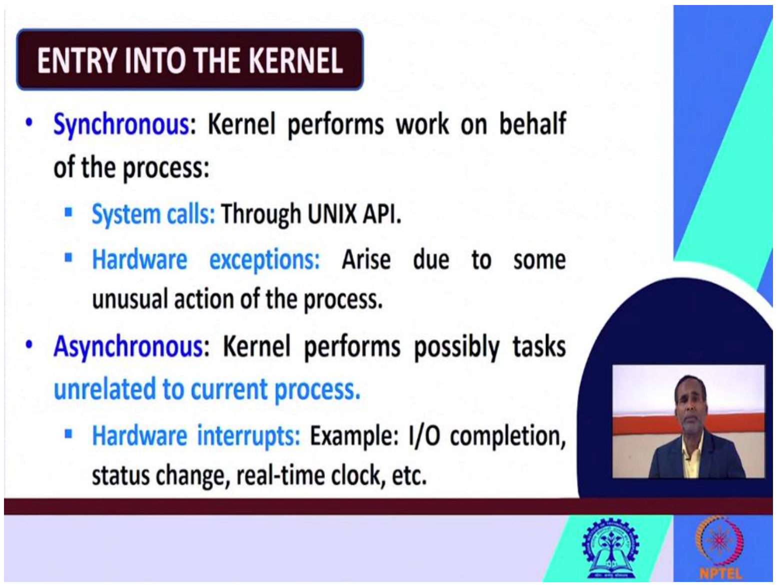
Then, let us say, how this process scheduling occurs in this Unix operating system? So, Unix uses preemptive round robin scheduling. I hope, you have already known these different scheduling strategies such as first come first serve, round-robin, and shortest job first etc in your traditional operating system paper.

So, a Unix operating system uses preemptive round-robin scheduling. Here, this Unix it uses fix time quantum, that means, this total time can be divided into some quantums, and that quantum is fixed. So, it uses fixed time quantum. The priority is dynamically adjusted using your computer CPU usage factor.

We have seen earlier there are two kinds of priorities, one is static priority, another is dynamical priority. So, in Unix, the priorities are dynamically adjusted. How? By using a computed CPU usage factor. In the next class, we will see, how this priority is dynamically computed using this CPU usage factor. Please remember within traditional Unix the priority is not statically assigned, the priority is dynamically changing, the priority is dynamically adjusted using a computed CPU usage factor.

The super user processes they are assigned a kernel priority, which is higher than the user priorities. So, normally, the user processes or the super user processes they are assigned a kernel priority, which had the end user priorities. And the priority levels I have already told you in Unix the kernel is assigned, the kernel processes are assigned priority in between 0 to 49 and the rest is seven, eight priorities are given to the user processes 50 to 127. These priorities are given to the user processes.

(Refer Slide Time: 17:59)



ENTRY INTO THE KERNEL

- **Synchronous**: Kernel performs work on behalf of the process:
  - System calls: Through UNIX API.
  - Hardware exceptions: Arise due to some unusual action of the process.
- **Asynchronous**: Kernel performs possibly tasks unrelated to current process.
  - Hardware interrupts: Example: I/O completion, status change, real-time clock, etc.

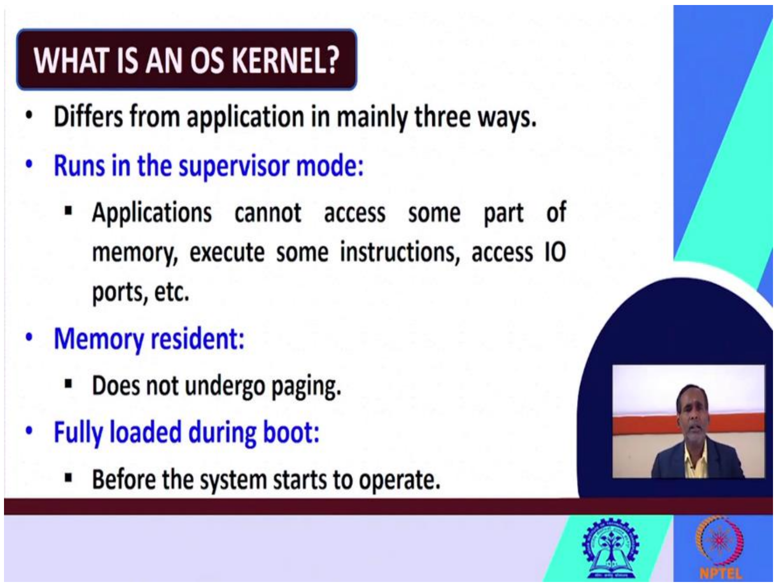
Now, let us see, how we can enter into the kernel, enter into the kernel. It can be either through synchronous mode or through asynchronous mode. So, synchronous mode means, here the kernel performs work on behalf of the process. So, as if, what, on behalf of the process the kernel performs each and every work.

So, in synchronous mode, the kernel performs the work on behalf of the process. And I have already told you about the system calls. The system calls are through Unix API. So, what is about the hardware exceptions? The hardware exceptions may arise due to some unusual action of the process, due to some unexpected action, due to some unusual action of the process, the hardware exceptions may arise.

And in a synchronous mode what is happening? The kernel performs possible tasks unrelated to the current process. Please see, in synchronous mode, the kernel performs work on behalf of the process, and in asynchronous mode the kernel performs possibly, the different tasks, which unrelated to the current process.

There can be some hardware interrupts in this asynchronous mode, such as, due to I/O completion, due to change of the status, due to use of a real-time clock etc. These could be the possible hardware interrupts in the synchronous mode.

(Refer Slide Time: 19:25)



WHAT IS AN OS KERNEL?

- Differs from application in mainly three ways.
- Runs in the supervisor mode:
  - Applications cannot access some part of memory, execute some instructions, access IO ports, etc.
- Memory resident:
  - Does not undergo paging.
- Fully loaded during boot:
  - Before the system starts to operate.

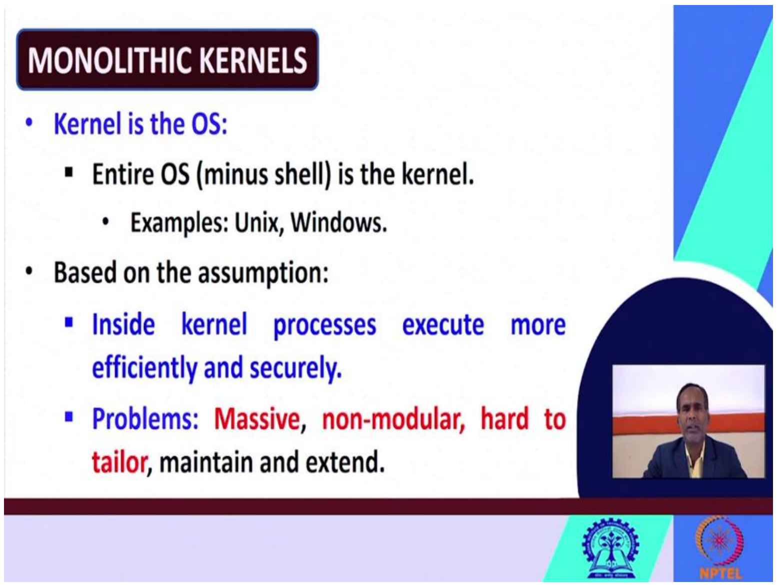
Now, let us see, what is an operating system kernel. What is an operating system kernel? So, how does it differ from an application process? So, this operating system kernel it defers from the application in mainly three ways. What are the three ways? First of all, it runs in the supervisor mode. So, this kernel it runs in the what supervisor mode. I have already told you, the kernel differ from the an application in mainly three ways. It runs in the supervisor mode.

The applications cannot assess some part of the memory. So, it runs in the supervisor mode means, the applications they cannot assess some part of the memory, execute some instructions, access I/O ports etc. Because the kernel has taken the supervisor mode. So, since kernel runs in the supervisor mode the applications, they cannot assess some part of the memory, execute some instructions and access I/O ports etc.

I have already told you that this Unix operating system, this Unix kernel it is memory resident, it does not undergo paging. So, since it is memory resident it does not undergo paging. So, also another important difference between this operating system kernel and the what the user or the applications is one more difference that is fully loaded during boot. So, during the boot the operating, the kernel is fully loaded.

So, fully loaded during boot that is before the system starts to operate. So, before the system starts to operate the kernel is fully loaded during boot. So, these are these three important differences between an operating system kernel and an application.

(Refer Slide Time: 21:15)



MONOLITHIC KERNELS

- Kernel is the OS:
  - Entire OS (minus shell) is the kernel.
    - Examples: Unix, Windows.
- Based on the assumption:
  - Inside kernel processes execute more efficiently and securely.
  - Problems: Massive, non-modular, hard to tailor, maintain and extend.

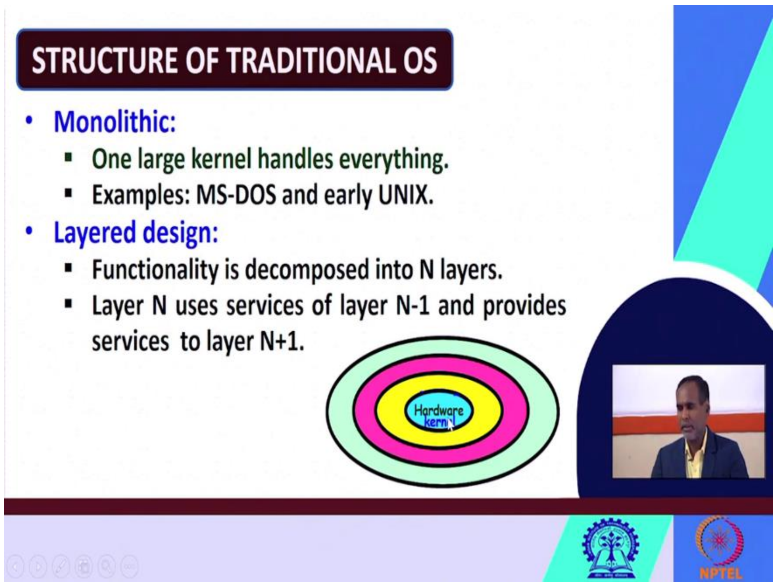
Now, let us see some of the types of the monolithic kernels. I will discuss here, two important types of the kernel, one is the monolithic kernel, another is microkernel. So, as its name suggests a monolithic kernel. Just what you can say that a monopoly kind of thing you can say. Here the kernel in the operating system, the kernel thinks it is the whole operating system, kernel in the operating system.

It is considered it is the entire operating systems minus the shell, excluding the shell. So, excluding the shell, the entire operating system is treated as the kernel. And the examples of this monolithic kernel like you can see this Unix, windows here the kernel is the OS and here these are the examples of monolithic kernels. So, what is the assumption? How does this monolithic kernel work? The monolithic kernel is based on the assumption that inside the kernel processes execute more efficiently and securely.

So, why the, here the entire operating system is treated at the kernel because, it is assumed that inside the kernel the processes, they execute their own more efficiently and securely. That is why, the whole kernel it is considered as the operating system, and inside that kernel whatever processes will be there, they will execute more efficiently and securely. Based on that assumption, monolithic kernel works.

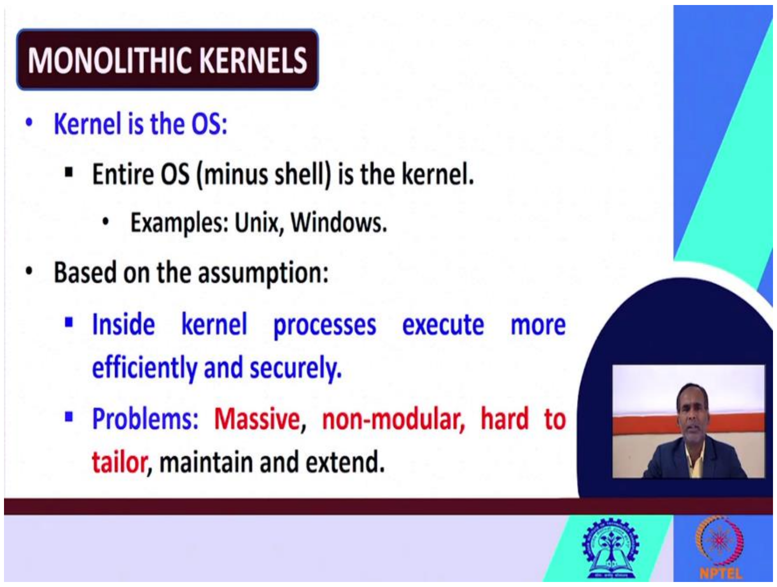
But there are some limitations of this monolithic kernel. What? Massive, because everything we are putting into the kernel. All the actions, all the tasks, all the activities they are carried out in the kernel. So that is why, these monolithic kernels they become more massive. You cannot modular them because everything else is packed into that, so non-modular and is very much hard to tailor. If you want to make any changes very much difficult to tailor also very much difficult to maintain and extend. So, these are some of the problems associated with monolithic kernels.

(Refer Slide Time: 23:16)



STRUCTURE OF TRADITIONAL OS

- **Monolithic:**
  - One large kernel handles everything.
  - Examples: MS-DOS and early UNIX.
- **Layered design:**
  - Functionality is decomposed into N layers.
  - Layer N uses services of layer N-1 and provides services to layer N+1.



MONOLITHIC KERNELS

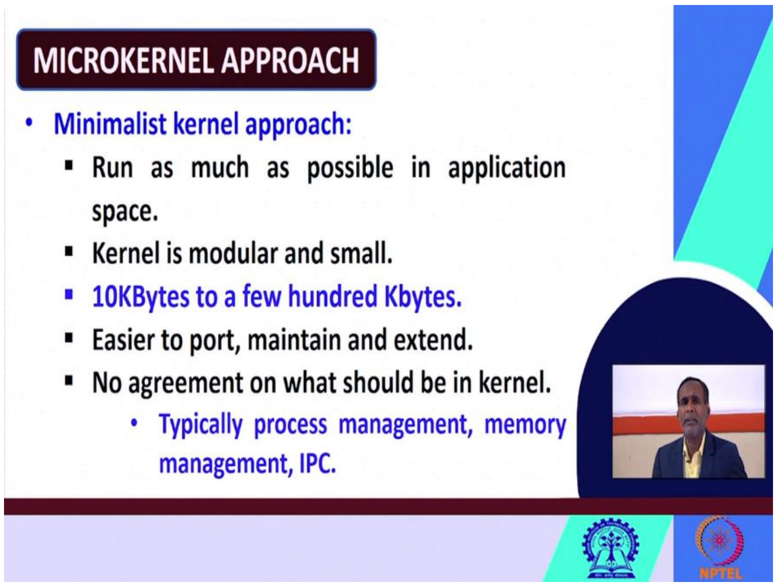
- Kernel is the OS:
  - Entire OS (minus shell) is the kernel.
    - Examples: Unix, Windows.
- Based on the assumption:
  - Inside kernel processes execute more efficiently and securely.
  - Problems: Massive, non-modular, hard to tailor, maintain and extend.

So, if we will see the structure. I have already given you the example of this monolithic kernel that is Unix and Windows. Let us see, what is the structure of the traditional operating system. So, the traditional operating systems normally they are, they use these monolithic kernels. And here, as its names of this monolithic there is one very large the kernel, which handles everything else. All the processes, all the tasks are handled by that large kernel that is why this is monolithic.

And examples are, I have already told you, these Unix or Windows or you can say that the MS-DOS and the early UNIX they are treated as the monolithic kernels. In layered design, what is happening? The functionality is decomposed into layers. All the functionalities that decomposed into some n number of layers. And in this layered architecture what happens, that, the layer N, it uses services of the layer N-1 and provides the services to the layer N+1.

So, in this layered, I have already told you the functionality is decomposed to N number of layers. In this layer architecture, layer N, it uses the services of the layer N -1 and it provides services to the layer N+ 1. Like this, this here, it follows a layered architecture or layered design. So, everything else you see that this kernel, so there is one large kernel, which handles everything else. So, this is the hardware kernel.

(Refer Slide Time: 24:42)



MICROKERNEL APPROACH

- Minimalist kernel approach:
  - Run as much as possible in application space.
  - Kernel is modular and small.
  - 10KBytes to a few hundred Kbytes.
  - Easier to port, maintain and extend.
  - No agreement on what should be in kernel.
    - Typically process management, memory management, IPC.

Now let us say, the other approach that is a microkernel approach. So, after discussing this, what monolithic kernel, let us quickly go towards the microkernel approach. So, this approach is known as a minimalistic kernel approach. That means, minimalist kernel approach means what, we will try to minimize the size of the kernel.

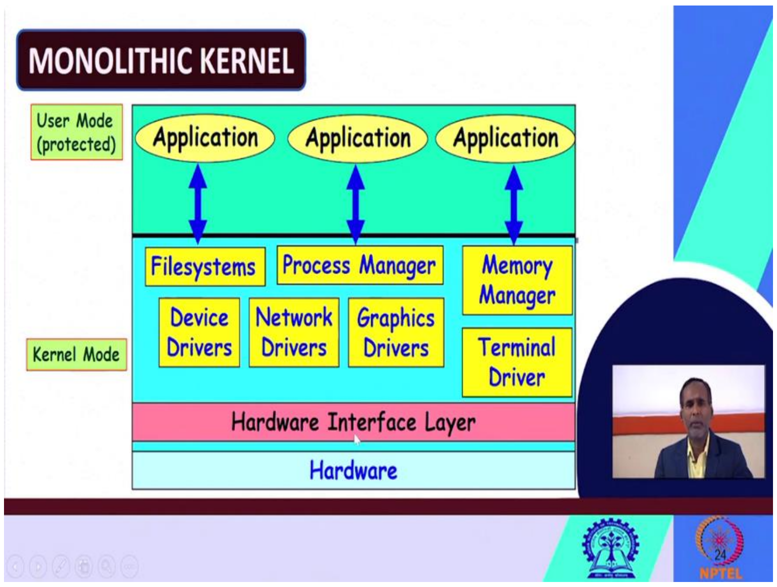
It runs, so run, you try to run, as much as possible in application space. So, everything else whichever are bare, which are at all required these important things you may run on kernel, otherwise, try to run as much as the processes the tasks as much as possible, where, in the application space. So, the cardinal is very much modular and small.

In comparison to the what monolithic kernel, the kernel is very much small and kernel is also modular. It is so small. What could be the size? The kernel is modular and small, and its size can be from 10 kilobytes to a few 100 kilobytes, you see, it is how small this is. So, since it is small and modular it is easier to port, it is easier to maintain and easier to extend.

While the monolithic kernel since its size is very large, because everything else you are trying to everything else you are trying to run or execute in the kernel so its size is very large, it is non-modular. And hence, since it is size is large, it is difficult to port, difficult to maintain an extent. But here, since in microkernel approach, the kernel is a modular and small, so it is easier to port them, easier to maintain them and it easier to extend them.

There is no such agreement that on what should be there in the kernel, what should not be there no such agreement, but typically, the process management, memory management and inter process communication IPC these activities are carried out in the kernel. So, this is how the microkernel approach looks like.

(Refer Slide Time: 26:37)

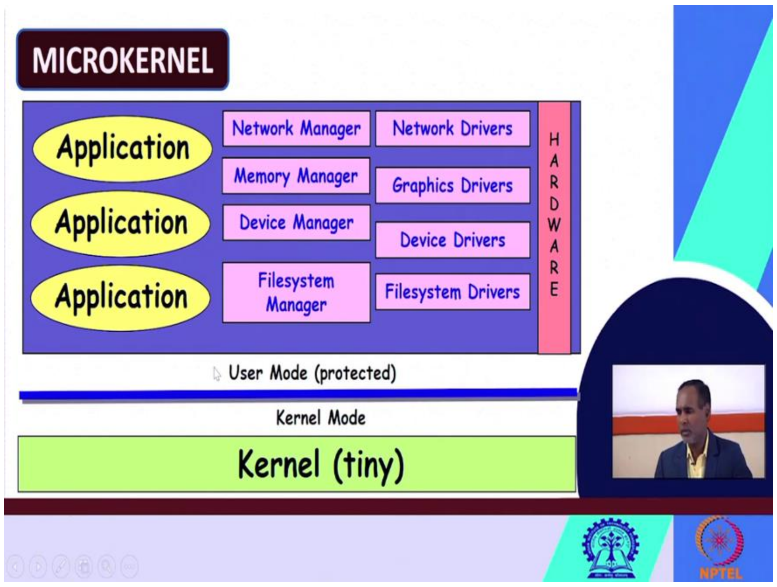


Now, let us quickly look at the, this what, architecture of the monolithic kernel. Then we will see the architecture of the microkernel. So, if you will see this monolithic kernel there are two modes I have already told you, the user mode, we can it is the protected mode and the kernel mode. In monolithic kernel I have already told you, all the tasks, all the services, they will be provided where, they will be occurred or they will be executed in the kernel mode.

So, see, all the services like file systems, process manager, memory manager, device driver, network drivers, graphics driver, terminal driver, everything else put in the kernel mode, that is why the kernel is so big and so it is voluminous. And in a user mode only applications will run. The applications they can interact, they can communicate with the different services in the kernel mode.

And, of course, at the bottom, the hardware are there, the kernel mode services can interact, can communicate to the hardware through the hardware interface layer. This is how the monolithic kernel it works.

(Refer Slide Time: 27:42)

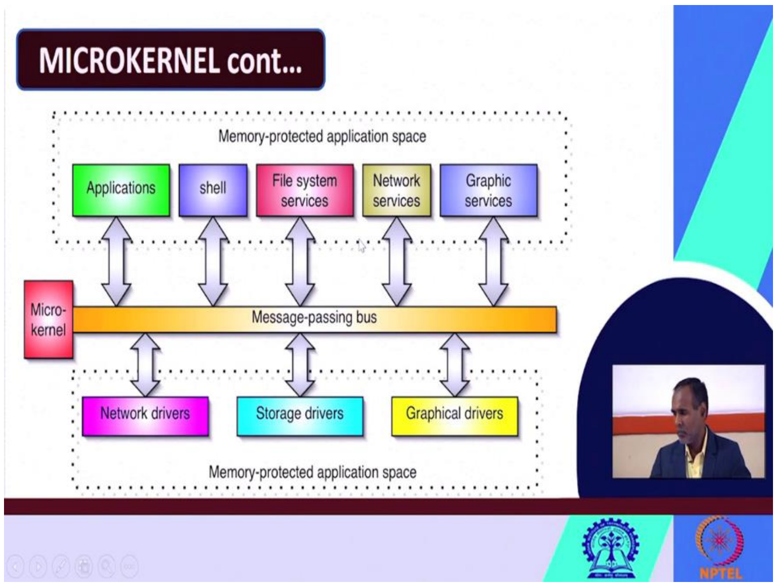

MICROKERNEL

Now, let us quickly look at the microkernel. How does it work? I have already told you. Here, the bare minimum things you will put in the kernel, all other things you will try to execute where, in the application, in the user mode. So, here you can see that, the upper portion is the user mode, this is the protected mode. This tiny person is the kernel, this is the kernel tiny portion.

So, only maybe, as I have already told you like this memory management and this inter-process communication etc they can be hard to perform at this kernel. All other things you have to execute, you have to run in the user mode through the applications. Like the network manager, or memory manager, device manager, file system manager all the drivers they will be there where, in the application space.

In the user mode through applications all those services can be provided. They can be executed. And here they hardware is there, and the kernel is very tiny. And only what services maybe inter-process communication or memory management can be done in the kernel. So, this is how, this microkernel architecture looks like.

(Refer Slide Time: 28:49)

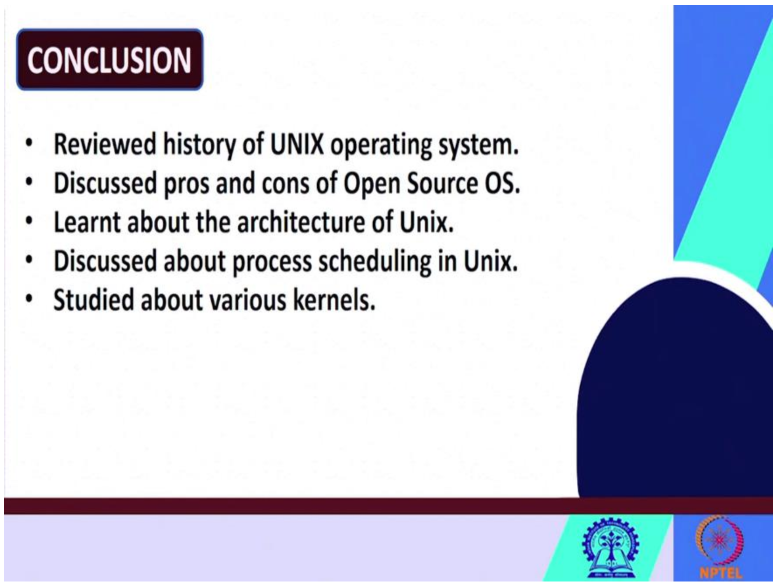


MICROKERNEL cont...

So, there is a more detail. I have given how the communication takes place in a microkernel architecture you see. So, here, this is the microkernel it is a tiny one, and you will see all the applications, all the services and the different drivers they communicate through a message passing bus. So, all the applications and the services and the storage drivers.

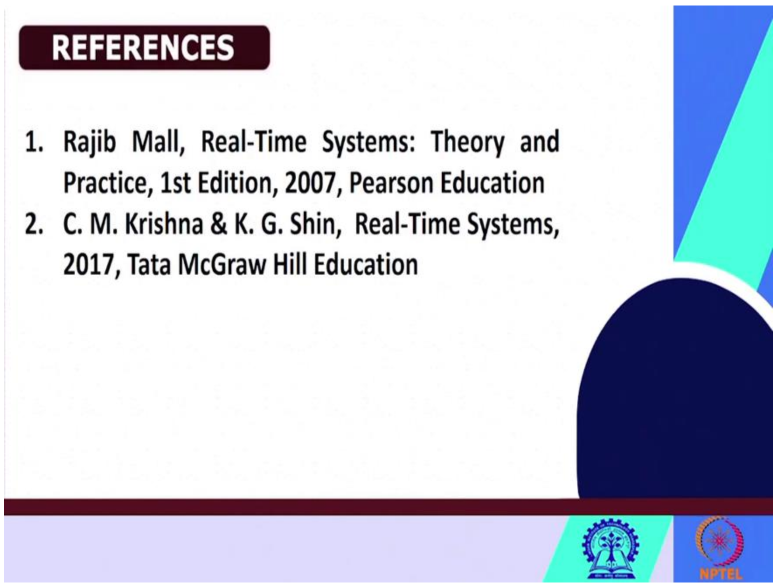
All the applications, services and the different drivers they can communicate among themselves through the message passing bus. Also, they can communicate or they can interact with the microkernel through the message passing bus. So, this is how, this microkernel architecture in the microkernel architecture the communication takes place. This is something about the microkernel architecture.

(Refer Slide Time: 29:40)



CONCLUSION

- Reviewed history of UNIX operating system.
- Discussed pros and cons of Open Source OS.
- Learnt about the architecture of Unix.
- Discussed about process scheduling in Unix.
- Studied about various kernels.



REFERENCES

1. Rajib Mall, Real-Time Systems: Theory and Practice, 1st Edition, 2007, Pearson Education
2. C. M. Krishna & K. G. Shin, Real-Time Systems, 2017, Tata McGraw Hill Education

So, today, we have discussed what? The review, we have reviewed the history of Unix operating system. How the Unix operating system was evolved. Then we have discussed the pros and cons of the open-source operating systems. We have seen some of the success to the use of open-source softwares open-source operating systems such as your Apache and the Perl, etc. We have learnt about the architecture of the Unix.

We have discussed about how process scheduling it happens in Unix. We have studied about two important kernels that is the monolithic kernel and the microkernel. I have also given. I have also shown you the architecture of the monolithic kernel and the microkernel. These are very fundamental things. May not be only restricted to these books.

Any fundamental books on operating system or from the internet you can get the details of the items. Thank you very much.