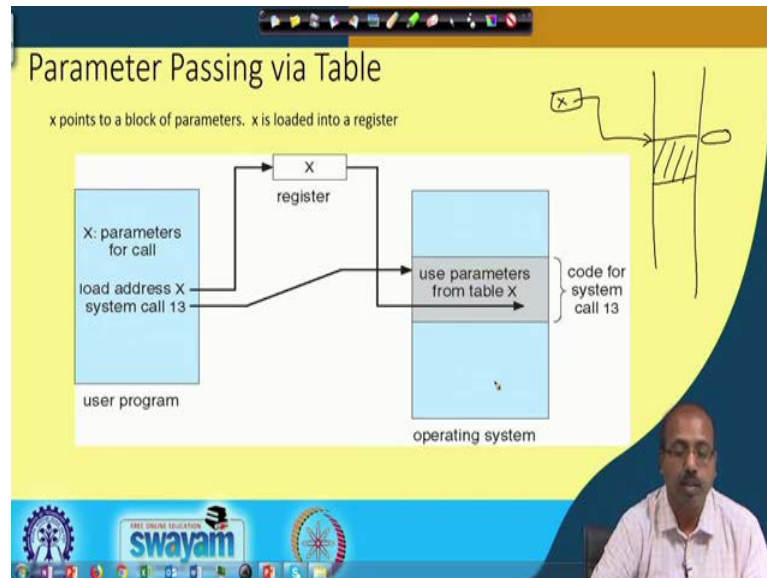


Operating System Fundamentals
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 09
Operating System Structures (Contd.)

(Refer Slide Time: 00:29)



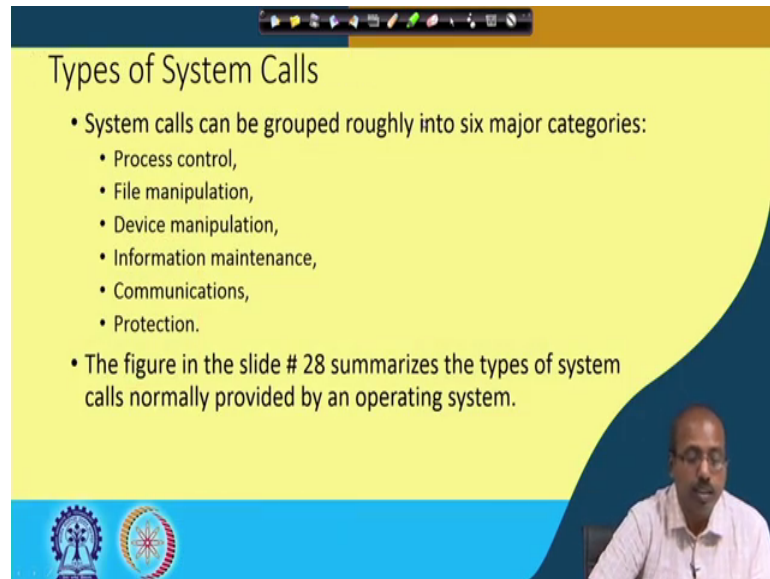
So, if you look into this parameter passing via table. So, this is the strategy that we said is very important or very one of the simplest way or simplest at the same time it has got flexibility. So, we have got the this for this procedure call. So, we have for this system call, so I have got a number of parameters to pass and those parameters that we want to pass, so they are loaded into some memory locations and then this parameters for the call, so we load the address x for the system call.

So, this is the system call 13 that is that the program is going to make. So, x is the parameter x is loaded onto a register and then that is passed onto this thing. So, this x is actually an address register and that holds the block of parameters the address of the block of parameters that we have in the system for this particular call. So, it is like this that. This if this is the memory, so maybe in this portion of the memory the parameters are all copy and in the x registers, so we have the address of this. So, when this x is properly loaded with this particular address then at the operating system implementation

level of the system call, so it can use that x register to locate the parameters that have been passed in the block.

So, this way by using this table type of structure. So, we can pass the parameter for this system call.

(Refer Slide Time: 02:05)



The slide is titled "Types of System Calls" and features a yellow background with a blue and orange border. It contains a bulleted list of six categories of system calls. A small video inset in the bottom right corner shows a man with glasses and a white shirt speaking. At the bottom left, there are two circular logos: one with a gear and a figure, and another with a star and a gear.

Types of System Calls

- System calls can be grouped roughly into six major categories:
 - Process control,
 - File manipulation,
 - Device manipulation,
 - Information maintenance,
 - Communications,
 - Protection.
- The figure in the slide # 28 summarizes the types of system calls normally provided by an operating system.

There are different types of system call, as I was discussing there are different types of services. So, each of the services that the operating system provides. So, they are a set of there is a set of system calls that are possible. So, they can be grouped roughly into six major categories some of the system calls are process control related, some of them are file related file manipulation, some of them are for device manipulation, some of them are for information maintenance, some are for communication, some are for protection, ok. So, this will see sometime later that how the system calls are there.

(Refer Slide Time: 02:41)

The slide is titled "System Calls – Process Control" and features a yellow background with a blue border. It contains a list of system calls and a diagram. The list includes: create process, terminate process; end, abort; load, execute; get process attributes, set process attributes; wait for time; wait event, signal event; allocate and free memory; Dump memory if error; **Debugger** for determining bugs, single step execution; and **Locks** for managing access to shared data between processes. The diagram shows two circles, P1 and P2, with an arrow pointing from P1 to P2. Next to P1 is the text "signal(e)" and next to P2 is "wait(e)".

- create process, terminate process
- end, abort
- load, execute
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory
- Dump memory if error
- **Debugger** for determining bugs, single step execution
- **Locks** for managing access to shared data between processes

So, to start with the system calls for process control they will create process and terminate process. So, these are some of the related system calls like create process or terminate process. So, we may want to end a process or about a process. So, ending is just terminating that process or aborting is in between there is some erroneous condition that has occurred, so that has to abort the call.

Then there are load and execute. So, maybe some program has to be loaded or some process has to be loaded and process as to be executed. So, load means it will be loaded from the content will be copied from the secondary storage or disc into the main memory so that the process becomes ready for execution. And later on, when the process is given CPU time, so it will be executing. So, that is the execute. So, these are the two these are two different operations, so one is making the program ready or the making the process ready for execution the by copying its content from the secondary storage and other one is to execute the process that is giving the CPU to the process.

So, get process attributes and set process attributes. So, some of the attributes of the process like who is the owner of the process, then how much time we should give it, what is the priority of the process. So, that way we can try to get the attributes of the process or sometimes with the we may want to set that the process attributes. So, it may so happen that we want to reduce or improve increase the priority of a process, so that

can be done by the system user the system administrator. But the operating system should provide some system call by which that can be carried out.

Then some system call, some process maybe like to wait for some time that between two operations it wants to introduce some delay, so it has it wants to wait for sometime. So, when the time is over then the process should be informed and the process will resume its execution from that point. Then wait for event or signal event, some time a process may like to wait for some event to occur or it may want to inform some process that some other process that, with the events some particular event has occurred, so that other process can continue.

So, it may so happen that I have got two processes p 1 and p 2. So, p 1 is p 2 is waiting for some event which will be triggered by p 1. So, p 2 for the p 2, so it is waiting on some event e ok. So, that is the wait system call and this process p 1, so it will be doing that signal on the e. So, that way it will signal that the event as occurred. So, that is the wait event and signal events. So, they are two different events for with their must be system call available so that they can be utilised by the processes.

Then allocate and free memory. So, some process may require more storage because as we know some way while writing in some high-level program, so we want to create some dynamic storage. So, the dynamic students creation is by means of this allocate request for some dynamic space and when the space is no more required would like to free that space. So, this is allocate and free, so these are the two things.

Then dump memory if error. So, if there is an error that has occurred. So, we were dynamically that error has occurred while the file the program was running and it may be familiar with getting this type of error. So, if particular if you are running program on UNIX or Linux, operating system something like this particular error message segmentation fault; segmentation fault core dumped. So, you might have seen this message.

(Refer Slide Time: 06:19)

The slide is titled "System Calls – Process Control" and lists the following system calls:

- create process, terminate process
- end, abort
- load, execute
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory
- Dump memory if error
- **Debugger** for determining **bugs**, **single step** execution
- **Locks** for managing access to shared data between processes

Handwritten note: *Segmentation Fault - core dumped.*

Diagram: A vertical rectangle representing a memory segment. The top is labeled "start address" and the bottom is labeled "end address". Inside the rectangle, there are three horizontal lines representing memory locations, with the middle one labeled "p". An arrow points to the "p" location from the left.

So, this what this message means is that so I have a program, so in that program I am writing say $x = \text{something}$ where this is for normal variable assignment. So, if I have got a pointer p and I write that $*p = \text{something}$. And until and unless this p is properly initialised, so when this operation is going to be executed. So, it if there is a very high chance that this address the current value of p that is the address of the location that should be modified is not within the valid address range for this particular process.

So, as a result, so this is segmentation fault it is going to it is trying to access some location which is beyond the segment that is available for this program. So, that is a segmentation fault. And when the segmentation fault occurs the user needs to check whether the fault has really occurred or not ok. So, how what was the situation that lead to p having some other values some undesirable value. So, for getting that trace, so we need to know like how this point was reached in program execution and that is the total runtime image of the process that is executed. So, that is called the core of the program.

Now, so when this segmentation fault occurs. So, if this core is dumped then later on some debugger can be used to trace why exactly this particular fault occurred. So, that is the segmentation fault core dumped. And its normal it is a huge files compared to a normal the actual program code because program code is small, but the program code maybe it is making some recursive calls and all. So, this core becomes very large.

So, that way we this is also very important. Like operating system it should be able to dump the memory content if there is an error, so that we can figure out what went wrong with the program. Then debugger for is required for determining bugs. So, we should be able to single step execution. So, debugger is for checking whether there was a why this particular situation as occurred when the program is not running properly, what is the problem, why is it not running properly. So, you may like to have this thing like for example, I might have written a program that does all this competition x equal to y plus z etcetera etcetera.

(Refer Slide Time: 08:59)

System Calls – Process Control

- create process, terminate process
- end, abort
- load, execute
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory
- Dump memory if error
- **Debugger** for determining **bugs**, **single step** execution
- **Locks** for managing access to shared data between processes

Now, after executing this line whether this line was properly executed or not. So, how do I check it? So, I should be able to see what is the value of y , what is the value of z and what is the value of x . So, after this line has executed. So, if I find this values and for see that x is really having the value of y plus z then I know that up to this much the program has executed correctly. But if I get something else that means, there is a problem somewhere may be this particular statement or some program or some statement in the other part in the earlier part of the program.

So, how to do this thing? So, for this we must have some debugging facility. So, either the system must have a debugger, so that I can do this check. So, it can it will allow you allow me to check the content of some of the memory locations and the CPU registers. And also, I should be able to run a program line by line. So, if these are the different

lines of the program in particular in the source level. Like this line maybe $x = y + z$, this line may be $p = x * m$. So, like that, so I should be able to stop.

So, normally if you start executing a program here, so it execute all these lines and then comes out it terminates. But I do not want it, I want that the after I have initiated the operations, so the program would should get suspended here and then I will be able to check the values of x , y and z . Then if I say go then it will execute the next line and again stop here, again I will be able to check the values of this variable then it should go like this. So, line by line it should proceed not whole thing in a single go. So, that is the single step execution that is there. So, single step execution is helpful in the program debugging phase.

So, normally when the program is being developed initially. So, there are lots of logical bugs in the program then this debugger and this single stepping facilities, so, these are really required; particularly, if the logic of the program is complex.

So, this way this debugger facility they are provided by means of system calls services. Then there are locks for managing access to share data between processes. So, as I said that if there is a variable shared variable x , now if process 1 is trying to write on to this and process 2 is trying to read this then when process 1 is modifying this value of x then this content of x is not final. So, it is in some the value that is read by p_2 may not be a correct one as the value x was undergoing some changes. So, what I can say when was p_1 is accessing x p_2 should not be allowed, similarly when p_2 is reading x p_1 should not be allowed to modify. So, for that I can have some sort of lock around this and to access this x I should get a hold of the locks. So, if I get the lock permission then only I should be able to proceed.

So, this type of locking mechanism must be provided by the operating system calls, so that this sharing becomes consistent. So, we look into this in process synchronization chapter as we will have a later.

(Refer Slide Time: 12:31)

The slide is titled "System Calls – File Management" and lists the following system calls:

- Create file
- Delete file
- Open and Close file
- Read, Write, Reposition
- Get and Set file attributes

Handwritten notes on the slide include:

- A diagram of a file with a pointer labeled "fp" pointing to the start.
- The word "loop" written next to the diagram.
- A code snippet: `read(fp, buf, <no bytes>)` with an arrow pointing from "fp" to the first parameter and from "<no bytes>" to the third parameter.

The slide also features a video feed of a presenter in the bottom right corner and a Windows taskbar at the bottom.

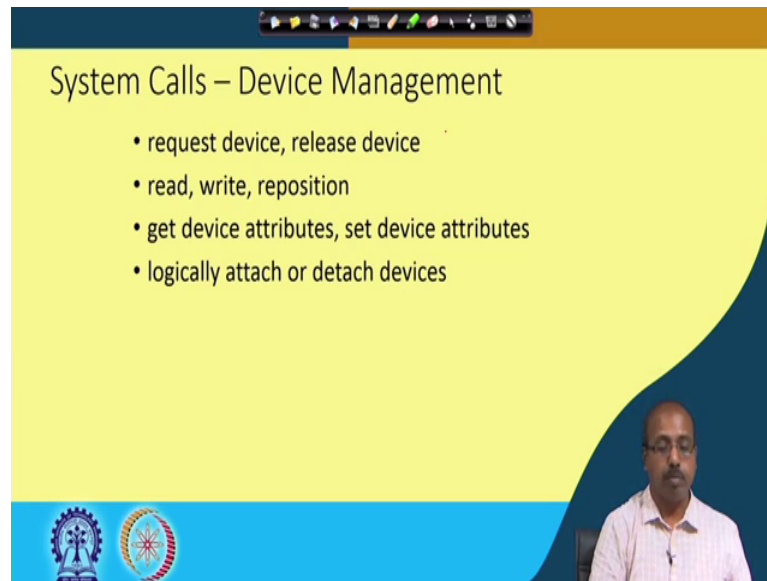
Then for file management, we have got a number of system calls like the creating a file, then deleting a file, open and close file, then read write or reposition. So, this read write we understand. So, reposition actually means that if this is; now when you open the file; the file pointer is set at the beginning of the file. So, if you read from here then you will be getting the lines one after the other, actually you will be getting what the character by character or this depending on the size of that read in the read system call we have to tell like how many bites we want to read.

So, here I have to tell that f p and the other parameter that I want that I need to give is the number of bytes that I want to read. There is of course, another pointer the buffer pointer is there. So, if this read is successful, it will read from the current position of the file pointer number of bytes and put it into the buffer.

Now, after that the file pointer will advanced to the next suppose this much was the number of bytes read. So, file pointer will come here. So, if you issue another read here, so it will start from this read from this point and continue. Sometimes, we want to skip some part of the file for reading and that is done by this reposition types of system calls. So, normally in C program, so you will find there is a call l seek, ok, so that will place that file pointer somewhere later in the file.

So, this reposition system call is their then we have got system calls for get and set file attributes. So, that will be allowing us to get who is the owner of the file, what is the size of the file, when was it created when was it last modified etcetera. So, this file management related calls, so they are there they should be system be there in any operating system.

(Refer Slide Time: 14:53)



The slide is titled "System Calls – Device Management" and lists the following system calls:

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

The slide features a yellow background with a dark blue curved shape on the right side. At the bottom left, there are two logos: one of a gear and a person, and another of a gear and a sun. A video inset in the bottom right corner shows a man with glasses and a mustache, wearing a light-colored shirt, speaking.

Next we will see the device management related call. So, there may be request for the device then release the device, then read write reposition. So, this are the same thing. So, we want to read the content from the device or write or something on the output device or we want to advance that the read write head, that repositioning the head.

Then get device attributes, set device attributes, logically attach or detached devices. So, the device is there, but we want to get it attached to the system or attached to the program, so that is the attachment or we want to we do not want that device any more in my system in my program, so you want to detach the device. So, these are the services provided by the under the device management category.

(Refer Slide Time: 15:39)

System Calls -- Information Maintenance

- get time or date,
- set time or date
- get system data,
- set system data
- get and set process, file, or device attributes

The slide features a yellow background with a blue wave on the right side. At the bottom, there are two logos: one of a gear and a person, and another of a circular emblem with a star. A video inset in the bottom right corner shows a man with glasses and a white shirt speaking.

Then maintenance related calls like get time or date. So, we make for example, if you are trying to keep the information about which user is using how much time using the system for how much time then these are important. Like get time or date set time or date get system data, set system data, then get and set process file or device attributes. So, these are some of the in maintenance related calls or maintenance related system calls that are available in operating systems ok. So, all operating systems will have some variant or the other of this calls.

(Refer Slide Time: 16:21)

System Calls – Communications

- create, delete communication connection
- if **message passing model**:
 - send, receive message
 - To host name or process name
 - From client to server
- If **shared-memory model**:
 - create and gain access to memory regions
- transfer status information
- attach and detach remote devices

The slide features a yellow background with a blue wave on the right side. A hand-drawn diagram on the right side shows two boxes representing processes, labeled 'm1' and 'm2'. Above them is the text 'Message passing'. Below each box, there are labels 'msg queue' and 'msg mailbox'. A dashed line connects the two boxes, indicating communication. At the bottom, there are two logos: one of a gear and a person, and another of a circular emblem with a star. A video inset in the bottom right corner shows a man with glasses and a white shirt speaking.

Then for communication between the processes. So, we have to have create delete this type of communication connection. So, this is one type of calls. So, if the message passing model, we have to we have got send receive messages. We have got shared memory model, then we want to clear created gain access to shared memory regions. So, it is like this, like whenever you got this multiprocessing system then one particular model that we have is the message passing model.

So, in the message passing model what is done? Between the processes that wants to communicate between themselves. So, there will be a message queue. So, this process will send a message to this one. So, every process, so you can think that this process has got a queue associated with it which we call the message queue; which we call the message queue and similarly this process also as got a message queue associated with it.

So, when p 1 wants to tell something to p 2. So, what it will do it will send the message to this message queue. Then accordingly this message queue content, so, the as soon as some message has arrived in message queue. So, p 2 wants to read from the message queue it will get that message into it. Similarly, p 2 if it want to send some message to p 1. So, it will actually be written on to the message queue or p 1 and later on when p 1 reads from the message queue will get the message.

So, this is one type of communication mechanism that many operating systems do support and naturally you need to have this type of messages you should have the system call for this send receive message to it may be sending to a hostname or to a process name. So, this is just shown in terms of two processes. But it may so happen that physically this M 1, this p 1 is in system M 1 and this p 2 is in system M 2 two different computers. So, then also we would like to have we would like to have this communication mechanism.

So, we should be able to send receive messages across host or across processes or it may be from client to server. The client wants to send some message to the server, so that way from client to server some message may go and from server to client also some message may go we must support this type of system calls in the operating system. Then the shared memory module is used. So, in shared memory model is a more tightly coupled system.

(Refer Slide Time: 19:07)

System Calls – Communications

- create, delete communication connection
- if **message passing model**:
 - send, receive message
 - To host name or process name
 - From client to server
- If **shared-memory model**:
 - create and gain access to memory regions
- transfer status information
- attach and detach remote devices

Hand-drawn diagram: Two boxes labeled p1 and p2 are connected by arrows to a central vertical stack of memory cells. The stack is divided into sections for p1, p2, and a shared section. Below the stack, there is a variable 'x' with a value of 5 and a pointer 'x8'.

So, in shared memory model what happens is that. So, I have got say these are the two processes say p 1 and p 2 and both of them are using the memory for accessing and getting. So, this p 1 this may be the portion of the memory which is dedicated for p 1. So, this may be the portion of the memory dedicated for p 2.

So, normally when p 1 is running, so it will be executing this piece of code and p 2 will be executing this piece of code. Now, if there is a variable x, which this p 1 and p 2 wants to share between them then you see there is no way because the copy of x that I have in p 1, so this memory location here and copy of x that I have in p 2 so, that is there. So, they are not the same x ok.

So, what we need to do is that we should have some portion of memory which we call shared memory ok. So, this is called shared memory. And then somehow p 1 should have a pointer to a location in the shared memory. So, it have has a pointer to this location in the shared memory, p 2 also has got a pointer to this location in the shared memory.

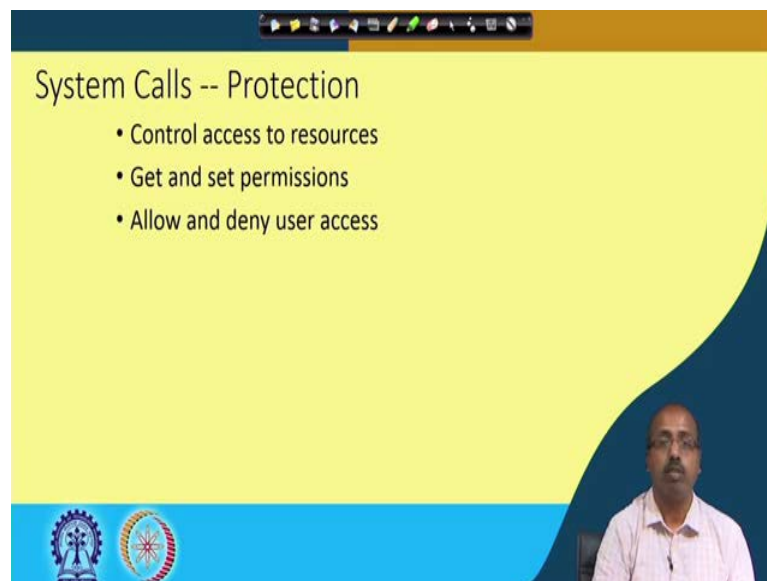
Now, if that can be established and there with the proper permission setting that this particular location is accessible by p 1 and p 2 only. So, then if p 1 wants to write something, so it can go by this pointer. So, suppose name of the this pointer is say p and this pointer is say q. So, if p 1 writes like star p equal to some 5, then what will happen? So, this particular location will get modified to 5. So, and then if p 2 it looks for the

value of star q then it will find that the value is equal to 5, so it will get this value 5 there. So, value of x there. So, this way we can have the sharing.

So, this for this shared memory realization we should there should be system calls by which we can create this memory regions to be shared and we should gain access to those regions. So, there are system calls which are useful for this creating the shared memory; we should be so, that is there. So, we will come to those calls later.

Then transfer status information. So, from one process, so we may need to tell what is the status of the process at this point of time. So, that way the status information needs to be transferred. Then attach and detach remote devices. Some device is on as remote system. So, for sometimes we want to use that system so you will make an attach call and sometimes, after sometime when the usage is over we want to detach from the device. So, that way it is detach system calls. So, those system calls are to be provided by the operating system.

(Refer Slide Time: 21:55)



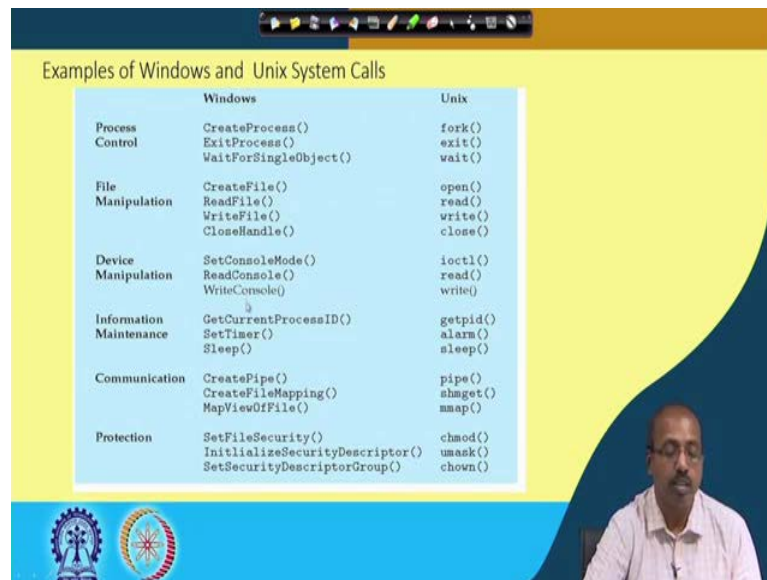
System Calls -- Protection

- Control access to resources
- Get and set permissions
- Allow and deny user access

For protection we should get control access to resources. So, this is important. So, you should have system calls that will control access to resources, get and set permission, allow and deny user access. So, these are the protection related system calls that should be available.

So, if you look into any operating system. So, this the system calls they can be grouped into this categories.

(Refer Slide Time: 22:19)



	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

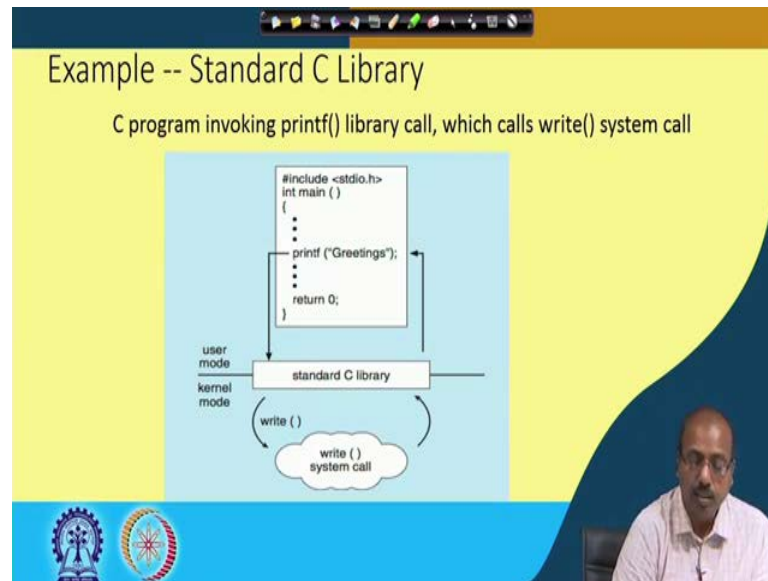
For example, so, this is say from the process control group. So, if you look into this windows operating system, so there are calls like create process exit process wait for sibling objects. So, like that similarly for Unix call, Unix system or Linux system, so we have got the corresponding system call like fork, exit and wait.

So, this create process of windows is similar to fork in Unix, exit process in windows is similar to exit in Unix. So, like that similarly for file manipulation in windows we have got create file in Unix or Linux we have got open, for reading a file we have got read file here and read here, write file for writing, write for writing. So, like that we can have different types of system calls. So, we have got close handle for closing the file. So, we have got simple close to close the file ok. So, this are the some of the cause. Like for the device manipulation we have got set console mode, similarly we have got this ioctl. So, that sets various parameters related to the IO operation

Then we have got read and write system calls from the device. So, here also we have got read write. So, in case of; in case of windows you see that device manipulation we are got read console, write console they are different from read file and write file calls whereas in Unix system. So, they are same the same read write system calls are used for

both file and devices. So, in Unix system the files and devices they are look in a more uniform fashion compare to say windows OS. Then for you know information maintenance, so we have got get current process id, in windows we have got get pid in Unix. So, like that there are many such system calls for every operating system. So, we can find this list documented in the manual.

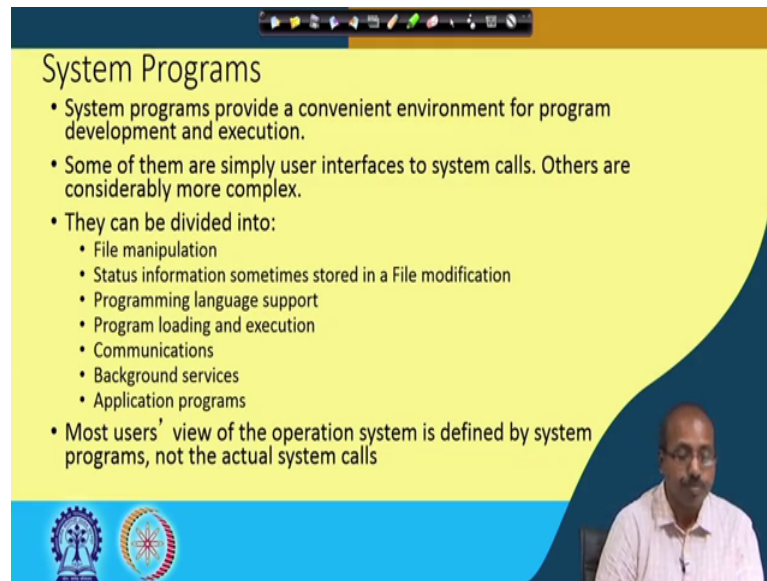
(Refer Slide Time: 24:09)



So, let us see how this standard C library routing printf is executed. So, have got a piece of program where there is a printf statement. So, this printf this will be a this is the library call library function they are available with the in the to the C program. And when this is done, so this program is was executing in user mode. So, it when this printf call is found, so, it goes to the standard C library and the standard C library it will transform this call into the write system call for the underlying operating system.

So, it makes a system call write. So, it here it is going into the kernel mode of execution. So, it is the code for printf. So, it will convert this into a write system call and the write system call is coming to the write a system call interface and then it will go to the actual execution of the write system call. After the write system call is over. So, it comes back to the standard C library. And from there it goes back there, so this way this printf library routine is not invoked from the user program and that intern invokes the write system call.

(Refer Slide Time: 25:21)



System Programs

- System programs provide a convenient environment for program development and execution.
- Some of them are simply user interfaces to system calls. Others are considerably more complex.
- They can be divided into:
 - File manipulation
 - Status information sometimes stored in a File modification
 - Programming language support
 - Program loading and execution
 - Communications
 - Background services
 - Application programs
- Most users' view of the operation system is defined by system programs, not the actual system calls

The slide features a yellow background with a dark blue curved shape on the right side. At the bottom left, there are two circular logos. A small video inset in the bottom right corner shows a man with glasses and a white shirt speaking.

So system programs they provide a convenient environment for program development and execution. Some of them are simply user interfaces to system calls, others are considerable more complex. So, system program some of them may be as simple as just making the system call some programs may be more complex. So, we will see how are they divided into different classes as. So, as there are different parts of the operating system.

So, system programs may be also of different types, there are some programs for file manipulation, some for status information and sometimes stored in a file modification, then programming language support, program loading and execution, communication background device, services and application program. So, most users view of the operating system is defined by system programs not the actual system calls because the system calls may not be visible to the user and they just see the system programs that are available.

(Refer Slide Time: 26:29)

System Programs

- **File management**
 - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- **Status information**
 - Some programs ask the system for information - date, time, amount of available memory, disk space, number of users
 - Others programs provide detailed performance, logging, and debugging information
 - Typically, these programs format and print the output to the terminal or other output devices
 - Some systems implement a **registry** - used to store and retrieve configuration information

Handwritten notes: /home / users, abc, copy -> cp, /bin/cp, open, read, write, close, close.

So, system programs for file management like this create, delete, copy, rename, print, dump, list and generic generically manipulate this files and directories. So, these are the file management related system programs.

So, why do we call them as program because of several reasons. Like if you look into for example, the Unix or Linux operating system, so you will be finding that whenever suppose I have got a user whose name is user 1. Normally, the user 1 will have a directory slash home plus user one and in this directory maybe the user has got a program abc when the user gives the program for execution, so it runs. But sometimes the user may need for running some other system program the system program likes to copy a file we have got a command cp.

Now, there is a specific directory in my system where the code for cp is located. So, it may be the directory slash bin and slash cp. So, this is the file in the slash bin directory will find a file whose name is cp this is the this is an executable file. So, this is actually the program that will be executed if you give the cp command.

So, this cp command will intern use the system calls. Like it has to a for executing this cp command. So, it has to open the source file, it has to open the destination file, it has to do a number of read and write system calls. So, as we have seen previously it has to do a number of read write system calls in a loop and then finally, close the files, but as if user

of the system we do not see this thing. So, this part is not visible to us, so we just see that there is a file cp. So, that cp is a system program and it internally uses the system calls.

So, system calls are not visible to the user what is visible is the system program. And the reason why is it called a program because they are also kept in certain directories and they can be executed from there. So, this in some cases this system programs can be modified like you can have your own version of the program cp and put it into the slash bin directory so that it will be using your program for copying rather than using the system program cp.

For some operating systems it may not be possible. So, there this system programs are not available separately, so they come they come as part of the command line interface directly. So, that way it is, so they are not modifiable, but say operating systems like Unix Linux, so they will allow you to do this thing. So, these are file management related system programs.

Then for status information. So, some programs asked the system for information date, time, amount, available memory disk, space, number of users. So, they are some there is some programs for doing that. There are some programs that provide detailed performance, logging and debugging information. So, again they will be internally using system calls, but that is not visible to the user. Typically, these programs format and print the output to the terminal or other output devices.

(Refer Slide Time: 29:57)

System Programs

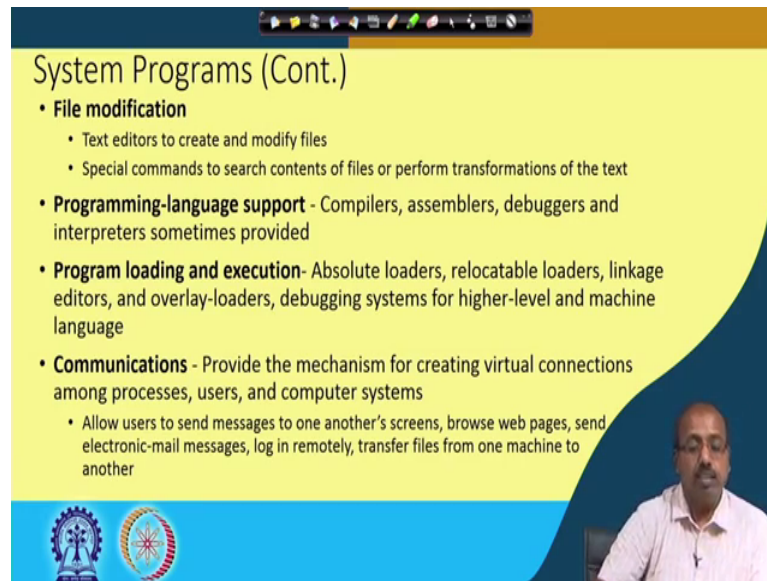
- **File management**
 - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- **Status information**
 - Some programs ask the system for information - date, time, amount of available memory, disk space, number of users
 - Others programs provide detailed performance, logging, and debugging information
 - Typically, these programs format and print the output to the terminal or other output devices
 - Some systems implement a **registry** - used to store and retrieve configuration information

So, it is like this as I said that particularly if you look into the Linux operating system the command ls that list the files that are there in the directory. Now, this ls command it has got a number of options like ls minus l, l s minus al, there are lots of commands, so lots of argument that you can pass to ls.

Now, when this ls is implemented as a system call. So, system, so it gets all the list of files and all, but that it gets the complete data is available to it, but that data how it will be presented to the users, so that is the concern. So, this is the whole raw data that the system call has returned and then it is formatted in proper fashion depending upon the options that are specified by the user so that it comes in the way the user was working for. So, this cosmetic changes or this format conversions are done by the system programs. And the system calls they will return the raw data from the system and the system programs they will format it in the form that the user has asked for.

So, this is the thing that they typically this they format and print the output to the terminal or other output devices. And some systems implemented registry, so used to store and retrieve configuration information. So, this configuration information are stored, so that for update and also it becomes easy. So, you know where to look for the update and all. So, this is there. So, the sometimes we have got this configuration.

(Refer Slide Time: 31:29)



System Programs (Cont.)

- **File modification**
 - Text editors to create and modify files
 - Special commands to search contents of files or perform transformations of the text
- **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided
- **Program loading and execution**- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- **Communications** - Provide the mechanism for creating virtual connections among processes, users, and computer systems
 - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

Then file modification. So, this we have got this text editor that can do file modifications, then we have got programming language support, we have got program loading and program loading and execution their there are system programs, for communication also there are system program. So, we will look into this in the next class.