

Information Security-3
Prof. V Kamakoti
Department of Computer science and Engineering
Indian Institute of Technology Madras
Basics of Unix and Network Administration
Operating Systems Introduction
Mod01, Lecture 04
Module 4: OS Security Issues

So welcome to module 4 that we have understood something about security and different functionalities of the operating system. So the services that the operating system need to take care and we concluded last module with security issues, we distinguish between protection and security and in this module, we will talk more about OS security issues. Please, understand that this course is essentially server and a network administration course for security engineers. So we will emphasize more on the security aspects of operating system.

(Refer Slide Time: 0:49)

How does OS "Protect" itself?

- **Dual-mode** operation allows OS to protect itself and other components
 - OS must not get corrupted
- **User mode and kernel mode**
 - **Mode bit** provided by hardware
 - Distinguish when system is running user code / kernel code
 - Some instructions designated as **privileged**, only executable in kernel mode
 - **System call** changes mode to kernel, return from call resets it to user.
- **Supervisor/Kernel mode**
 - Can execute all machine instructions
 - Can reference all memory locations
- **User mode**
 - Can only execute a subset of instructions
 - Can only reference a subset of memory locations

Operating Systems

NPTEL

So typically how does an operating system protect itself? So the operating system will basically give two modes of execution what we call as a dual mode. In a dual mode, there is something call the user mode, in which the user process, the application software. We will do some activities, right. So and basically in the user mode, a process which is a program in execution when it is executing in the user mode, we will have restricted privileges, it cannot go and touch certain parts of the operating system or it cannot go and touch certain part of the memory. So it cannot go and change certain values and there is another mode, which we call as the kernel mode wherein from the user mode we basically execute a system call and come to kernel mode.

In the kernel mode there is more privilege. So then certain parts of the operating system can be checked out. So the operating system, so it will be in two modes of execution when a user process starts executing then it will be in the user mode. So that user process cannot come and touch anything within the operating system, right and when certain important operations have to be taken, which are very close to the hardware then there is a switch into the kernel mode and in the kernel mode certain operations can happen, for example, let us take a very simple hallo world program, some part of the program say, there is a printf statement there are other statements other than the printf. All those things are basically executed in the user mode and when I do a printf then there is a system call basically which will go and touch the device driver of the monitor and the graphics adapter and then the hallo world will go and print there and that part is basically a kernel mode.

So when a process executes some part of it will be executed in the user mode and whenever it does a system call we go to a kernel mode and starts executing in the kernel mode. So to distinguish between these two modes, let us look at supervisor slash kernel mode, where one can execute all machine instructions, one can reference all memory locations and there is a user mode where only a subset of the instructions can be executed and only a subset of memory locations can be accessed.

So if we go back to the information security 2 course on architecture layer to information security, there we have introduced four privilege levels or four modes, privilege 0, 1, 2, and 3 in the context of the X axis (0)(3:15) architecture and we have explained how this type of intra-process protection can happen and how you switch from one mode to another modes. There is a dedicated assignment on task switching which basically talks about how we can go from 0 to 1 to 2 to 3 and back 3 to 1,0 and how one can be at a level three that is probably the user mode and go and execute a program to that is level 1. How are interrupts handled? So are the interrupts in user mode or are they in kernel mode? Can I execute an interrupt service routine in the user mode? Can I execute it at a kernel mode? All these answers are available in our architectural layer to information security 2 course, I urge (0)(3:59) the readers to please look at the assignments specifically with respect to what we called as task switching in that course.

(Refer Slide Time: 4:06)

Kernels and Memory

3

- The part of the OS critical to correct operation (trusted software)
- Executes in supervisor mode
- The trap instruction is used to switch from user to supervisor mode, entering the OS

The diagram illustrates the mapping between processes and memory spaces. On the left, two ovals represent 'User Process' and 'Supervisor Process'. On the right, a vertical rectangle is divided into two sections: 'User Space' (top) and 'Supervisor Space' (bottom). Lines connect 'User Process' to 'User Space' and 'Supervisor Process' to 'Supervisor Space'. Below the diagram, a small video inset shows a man speaking, and the text 'Operating Systems' is visible.

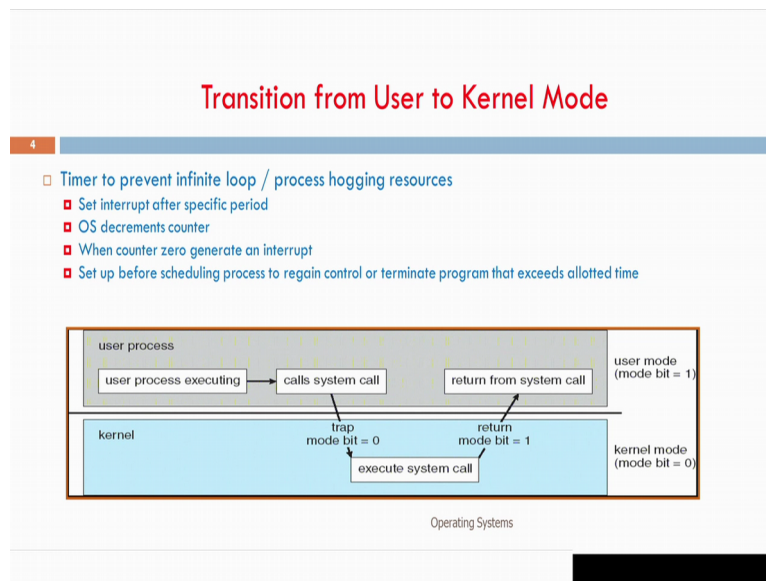
Operating Systems

NPTEL

So just to give glimpse of what we covered in that course, there is a notion of a kernel; there is a notion of a memory. So what do you mean a notion of a memory for a kernel? So in the memory essentially is basically split into two parts, one part is for the user space another is part is for the supervisor space or so when the kernel is executing, can access the entire memory namely a user space and the supervisor space, but when user process is executing, it can handle only the user space and so the supervisor process will work in the supervisor mode the user process will work in the user mode.

There are instructions for example, if we go back to the information security 2 course, there instructions like `int int` etcetera, which are like `LGDT`, `LLDT` load global descriptor table, load local descriptor table many many important instructions that are all what we call as privilege instructions, those can be executed only in the supervisor mode, it cannot be executed in the kernel mode and we could go and reserve memory every bite of the memory can be given a privilege level. So if a memory location has been given a supervisor level privilege then the user process cannot access it. So these are some of the very interesting things that form the basis of protection and security in the operating system context.

(Refer Slide Time: 5:31)



And then there are lot of care that is taken while there is a switch from a user to kernel mode. One important care is that I ask for certain kernel functionality and what happens I go it goes in the kernel mode and never returns back there something that has happened there and the system call does not happen.

So one of the things that many operating system specifically real time operating system does (())(5:55), when we go from the user mode to the kernel mode, they go and setup what we call as counter and view setup some time quantum in the counter and the when the counter reaches zero, immediately it generates an interrupt. This can also happen from user process to a kernel and also, it can happen from a kernel to user process. So as an operating system, I start executing user process and then what happens is that I need to have a response from the system and if the system goes on to an infinite loop and I want to pull it out after some time then I actually set timer and based on the timer interrupt, I pull it out. This is the basis of round robin scheduling.

The other part is that when I am looking at real time processing and there is a user process which is requesting some service and it does not come within sometime, I can still have a timer which looking (())(6:48) it back, it may not take it back to the user process but it can take it back to the another OS routine, which can do some file save of activities. So there is a notion of an hardware timer. It is very very important in the context of operating systems, which can basically be programmed to get quick responses to ensure that responses are quick and in cases the response is not quick then it can take some (())(7:13) actions to see that no major damage is done, because of the lack of response.

(Refer Slide Time: 7:19)

Shorter look -Services of an OS

5

The diagram illustrates the layers of an operating system. At the top is 'User programs', followed by 'User interface', 'System Calls', and 'Hardware'. A vertical label 'Operating System' is on the left. Below 'System Calls' is a row of services: Program Control, I/O, File System, Comms, Error Mgmt, Resource, Auditing, and Security.

- OS provides services to other part of software that resides on the computer through a "system call".
- Each of these services typically are "abstraction" of a hardware resource available.
 - That is how to make use of the physical "bare hardware resource" with the help of software

Operating Systems



So to look at in a nutshell, if you look at an operating system, it gives there is an underlying hardware and that hardware has different things that it do. The hardware has the IO devices, it has it stores files, it needs to be secure, it needs to be auditor, there are multiple resources that needs to be manage, there are lot of error that need to be managed IO devices are there. So what is the operating system do? It actually provides services to other part of software's that recites on the computer through what you call as system call.

So there is user program, the entire setup is to basically run that user program properly and the operating system provides an user interface using, which the user program can call the system calls and what does system call do? It actually provides an abstraction for of the hardware resource available and what are all the abstractions that you see, I can look at mechanism by which I could control the program, there are Io devices I see a file system, there are communications that could be established, error management, resource management, auditing, security everything are abstractions and the system calls can be used to basically go and achieve these abstractions, namely I want to control a program, I use a system call to go and control the program, for example in Unix fork is a very very interesting program control mechanism, which is basically a system call.


(Refer Slide Time: 8:49)

Another look in to the services provided by an OS

6

- **Facilities for program creation and execution**
 - Editors, compilers, linkers, debuggers, etc.
 - Loading in memory, I/O and file initialization.
- **Access to I/O and files**
 - Deals with the specifics of I/O and file formats.
- **System access**
 - Resolves conflicts for resource contention.
 - Protection in access to resources and data.
 - **File-system manipulation**
 - read and write create delete search list files and directories, create and delete files
 - Permission management.
- **Error detection and response**
 - internal and external hardware errors
 - memory error and device failure
 - software errors
 - arithmetic overflow and access forbidden memory locations
 - operating system cannot grant request of application
- **Resource allocation**
 - CPU cycles, main memory, and file storage, and I/O devices to be allocated for process.
- **Accounting(optional)**
 - To keep track of which users use how much and what kinds of computer resources

Operating Systems



So to sum up the operating system provides facility for program creation and execution. It gives you access to IO and files. It provides system access. It also provides lot of resilience in terms of error detection and response. It is responsible for allocating resources that a process a program in execution wants. It is also very important for us to account in many of the systems, which are paid systems, where you pay for the service, there is accountability and then there are for legal reasons we need to know, who has access the system. So there is logging that needs to be done, which is also a under the head of accounting.


(Refer Slide Time: 9:26)

Summary of Services

7

- Program execution – system capability to load a program into memory and to run it.
- I/O operations – since user programs cannot execute I/O operations directly, the operating system must provide some means to perform I/O.
- File-system manipulation – program capability to read, write, create, and delete files.
- Communications – exchange of information between processes executing either on the same computer or on different systems tied together by a network.
 - This is implemented via shared memory or message passing.

Operating Systems



Operating system is also responsible for program execution, for IO operations, for file system manipulation and also, communication between the processes within the particular system or

across systems, in the case of a network distributed operating system and (9:40) these are all some of the services, which the operating system provides for which we have understood to some level so far (9:47).

(Refer Slide Time: 9:48)

Summary of Services

- Error detection – ensure correct computing by detecting errors in the CPU and memory hardware, in I/O devices, or in user programs.
- Additional functions exist not for helping the user, but rather for ensuring efficient system operations.
- Resource allocation – allocating resources to multiple users or multiple jobs running at the same time.
- Accounting – keep track of and record which users use how much and what kinds of computer resources for account billing or for accumulating usage statistics.
- Protection – ensuring that all access to system resources is controlled

Operating Systems

In addition the operating system should have error detection, it should have additional functions exist not for helping the user, but rather for ensuring efficient system operations, for example, thermal management today, it is a very very interesting example of additional functionality, when a chip heats up then automatically some of the CPU cores (10:08). So if we have a multi cores system, some of the cores are put into sleep and then resource allocation as we have seen accounting protection etcetera. So these are all some of the services that the operating system provides.

(Refer Slide Time: 10:23)

The slide is titled "How do we request the services?" in red text. It contains a list of two methods for requesting services from the OS:

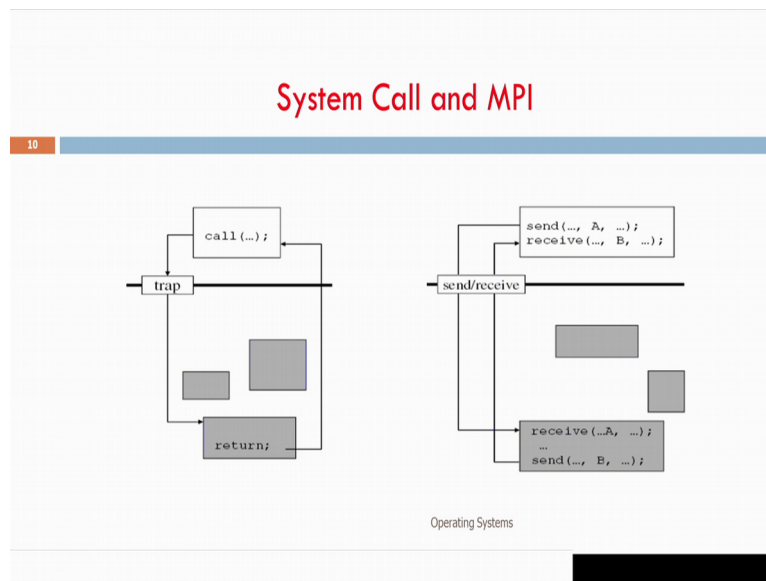
- **System Call**
 - Process traps to OS Interrupt Handler
 - Supervisor mode set
 - Desired function executed
 - Returns to application
- **Message Passing**
 - User process constructs message indicating function (service needed)
 - Invokes send to pass message to OS
 - Process blocks
 - OS receives message
 - OS initiates function execution
 - Upon function completion, OS returns "OK"
 - Process unblock...

At the bottom right of the slide, there is a small video inset showing a man in a striped shirt speaking. The text "Operating Systems" is visible at the bottom center of the slide.

Now how do we request for the services are two ways by which one can request for the services. In many of the operating system today, this basically done through system calls. What are system calls? It is basically a process when it is executing, it needs to get some service from the operating system. It executes a trap and the moment I reviewed a trap we go into an operating system interrupt handler as I told you every trap will have a number we go to an interrupt table and that particular interrupt service routine starts executing and after that interrupt service is over, we come back to the application. So this is a notion of a system call.

There is also something called message passing. What you mean by message passing? The user actually constructs a message, which it wants to deliver to the operating system and it invokes `send()` to pass the message to the OS and once it sends the message to the OS, it gets blocked, it starts waiting. So some other process can start working and then the OS actually from its end it receives the message from this block then it execute some functions and after the function is completed, returns back message. Now the process which is blocked again gets unblocked and starts executing. So this is another way of getting a service done from the operating system. So one is call the message passing another is called system calls.

(Refer Slide Time: 11:47)



So this basically graphically, it teaches you what is the difference between a system call. Systems call nothing but a call of some nature, which creates a trap, which goes as an interrupt and you return from that. In a messaging passing interface, which we call as MPI we send a data from the user process to a kernel process, the kernel process reads the data after finishing some computing, it sends back an answer and which is received by the system call and immediately, it starts executing. So these are two ways by which one can get operations done out of an operating system.

(Refer Slide Time: 12:23)

System Calls

- System calls provide the interface between a process and the operating system. These calls are generally available as assembly language instructions
- Some systems also allow to make system calls from a high level language, such as C.
- Three general methods are used to pass parameters between a running program and the operating system.
 - Pass parameters in registers.
 - Store the parameters in a table in memory, and the table address is passed as a parameter in a register.
 - Push (store) the parameters onto the stack by the program, and pop off the stack by operating system.

Operating Systems

The slide includes a small video inset in the bottom right corner showing a person speaking.

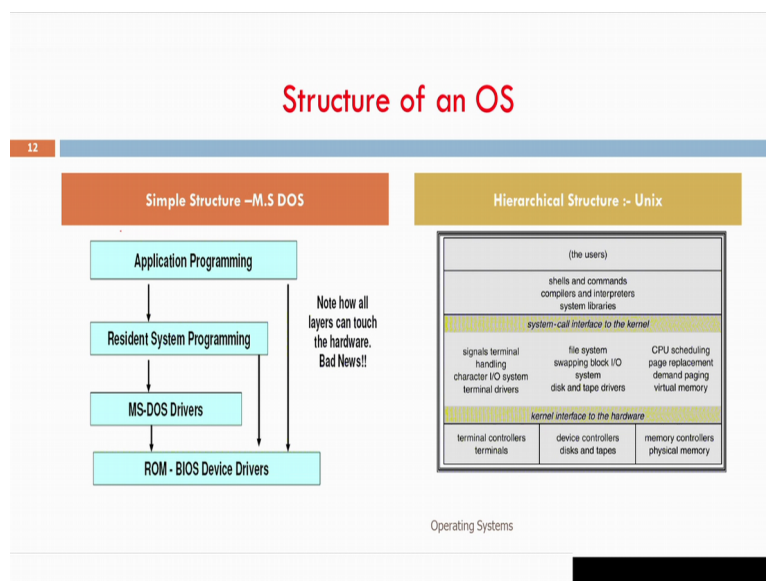
So what are system calls? System calls basically or as I told you like `printf`, `scanf` etcetera `malloc` is an example of a system call, allocate memory and whatever you see on the libraries

of the `stdlip` or `stdio`. These are all basically system calls and when we do a system call essentially we need to give some parameters, for example, if I want to do a `scanf`, please understand, I have to give the address to which I need to scan the data scan in the data, when I want to `printf` I need to give the value.

So there are some parameters that are passed between the user process and their corresponding system call and that basically happens. There are many ways by which one can pass the parameters, one can pass it through the stack, one can pass it through specific memory locations, even pass it through registers. So depending upon the type of things that we passed depending up on the mechanism that is decide at ((13:14) user process can pass parameters as a part of the system call. So it is like any other one subroutine passing parameter to some other subroutine, the same process can be there, but with lot of security here, right.

So in the information security 2 course, please understand that we had covered in depth about how data is passed between privilege level 1 code and a privilege level 0 code. There are different stacks that are involved and why do we need multiple stacks? All these things have been explained very in detail, in the information security 2 course, which would be of very high importance when you want to appreciate how parameters are passed between an user process and the corresponding supervisor process.

(Refer Slide Time: 14:04)



Now if we take some of the old operating systems like MSDOS, so all the levels namely the application programming and the resident system programming, MSDOS drivers everybody

directly talks to the hardware device driver. They actually talk directly every layer can touch the hardware bypassing the intermediate layers as you see in this particular thing on your left hand side. So but on a hieratical structure like Unix, where you have a layered operating system.

So there are different layers as you see here, there is the undermost layer is a terminal controllers and terminals device controller, device memory controller, these are all the hardware on just top of that hardware as you see, there are kernel interface to the hardware and then then there this is the operating system there, where you have signals and terminals handling character IO stream and IO system and terminal drivers, file system and then CPU scheduling, page replacement etcetera. So that is the operating system. Then there is a system call interface on top of it and then you see, there are shells and commands and compilers and then of course the last one is the users. So you see at least there are four layers that we are looking for, perhaps there is the reason why may be there is reason why a the X axis (()) (15:22) architecture are four privilege levels.

(Refer Slide Time: 15:26)


NOTE

13

- Although some OS (MS-DOS) have simple structure, its interfaces and levels of functionality are not well separated.
- Any OS(Linux , Windows Vista) with hierarchical structure has two important separable parts
 - The system (space/program)
 - The kernel(space/program)

(the users)		
shells and commands compilers and interpreters system libraries		
system-call interface to the kernel		
signals terminal handling character I/O system terminal drivers	file system swapping block I/O system disk and tape drivers	CPU scheduling page replacement demand paging virtual memory
kernel interface to the hardware		
terminal controllers terminals	device controllers disks and tapes	memory controllers physical memory

Operating Systems

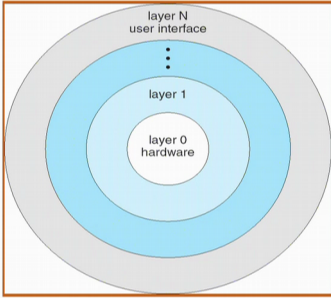


So all the any OS like Linux, windows vista will follow this hierarchical structures has and it has two important separable parts, the system, which is the space slash program, the kernel space which is again the space program. So the kernel as you see in the slide, is comprises the system call, interface and the kernel interface and then we have all the files system, CPU scheduling, memory management all these modules. They form the kernel there.

(Refer Slide Time: 16:00)


Layered OS Structure

14



- The OS is divided into a number of layers (levels), each built on top of lower layers
 - The bottom layer (layer 0) is the hardware
 - The highest (layer N) is the user interface
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers
- Major difficulty: appropriately defining the layers
- They tend to be less efficient

Operating Systems



So this leads us to what we call as a layered operating system, which I have been talking off. So we have layer zero, which is the highest privilege and which is very close to the hardware then top of it layer one with lesser privilege and go on, I can go up to layer n so, but the bigger difficulty here is, how do we define which part of the operating system goes to which layer and what sort of capabilities and user will have on each of these layers and so sometimes, this definition of generation of this layered operating system tend to be less sufficient, because if a layer n code wants to execute on the hardware, it may have to through layer n minus 1 to layer 1. So there could be lot of system calls and stuff like that and lot more of before a program can touch the hardware, there could be several other programs which execute on the same hardware to enable this to go and access that hardware and that causes this efficiency loss.

(Refer Slide Time: 16:59)


Micro Kernel Based OS Structure

15

The diagram illustrates the Micro Kernel Based OS Structure. It shows a 'kernel environment' containing three layers: 'application environments and common services' (top, light blue), 'BSD' (middle, white), and 'Mach' (bottom, grey). Arrows indicate communication between the application layer and the BSD layer, and between the BSD layer and the Mach layer.

- Moves as much from the kernel into "user" space
- Communication takes place between user modules using message passing
- Benefits:
 - Easier to extend a microkernel
 - Easier to port the OS to new architectures
 - More reliable (less code is running in kernel mode)
 - More secure
- Detriments:
 - Performance overhead of user space to kernel space communication

Operating Systems



One of the interesting thing that has come up in the recent time is this microkernel based OS structure. So I have a kernel, I take that kernel and lot of things that are very close to the user space we push it into the user space and essentially then what remains is a very part of this kernel which we called as a microkernel. Now the microkernel, so since all the other parts have been pushed, the microkernel actually becomes much smaller and it becomes easy for us to port it across multiple architectures and it is also very much possible that I can go and do certain formal verification of these microkernels, because it is all very small in size and I can ensure that decide (())(17:40) function not only that the decide functionalities executed by the microkernel, but functionality that we do not decide that will not be executed. This answering to this second question is to very difficult.

Will the software execute what it is supposed to do? The answer could be easily done we can give an answer for that easily. will the software can you ensure that the software will not do what it is not supposed to do that is a very difficult question to answer, because we do not know what it is not supposed to do, I know what it is supposed to do, but I do not know what it is not supposed to do and so for answering that type of question a more formal approach is necessary and if I have a huge kernel, I cannot do any formal approach or formal verification on it. If I have a microkernel then one can do a formal verification on that and that makes the notion of a microkernel very very important in the context of building secure operating system.

(Refer Slide Time: 18:32)

Modular OS Structure

16

- Most modern OS implement kernel modules
 - Uses object-oriented approach
 - Each core component is separate
 - Each talks to the others over known interfaces
 - Each is loadable as needed within the kernel
- Similar to layers, but more flexible
 - any module can call any other module
- Similar to microkernel, but more efficient
 - No message passing among modules

Operating Systems

So today if you look at now operating system, there are multiple modules, as you see on the slide you have at least seven modules, 1 for scheduling, 1 for file system, another for system calls, which you can load some we can say, I could have executable formats, I could have streams, I could have other miscellaneous modules. So what is the interesting thing about this modular operating system? I can go and load some module as I need, suppose I do not want some devices are (19:03), I will not go and load the drivers at all (19:05), right.

If I do not want to support certain system calls, I need not load those systems calls. So I can built up a new core of say, for example, it is quite true solar is (19:17) the operating system in the past, which basically give the notion of a modular voice structure in a very big way. So the thing is that I need not load certain one (19:27), for example, (I am) I want to disable USB , we can go and say do not even load the system driver or anything related to USB just do not compile your kernel by it removing all those things that make your kernel extremely secure that can give you much much more guaranty that somebody cannot break (19:46) open by you know the traditional means to get information out of your system. So a modular OS structure is extremely important towards building up a secure OS.



(Refer Slide Time: 19:59)

Virtual Machines

17

- A **virtual machine** takes the layered approach to its logical conclusion
- It treats hardware and the OS kernel as though they were all hardware
- It provides an interface *identical* to the underlying bare hardware
- The OS host creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory
- Each guest provided with a (virtual) copy of underlying computer

Operating Systems



The last thing that we will cover today is about in this module is about virtual machine. What is a virtual machine? Virtual machine is nothing but a software box and many such box could exist on a single hardware what you see in. So what normally we see is that in a normal machine, there is an hardware, which you see on your extreme left in your screen, there is a hardware then there is a kernel and that kernel is executing on the hardware and there be processes that are executing on that kernel and which will be placing () (20:31) the hardware through the kernel interface. So this is so there is only one operating system and this hardware is dedicated to this operating system.

Now you see what is there on figure B that is on a middle of the slide. Now you see, there is single hardware then there is a virtual machine implementation like, there are multiple virtual machine, KVM is an freely available virtual machine today, you implement a virtual machine layer and what will that virtual machine layer give you? It will give you what you call as boxes virtual boxes, which we call as VM1, VM2, VM3 I can create n such boxes and on each one of this box I can run different operating system. So the kernel on VM1 can be windows, kernel on VM2 can be you know Linux etcetera.

So I can run multiple operating systems on this and in each of these operating systems, I can run processes. So now the processes corresponding to VM1 thinks that the entire hardware is dedicated to it, processes corresponding to VM2 thinks that the entire hardware is dedicated to it, but () (21:33) it will not even know that VM1 exist. Similarly, VM3 thinks all the processes there we will use we will think that the hardware is dedicated to it; it may not know that VM2 exist or it may not know that VM1 exist. All the processes in VM1 will execute in

isolation isolated from the processes of VM2 and VM3. Similarly, VM2 processes will be isolated from VM1 VM3 and VM3 processes will be isolated from VM1 and VM2. This is the notion of a virtualization and this is suppose to in practice also improves the hardware utilization.

Now one of the important thing that comes out of this is, now again I am shearing resources. So there should not be the isolation that I have talked should be perfect isolation should be provable isolation such that a process in VM2 should not be in a position to go and leak information about generated or used by process in VM1. So there should be an complete proved isolation between these VMs, which are executing on the same hardware. The real challenge here is that all the hardware resources are sheared between VM1, VM2 and VM3, right and with that shearing we need to ensure that there is no information leakage, which makes the whole thing a very complex (22:54).

(Refer Slide Time: 22:55)

Virtual Machines

(a)

(b)

- The resources of the physical computer are shared to create the virtual machines
- CPU scheduling can create the appearance that users have their own processor
- Spooling and a file system can provide virtual card readers and virtual line printers
- A normal user time-sharing terminal serves as the virtual machine operator's console

Operating Systems

So that is what we have mentioned here. So today virtual machines are very very important today what they form almost every data center in the world would have at least one insulation of a virtual machine. Virtual machines not only gives notion of more efficient hardware utilization, it also helps in what we call as consolidation of applications or consolidation of your server and that way virtual machines are going to be extremely crucial in days to come (23:25) with this, we complete this module 4, we will talk more about operating system from the process and file management in great detail in the module. Thank you.