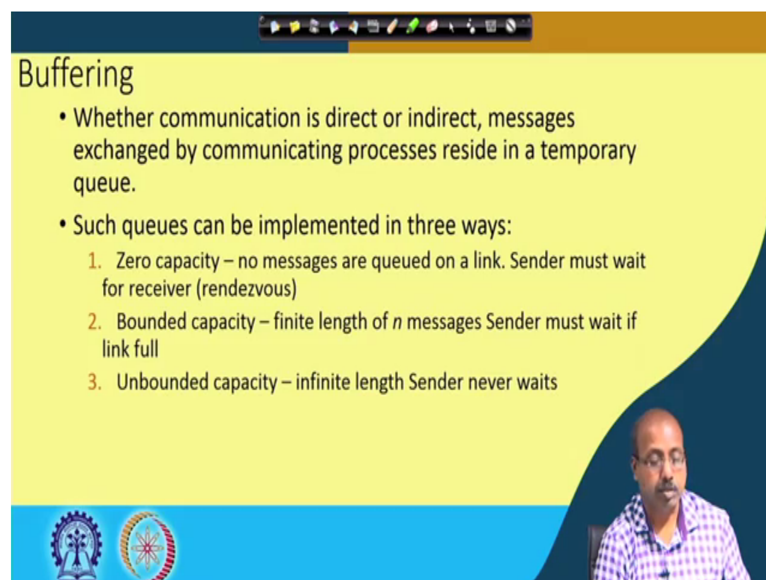
Operating System Fundamentals
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 18
Processes (Contd.)

So, this buffering type of situation when these messages are being passed. So, whether the communication is direct or indirect messages are exchanged by communicating processes decide in a temporary queue.

(Refer Slide Time: 00:37)



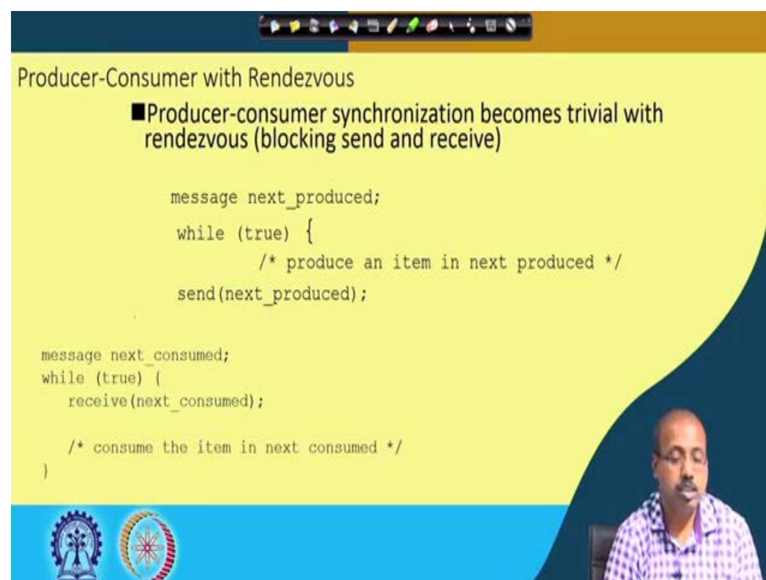
So, we have got a temporary queue in which the messages are kept and these queues may be of different implemented in three ways. So, zero capacity. So, no messages are queued on a link. The sender must wait for this receiver. So, this is a situation when there is no buffering. So, sender sends a message and the sender will be waiting for the message to be consumed. So, this is one type of situation. We can have some bounded capacity. There is a finite length of n messages and sender must wait if the link is full. So, we can say that they are at most n messages can be sent and messages they have got some fixed size.

So, at most n messages can be sent and then they are kept in the buffer. So, produce the sending process, so it need not wait up to these n messages, but after these n messages has been transmitted if none of them are received by some receiver message or process

then this sender process is made to wait and there can be unbounded capacity like infinite length sender as a sender never waits. So, of course, this is a hypothetical situation because I cannot have unbounded capacity we are due to this limited memory size, but practically we can do that because this never in practicality. So, this receiver will definitely be there in the system.

So, it will also consume some messages. It will receive some messages. So, as a result this the sending process it will never be made to wait for the, all the messages to be taken by the receiver.

(Refer Slide Time: 02:17)



So, this is one example for synchronization becomes trivial with this blocking, send, and receives with it in diverse type of situation. So, this message next, so this is a producer or the sending process so while true so, it produces and item and send the item produced. So, I am sorry there should be a closing brace here that is missing. So, this producer process or this producer process ends at this point and this consumer process is just receiving messages.

So, that way so the operating system will ensure that the buffer full condition is taken care of by itself. So, we do not have to do anything any point at check or anything like that because the buffer full or buffer empty conditions are taken care of by the underlying message passing mechanism that the operating system supports; so, this is blocking, send, and receive.

So, if this if the sender the producer process sends some message that that contains the next produced item and it finds that there is no receiver for that. So, the center process will be made to wait at this point. So, it will not, it will cannot it will not go for go on filling of the buffer space completely and overwriting them. On the other hand these communications they become the receiving process that acts as the consumer.

So, this will be executing the receive system call. So, if the buffer space is empty there is no message in that here in the queue then that process will also be made to wait. So, this receiver next consumes. So, it will be waiting for the message to be ready for next consumption. So, this way we have got this producer consumer problem solved with the message queue operation.

(Refer Slide Time: 04:01)

Pipes

- Acts as a conduit allowing two processes to communicate
- Issues:
  - Is communication unidirectional or bidirectional?
  - In the case of two-way communication, is it half or full-duplex?
  - Must there exist a relationship (i.e., ***parent-child***) between the communicating processes?
  - Can the pipes be used over a network?
- Ordinary pipes – cannot be accessed from outside the process that created it. Typically, a parent process creates a pipe and uses it to communicate with a child process that it created.
- Named pipes – can be accessed without a parent-child relationship.

There is another type of IPC which is known as pipe. So, this acts as a conduit allowing two processes to communicate. So, it is like this that we have got two processes say P 1 and P 2 and we have got a pipe established between them. So, there is a pipe here. So, whatever this process P 1 writes here P 2 can read it. Similarly whatever P 2 writes here P 1 can read it.

So, acts as a conduit allowing two processes to communicate. So, instead of going into this message send, receive, and this formatting and all so, you can simply take this pipe and then this read write we done in terms of the pipe. So, there are issues like is the communication bi-directional or unidirectional? The example that I have taken here so is

a bidirectional pipe of course, we can have unidirectional pipe also. In the case of two way communication is it half or full-duplex? So, half-duplex means at one point of time communication is from P 1 to P 2 only and at some other time it is from P 2 to P 1 only.

So, that is the half-duplex type of communication and full-duplex means so, they can be occurring arbitrarily. So, P 1, so, the communications can go on simultaneously in both the directions, but in case of this half half-duplex communication can go in one direction at a time only. So, we have got this two way communication. So, we have to answer whether it is half-duplex or full-duplex. Again this is OS dependent. So, different OS will allow different level of communication.

Must there exist a relationship between the communicating processes, is it mandatory that P 1 and P 2 should have a parent-child relationship between them? Some operating systems will tell that there should be a parent-child relationship, some operating systems will tell no it is not necessary. So, you can establish pipe between any pair of processes.

So, that way we can have the relationship as a constraint and can the pipes we used over a network? So, they physically they have two processes they do not belong to the same processor. So, can we have that type of situation or the same computer can we have this pipe established over a network? So, these are some of the issues. So, different operating systems will answer these issues to different extents. The ordinary pipes so the pipe can be classified into two categories. One of them is this ordinary pipe and the other one is known as the named pipe.

In case of ordinary pipes, so, they cannot be accessed from outside the process that created it. So, one process creates a pipe and it can be accessed from within the process only. Typically a parent process creates a pipe and uses it to communicate with child process that it create it. So, what can happen is that, we can have these parent process P.

(Refer Slide Time: 07:03)

Pipes

- Acts as a conduit allowing two processes to communicate
- Issues:
  - Is communication unidirectional or bidirectional?
  - In the case of two-way communication, is it half or full-duplex?
  - Must there exist a relationship (i.e., ***parent-child***) between the communicating processes?
  - Can the pipes be used over a network?
- Ordinary pipes – cannot be accessed from outside the process that created it. Typically, a parent process creates a pipe and uses it to communicate with a child process that it created.
- Named pipes – can be accessed without a parent-child relationship.

And, in its code somewhere it has created a pipe and after that it makes a fork system call as a result a child process C is created. Now between this pipe is between the parent and child. So, this type of situation is called ordinary pipe. So, the ordinary pipes so, since there is no other idea associated with the pipe you cannot have some other process communicate via this pipe some other process P dash it cannot communicate with this pipe.

So, typically a parent process creates a pipe and uses it to communicate with a child process that it created and there are named pipes it can be accessed without parent-child relationships. So, this is another so, you can you have got some identifier for the pipes, for this pipe and then you can have this without having this parent-child relationship also you can make multiple processes to communicate via the pipe.

So, these are the different alternatives we have and again different operating systems will support it to different extents.

(Refer Slide Time: 08:09)

Ordinary Pipes

- Ordinary Pipes allow two process to communication in standard producer-consumer style
- Producer writes to one end (the **write-end** of the pipe)
- Consumer reads from the other end (the **read-end** of the pipe)
- Ordinary pipes are unidirectional, allowing only one-way communication.
- If two-way communication is required, two pipes must be used, with each pipe sending data in a different direction.
- Require parent-child relationship between communicating processes

So, we will look into these ordinary pipes first. So, in case of ordinary pipes they will allow two processes to communication in standard producer-consumer style. So, in a producer consumer style, so producer writes to one end, the right-end of the pipe and the consumer reads from the other end the read-end of the pipe.

So, ordinary pipes are unidirectional only one-way communication is possible. If two-way communication is required two pipes must be used with each pipe sending data in a different direction. So, this requires read-write parent-child relationship between communicating processes. So, this is ordinary pipe as such. So, this is going to be a unidirectional pipe. So, this is the parent process which has created the pipe and I have it erase communicating with a child. So, there are two ends of a pipe.

So, this is end-1 and this is end-2. This end-1 is the write-end, this is the write-end and this is the read-end or if I take this one as read-end then the other one will be write-end. So, you can so, producer if you keep the producer here, so producer will write on to this read on this end-one and consumer will read from the read-end that is end-two of the pipe. So, they are so, we have got only one pipe. So, this at the write-end it will write it and at the read-end it will read, but it cannot be done side two way like it you cannot do it like this that this producer process which is fitted with the parent here.

So, it writes here and also it reads from the pipe. So, that cannot be the case. So, whatever is written here can be read from this one only and whatever is written at this

ends so, can be read from this end only. So, essentially so there are there are basically two pipes that are created. So, if we want this two-way communication then I have to have two per two pipes created. So, if you want to per so, if you want both of these to happen that is you want to write at end-1 and read at end-2 and also whatever is written at end-2 you want to read it a trend or a read at end-1. So, in that case I should have two pipes created.

So, this is a two-way. If two-way communication is required, two pipes must be used and each pipe sending data in a different direction. So, require parent-child relationship between communicating processes. So, until and until and unless that is there will not have this pipe we cannot have this pipe implemented.

(Refer Slide Time: 11:07)

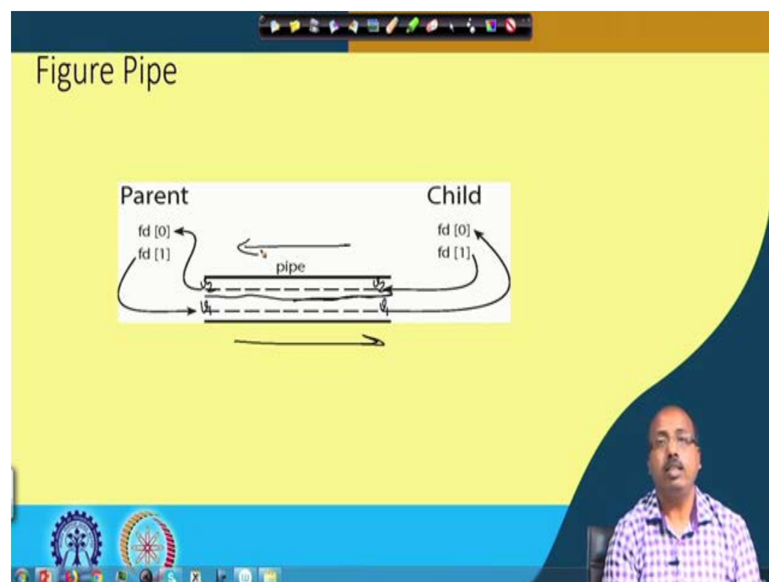
Now, so, this is one situation. So, this is for UNIX operating system. So, this is the thing. This is an example from the UNIX operating system. Ordinary pipes are constructed using the pipe system call and pipe int fd. So, it is a file descriptor. So, file descriptor so, this size should be 2 ok. So, this size should be 2.

So, I should have this fd sides as 2. So, what it does is that. So, we have to have this fd defined to be 2, size is equal to 2. So, there are 2 descriptors, one is fd 0 and other is fd 1. If fd 0 is the read-end of the pipe and fd 1 is the write-end of the pipe. So, this is so, this fd 0 the parent process if you want when it wants to read something. So, it will be doing it via fd 0 and if it wants to write something it will be doing it via fd 1. So, for this so,

this arrow I think you better think in this drive this direction. So, for fd 1 is the write-end of the pipe.

So, when child is trying to write so, it is writing on to this fd 1 as a result if the parent reads it and fd 0. So, it will get it here. Similarly if the parent is trying to write something it will write in the fd 1, it will write it onto fd 1 and it will write it onto fd 1 and if it wants to read it, so, it will do it in the fd 0. So, this pipe handling is same as file handling that we have in UNIX. There are read and write system calls by with these pipes can be implemented.

(Refer Slide Time: 12:59)



So, will be so, we can say make some. So, this is the thing. This diagram actually explains it more clearly. So, whatever is child whatever parent is writing. So, parent is writing on fd 1. So, for both the ends parent and child fd 0 is the read pointer read descriptor and fd 1 is the right descriptor. So, similarly for the child fd 0 is the read descriptor and fd 1 is the right descriptor, but the system will ensure that when this child is writing on to fd 1 if the parent reads by fd 0, so it will get that content. Similarly if parents write something on to fd 1 the when the child is reading it by fd 0 it will get that content.

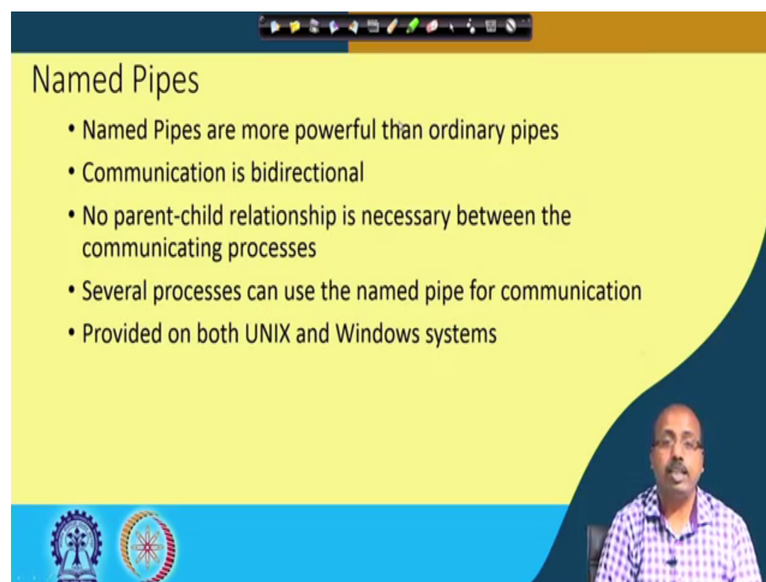
So, it will not happen like this that the child has written using fd 1 something here and when the child executes say read it will get that content. So, it will not happen like that. So, when the child executes say read on fd 0 it will get whatever is written by parent in

fd 1. So, that is the thing. So, if some value v 1 is written here, if some value v 1 is written here so, that v 1 will be available here.

Similarly some v 2 that is written by the child this v 2 will be available here. So, it is not that the child the child has written v 2 so, while it does a read it gets this v 2 it will get the v 1 which the parent has written. So, essentially what happens is that in case of UNIX you can understand that this is virtually divided into 2 pipes ok. So, one is in this direction.

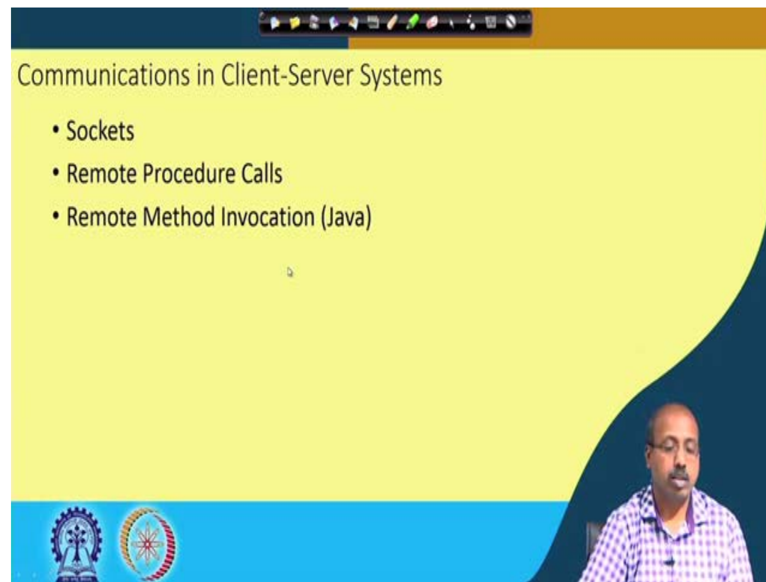
So, this pipe is in this direction and the upper pipe is in this direction. So, we have got 2 pipes clubbed together in the UNIX operating system. So, we have got 2 pipes implemented there. So, this is. So, different operating systems will handle them differently. So, this is the way UNIX handles the pipes.

(Refer Slide Time: 14:57)



So, there can be another type of pipe which is called named pipes. So, named pipes are more powerful than ordinary pipes and the communication is bi-directional, no parent-child relationship necessary between the communicating processes and several processes can use named pipe for communication. So, we can have different types of communication implemented using named pipe and provided on both UNIX and Windows operating systems. So, we have got this named pipe type of implementation.

(Refer Slide Time: 15:27)



So, you I will suggest that you look into the manual of either of these two operating system to get some idea about how this is done, because that is a bit complex. So, it also with the idea of understanding this unnamed pipe so you can also be able to understand that the calls will be slightly difficult different. Then in case of client-server environment, so there is another type of communication.

So, client-sulfur environment so, then there are client processors client processes and there is server processes they may be located on the same computer or they may be distributed over the network. So, as a result we will have different types of communication mechanisms that we have, one is first one is called socket. So, if you are think simply of data transfer between two systems or two processes or one client and one server.

So, we can there are sockets for doing that. There are remote procedure call by which you can invoke a procedure on a remote processor. So, remote procedure called a remote system. So, remote procedure call is there and there is a remote method invocation which is specific for the Java type of situation.

(Refer Slide Time: 16:37)

Sockets

- A socket is defined as an endpoint for communication
- Concatenation of IP address and port – a number included at start of message packet to differentiate network services on a host
- The socket **161.25.19.8:1625** refers to port **1625** on host **161.25.19.8**
- Communication consists between a pair of sockets
- All ports below 1024 are ***well known***, used for standard services
- Special IP address 127.0.0.1 (loopback) is used to refer to system on which process is running. That is, when a computer refers to address 127.0.0.1, it is referring to itself.

So, first we talk about the sockets, a socket is defined as an end point for communication ok. So, any system that we have, so, if it supports a number of sockets, so, there is also basically if this is the system. So, conceptually you can think that there are a few sockets here, but each socket is an endpoints of the communication can take place through these sockets. So, this is some system it supports the number of sockets like that. So, this concatenation of IP address and port a number included in the at the start of message packet to differentiate network services on a host.

So, any message that is coming on to this through the network, so, they are ultimately connected over a network. So, this message that is coming. So, if this will have the IP number and the service this port number. So, there are suppose this port number is 100. So, this is 110, this is 120 like that. So, there the port ID will be mentioned 100. So, as a result this message is coming to this system over a network and then it can identify to which port this message should go. So, this number is included at start of message packet to differentiate network services on a host. So, for example, the socket this 161.25.19.8: 1625 refers to the port 1625 host this one.

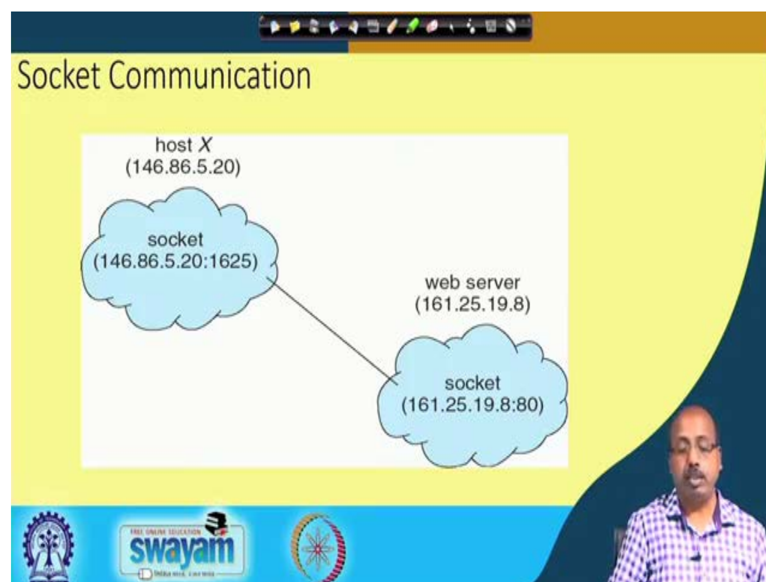
So, for two different ports so, we can associate different services and communication consists between a pair of sockets. So, for, so this is system 1 and so, there is another system, system 2 for example so, from where this communication is coming. So, system 2 a also has got some ports here and this system 2 is sending this message to system 1 by

telling the corresponding IP address and the socket port ID. So, all ports below 1024 are well-known and they are used for standard devices.

So, across any operating system so, there is some worldwide fixed services for these ports and therefore, they are fixed for that purpose only. So, up till 1024 they are fixed. So, if you have some specialized service on some system on some server, so, they should be numbered beyond this. The port number should be beyond this 1024. So, we have got special IP address 127.0.0.1, which is loopback. It is used to refer to system on which the process is running that is when a computer refers to address 127.0.0.1 it is referring to itself.

So, it can be used to give a loop back and send some message to itself and see whether the server that is running on this system can respond to the messages coming like coming to it. So, if it is even if there is a network problem so, the weather my server is or not. So, I can send it by, I can check it by sending this message here to itself. So, this special IP address the loopback address 127.0.0.1 then. So, these sockets are basically used for a distributed system.

(Refer Slide Time: 19:59)



So, like this, so, we have got a host X and a web server. So, host X it sends a message through that through this socket 146.86.52:0:1625. So, this host X has got a number of ports out of that this particular port I am sending a message and this is sending a message to the port 80 of this web server. So, this particular server 161.25.19.8:80 so, this is a

server and the corresponding socket port ID is 80. So, it is sending a message to this one. So, this way over a network we can specify the IP address and the socket port number by that we can tell to which communication to which port the communication should take place.

(Refer Slide Time: 20:45)

Remote Procedure Calls

- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems
  - Again uses ports for service differentiation
- Stubs – client-side proxy for the actual procedure on the server
- The client-side stub locates the server and marshalls the parameters (marshalling involves packaging the parameters into a form that can be transmitted over a network).
- The server-side stub receives this message, unpacks the marshalled parameters, and performs the procedure on the server
- On Windows, stub code compile from specification written in Microsoft Interface Definition Language (MIDL)

There is another type of communication or activity which is known as remote procedure call. So, remote procedure call or RPC, it abstracts procedure calls between processes on networked systems. So, as a user of the system we view it as if the procedure has been invoked in the host machine, but in reality it has gone to a distant processor over the network and then it is executing there. So, again we have got ports for process i a differentiation there are stubs at the client side proxy for the actual procedure on the server. So, what I mean is that, suppose I have given a call to a procedure says proc 1, but this proc 1 happens to be a remote process. So, this is executing some, this proc 1 is executing on some other process other processors.

So, this is on P 1, this is on processor P 2 over the network. So, they are connected. So, this P 2 this proc 1 is actually here. So, when this message comes via this network to this P 2 so, it will invoke this process one, but for the sake of compilation and also we keep some dummy processes in process P 1 which is called P 1 and they are called stubs. So, they are the proxy for the actual procedure on the server. The client side stub it locates

the server and Marshalls the parameters. So, what it happens is that this stub routine that we have here.

So, this will this is called here so, it will locate where exactly this routine is located accordingly the parameters that is passed. So, they are put onto a network format and then they are sent over the network. So, this message passing is done by this, stub procedure proc 1 at processor at processor P 1. Whereas, a processor P 2 it will do the actual execution. So, there also we have got a server side also has got a stub. So, it also has got a stub here. So, it receives this message.

So, this message that is sent so, it gets this message and unpacks the Marshall parameters and performs the procedure. So, there it will make a local call to this procedural P 1 and then this is actually returned via this to this to this client server client side stub and that will go back to the original call process one. So, this way by means of stub processes at this client side and receiver side they with this procedure remote mode procedure calls are executed. So, on Windows the stub code compiled from specification written in Microsoft Interface Definition Language or MIDL

So, different operating systems they will have their own implementation. So, I would suggest that you look into the remote procedure called mechanism font for different processors or different operating systems that we have.

(Refer Slide Time: 23:43)

Remote Procedure Calls (Cont.)

- Data representation handled via **External Data Representation (XDL)** format to account for different architectures.
- Must be dealt with concerns differences in data representation on the client and server machines.
- Consider the representation of 32-bit integers.
  - **Big-endian.** Store the most significant byte first
  - **Little-endian.** Store the least significant byte first.

  Big-endian is the most common format in data networking . It is also referred to as **network byte order**.
- Remote communication has more failure scenarios than local
  - Messages can be delivered ***exactly once*** rather than ***at most once***
- OS typically provides a rendezvous (or **matchmaker**) service to connect client and server

So, data representation handled via external data representational or XDL format. We account for different architectures, in different architectures the data representation format may be different.

So, this MIDL, so it will allow to have different types of external data representation and must be dealt with the concerns differences in data representation on the client and server machine. One particular configuration change that we have is that some processors they will follow big-endian format. Some processors they follow little-endian format. So, for 32 bit integers, so, 32 bit integer if I have. So, this is a 32 bit integer ok. So, it has got 4 bytes in it. Now you see that the memory is the byte organized. So, this is this is memory is not 32 wide.

So, if I say that this is the least significant byte and this is the most significant byte. Now if I say that this number is stored from location 1000 then how do you store the bytes. If this is the location 1000, so, the number will span over 4 such bytes here 1000, 1001, 1002, and 1003. Now where do you store the most significant byte and where do you store the least significant byte? In case of big-endian format it stores the most significant byte first.

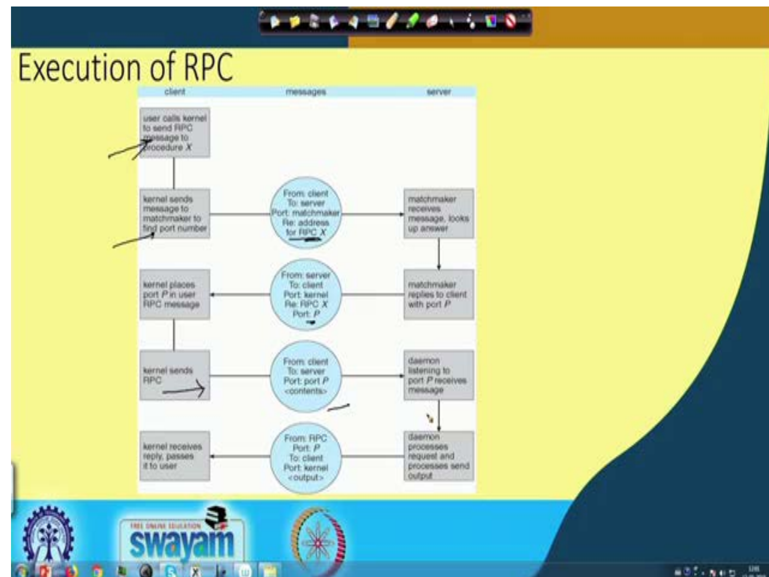
So, it will be storing the MSB here and LSB at the end. So, that is the big-endian format and in case a little-endian format it will store the least significant byte first. So, it will store the LSB at 1000 and MSB at 1003. So, we have got both the formats are valid and you see that when you are using it on the same processor, it does not make any difference really this big either way you store it does not matter, but in case of this little-endian big-endian difference we are across the host or machines.

So, we should have this format conversion. So, big-endian is the most common format in data networking. It is also referred to as network byte order because of the because of its a wide acceptance and this remote communication has more failure secure scenarios than local scenario. Because, there can be that may be the computation has been carried out correctly, but this message was message did not receipt is not received or the messages they can be delivered exactly once rather than at most once.

So, that is one problem and OS typically provides a matchmaker service to connect client and server. So, that is there like which server is having which services available. So, there is some sort of directory service and using the directory services is finds out like

which port has to be connected and then it connects to that particular port. So, this way we have got this remote procedure calls implemented in different operating systems.

(Refer Slide Time: 26:49)



The next we have, so this is the execution of the RPC. So, starts so, this in on the client side the user calls this kernel to send RPC message to proceed X then kernel sends message to matchmaker to find the a port number. So, from client from client to server the matchmaker it sends a message to the port number matchmaker and with the address of RPC X and the matchmaker it receives the message and looks up for answer like which one what is the exact port number for that? The matchmaker now replies to the client with port P. So, that message comes to the kernel sorry comes to the client machine where it has got the port number and the port number that we have.

So, that comes to the kernel port. Now this so, then it sends the RPC the remote procedure call request and this remote procedure call request that is also coming as a message and there is a daemon which listens to process port P receives the message and this daemon processes the request and processes and the sent the output and after the output is obtained, so it will be sent back to the client machine.

So, this way a number of messages are exchanged for this remote procedure called execution and in the process a number of messages will be transferred between them and one of them is the directory service where it is find by means of this matchmaker port it is trying to find out the corresponding service number service port and then once that is

informed to the client then the client will send the request to the appropriate port of the server. So, this way RPC is execute and that helps in invoking remote procedures over a network.