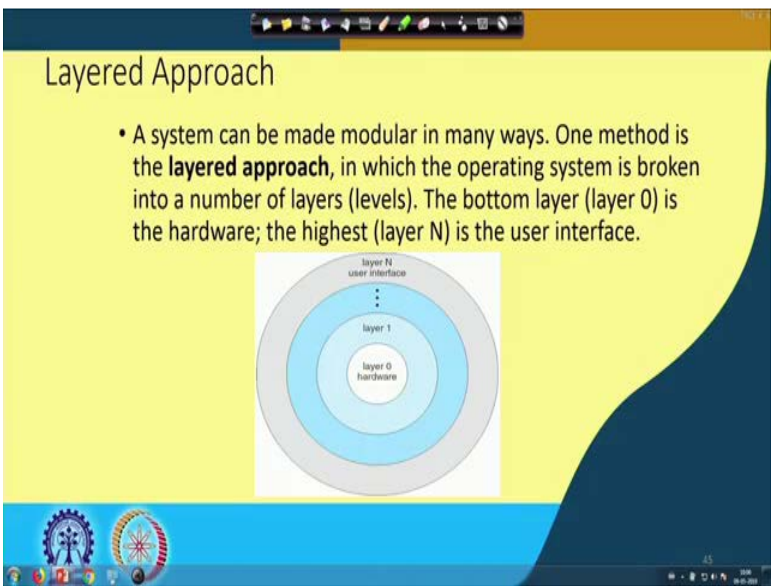
Operating System Fundamentals
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 11
Operating System Structures (Contd.)

In our last lecture we were discussing on modular approach for operating system design.

(Refer Slide Time: 00:29)



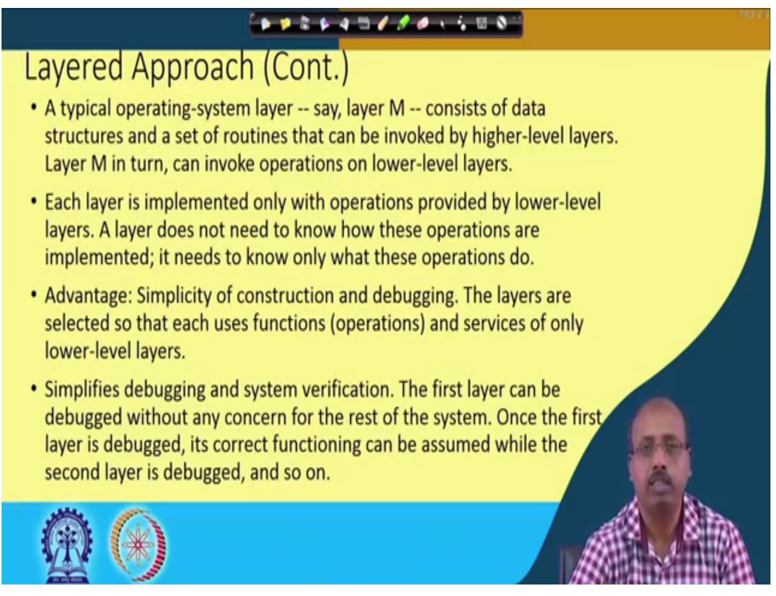
So, one possible way of doing that modular design is known as the layered approach so, in which the operating system is developed in terms of layers. So, a lower level lower that actually is close to the hardware and as we go higher and higher in the levels. So, it is more it is the level at which users will be interacting.

So, in summary you can say that a system can be made modular like this one is the layered approach, where the hardware is the bottom most layer. So, layer that is layer 0 and this bottom layer above that we have got different layers. So, on top of this basic hardware layer, so we have got layer one where we have got the software that can directly talk to the individual devices. So, which are commonly known as the basic input output services of the operating system. So, that talks to different hardware the modules that we have in the system.

Then on top of that layer one so, another layer 2 will be made so, that will be utilizing those bio services for doing some operations at a slightly higher level. For example, in a bio service we can have a facility to display single character on to the screen and at a higher level so, we can utilize that function to display string of characters or maybe we will have at a bias level 1 5 1 routine that reads the character from the keyboard and that can be put at a higher level. So, that can read a sequence of keys from the keyboard.

So, this way we can define hierarchal level of this operations and the lowest layer so, that is the layer one which is closest to the hardware. And, as we go high and high in this hierarchy so, at the top most layer we have got this user interfaces that has got modules that can talked directly to the users. So, that maybe in terms of these editors, compilers, linkers, loaders, like, so they are the next higher level in though the highest level module that you have. Given the database packages that we may have so, any application software that we have so, that will consist of this layer N user interface.

(Refer Slide Time: 02:38)



So, a typical operating system layers so, at layer M it consists of data structures and a set a routines that can be invoked by higher level layers. So, this is what I was telling so, at lower level you will implement some data structures and routines that are invoked by the higher level layers. So, layer M in turn can invoke operations at lower level so, that is the complete hierarchy. So, each layer is implemented only with operations provided by

lower level layers. So, it will not try to do a step jumping or it will not try to bypass a lower layer and directly talk to the lowest level.

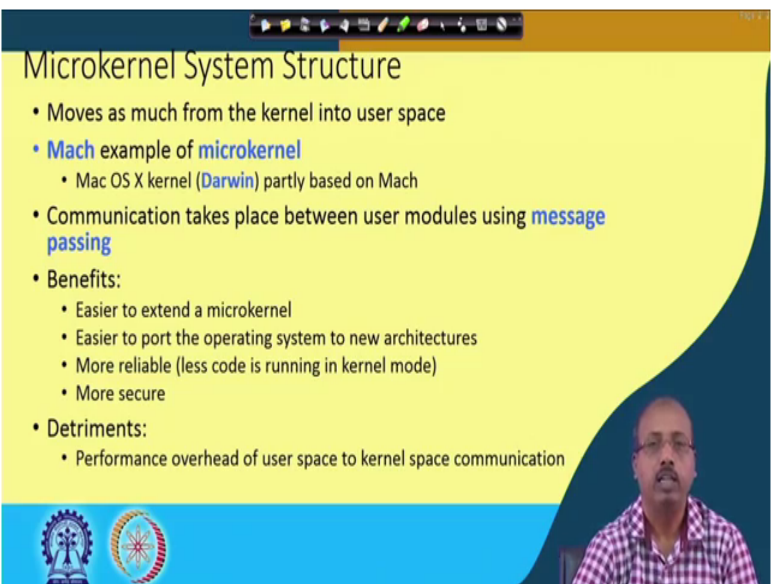
So, like that it for example, if you are developing a routine for displaying a character string. So, we will be using the lower level routine that is available for displaying a characters. So, we will not send directly the characters to character stream to the basic display unit. So, layer does not need to know how these operations are implemented, needs to know only what these operations do. So, that is that differentiation between interface and implementation.

So, at a higher level when you are looking into lower level routine so, we are just looking into the interface part and in the implementation of those routines so, that is done at a lower level. So, if you modify the implementation the higher level routines are not affected because interface is not modified. Similarly, if you modify the interface not the implementation part, then the higher level they should know that the interface has been changed. So, accordingly that those routines can be designed; so, advantage that we have is the simplicity of construction and debugging, the layers are selected.

So, that each uses functions and services of only lower level layers. So, because it because of this modular structure so, your debugging becomes easy. And simplifies debugging and system verification so, first layer can be debugged without any concern for the rest of the system because the hardware layer. So, we can have some hardware level tests that can verify that the hardware is working correctly. On top of that hardware so, the layer 1 software so, it can be tested by checking whether those routines are working in the context of operations by the hardware. So, once the first layer is debugged. So, based on that we can it we assume its correctness and based on that we try to debug the second level then that one go on.

So, whenever we are working with whenever we have debugged layer N for layer N plus 1 we will assume that layer N up to layer N everything is ok. So, we try to test the functionalities that we have at layer N plus 1.

(Refer Slide Time: 05:18)



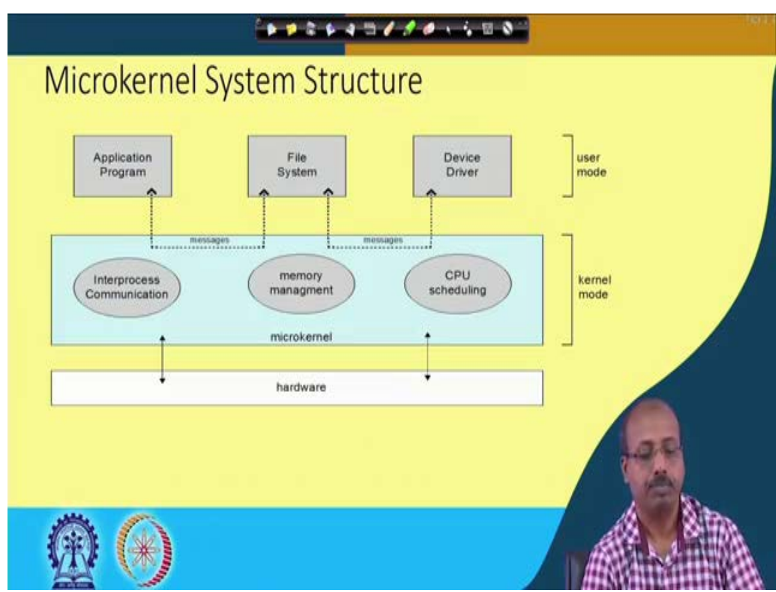
So, Microkernel system structure so, what that, this particular approach so, it moves as much from the kernel into the users space. So, kernel is made very simple and most of the operations that are moved to the users space. So, the advantage that we get is when a program is executing. So, for many of the operations it need not go to the kernel mode. So, kernel mode maybe the kernel mode is implemented in such a fashion that at one point of time only one process maybe in the kernel mode of operation. So, if multiple processes they try to enter into the kernel mode so, there will be blocking. Now, if most of the functions are moved to the user space then this kernel switching will not be required. So, the process blocking will be less.

So, Mac is an example of microkernel so, Mac OS X kernels so, partly based on Mac so, this is that is there. Communication takes place between user modules using message passing. So this simple message passing mechanism is used for communication between modules. The benefit that we get easier to extend a microkernel, because microkernel is very simple so we have to so, it is structural it is simple so extending becomes easy. Easier to port the operating system to new architectures because the microkernel parts that actually talks to the hardware. So, if you are putting onto a different set of hardware only this microkernel part needs to be changed so, that way the portability becomes simple.

Then more reliable because less code is running in kernel mode so, you do not have to you can be sure about in the kernel mode there are not many processes most of the time there will be only one process. So, we do not be we need not be bothered about concurrency synchronization and all. And security is also ensure because if we allow only one process inside the kernel mode when naturally there will be if the problem for of inconsistency will be less so, the security that is taking care of.

Detriments, so performance overhead of user space to kernel space communication so, naturally users space to kernel space. So, if you want to switch then lot of information are to be sent so, that is there.

(Refer Slide Time: 07:35)

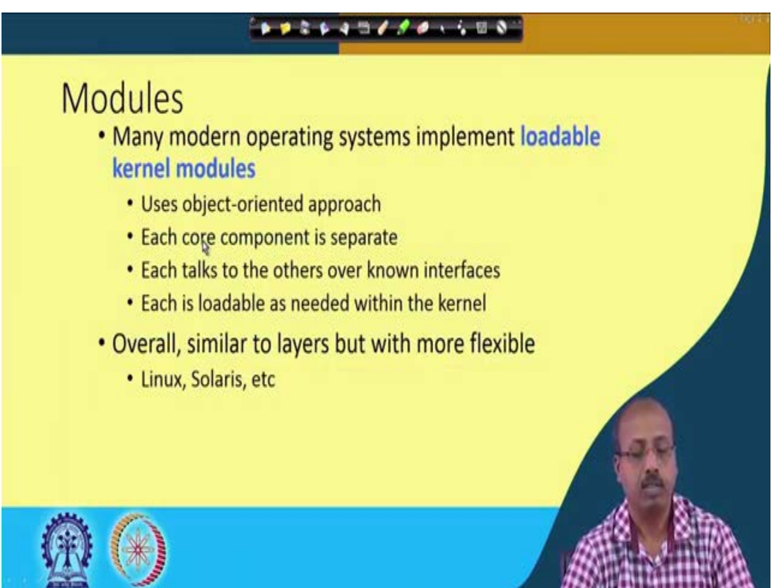


So, this is a typical structure of a microkernel structure. So, basically on top of hardware we have got microkernel and microkernel it has got inter process communication primitives, memory management and CPU scheduling policy. So, they are actually consisting of this kernel mode of operation and other operations like application programs, file system, device driver so, they are all run they will be running in the user mode only.

In a normal kernel what happens is that many of these functions from this file system device driver etcetera. So, they are moved into the kernel mode, but here to make the kernel simple. So, they are put on to a different modular, they are put into the user mode only and most of the operations are done in the user mode and only the very important

part they have been kept in the kernel mode. However, the messages when the when you try to send messages between this subsistence. So, that message is communicated via this inter process communication primitives so, that goes to this microkernel. So, communication part is through the through the microkernel. So, that is definitely an overhead that we have.

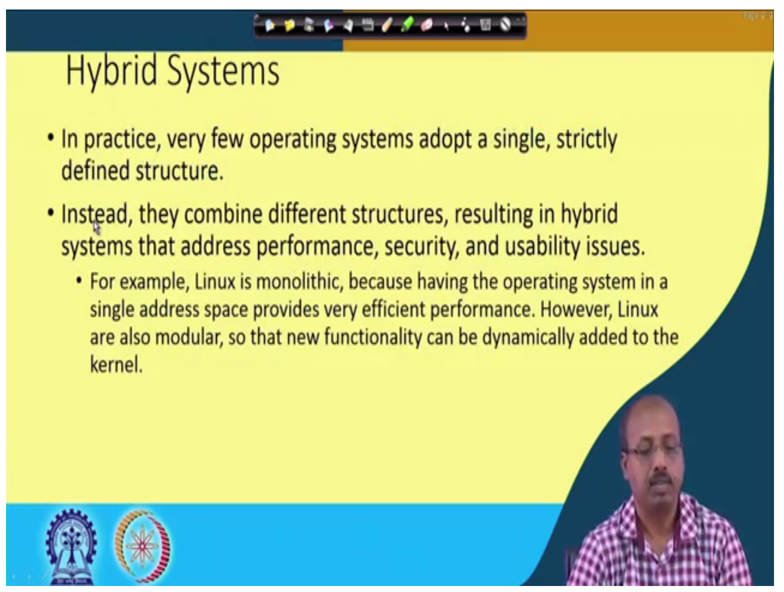
(Refer Slide Time: 08:46)



Many modern operating systems implement loadable kernel modules. So, loadable kernel models means as and when required the kernel modules will be loaded. So, entire kernel is not loaded into the main memory at the time of booting. So, it uses object oriented approach so, there is the object oriented approach means we have got interface and implementation part different inheritance and all those properties object oriented approach so, that they are used in this kernel design.

Each core component is separate, now each talks to others over known interfaces and each is loadable as needed within the kernel. So, whenever we need to load it so, then it will be loaded. It is similar to layers, but with more flexibility so, Linux, Solaris, so, they are based on this loadable kernel modules policy.

(Refer Slide Time: 09:36)

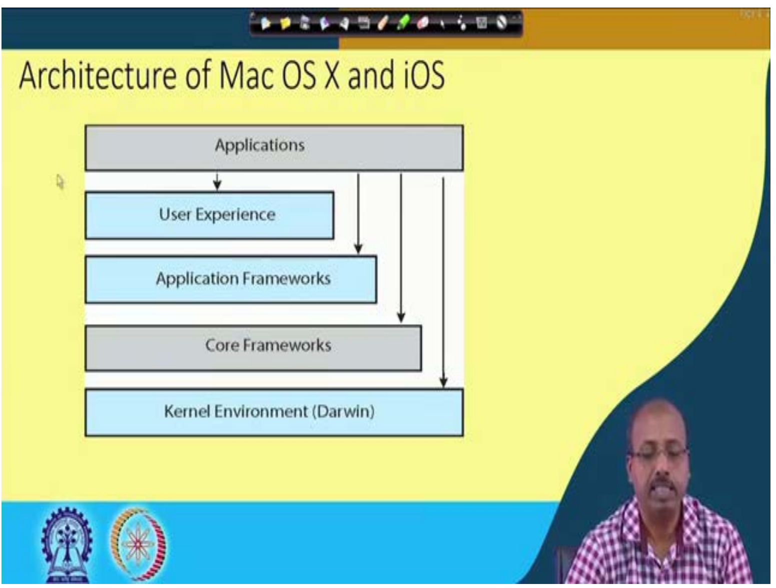


There are hybrid systems very few operating systems adopt a single, strictly defined structure, because after all operating system is designed by human being and this it is very difficult to follow a single design paradigm throughout the entire part. So, it is varies from person to person and definitely depending upon the requirements of the operating systems many a time we have to do a choice between the alternatives within the same operating system implementation.

So, instead of going for a single strictly defined structure we can combine different structures resulting in hybrid systems that will have effect on the performance security and usability of the issues, usability issues. So, for example, Linux is monolithic because having the operating system in a single address space provides very efficient performance. However, Linux is also modular so, that new functionality can be dynamically added to the kernel. So, you can have your own routines that will act as different that will act as a replacement for the standard Linux kernel routine.

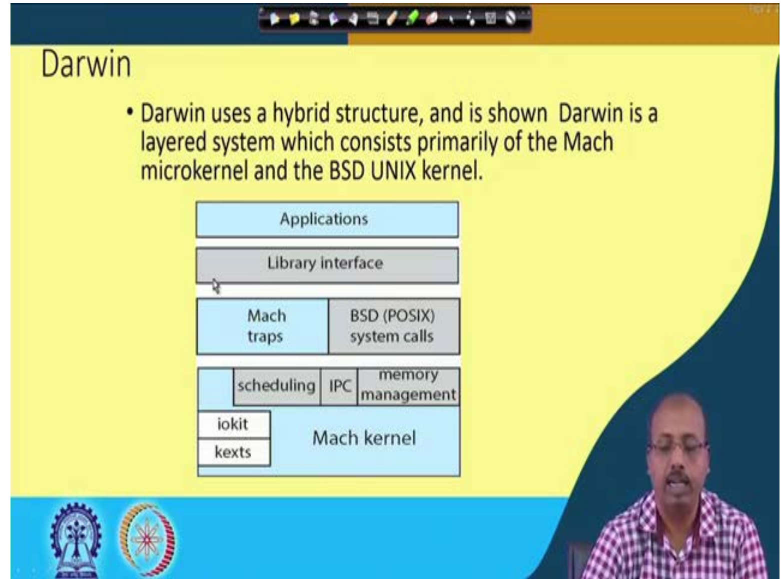
So, that can be done so, that way it can be customized, but at the same time Linux is monolithic because entire operating system is a single piece and they are that is put into the single address space. So, in this way you can find the mix of different design philosophies.

(Refer Slide Time: 11:08)



Now so Mac OS X and iOS, so these are so, kernel is there on top of that, we have got core frame works on top of that, we have got application frameworks, then on top of that you have got user experience and then the applications at the top level. So, applications they can talk to all other layers, but these individual layers of they will be talking to the next lower layer for doing the interaction. So, this is architecture of Mac OS X.

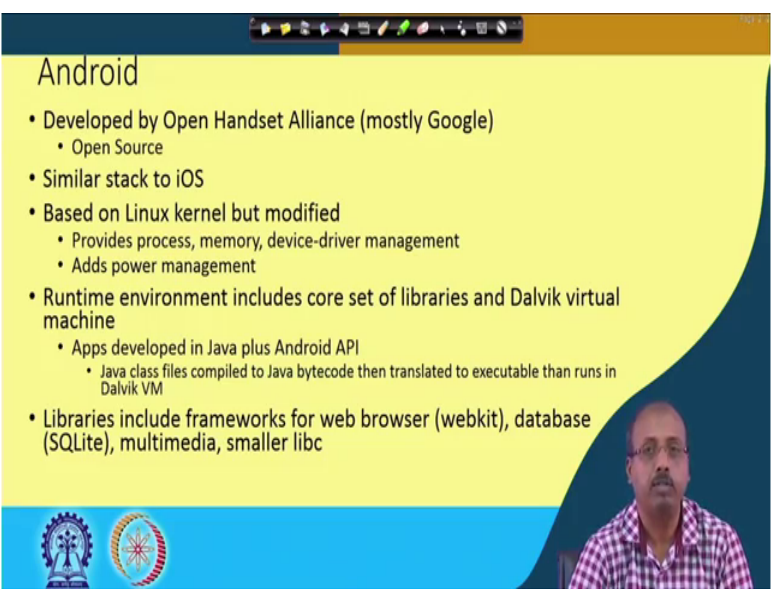
(Refer Slide Time: 11:34)



Then Darwin , it is a hybrid structure the kernel part and it and shown Darwin is a layered system which consists of primarily the; consist primarily the Mach microkernel and BSD UNIX kernel.

So, we have got this type of a mechanism so, this is the Mach kernel, on top of that we have got scheduling IPC and memory management units. So, you have got mach traps and BSD for system call the library interface application, apart from that there are some small models in the mach kernel iokit and kexts. So, these are for some IO handling IO processing and all. So, this is the very simple system so, that is again a hybrid system.

(Refer Slide Time: 12:18)



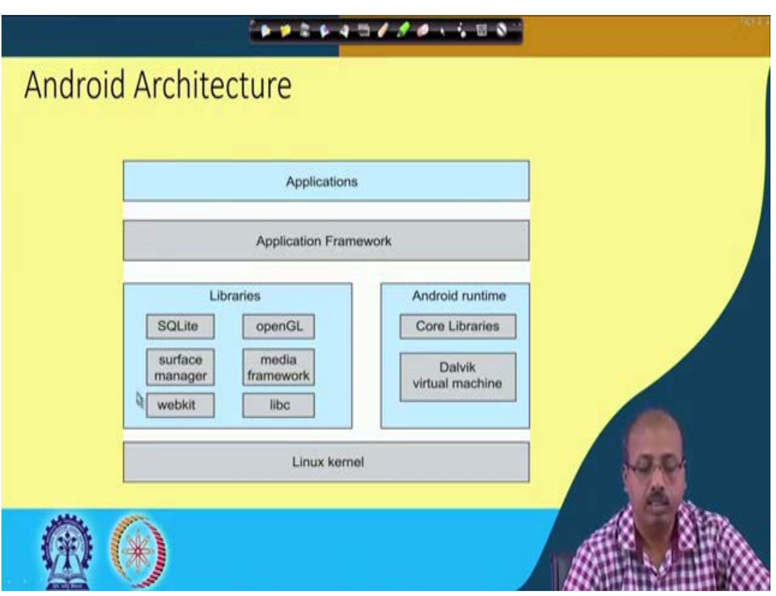
So, Android as we know it is developed by Open Handset Alliance mostly by Google. So, open source similar to similar stack iOS so, iOS stack we have just seen. So, this Android OS stack also like that, based on Linux kernel, but it is modified it provides process memory and device driver management and it adds power management. So, this because for Android most of the time it is working on these mobile phones and all via power is very important issue. So, we cannot have high power consumption.

So, power management so, whenever some hardware part is not use so, it has to be shut down and all. So, this power management is a very important issue. So, Linux kernel has been modified in Android to take care of this power management. Runtime environment includes core set of libraries and Dalvik virtual machine. So, this is so, the apps will be developed in apps are developed in Java plus Android API; Java class files complied

with java byte code and then translated to executable and then runs on the Dalvik VM. So, these are the standard Android development environment.

Libraries that we include are the frameworks for web browser that is webkit, database SQ Lite, multimedia, smaller libc. So, these are actually some simplified version of standard as standard so, modules and softwares that we have in say for example, in Linux. So, that way it helps in making the operating system lighter so, that it can be put on to this mobile devices.

(Refer Slide Time: 13:58)



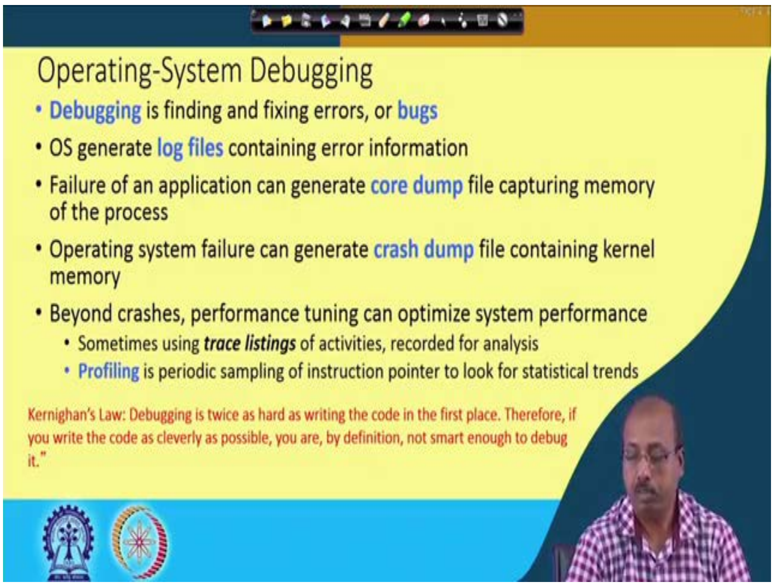
This is the Android architecture. So, we have got at the lowest level we are got the Linux kernel, on top of that we have got a set of libraries, SQ Lite, open GL, surface manager, own media, and it framework, webkit and libc. So, these are actually some mechanism by which we can handle database graphics and this multimedia and all so, these are there.

On the other hand for Android runtime you have got core libraries and the Dalvik virtual machine. So, this machine actually whatever be the underlying hardware so, it is this ultimately this Dalvik machine so, it will virtualize that. So, any application that is developed so, it will be thinking that the underlying processor is a Dalvik virtual machine.

So, it will be having programs translated into that machine and this machine is ultimately implemented by the underlying hardware. So, as a result we can have same software

running on different platforms, only this interface from this virtual machine to actual machine. So, that interface has to be generated and rest of the thing is unaltered.

(Refer Slide Time: 15:04)



Now, how to debug the operating system, so that is also a very important issue; so, debugging is defined to be finding and fixing errors or bugs. Now OS normally generate log files that contain error information and failure of an application can generate core dump file capturing the memory of the process.

So, many times what happens is that if we are running a program and it comes across some situation which is extraordinary, like say I have got a division by division by a variable y. So, say k equal to x by y and somehow the value of y becomes 0. So, that is a divide by 0 error, now how that divide by 0 came, how did the y became 0 become 0. So, that is difficult to understand until and unless we know the actual trace. So, this is actually available in the dump files. So, in the core dump file so, it is there. So, that way this we have often this core dump is produced.

So, operating system failure can generate crash dump file containing kernel memory. So, many times what happens is if system comes across unexpected errors which it cannot handle. So, if the system crashes and when it crashes so, it produces a crash dump. So, that some debugger or some designer can try to figure out what exactly went wrong at the time the system crashed.

Beyond crashes, performance tuning can optimize system performance. So, there are in an operating system there are many tunable parameters as we will see later, one possible one thing is that the block size like whenever the whenever a process or program is accessing secondary storage. So, it is (Refer Time: 16:50) in terms of blocks of data. So, what is the block size? Typical block sizes are 4 kilobyte or 8 kilobyte like that. So, depending upon the hard disc system that we have so, may be the block size different block size will be suitable.

So, what is a good block size so, that is an important issue. Second another parameter that we may need to tune is the time quantum given to individual processes for execution. So, that way if there are large number of users so it may so, happened that we want to give small time quantum to each of them. Now, what is that small time that we are talking about so, what is the exact time duration so, that is also a tunable parameter. So, we have got many tunable parameters and then this we can do that, this tuning can be done.

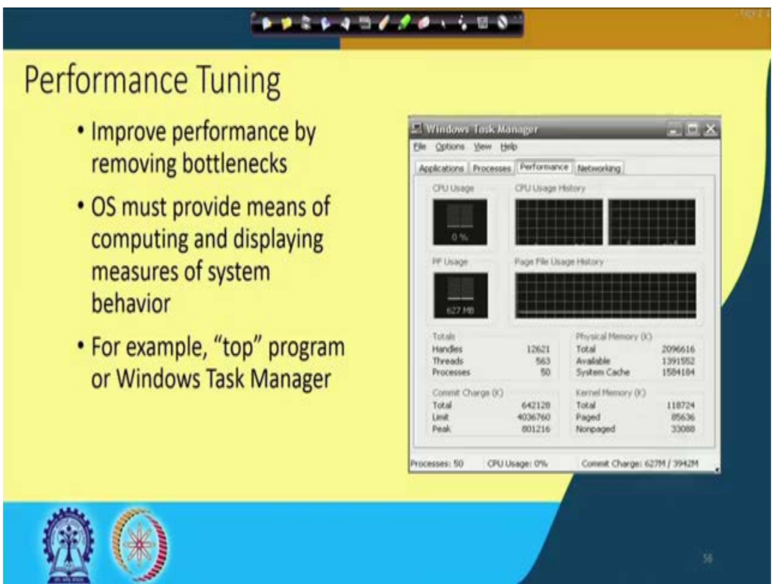
Sometimes using trace listing of activities recorded for analysis. So, you just keep a trace like what was happening for say last one day and based on that we may try to figure out that what can be done for tuning the system better. Profiling is periodic sampling of instruction pointer to look for statistical trends, like which type of codes executed more like is it doing some IO access or is it doing some competition so, that type of profiling can be done. So, that is also taken care that data is taken and later on the somebody may do an statistical analysis to see whether we can do something in the tuning process.

So, there is something called a called Kernighan's law which says that debugging is twice as hard as writing the code in the first place. So, this is very important because somebody else have developed the code and now we are trying to find the what is the difficulty with that code. So, that way it is very difficult and even if it divide by done by the same person who developed the code it is very difficult often to find out the exact the root cause of the situation.

So, if you write the code as cleverly as possible you are by definition not smart enough to debug it. So, this is the point cleverly written code means there will be even many tricks in the code so, understanding that code becomes difficult. So, it is advisable that while designing such a big piece of software like operating system. So, we try to avoid

these clever tricks as much as possible, because that will make the program debugging very difficult.

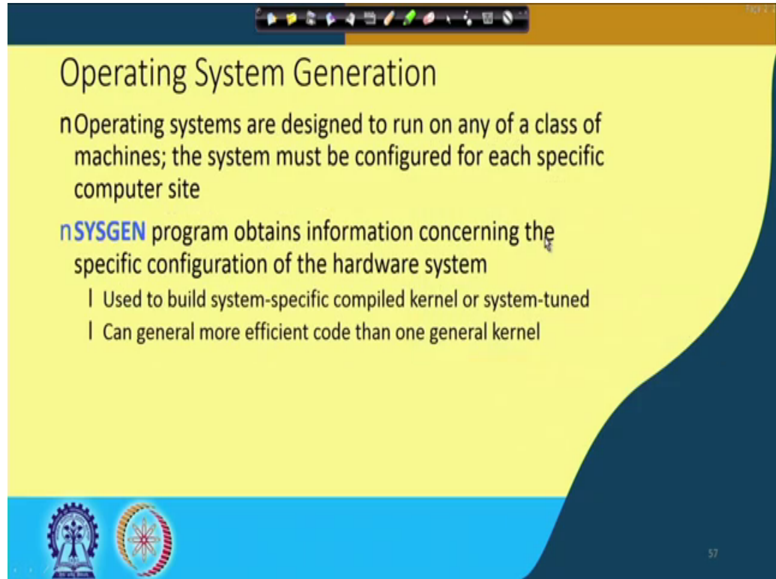
(Refer Slide Time: 19:15)



Then this is an example of performance tuning. So, improve performance by removing the bottlenecks. So, we try to figure out what are the bottleneck like if you are familiar with the Window system this Windows task manager. So, it has got a performance button so, where if you press it so, it will tell you what is the CPU usage, what is the page fault rate and like that so, page file usage so, that way. So, it also gives you the history that tells like what happened over last some time units. So, what like say so, what is the CPU usage like if the CPU usage is low; that means, that there are not much processes in the system.

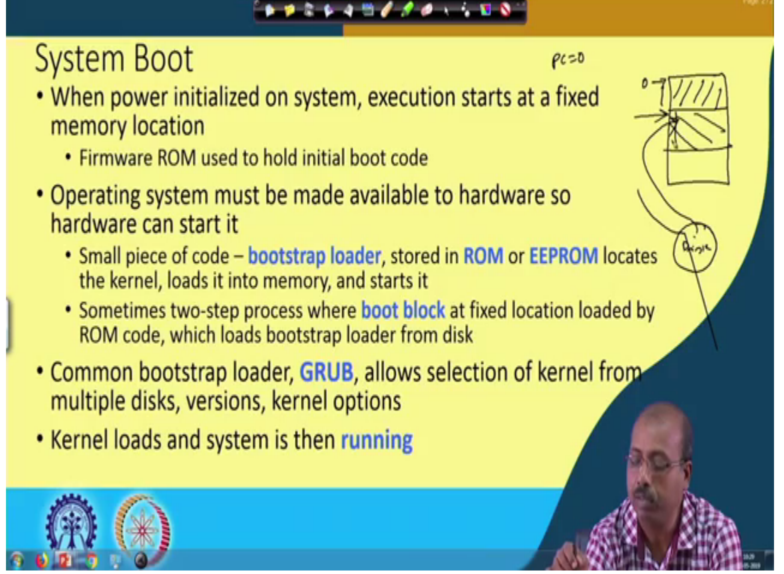
And if here for example, trying to see the reason for this throughput being low then if you see that the CPU usage is low; that means, that user the number of processes in the system is also less. So OS must provide means of computing and displaying measures of system behavior so that is important. So for example, there is a top program of Windows task manager so, that gives you the task details that are there.

(Refer Slide Time: 20:30)



Now there are generations of operating system. So, operating systems are designed to run on any of a class of machines, the system must be configured for each specific computer site. There is a nSYSGEN program that obtains information concerning the specific configuration of the hardware system. It is used to build system specific compiled kernel or system tuned kernel or it can also generate general more efficient code than one general kernel.

(Refer Slide Time: 21:03)



Now, the booting of the system so, this is very important like when a system boots so, when you switch on press the power button or the reset button of the computer then what happens. So, when power initialized on system execution starts at a fixed memory location. So, this is fixed by the processor and though so, if you reset the processor then there from the manual of the processor you can find that the program counter value is loaded with some specific one some specific value so, execution starts from that point. So, I should put some meaningful code at those memory location starting at that memory location.

For example in most of the processers we will find that if you press the reset button, if you give a reset pulse to the processor the program counter value becomes 0. So, that next instruction that the processor expects is from memory location 0. So, form memory location 0 we should put the appropriate code so, that this system boots properly. So, operating system must be made available to hardware so hardware can start it. So, from 0 location 0 I should have instructions such that it either it is the beginning of the operating system or it is the it is a symbol code that loads the operating system from the secondary storage into the main memory and then transfers control to the operating system.

So, there is a small piece of code which is called the bootstrap loader. So, what happens is that it is like this. So, as I was telling that from memory location 0 we have got. So, this is the memory location 0 now program counter value at the beginning it has become 0. So, in this part of the memory so, we put some code whose responsibility is that from the disks so if this is the disk so, it will copy the operating system part from here and put it on to the system.

So, that is the from the disk it will take it and it will be copied onto the system and as a it is copied in this part then at the end of this routine. So, it will transfer, it will make the program counter value equal to this so, that the program execution can start from this point onwards. So, this is so, what is done here. So, this operating system operating system should have this bootstrap loader where this bootstrap loader is it is loading the portion from the secondary storage into the main memory and then this is the this is bootstrap loader should always be present when the system boots.

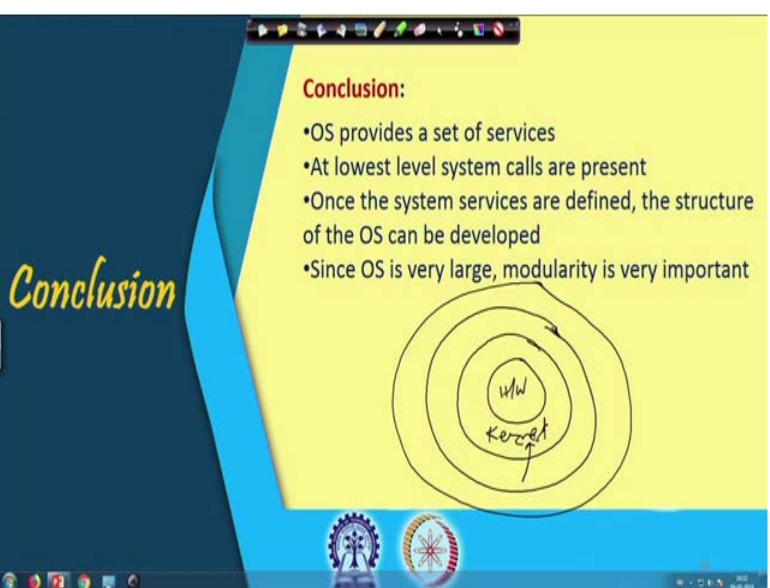
So, it is normally kept in some ROM or EEPROM that that is that is actually the kernel part of it. Sometimes 2 step process where a boot block at fixed location loaded by ROM

code which loads the bootstrap loader from disk. So, this is also there like in many cases that boots making that bootstrap loader very simple for that what we do is that. So, here we just have a piece of code that starts reading the disk from a particular boot block ok.

So, many of the operating system so when you format a CD you might have seen this that, whether you want to make it a boot disk or not so, then if you are trying to make it a boot disk when at a particular sector. So, it will be writing some booting information and that disk can be used for booting the system, because that this bootstrap loader that we have. So, it will be accessing that particular sector and we loading the portion from there into main memory.

So, common bootstrap loader GRUB is one such thing it allows selection of kernel from multiple disks versions and kernel options. So, GRUB is a software loader software bootstrap loader so, it has got many options. So, it can probably can have multiple operating system based situation so, that way it can be done. So, kernel loads and system loads the system and then it is then running. So, it after the kernel has been loaded so, system starts running phase.

(Refer Slide Time: 25:12)



So, to conclude so, we can say that OS provides a set of services at lowest level we have got system calls and once the system services are defined the structure of the OS can we developed and the since OS is very large modularity is very important. So, these are the things that we have. So, what we have seen is that at the lowest level we have got the

hardware at the lowest level we have got the hardware, then on top of that we have got the different layers of software. So, on top of that we have got these layered approaches and that way we make it both layered and modular.

So, that helps us in making this designing this OS very easy, because at higher and higher levels so we can utilize the routines that we have that we have at the next lower level and then we can do it in a modular fashion. So, and in general this lowest layer it is implemented as the kernel; so, to do protection and all so this whenever we are trying to go to this lowest layer of OS design.

So we have got some sort of protection by which we can take care of this kernel mode of operation. So, that way we can go from go into the OS design and with a good knowledge of computer architecture. So we can be that is we will be able to design operating system properly and in our successive classes. So, we will see different models of the operating system what are the issues there and how are they going to be designed in the way for any system.

Thank you.