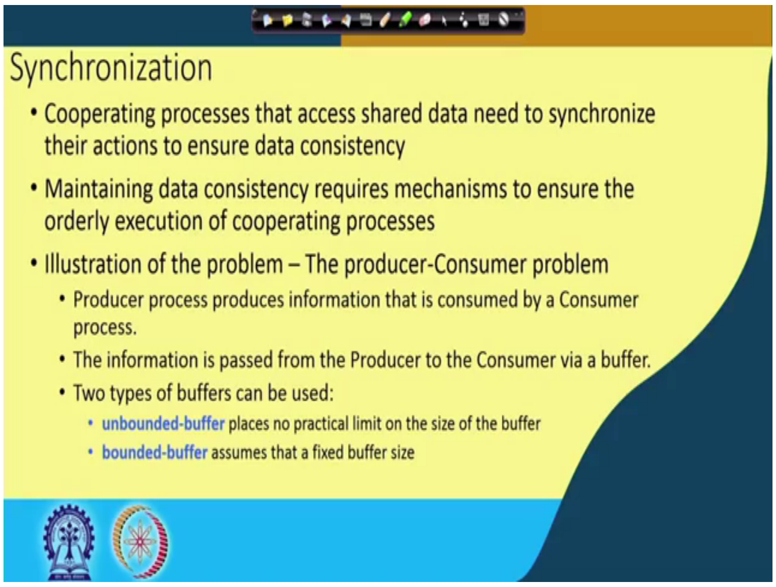
Operating System Fundamentals

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

Lecture – 17
Processes (Contd.)

So, we were discussing about this producer consumer problem. So, in case of producer consumer problem what happens is that is a very standard problem that we have in operating system.

(Refer Slide Time: 00:33)

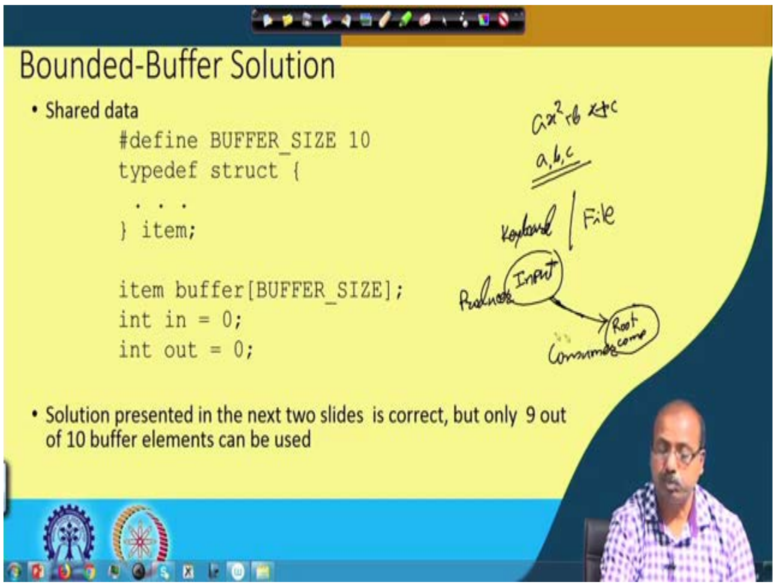


Synchronization

- Cooperating processes that access shared data need to synchronize their actions to ensure data consistency
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes
- Illustration of the problem – The producer-Consumer problem
  - Producer process produces information that is consumed by a Consumer process.
  - The information is passed from the Producer to the Consumer via a buffer.
  - Two types of buffers can be used:
    - unbounded-buffer places no practical limit on the size of the buffer
    - bounded-buffer assumes that a fixed buffer size

So, there are some processes which are producing some data and there are some process which are which is actually consuming that data. And, example of it that we have told in the last class like there may be I have a program written that finds roots of a quadratic equation.

(Refer Slide Time: 00:54)



Bounded-Buffer Solution

- Shared data

```
#define BUFFER_SIZE 10
typedef struct {
  . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

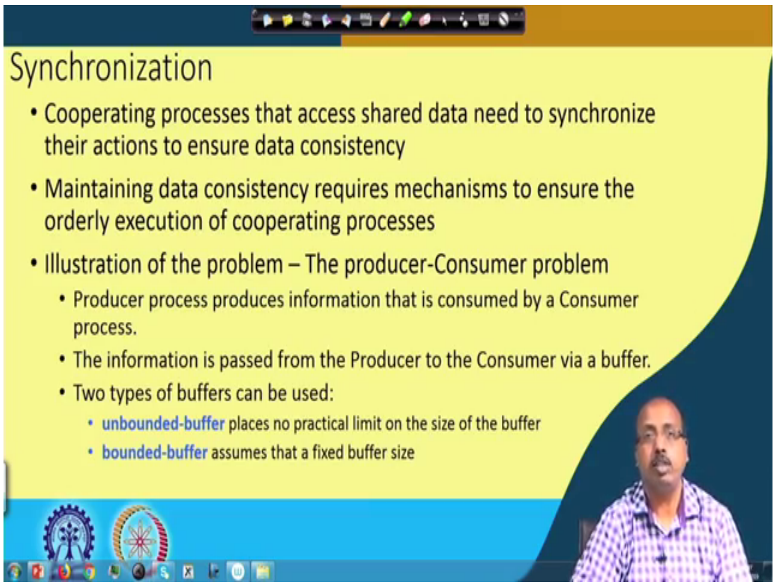
- Solution presented in the next two slides is correct, but only 9 out of 10 buffer elements can be used

So, that program so, that is so, it is doing like a x square plus it is finding roots for the equation $ax^2 + bx + c$. Now this program so, it needs to read the values of a b and c from the user, and then it will proceed finding the roots. Now this reading process of this abc so, they are from either from say keyboard or they may come from some file etcetera, they may come from some file etcetera. So, whatever it is so, there is an input process. So, I can say that there is an input process from where the data is coming and then this input process is there.

So, it is a in communication with the program that is actually doing this computation. So, this root computation program that with that is there. So, it is actually there in communication with each other. Now if this input process is slow which is often the case when, human beings are involved. So, then it is root computation process it reads something from this is abc. So, they are in some shared memory location as you can understand, because they are shared between the input process and this root competition module. Now, if this abc is the if the root competition module is very fast then of course, it will not get the proper values of a b and c.

On the other hand if this, if there is a continuous flow of roots. So, if I just modify the statement of the program that it gets a series of values of a b and c and for each of them it computes this value or the roots ok. So, if we say that it will go like that, then there may be a case that this data set is coming at a much faster rate then than the rate at which those roots can be computed the corresponding roots can be computed by the root finding routine. Now that type of situation how to handle? So, there basically what I want to mean is that there is a speed mismatch between the input process and the computation process or in terms of this the producer process. So, the input process that is the producer process and we have got this consumer process so, that is the root finding process. So, if there is a speed mismatch between this producer process and consumer process then how to handle the situation.

(Refer Slide Time: 03:27)



Synchronization

- Cooperating processes that access shared data need to synchronize their actions to ensure data consistency
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes
- Illustration of the problem – The producer-Consumer problem
  - Producer process produces information that is consumed by a Consumer process.
  - The information is passed from the Producer to the Consumer via a buffer.
  - Two types of buffers can be used:
    - unbounded-buffer places no practical limit on the size of the buffer
    - bounded-buffer assumes that a fixed buffer size

So, that if we want to handle, then we can have we can put a buffer in between we can put between the producer process and consumer process a buffer and so, that if the producer processes fast then it will be writing on to that buffer and the consumer will be consuming from this buffer. So, some of the some part of the speed mismatch will be taken care of.

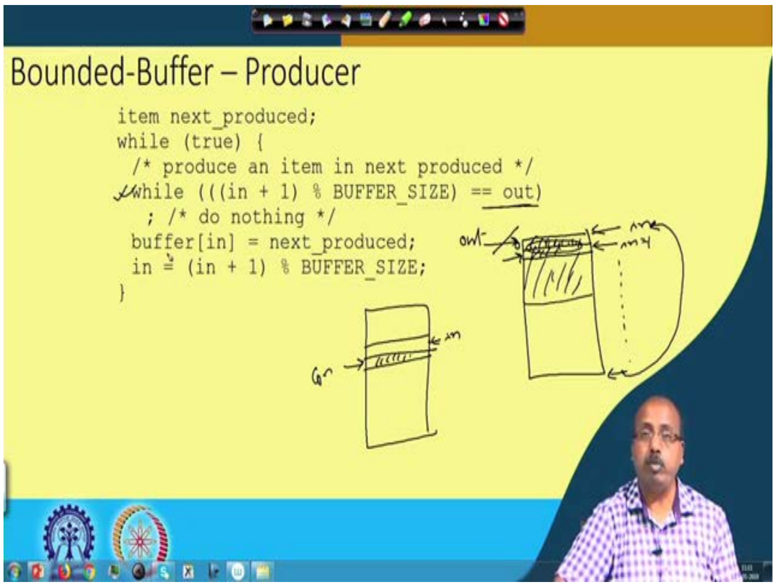
And again this buffer can be or two types one possibility is that buffer is unbounded in size. So there is no practical limit on the size of the buffer. So, though it appears to be impossible, but for all practical purposes; so, we can put a sufficiently large buffer so, that this producer will never be able to fill up this buffer if the consumer is also running. Other possibility is that which is more realistic in nature the buffer size is bounded. So, that is the bounded buffer situation, the buffer size is buffer has got a fixed size.

So, in that case when the data is produced by the producer after sometime producer has to be made to wait so, that the consumer consumes some of the buffer content and then only the producer will be able to proceed again. So, we will see that there are two variants of this problem and each of them gives rise to different levels of synchronization problems.

So, if I say that for the bounded buffer problem. So, if you have to define some buffer size and this then this item. So, we have to do whatever be the content of this buffet type so, that is not written here. So, it can be depending on the application that we have. So, there can be different types of items in the buffer.

So, then we can so, we are defining the type def item and then item buffer. So, this buffer is a size buffer size then there are input pointer and output pointer input index and output index. The solution presented here is correct, but only 9 out of 10 buffer elements can be used.

(Refer Slide Time: 05:29)



Bounded-Buffer – Producer

```
item next_produced;
while (true) {
  /* produce an item in next produced */
  while (((in + 1) % BUFFER_SIZE) == out)
    ; /* do nothing */
  buffer[in] = next_produced;
  in = (in + 1) % BUFFER_SIZE;
}
```

So, let us see how is it done. So, bounded buffer producer so, the producer problem so, we it what we does is that the process is an infinite loop. So, while true; so, this is an infinite loop if produces an item in the next produced this variable. So, next produced is a variable of type item. So, the time is produced in next variable. So, while in plus 1 person buffer size not equal to out. So, do nothing. So, basically is equal to equal to buffer. So, I have got a buffer sorry.

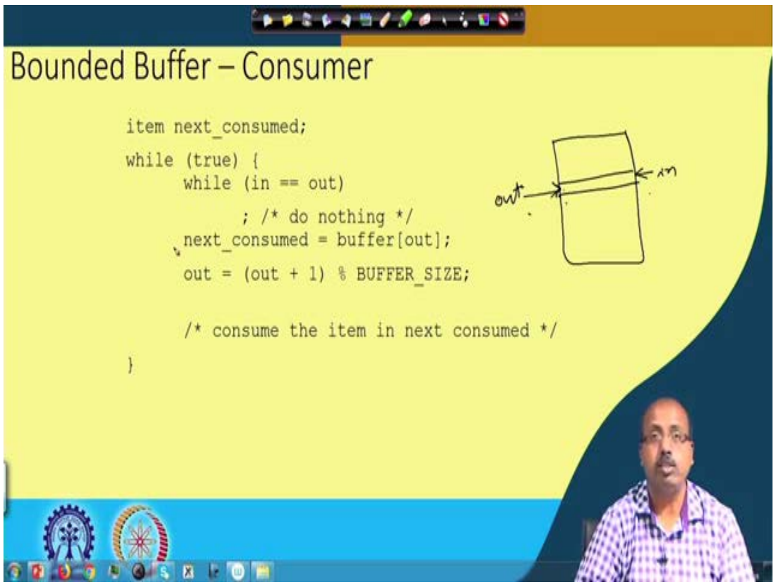
So, I have got a buffer and initially this input and output so, both of them there. So, this input and output both the indices so, they are made equal to 0. So, this is the buffer so, the input pointer and output pointer, so, both are equal to 0 this is the location 0 now. So, in plus 1 percent so, buffer is empty; so, in plus 1 percent buffer size so, in plus 1 is 1 percent buffer size is not equal to out. So, it will not be here. So, it will be executing this statement buffer in equal to next produce at location in. So, location 0 it will produce an item and write it here, then this in pointer will advance in will come to be equal to 1.

So, this way assuming that the consumer is not there so, this buffer will be filled up 1 by 1 and this in pointer will come to this last space buffer this value of 9 and then it will go back to in equal to 0. So, it will go back to their now that is how this producer process runs and only thing is that; so, after so, maybe this consumer process is present, but the consumer is a bit slow. So, after this producer has produced say up to this much of item. So, consumer came and consumer consumes something.

So, this consumer pointer will come to the next place ok. So, it will come to the next place. So, that when this producer comes back it finds that, some part of the buffer is already consumed so, in this part so, it can continue writing a here. But it may so, happen if the consumer is slow, but producer is fast then it is at a situation where this producer. So, consumer is currently at this point say it will next consume this one this item and producer is at this point the input pointer is somewhere here.

So, it is trying to write on the next location after this. So, at that point this while loop will make it to wait, while in plus 1 percent buffer size is equal to out so, in that case the producer is made to wait. So, that is how this buffer if the buffer is filled in the producer is made to wait till the consumer as consume something. So, the reverse things should happen about the consumer because consumer should not consume from a from an empty buffer. So, how that is done?

(Refer Slide Time: 08:45)



Bounded Buffer – Consumer

```
item next_consumed;
while (true) {
      while (in == out)
            ; /* do nothing */
      next_consumed = buffer[out];
      out = (out + 1) % BUFFER_SIZE;

      /* consume the item in next consumed */
}
```

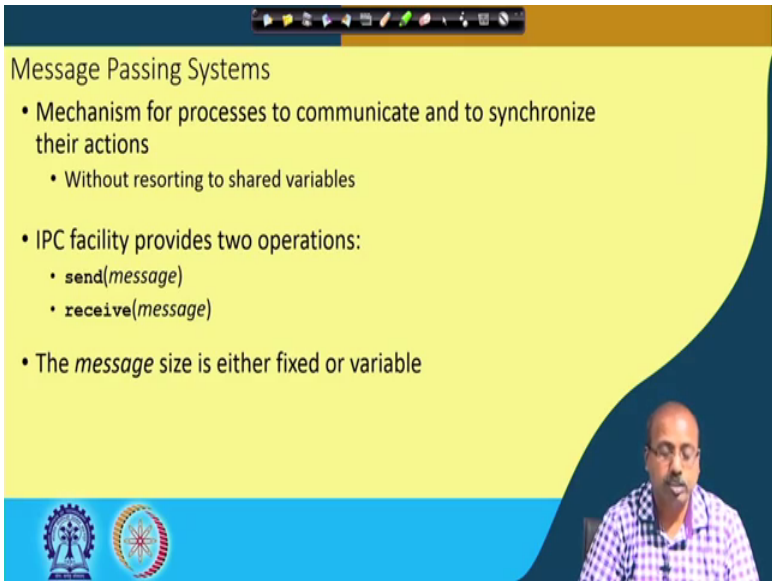
So, that routine is here next consumed. So, if in equal to out so, in and out they are same; that means, whatever the producer was whatever the producer was going to produce. So, it will write on to that slot at which the consumer is also pointing to. So, if this is the buffer. So, if in equal to out means; so, it is somewhere here ok. So, in is here. So, the producer will write to in this will try to write on to this location and the consumer is also here.

So, that means, consumer here means consumer has consumed up to this and next it will try to consume the next one; but there is nothing in the in the buffer because this in and out both of them are same. So, as a result the consumer should be made to wait. So, if while in and out are same then the consumer is made to wait otherwise the consumer is made to consume from the out location and this is the out pointer is incremented to out plus 1 percent buffer size then it will consume it and produce.

So, this way we can have synchronization between this producer and consumer and with respect to the buffer size if the buffer becomes full then the consumer producer is made to wait, if buffer becomes empty then the consumer is made to wait. So, this way this producer consumer this bounded buffer version can work. Of course, there are other issues like say this input output modifications when being done.

So, we have not ensure that producer process is not reading from the buffer when consumer is also trying to producer process is not writing on to the buffer when consumer is also reading from the buffer or vice versa. Because it may so, happen that they producer and consumer they are trying to access this buffer space simultaneously as a result there is some synchronization mismatch, but this type of problems will see later in the course. So, right now we just keep ourselves satisfied with the idea that we can synchronize between producer and consumer and try to do something to take care of their speed mismatch.

(Refer Slide Time: 10:57)



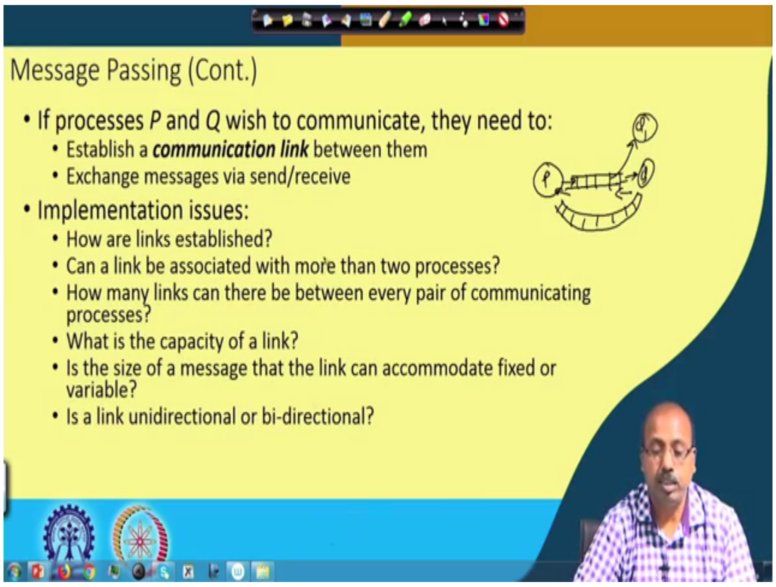
Message Passing Systems

- Mechanism for processes to communicate and to synchronize their actions
  - Without resorting to shared variables
- IPC facility provides two operations:
  - **send**(*message*)
  - **receive**(*message*)
- The *message* size is either fixed or variable

So, then another type of situation that we can have is the message passing case. So, message passing systems. So, they have called so, there for processes to communicate and to synchronize their actions without resorting to shared variable. So, there is no shared variable, so, the process is if they want to communicate they will send explicit messages to the from the sender to the receiver and this there are two system calls send and receive by means of this messages can be sent and received ok.

So, sender process will execute a will use send system call and the receiving process will execute receive system call message size can be fixed or variable. So, depending upon the operating system some operating systems will allow you to have variable sized messages, some operating systems will have only fix size messages. So, we can have different a message sizes. So, this is there so, this message passing systems are also there in different operating systems.

(Refer Slide Time: 11:55)



Message Passing (Cont.)

- If processes *P* and *Q* wish to communicate, they need to:
  - Establish a ***communication link*** between them
  - Exchange messages via send/receive
- Implementation issues:
  - How are links established?
  - Can a link be associated with more than two processes?
  - How many links can there be between every pair of communicating processes?
  - What is the capacity of a link?
  - Is the size of a message that the link can accommodate fixed or variable?
  - Is a link unidirectional or bi-directional?

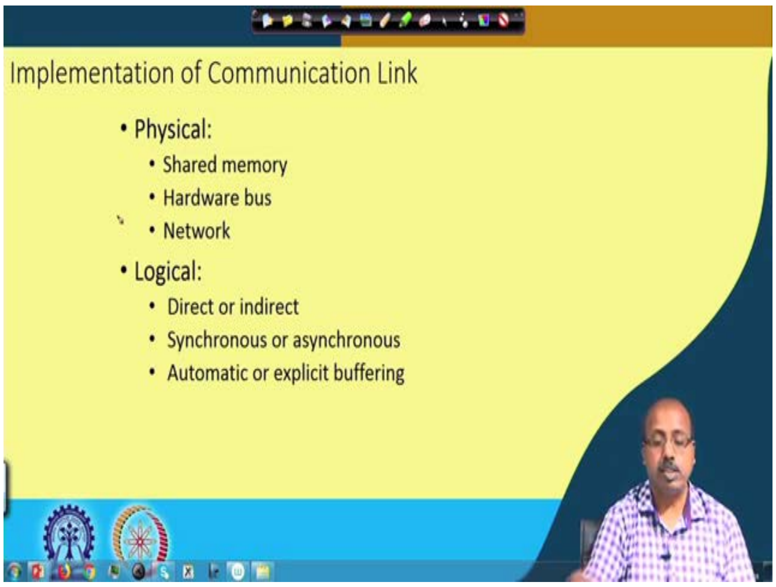
So, if processes P and Q wish to communicate, they need to establish a communication link between them. So, this is the first thing. So, I have got two processes P 1 and P 2. So, that P and Q so, between them I must establish a communication link between them so, that this process P can write on to this Q into this link and process Q can read it. Similarly process Q can write here and process P can read from here. So, that is exchange messages via send or receive so, process P can send messages, process Q can receive messages or vice versa.

Now the questions that we need to answer is how are the these links established that is the first thing, second thing is that can a link P associated with more than two processes can I have a message Q that covers more than two processes? Then how many links can there be between every pair of communicating processes? So, is there a single Q or there are multiple Qs. So, this is another Q like that ok. So, many things to be answered or maybe the same message Q it feeds not only Q, but also another process Q 1. So, can it be there? What is the capacity like how many messages can I send or how big messages can I send through this Q that is through this link.

So, that is also another thing is the size of a message the link can accommodate fixed or variable. So, message size. So, message format can it be of fixed length or the is it needed that there should be a fixed length or can I accommodate variable length also? Is the link unidirectional or bidirectional? Unidirectional means that whether it always goes from P to Q only or it can also come from Q to P. So, these are the different alternatives that we can think about a communication link between two processes or multiple number of processes and many operating systems they will try to answer this questions in different ways ok.

So, each of them have as you are making the system more and more complex and you are trying to accommodate more and more flexibilities into the message Q then the into this message passing mechanism. So, this implementation will become more and more complex. But, this depending upon the flexibility that we need so; we may like to do many things.

(Refer Slide Time: 14:25)



Implementation of Communication Link

- Physical:
  - Shared memory
  - Hardware bus
  - Network
- Logical:
  - Direct or indirect
  - Synchronous or asynchronous
  - Automatic or explicit buffering

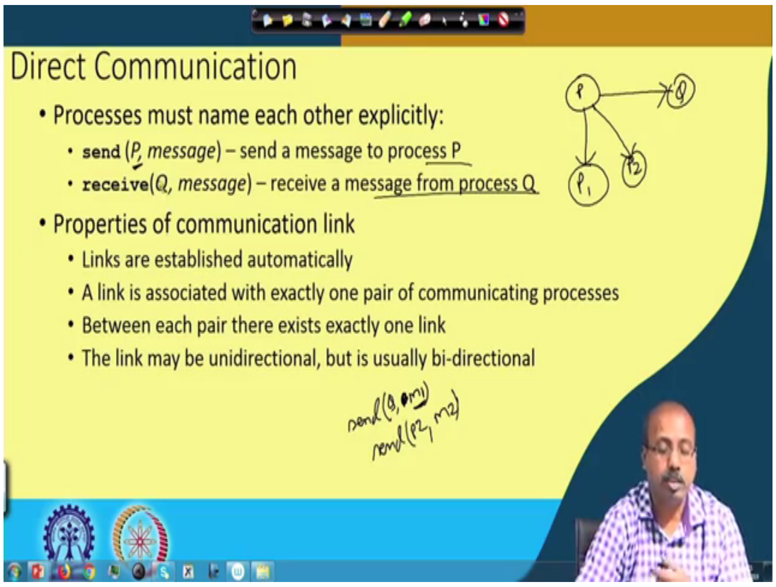
So, there are as far as the implementation is concerned. So, we can have physical communication. So, physical communication the communication link so, there has there should be a physical part and there should be a logical part; because physical means they are the communication maybe via shared memory. So, just like we have discussed about this shared memory creation like that SHM get system call and all that.

So, similarly we can have some dedicated part of shared memory which acts as message q between two processes. So, that is one possibility. So, we can have one physical communication like that; there can be hardware bus. So, maybe I have got multiple processors. So, they are connected to a bus now over the bus the data transfer takes place. So, that is one possibility.

So, over hardware bus we do this physical communication or over a network we do the communication. So, they are the processors or they are distributed over a network and over the network we sent messages receive messages like that. So, we can have this type of physical platform over which this communication link can be established, and logically there can be different classification like whether the link is direct link or indirect link whether it goes via some other nodes of things like that or it is a direct point to point sort of link, then synchronous or asynchronous.

So, is there any global clock that synchronizes this operation of this transmitter and receiver and so, that is another thing; automatic or explicit buffering. So, buffering that we need maybe if the message is long or multiple messages are being sent without waiting for this receiver to receive it so, are you doing it using some buffering or not? So, that way again different questions need to be answered; different mechanisms can be there.

(Refer Slide Time: 16:18)



Direct Communication

- Processes must name each other explicitly:
  - **send** (*P, message*) – send a message to process P
  - **receive**(*Q, message*) – receive a message from process Q
- Properties of communication link
  - Links are established automatically
  - A link is associated with exactly one pair of communicating processes
  - Between each pair there exists exactly one link
  - The link may be unidirectional, but is usually bi-directional

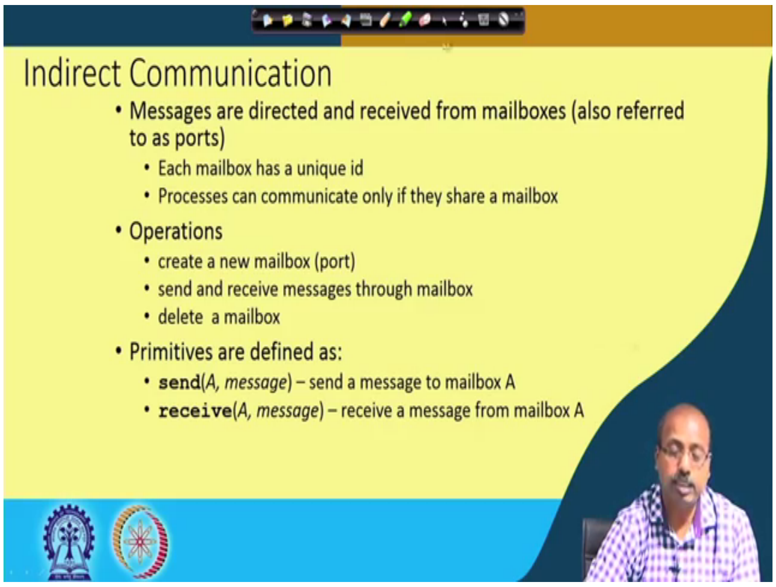
So, as far as direct communication is concerned, so, process is must name each other explicitly like there are send and receive this two system calls that are used for sending messages. So, this call like send P comma message. So, this essentially means that I want to send a message to the process P. So, send a message to process P. Similarly receive Q message. So, receive a message from process Q. So, that way we can have explicitly we can mention to which process we want to send the message send the message. So, it may so, happen that this is P and this is Q. So, P has got Q it wants to send some message to Q and also it wants to send some other message to process P 1 so, another message to process P 2.

So, it has got communication with many of them, but it wants to send this particular message to Q only. So, it can use like send Q comma message 1. Where message 1 is a data memory structure some variable structure the in the that is put into the form of message and then this message is sent to Q. So, at the and when it wants to send to P 2 so, it will write like send P 2 comma some message M 2. So, it may like to do that.

So, we have got the same system call similarly this receiving end. So, it can tell from which source it is expecting the message. So, that is from a will receive statement example that we have here. So, it is expecting that it will get a message from process Q. Now, this properties of communication links that we have. So, communication links are established automatically a link is also is associated with exactly one pair of communicating processes.

So, between every pair of communicating process we have got one link; between each pair there exists on exactly one link. So, and the link maybe unidirectional, but it is usually bidirectional. So, these are some of the properties of communication link that is generally followed. So, we have got different type of different parameter of different properties of them they are generally there whenever a message is being sent the link is automatically established. So, it is between one pair of communicating process. So, each pair exactly one link and the link is bidirectional in nature.

(Refer Slide Time: 19:02)



Indirect Communication

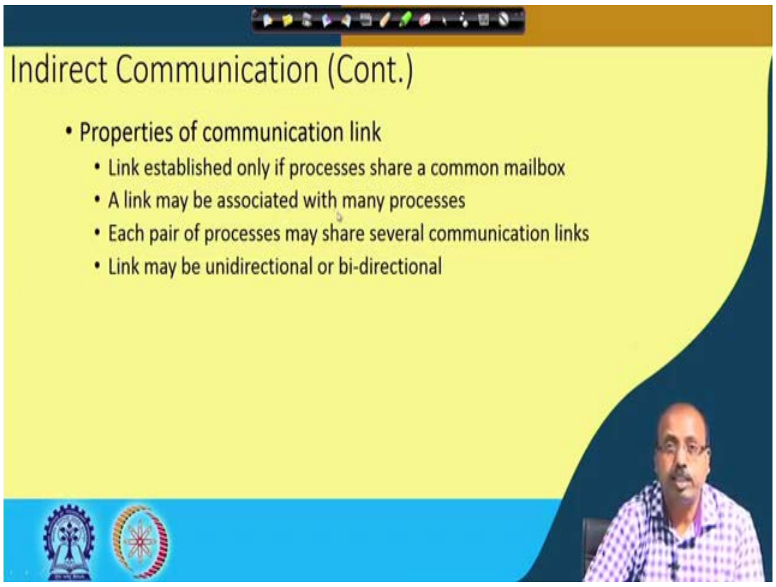
- Messages are directed and received from mailboxes (also referred to as ports)
  - Each mailbox has a unique id
  - Processes can communicate only if they share a mailbox
- Operations
  - create a new mailbox (port)
  - send and receive messages through mailbox
  - delete a mailbox
- Primitives are defined as:
  - **send**(*A, message*) – send a message to mailbox A
  - **receive**(*A, message*) – receive a message from mailbox A

In case of indirect communication; so, messages are directed and received from mailboxes also referred to as ports. So, each mailbox will have a unique id and processes can communicate only if they share a mailbox. So, every process has got an associated mailbox. So, anybody wants to send a message to a process. So, it will send a message to the corresponding mailbox and the receiving process so, it will just read it from the mailbox instead of telling from which receiver it is expecting a message.

So, it will just read it from the mailbox and find out from which receiver which transmitter the message has arrived. And of course, there should be there should be permission. So, for this mailbox sharing and all so, that is there, but once the shape permission is established. So, it is done like that. So, operations that we have so, create a new mailbox or port send or receive messages through mailbox and delete a mailbox. So, these are some of the operations that is that is there. So, for example, primitives we have got send A comma message.

So, it is sending a message to mailbox, receive A comma message receive a message from mailbox A. So, you can have this type of things. So, so they are all associated with some mailbox. So, if two processes they share some mailbox then mails send to one mailbox to buy one process will be visible to the other one also. So, this is indirect way of communication because we are not telling directly to which process we are wishing to send the message, but we are mentioning a mailbox only. Similarly at the receiving end we are not telling from which process will be receiving the we want to receive the next message, but we are just telling that which is the next message that we have in the mailbox and accordingly take some action for doing that handling it.

(Refer Slide Time: 20:58)

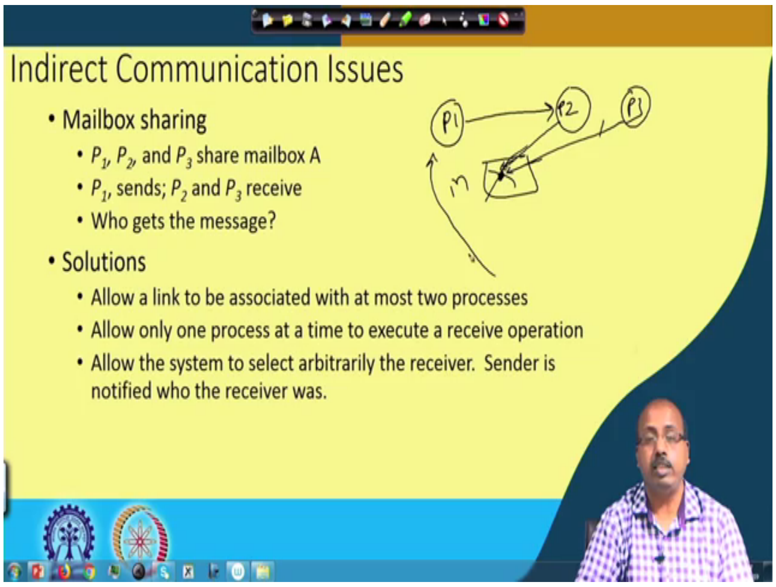


Indirect Communication (Cont.)

- Properties of communication link
  - Link established only if processes share a common mailbox
  - A link may be associated with many processes
  - Each pair of processes may share several communication links
  - Link may be unidirectional or bi-directional

Then properties of this communication link; so, link established only if processes share a common mail box, a link may be associated with many processes each pair of processes may share several communication links and link may be unidirectional or bidirectional. So, these are the properties of communication link in case of indirect communication.

(Refer Slide Time: 21:20)



Indirect Communication Issues

- Mailbox sharing
  - P_1, P_2, and P_3 share mailbox A
  - P_1, sends; P_2 and P_3 receive
  - Who gets the message?
- Solutions
  - Allow a link to be associated with at most two processes
  - Allow only one process at a time to execute a receive operation
  - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

Some other issues like mailbox sharing. So, P 1 P 2 suppose this is a situation P 1 P 2 and P 3 they share mailbox A, P 1 sends and P 2 and P 3 receives now who gets the message like for whom P 1 did send it? And since they are sharing the mailbox; so, we do not assume that they are there is a linking of information from P from for them we assume that anything that is send should be if it even if it is scene by some other process.

So, between this P 1 P 2 and P 3; so, it is not hampering the security issue, but the point is like a who should read this that who should read the next message. So, what can be the solution? Solution is to allow a link to be associated with at most two processes. So, if I have got mail box single mailbox A in this case was associated with three processes P 1 P 2 P 3 instead of that we can create two mailboxes one between P 1 and P 2 and one between P 1 and P 3.

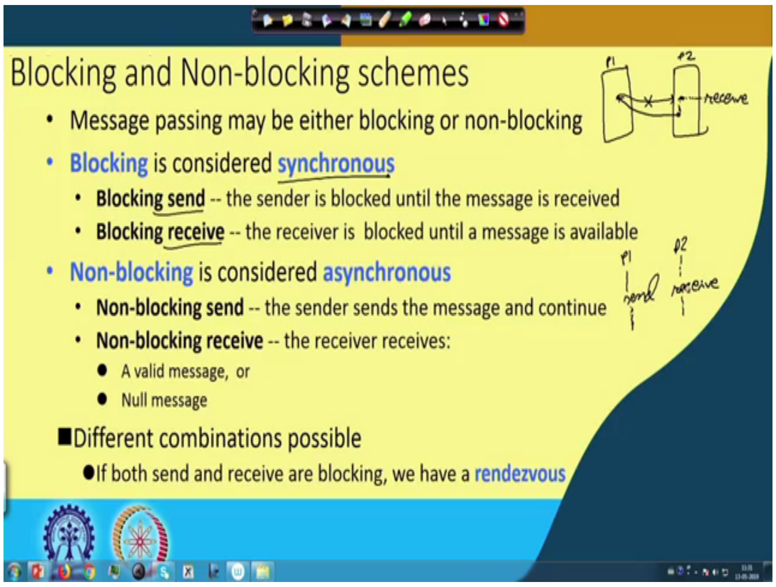
So, that way we do not have this type of problem ok. So, we have we do not have this which process gets the message so, that that problem will not occur. Or we can allow only one process at a time to execute a receive operation. So, somehow we ensure that two processes will not execute this receive simultaneously. So, how do how do we do this. So, that is another issues operating system should ensure that only one process will be executing the receive operation. So, otherwise what will happen is that maybe we have sent this P 1 it has got it has sent some message.

So, P 1 has send some message and. So, so this I have got P 2 and P 3. So, this P 2 P 3 this processes are there now P 1 has send a message to the mailbox. So, in the mailbox some message has come now both P 2 and P 3 once to tries to access it and as a result. So, there may be some inconsistence situation that can occur because maybe P 2 has read it half and then P 3 is time to read or maybe P 2 is reading has read it and it has deleted the message from here, but P 3 was also reading it from here. So, that way there may be several inconsistencies that can come.

So, we have to somehow ensure that only one process will execute the receive operation at a time. So, so operating system must ensure that receive is done by only a single process or we can allow the system to select arbitrarily the receiver. So, and then or the sender is notified who the receiver was. It nay so, happen that the system decides like who gets the message. So, if P 2 P 3 both of them issue a receive call, then maybe it will take up one of them maybe P 2 and then the operating system will involve in will inform

P 1 that you are message was picked up by process P 2. So, this is one possibility that the arbitrarily the receiver is selected and the sender will is notified about the receiver.

(Refer Slide Time: 24:39)



Blocking and Non-blocking schemes

- Message passing may be either blocking or non-blocking
- Blocking is considered synchronous
  - Blocking send -- the sender is blocked until the message is received
  - Blocking receive -- the receiver is blocked until a message is available
- Non-blocking is considered asynchronous
  - Non-blocking send -- the sender sends the message and continue
  - Non-blocking receive -- the receiver receives:
    - A valid message, or
    - Null message
- Different combinations possible
  - If both send and receive are blocking, we have a rendezvous

So, there is another important concept with respect to this messages blocking and non blocking scheme. So, blocking scheme is considered to be synchronous. So, basically what can happen is that, suppose we have got this process P 1. So, which is executing a piece of code; so, this is for P 1 and P 1 sends a message to another process P 2. Now since it has sent the message.

So, the message contains some data or information with which P 2 will proceed and when P 2 has finished using the data it should tell P 1 that I have done with my operation. So, this is one type of situation that P 1 waits until and unless the message has been the message is received by the receiver. So, this is called blocking type of send. So, blocking send the sender is blocked until the message is received by the receiver and this is a blocking receive. So, this blocking receive is a receiver is blocked until a message is available. So, maybe P 2 was executive.

So, at this point it was expecting a message, but this P 1 did not send anything so, far P 1 has not yet send it may be P 1 is slow or it was not schedule etcetera. So, at this time at this point when P 2 executes a receive call. So, P 2 is made to wait. So, P 2 is made to wait that since no message is available. So, it is made to wait. So, we have got this blocking send and blocking receives. So, blocking send is there something like this message is sent, but there is no receiver message is not received. So, that is a blocking send and blocking receive is that receiver is blocked until some message is available for receiving. So, this is one type of mechanism.

So, this is called synchronous because we have got synchronism between the sender and receiver of messages. There is another type of situation which is known as non blocking or asynchronous situation. The non-blocking sent the sender sends the message and continue and so, so what happens is that, here in this case P 1 it at the sum point it sends the message, but without waiting to see whether the message is received by the receiver or not so, it will continue doing the next operation. So, eventually the receiver will receive it and do the corresponding operation.

So, the sender sends the message and continue and other case is a non blocking receive and the receiver receives a valid message or a null message like if the P 2 is the receiver and at some point it executes a receive call. So, it executes a it executes a receive call, now it may so, happen that at this point some messages available. So, it gets the message or if P 1 is slow then it will not get any message. So, it will get a null message. So, whether the valid message or null message with that the receiver will continue, but receiver will not wait for the message to be available.

So, it executes a receive, it is good if it gets a valid message otherwise it gets a null message, but whatever be the situation it continues with the further statements in the program. So, we can have different combinations of possibly both sender and receiver blocking. So, we have got this rendezvous type of situation. So, that way we can; so, different operating systems so, again they will they will implement different combinations of this say message passing blocking or non-blocking. And, you have to consult the corresponding operating system manual to see what type of message passing is supported by that.