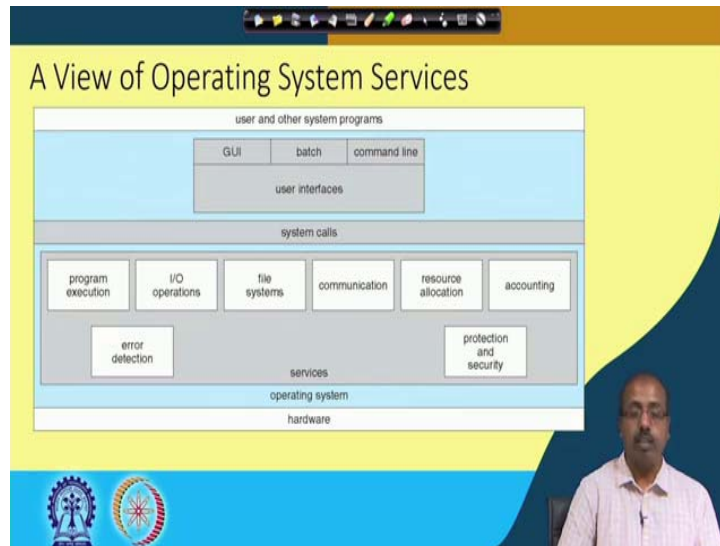


Operating System Fundamentals
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 08
Operating System Structures (Contd.)

(Refer Slide Time: 00:29)



So, if you look into this Operating System services we can look into this big picture. So, at the top level we have got this user and other system programs, then they talk to the system underlying system by means of this user interfaces which may be a graphical user interface, may be a batch interface, or maybe some command line interface.

So whatever it is by means of this user interfaces so, this users user and other system program. So, they will talk to the operating system via this operating system services. So, these are different services that are provided by the operating system like program execution, IO operations, file systems, communication, resource allocation, accounting, error detection and protection and security. So, these are the different classes of services that are provided by the operating system and for getting access to any of these services there is an another interface called system calls.

So, this from this user domain so, you cannot directly land on to any of these services, you have to request it via an interface provided by the operating system which is known

as system calls. So through will see how the system calls they are organized so by this so, each service that is available in this operating system so, it has got a unique number for that service. So, depending upon the service that it needs also the program needs so, it will say it will request for that particular number.

For example, the printing service may be service number 54, then opening a file may be service number 57 so, like that. So, accordingly the user program they will be requesting for those services and finally, the operating system will get those services done by means of this underlying hardware. So, the operating system will use the basic input output service routines of this opera of it to talk to the underlying hardware. So, that the operations are turned up and performed by the system.

(Refer Slide Time: 02:26)

Command Interpreters (CLI)

CLI allows users to directly enter commands to be performed by the operating system.

- Some operating systems include the command interpreter in the kernel.
- Some operating systems, such as Windows and UNIX, treat the command interpreter as a special program that is running when a job is initiated or when a user first logs on.
- On systems with multiple command interpreters to choose from, the interpreters are known as **shells**. → sh → Bourne Shell
csh → C shell
- The main function of the command interpreter is to get and execute the next user-specified command.
- Sometimes commands built-in, sometimes just names of programs
 - If the latter, adding new features doesn't require shell modification

Handwritten annotations on the slide include: "cat", "rm", "rm -rf", "rm -r", "\$ rm -rf", "\$ rm -r", and "\$ rm -rf".

Coming back to the Command Line Interpreters or CLI so, CLI it allows users to enter commands to be performed by the operating systems. Some operating systems include the command line command interpreted in the kernel so, kernel that is the main part of the operating system, where it that has got direct interaction with the hardware. So, sometimes what we will have is that this command interpreter that is after the user has entered command to the keyboard what does it mean. So, that part becomes a part of the kernel part of the operating system. So, that is the major the innermost part of the operating system.

So, we will understand this kernel slightly more when we go to these services in more detail, some operating systems such as Windows and UNIX treat the command interpreter as a special program that is running when a job is initiated or when a user first logs on. For some cases there is this UNIX and Windows operating systems so, they have got a command interpreter. So, they are commonly known as shells.

So they are given they put the user into some shells structure into which certain set of commands are available and from that the user can enter those commands and that the command interpreter starts working from there. So it takes the commands the command interpreter routine it takes the commands on the user and then passes it to the underlying operating system.

So, we have got some of a sum of system. So, there may be multiple such command interpreters to choose from so, they are called shells. For example, if you look into this UNIX operating system so, we have got several shells there is a shell which is named as sh which is the Bourne shell; which is the Bourne shell, then we have got csh which is called C shell. So, like that there are different types of shells.

So what is this shell? So, shell actually will tell you certain type the commands that are available in a particular interpreter. So, overall function is almost same for all the shells, but the pair of the commands that are there. So, that will be varying slightly and some shells they have got a history mechanism. So, that you can it will remember the last 20 commands. So, if you repeat some commands you do not need to write the entire command day again so, you can just select from the history and ask the system to run.

So, the this type of cosmetic changes are there across the shells, but after all if shell is nothing, but a command interpreter. So, it takes the command from the user if required it will partition it into several parts and then it will be getting them executed. For example, if the user enters the command say cat a. So, in Linux or UNIX operating system so, this cat means concatenate. So, it will print the file a on to the screen. So, it print if you after this the file this file is printed on to the screen. Now sometimes we may like to count like how many particular character is there a particular word is there say for how many times the word say how many times the word rat appears in this. So, it maybe we can find to we can do it like this grep "rat" and then we can see like wc -l.

So, assuming that, in each line the command that the word “rat” will come only once, then if first this commands so this will be; so, this will be outputting the file, but there is a pipe here. So, pipe means that this output of this program will it will go to the second program and the second program will take out the lines that has got the word “rat” in it and in the third part of the command. So, we are counting how many such lines are there. So, there are 3 commands that are given in one shot by the user.

So, this whole thing is given to the shell and the shell will find that there are 3 commands to be executed, one is this cat, one is grape, another is wc and not only that the output of the first command should be given as input to the second command and output of second command should be given as input to the third command. So this shell program so, it will find out all these details and it will figure out a way by which these commands can be executed in the proper meaning whatever the user has asked for. So this is the thing that we have got these shell structures.

So, this the main function of this command interpreter is to get and execute the next user specified command. So, basically when there is no new command so, the shell will be waiting for the next command to come. So normally we have got some sort of a prompt and that prompt may be in some system it may be dollar it may be greater than. So, that way you can design your own prompt also in some system so, given this prompt this prompt is primed printed and until and unless the user has entered the data the system will be waiting.

So that is the main function of the command interpreter is to get and execute the next user specified command sometimes the command in the built in sometimes just names of program. So this command that the examples that I have taken so, they are all built in commands like cat, grape, wc, they are all built in command, may I have a program which computes the square of a number maybe so, you can when in the dollar prompt. So, you can just write square the command interpreter it will see that square is not a built in command. So, then it will try to look for a file who is whose name is square and it will execute that particular program. So, that is also for it may be just names of some programs instead of some built in command.

(Refer Slide Time: 08:44)

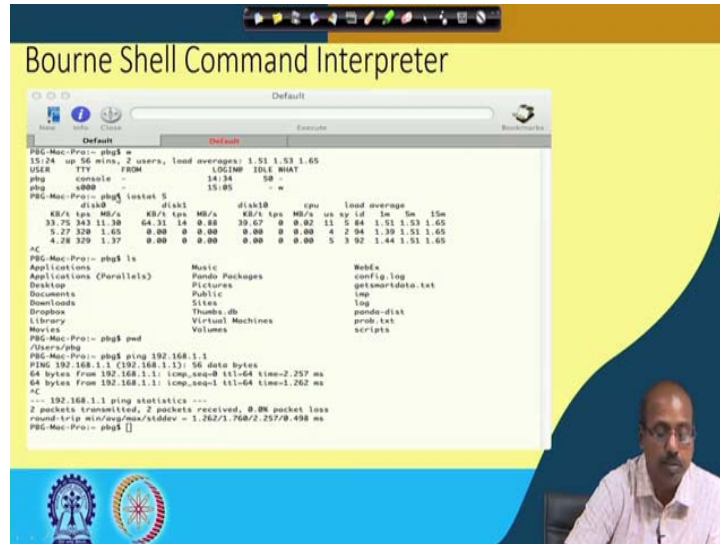
```
The Bourne shell command interpreter in Solaris

1.root@r6181-d5-us01~ (ssh)
Last login: Thu Jul 14 08:47:01 on ttyd02
[MacPro: ~] $ ssh root@r6181-d5-us01
root@r6181-d5-us01's password:
Last login: Thu Jul 14 06:01:11 2016 from 172.16.16.162
[root@r6181-d5-us01 ~]# uptime
 9:52:48 up 16 days, 10:52,  3 users,  load average: 129.52, 80.33, 56.55
[root@r6181-d5-us01 ~]# df -kh
Filesystem      Size  Used Avail Use% Mounted on
/dev/mapper/vg_ks-lv_root
 50G   19G  28G  41% /
tmpfs
 127G  520K 127G   1% /dev/shm
/dev/sdals
 477M   71M  381M  16% /boot
/dev/dss0000
 1.0T  488G  545G  47% /dssd_ufs
tcp://192.162.150.1:3334/orangefs
 12T  5.7T  6.4T  47% /mnt/orangefs
/dev/gpfs-test
 23T  1.1T  22T   5% /mnt/gpfs
[root@r6181-d5-us01 ~]#
[root@r6181-d5-us01 ~]# ps aux | sort -nrk 3,3 | head -n 5
root   97053 11:2  0.0 4266344 1727036 ?   Ss1  Jul13 166:22 /usr/lib/mfcs/bin/mfcd
root   69849  6.4  0.0  0  0 ?        S    Jul12 181:54 [vsthread-1-1]
root   69658  6.4  0.0  0  0 ?        S    Jul12 177:42 [vsthread-1-2]
root   3829  3.0  0.0  0  0 ?        S    Jun27 738:04 [rp_thread 7:0]
root   3026  3.0  0.0  0  0 ?        S    Jun27 728:08 [rp_thread 6:0]
[root@r6181-d5-us01 ~]# ps -l /usr/lib/mfcs/bin/mfcd
-rw-r----- 1 root root 28667161 Jun  3 2015 /usr/lib/mfcs/bin/mfcd
[root@r6181-d5-us01 ~]#
```

So, next we have the bone shell command interpreter in solaris system that example is given. So, you see that this prompt comes in this format this the name of the this is the prompt the current directory structure then this hash is there and then we can give the command like this is one command ps aux. So, this ps command tells to tell what are the processes that are currently running, then aux these are the options that are passed away that are accepted by the ps command. So, for this detail you need to look into this manual for this Solaris or the unique system and then it is sorting them in some fashion and then it finding the head part of it.

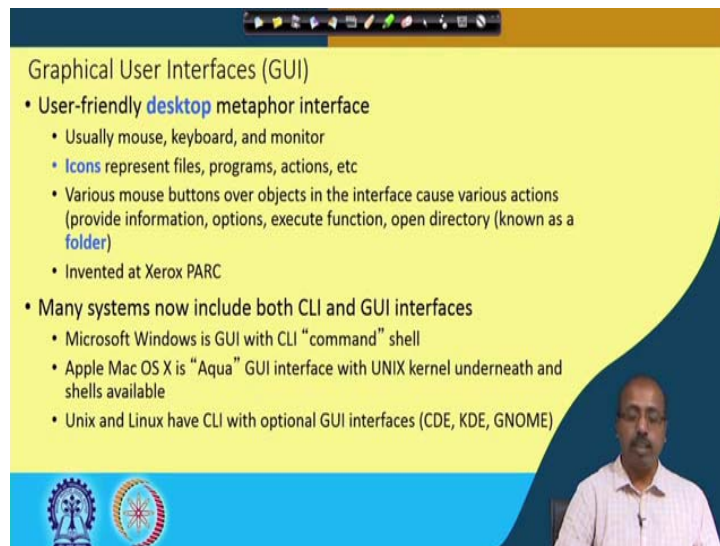
So, that way it is printing this command. So, here also we have got these pipes between them or for example, says this command so, this uptime is a built in command. So, these this measures how many times the system is up since what time the system is up, what is the average load and all that. So, this way it can come up with a number of different command that can be accepted. So, this is a bone shell type of interface.

(Refer Slide Time: 09:57)



There are other interfaces also like say this is another interface say, w so, this is a mach for a Mac computer this interface. So, this is w so, this gives the users that are there from which console they are login logged in. Then IO status this gives the IO status for different discs IO statistics for different discs, then this ls command so, this prints the files that are there in the system. So, like that we can have another user interface.

(Refer Slide Time: 10:31)



Then next look into graphical user interfaces, then the graphical user interfaces are much more friendly so, they are user friendly desktop metaphor interface usually we have got a

mouse, a keyboard and monitor. So icons represent files programs and actions etcetera and there are various mouse buttons over objects in the interface that cause various actions.

So as we click on the mouse button so on a particular object or interface. So maybe that file will be opened or that particular program will start executing or we may like to it may give us the properties of that particular program or that particular object, like that we can have different types of actions defined. So, this was invented by Xerox PARC. So by now almost all the computer systems that we are getting so they have got this graphical user interface.

Many systems now include both CLI and GUI interfaces like Microsoft windows with GUI with CLI command shell like we have got in windows, my name is window. So, you have got command shell also for running some older programs. So, sometimes we need to go to this command mode and then we can invoke that command shell and go to the command line interface mode.

Then this Apple Mac OSX is “Aqua” GUI interfaces with UNIX kernel underneath and shells available. So, you have got this UNIX kernel and the shells are there. So, based on there so, you can choose between different shells that are there in UNIX system like a say bone shell, c shell, k shell, like that and based on that it will take the commands from the user.

UNIX and Linux have CLI with optional GUI interface like this CDE, KDE GNOME. So, these are some GUI interfaces that are available under this Linux platform. So, they normally have both this command line interface as well as this GUI interface. So, now, almost everybody is comfortable with GUI interfaces. So, you will see more of GUI's then CLI's, but CLI's are existing because mainly because of legacy reason that certain command it is easier to be given in a as a comment typed by their super user and when the system is just starting maybe the GUI has not yet started there is an error that has occurred even before that so, GUI system cannot come up.

So, in that case the CLI interface can be utilized for debugging the system or asking the system to do some rectification action so, that the problem can gets resolved.

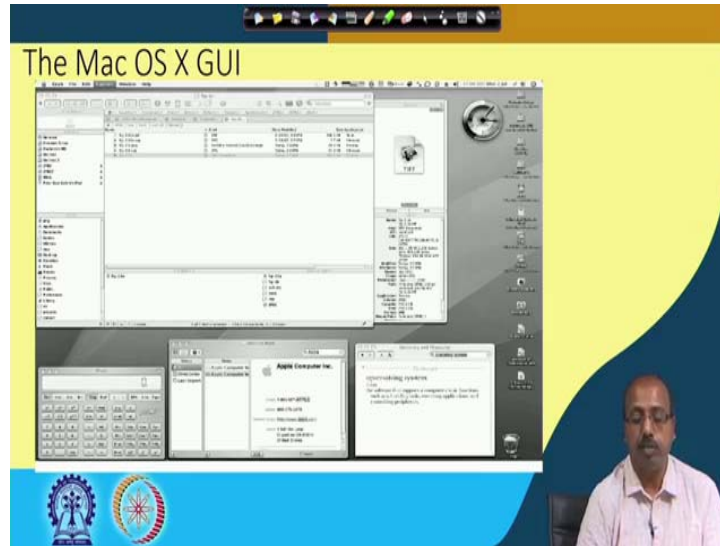
(Refer Slide Time: 13:17)



Then we have got this touch screen interfaces. So, a touch screen they require new interfaces like mouse is not possible or not desirable. Then actions and selection based on gestures like sometimes we make some hand gestures or eye gestures. So, based on that the actions take place and there is often a virtual keyboard for text entry. So, these are standard that we have got on mobile devices mostly. So, they will have got this touch screen devices that requires new type of interfaces.

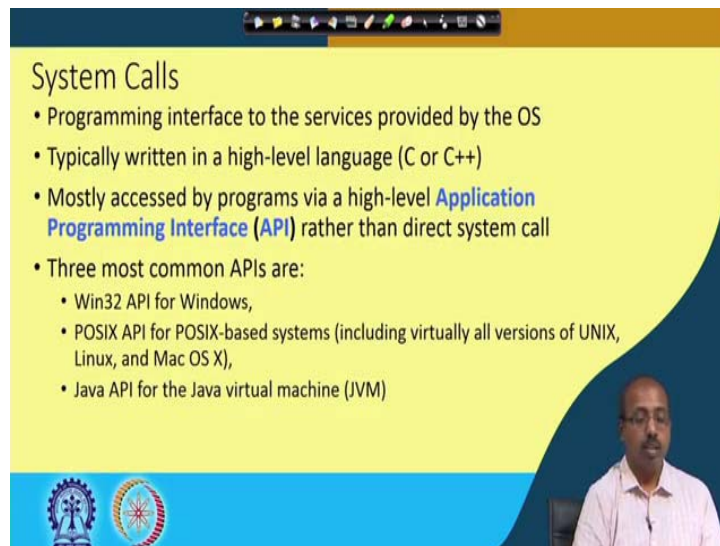
Then there are voice commands like we can give the commands verbally, so, that these mobile devices they are accepting it. So, that way we have got another type of interface this touch screen interfaces. So, they are also behavior very popular now.

(Refer Slide Time: 14:05)



So, this is again one Mac OS X; Mac OS x GUI so, it has got it is own feature. So, people familiar with this Mac with the Apple, Lava operating system. So, they will be looking into this.

(Refer Slide Time: 14:19)



So, apart from the interfaces, interface is just to take the command from the user or gate or knowing what the user wants to do. Now the next the question that we will answer is how is it being done by the underlying operating system, how the interface tells the

operating system what to do ok. So, they are done by means of system calls. So, system calls are programming interface to the services provided by the operating system.

So, operating system designer will tell I provide all these services for example, file related services open a file, close a file, read a file, right onto a file check the permission of a file, change the mode of a file. So, these are certain type of facilities or type of services that are provided on files by the operating system.

Now, each of the services as I said previously has got a number. So, this services they are typically written in a high level language like C or C plus plus and they are mostly accessed by programs via high level application programming interface or API rather than direct system calls. So, most of the time nowadays you will find these API's and these API is so, they will ultimately call the underlying system underlying operating system using those system call numbers.

So, normally there are different types different API is that have come up there are 3 most common API the Win 32 API for Windows, POSIX API for POSIX based systems including all almost all versions of UNIX, Linux and Mac OS X and we have got Java API for Java virtual machines. So, these are some of the API is that are very common that are used by the operating systems.

(Refer Slide Time: 16:09)

The slide is titled "Example of System Calls" and features a yellow background with a blue header and footer. At the top, there is a navigation bar with various icons. Below the title, a bullet point reads: "• System call sequence to copy the contents of one file to another file". A diagram shows a box labeled "source file" on the left and a box labeled "destination file" on the right, with an arrow pointing from source to destination. In the center, a light blue rounded rectangle contains the following text:

```
Example System Call Sequence
Acquire input file name
Write prompt to screen
Accept input
Acquire output file name
Write prompt to screen
Accept input
Open the input file
If file doesn't exist, abort
Create output file
If file exists, abort
Loop
Read from input file
Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

At the bottom of the slide, there are two logos on the left and a small video inset of a man in a white shirt on the right.

So, how this system calls how what are these system calls. So, here we have got an example. So, system called sequence to copy the contents of one file to another. So, we want to copy the source file to the destination file. So, first thing is that acquire input file name. So, write prompt to the screen and accept. So, this is the first operation to be done acquire if the source file name has to be obtained. So, for that we need to write a bright prompt to the screen like enter file name and then accept the input.

Then I have to see to which file this has this is this will be copied. So, that way it will acquire output file name. For acquiring output file name the prompt should be written to the screen and the input should be accepted from the user, then we have to open the input file. Now the file that the user file name that the user has provided the corresponding file may or may not exist. So, that way if file does not exist then it has to abort, if the file exists then I have to first create the output file.

Now, for the output file if the file already exists; that means, that is an error. So, I am trying to overwrite a file so, that is aborted and then it goes into a loop; in a loop so, it reads from the input file and writes onto the output file until read failed. So, as long as we are reading from the file so, maybe we are reading this file read is done in terms of blocks. So, maybe one block maybe the 1 kilobyte or 2 kilobyte or 4 kilobyte like that there is a system defined block size. So, in one read we read one block of data from the input file and then we write that block to the output file. So, this thing goes on in a loop till there is there are no more blocks available on the input file.

So, this input file read pointer has reached the end of the file then if you try to read it again that gives a read failure. So, until this read fails so, it goes on in a loop like this and once the read fails; that means, entire file has been read and copied. So, we closed the output file and write completion message to the screen and then terminate normally. So, this is the sequence of operations that should take place when we want to copy a file to the output copy file from input file to output file.

(Refer Slide Time: 18:26)

Example of Standard API

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the man page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count)
```

return value function name parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns `-1`.

As an example of the standard API consider the `read` function that is available in UNIX and Linux system. The API for this function is obtained from the man page by. So, if you do this `man read` so, this UNIX and Linux system. So, they provide online manual pages so, you can just do this `man` and then the name of the command. So, accordingly it can find out the manual page and then it can print this thing.

So, it says that the API description is like this that if you want to use this copy then this particular file has to be include `<unistd.h>` and this is the `read` command. So, this `read` command is like this so, it has to it will need 4 parameters the file descriptor. So, file descriptor is the pointer to the file from where you want to read the data then a buffer is a pointer. So, buffer where the value will be read so, that there has to be a free allocated buffer associated here.

So, that the value that the 1024 bytes of block that is read though that is kept on to these buffer and the `size_t` tells how many per bytes you want to copy. So these are the meaning of the 3 parameters `fd` is the file descriptor to be read, `void star buffer` is a buffer where the data will be read into and `size_t count` is the maximum number of bytes to be read into the buffer.

So, a program that uses the `read` function must include the `<unistd.h>` header file as this will define these particular types this is `ssize_t` and the `size_t`. So, these two variable these

two data types are defined is this in this <unistd.h> header file. So, this file must be included and the parameters first are like, this if the reading is successful the number of bites read is return and the return value of 0 indicates the end of file and the return value of minus 1 indicates that there was an error when this return was when this a read was tried out. So, that way we can have this standard API.

(Refer Slide Time: 20:45)

The slide is titled "System Call Implementation" and features a list of bullet points on the left and a hand-drawn diagram on the right. The diagram shows a table with three rows and two columns. The first column contains the numbers 0, 1, and 2. The second column contains the words "create", "open", and "close". Arrows point from each row of the table to a corresponding box on the right side of the diagram, which represents memory locations. The slide also includes a video feed of a man in the bottom right corner and a Windows taskbar at the bottom.

- Typically, a number is associated with each system call
 - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values
- The caller need not know a thing about how the system call is implemented
 - Just needs to obey the API and understand what the OS will do as a result call
 - Most details of OS interface hidden from programmer by API
 - Managed by run-time support library (set of functions built into libraries included with compiler)

Now how do we implement the system call, typically a number is associated with each system call. As I was telling that normally we have got some number for each system call and system call interface it maintains a table indexed according to these numbers. So, that the table is indexed and then if I have got a table where says this is the table and So, we have got the numbers of 0 1 2 3 etcetera.

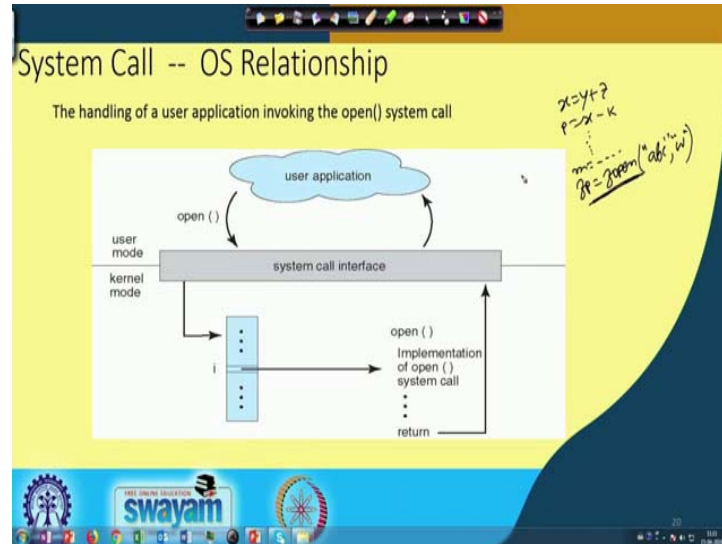
Now, the 0 may be for create to create a file this may be for opening a file, this may be for closing a file, so, like that. So, what will happen is that this entry. So, in the memory if the routine for creating a file has been loaded in this part of the memory then create will point to this. Similarly, if the open is somewhere here the code for open is somewhere here so, this will point to this particular memory location. So, like that so, this table is indexed by that. So, that using this call so, it can using this pointer. So, it can come to the corresponding memory location where the routine for doing that system call is loaded.

So, a system call interface invokes the intended system call in OS kernel and returns the status of the system call and any return values. So, these execution is done in the kernel mode of execution not in the user mode. The caller need not know a thing about how the system call is implemented, because what the caller needs to know is only this table. So, OS as a documentation the OS designer will provide this table to the users telling that if you for 0, if you want to create a file the number is 0, if you want to open the file the number 1 so, this system called indices are provided as the information and actual implementation is hidden from the user.

So, user does not know how these calls are actually implemented by the operating system. So, the caller need not know anything about how the system call is implemented just needs to obey the API and understand what this what the OS will do as a result call and most details of the OS interface hidden from the programmer by this API and it is managed by runtime support libraries. So, set of functions built into libraries included with the compiler.

So, this actually this where this files are actually loaded so, that information is available with the operating system and that is that becomes a part of this compiler when it generates the code. So, the runtime libraries are included by in the operating system in the program code by means of the compiler and that will be managed at runtime by the support library. So, this will be more clear so, if we look into a compiler how does it link the dynamic libraries and all. So, if we understand that part then it will be more clear which is beyond the scope of this discussion so, we do not go into that.

(Refer Slide Time: 23:54)



Now, so this is the overall thing that we have. So, the handling of a user application invoking the open system call is like this. The user application the first thing is that the user application makes this open call and that is by means of system call interface. So, as long as this open is not executed the program is executing in the user mode.

So, as I told previously like there is there may be a set of operation like x equal to y plus z , p equal to x minus k . So, like that when these simple executions are being done. So, they are all in user mode of execution in the program, but some time later you may like to write the value of say value of the variable m that has been computed here to a file. So, for that I need to open a file so, in a c program we write it like this if p equal to f of m some file abc and we want to open it in right mode.

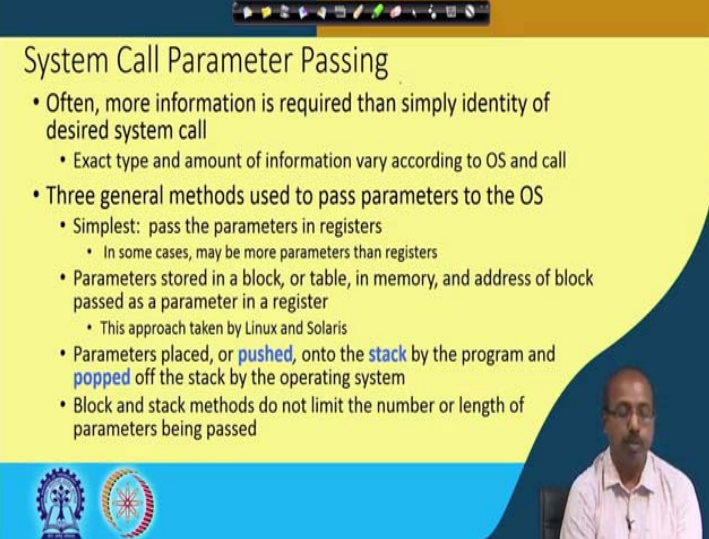
So, when this statement is to be executed so, it is no more in the user mode of execution. So, you see that this f of m library function of c so, it will ultimately translate to the open system call for the operating system and then this open system call it will be; it will be crossing this user mode of operation and it will enter into kernel mode of operation. So, this system called interface what it will do, it will be transferring the continuity look into the index and accordingly they according to this index value so, this may be that this system called table that I was talking about previously. So, this is there and this will transfer the call to the actual code where the open open system call and there it will be having this implementation of open system call. So, this particular piece of code will be

executed and after that it will return to the system call interface from where it will return to the user application.

So, if there was an error while opening this file so, this error code will be returned otherwise if this open was successful then it will return is 0 and the user program will be getting back a value 0. So, ultimately this there so, sorry if there is if there was no error then it will return some file descriptor and this file descriptor will be coming back to this system to that user program and that will be coming to this f p value.

So, this way we can see that this there will be this open system call is implemented as long as the program is executing in the user mode. So, normal program statements are being executed in the user mode, but whenever it requires some services from the operating system. So, it will go to the kernel mode of execution and then in the kernel mode of execution so, it will go to the exact code where this operating system code calls are implemented and accordingly it will be done.

(Refer Slide Time: 26:51)



The slide is titled "System Call Parameter Passing" and contains the following text:

- Often, more information is required than simply identity of desired system call
 - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
 - Simplest: pass the parameters in registers
 - In some cases, may be more parameters than registers
 - Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register
 - This approach taken by Linux and Solaris
 - Parameters placed, or **pushed**, onto the **stack** by the program and **popped** off the stack by the operating system
 - Block and stack methods do not limit the number or length of parameters being passed

The slide also features a small video inset in the bottom right corner showing a man speaking, and logos of institutions at the bottom left.

Now, how do we pass the parameters to the system call like does that open the f open or open system call it requires the name of the file, then what is the permission in which you want to open the file the operation that a read operation or write operation append operation or read write operation, so, like that. So, often more information is required than simply the identity of the desired system call extract. So, exact type and amount of

information will vary according to the OS and the call so, that there may be variable number of things.

Sometimes some of the calls may not be requiring any more parameter for example, it may be just asking for say wait it may be asking for say terminating something so, terminating the program itself. So, in that case no more parameter is necessary, but if it is asking for say opening a file then a number of parameters are to be there.

Now, there are three general methods that are used to pass parameters to the operating system, the simplest way is to pass the parameters in registers. So, that what the user level what is done is that, the parameters that we want to pass their values are copied on to CPU registers and in the operating system call implementation of a system call implementation it accesses from those registers.

So, it may be more parameters, but bypass parameters by registers. So, this is a simplest one, but CPU registers are limited. So, if you need to pass more number of parameters then registers are not sufficient and all the registers may not be available also. So, they may be already having some important data in them so, they are not parable for passing the parameters.

Second possibility is the parameters are stored in a block or table in memory and address of the block passed as a parameter in a register. So, you copy the parameters that you want to pass in a block of memory locations and then address of that block is copied on to a CPU register and the call is done. So that at the receiving end at the system call execution so that register value will be used by the operating system to locate the parameters that are passed for this particular system call. So, this is the approach taken by Linux and Solaris operating system.

Or it may be the parameters are placed or pushed onto the stack by the program and popped off the stack by the operating system. So, there may be if there is a stack then maybe at the time of calling thus making the system called the user program it pushes all the parameters onto the stack and the system whether the system call implementation it just pops out those values from the stack and gets the parameter values. So, this is one possibility and block and stack methods do not limit the number or length of parameters

being passed. So, this is the advantage of this either the block method or the stack method.

So, the number of parameters is not restricted so, it you can pass a large number of parameters whereas, the first one where I am using CPU registers for passing the parameter. So, they are restricted because number of CPU registers available is also restricted. So, this way we have got different ways of passing parameters. So, we will discuss kind of on this in the next class.