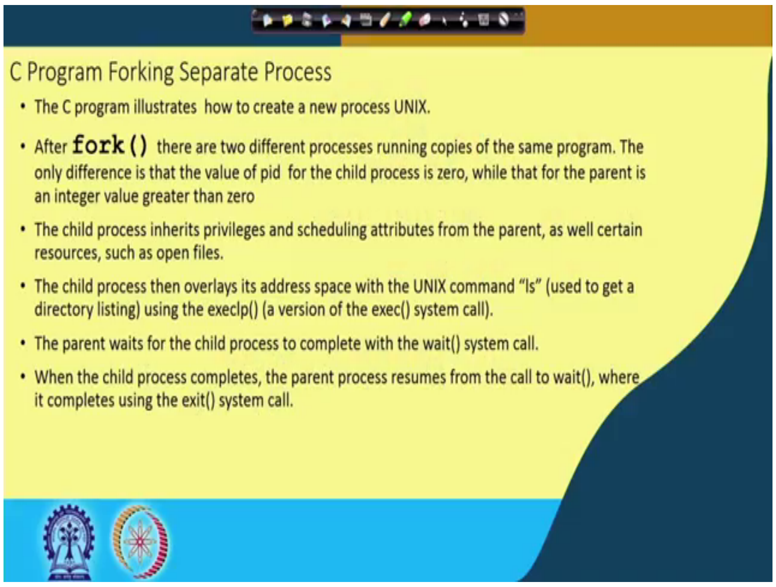
Operating System Fundamentals

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

Lecture – 16
Processes (Contd.)

So, to summarize what we have discussed in the last class like this forking and all.

(Refer Slide Time: 00:27)



C Program Forking Separate Process

- The C program illustrates how to create a new process UNIX.
- After `fork()` there are two different processes running copies of the same program. The only difference is that the value of pid for the child process is zero, while that for the parent is an integer value greater than zero
- The child process inherits privileges and scheduling attributes from the parent, as well certain resources, such as open files.
- The child process then overlays its address space with the UNIX command "ls" (used to get a directory listing) using the execlp() (a version of the exec() system call).
- The parent waits for the child process to complete with the wait() system call.
- When the child process completes, the parent process resumes from the call to wait(), where it completes using the exit() system call.

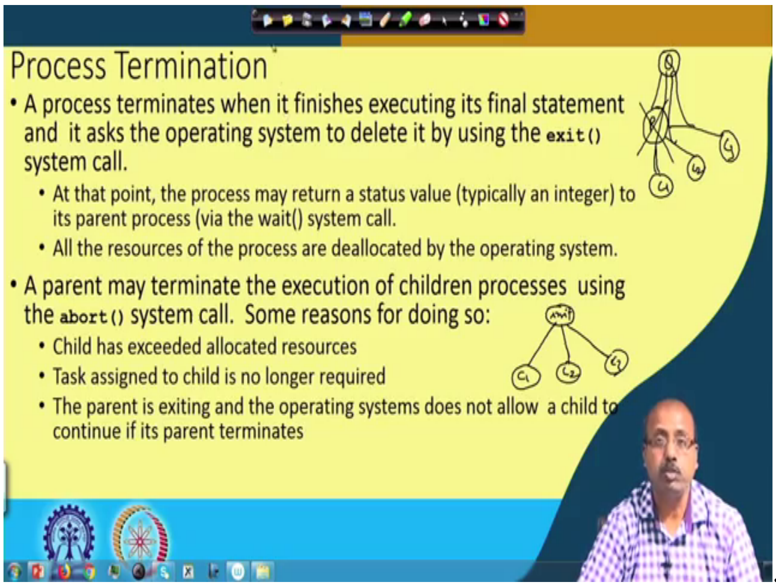
So, we have seen a C program that does forking to create new processes in UNIX operating system. After fork there are two different processes running copies of the same program. The only difference is that the value of pid for the child process is zero, while that for the parent is an integer value greater than zero and which happens to be equal to the pid of the child process.

Child process inherits privileges and scheduling attributes from the parent. So, whatever the privilege in terms of 5 lakh says etcetera, it has and whatever the scheduling attributes like this CPU usage disk usage IO waiting and all that. So, whatever scheduling parameters are used for the parent process, so, they are also available to the child process.

Then after that, child process in the particular example that we have looked into so, it over lays its address space with the UNIX command "ls" that is used to get directory listing using the exec lp. So, it is a variant of this exec system call so, for overwriting the address space of itself.

Now the parent process it was made to wait for the child process to complete by the wait system call and when the child process completes. The parent process resumes from the call to wait and where it completes using the exit system call. So, basically the child process terminates and when the child process terminates so, with the parent processes in informed. So, when the wait system; call it executing a wait call so, it is informed and the parent process will continue from that point.

(Refer Slide Time: 02:09)



Process Termination

- A process terminates when it finishes executing its final statement and it asks the operating system to delete it by using the `exit()` system call.
  - At that point, the process may return a status value (typically an integer) to its parent process (via the wait() system call.
  - All the resources of the process are deallocated by the operating system.
- A parent may terminate the execution of children processes using the `abort()` system call. Some reasons for doing so:
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - The parent is exiting and the operating systems does not allow a child to continue if its parent terminates

Now, as for as termination is concerned, a process terminates when it finishes executing its final statement. So, that is the obvious one. So, last statement of the parent process of a process is executed. So, it is over, or it ask the operating system to delete it by using exist system call. So often so this is not visible to the programmer. So many a time the compiler will introduce some exit system call at the end of the code that is generated for a process. So that way the process terminates and this control is given the back to the operating system.

The process may return a status value typically an integer to its parent process so, that why are the wait system call. So, basically sometimes we need to know that a child process that has been created. So, what was the outcome of its execution? So, there may be different types of termination of a child process and this parent process may be interested to know in which way the child process terminated. So, as a result there can be different outcomes of the child process and they can be; this outcome can be informed to the parent process by means of this return value.

So, when it exits normally with the exit system call we can assign some return value. So, that return value is passed to the parent process by the operating system; so, parent process will also know how the child process terminated. And all the resources of the process are deallocated by the operating system. So, all the resources that we had so, that is taken back by the operating system.

So, this is one possible way, other possible way if parent may terminate execution of children processes execute in the about system call. So, sometimes so, parent may execute an about system call and on the children process they will get terminated. So, why is it necessary? So, this may be necessary that child has exceeded allocated resources. So, as we know particularly in UNIX operating system; so, there is a parent child relationship. So, a user may spawn a process that is the taking lots of resources ok. So, that maybe some either a misbehaving process or the process may be a highly demanding one. So, that way; so, that cannot be run at some specific time of the day. So, that may be run only when the system load is low maybe at night or something like that.

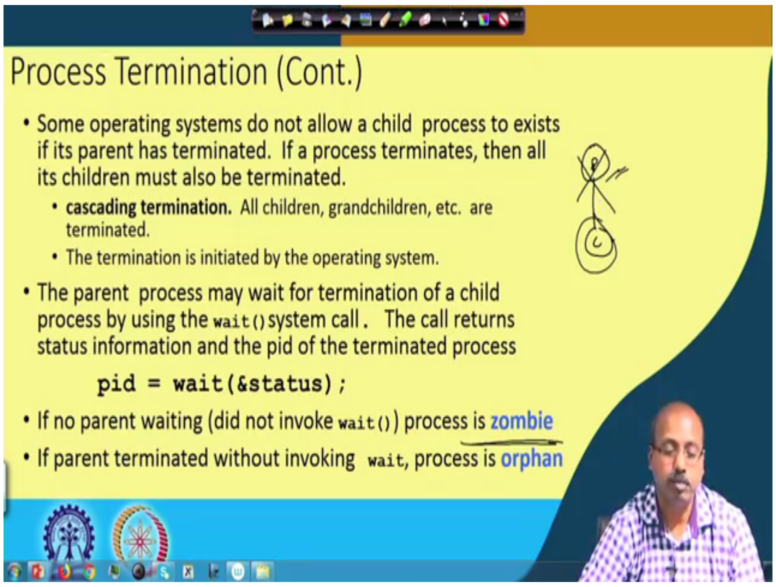
So, if such a process has to be about it, such a process has to be stopped then so, that is the situation they have allocated; they have exceeded the allocated resource and then we want to about it. Then task assigned to child is no longer required. Many a time like we start a task, but that task is no more required. So, why do you schedule that task? This may particularly important like say suppose on occurrence of some interrupts, say maybe in a real time system. So, a fire sensor has sensed some smoke and then accordingly something has to be done, but you see that after say two or three minutes also the appropriate action is not taken. So, there is no point continuing to invoke those tasks so, that way we may want to about that particular task.

Other option is the parent is exiting and the operating system does not allow with child to continue if its parent terminates. Like we have got this parent child relationship between the processes; so, this parent child relationship, so, this parent is there and we have got the child process it has created a number of child processes C 1, C 2, C 3 like that. Now what happens if the parent itself dies parent itself finishes? When what happens to this children process is C 1, C 2 and C 3. So, now, there can be different types of thoughts, one thought is that there in the parent is not there. So, there is no point keeping the children processes so, they will also terminate; so, this is one possibility.

Other possibility is that this parent P that we had; so, it was and itself was the child of some other process Q. So, if this process dies then make all the C 1, C 2, C 3 as children of this Q process; so, that is another possibility. And the third option that we have is there is a init process in the system. So, whenever a process terminates and whenever so, this situation occurred that the process P terminate and we have got all these C 1, C 2, C 3 as the orphan processes. So, this orphan processes so, they are all made children of this init process ok.

So, since this init is always there in the system so, we do not have any confusion like the init will not stop until and unless the system is crashed. So, that way we have got these children processes they can become part of init and continue like that. So, these are the different types of options that we can have. So, different operating systems so, they will follow different type of conventions.

(Refer Slide Time: 07:13)



Process Termination (Cont.)

- Some operating systems do not allow a child process to exists if its parent has terminated. If a process terminates, then all its children must also be terminated.
  - **cascading termination.** All children, grandchildren, etc. are terminated.
  - The termination is initiated by the operating system.
- The parent process may wait for termination of a child process by using the `wait()` system call. The call returns status information and the pid of the terminated process

```
pid = wait(&status);
```

- If no parent waiting (did not invoke `wait()`) process is zombie
- If parent terminated without invoking `wait`, process is orphan

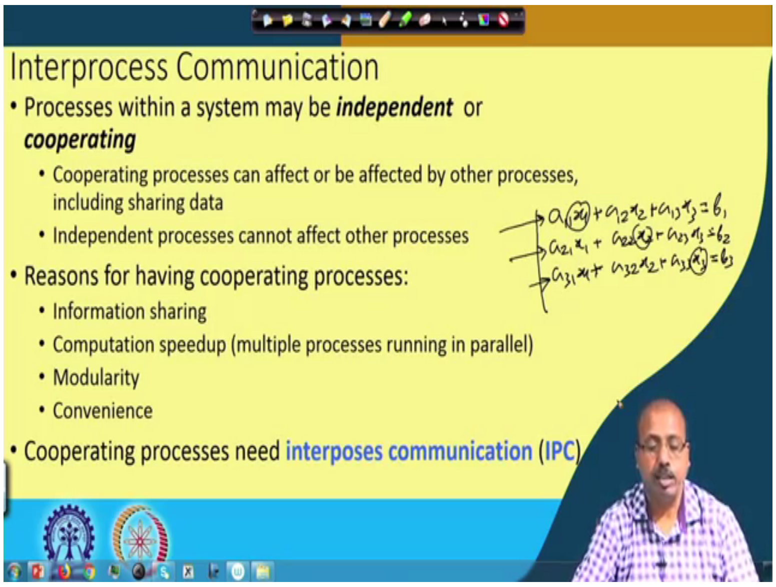
So, some operating systems do not allow a child process to exist if its parent has terminated; if a parent terminates all its children must also terminated. So, this is called cascaded termination all children, grandchildren etcetera all are terminated. So, entire a branch starting at that particular process which is being terminated so, that is they are also terminated. So, this termination is initiated by the operating system. So, this is one by one all the process is terminated. Then a parent process may wait for termination of a child process by using the wait system call that we have seen previously. So, this wait system call if a process executes; so, it is waiting for the children process to be over the call returns.

Status information and the pid of the terminated process. So, this pid equal to wait status; so, this will be whichever process it was waiting. So, when this fellow finishes when pid will have the pid of the terminated process and the status information will have the return value which is returned by the child process. If the parent; if no parent is waiting then the process is zombie state so, that is we do not the parent was not waiting. So, it may so happen that there are the parent process has created a child process ok; so, parent process has created a child process and it is not waiting. So, the parent code not have a wait call.

So, happen that the operating system scheduler or will schedule the parent process first and the parent process terminates before the child process comes. So, before the child process terminates; so, this process is not there.

So, when this fellow terminates it finds that I do not have any parent ok. So, in that case we say that this process is in zombie state, this plus child C. Then so, that is there and if the parent terminate it without invoking wait then the process is orphan. So, when this process is a parent terminates, but at the time child is not over. So, child is there; so, child is a child is said to be orphan. So, that we have discussed previously.

(Refer Slide Time: 09:33)



Interprocess Communication

- Processes within a system may be ***independent*** or ***cooperating***
  - Cooperating processes can affect or be affected by other processes, including sharing data
  - Independent processes cannot affect other processes
- Reasons for having cooperating processes:
  - Information sharing
  - Computation speedup (multiple processes running in parallel)
  - Modularity
  - Convenience
- Cooperating processes need interposes communication (IPC)

$$a_{11}\boxed{x_1}+a_{12}x_2+a_{13}x_3=b_1$$
$$a_{21}x_1+a_{22}\boxed{x_2}+a_{23}x_3=b_2$$
$$a_{31}x_1+a_{32}x_2+a_{33}\boxed{x_3}=b_3$$

Next we will be; now how this orphan zombie so, this situations will be handle. So, that is operating system designer specific and different operating system so, they will follow different type of policies for them. So, we will be getting more ideas so, if you are looking into one particular operating system and how does it handle it and all. So, one possible case we have discussed it with respect to UNIX, where this if the children process is the parent is the, parent dies and then all the children. So, they made the children of the parent process or the init process. So, that is one possible option.

Next we will be again looking into another very important issue in case of processes which is known as inter process communication. So, inter process communication means in a system so, if there are number of processes and particularly there is parent child relationship between the processes then they are part of a big system. So, as a result they need to talk to each other, they need to communicate between each others. So, they are has to be some communication between the processes.

So, this cooperating processes can affect or the affected by other processes including sharing of data. So, this is the situation that is quite common say if I have got a number of processes which are part of a big system. So, they will be communicating between themselves they will share data and all. Independent processes cannot affect other processes. So, since they are if they processes are independent, then naturally they may be doing totally different types of jobs, maybe from two different users they are coming.

So, as a result so, they are; so, they do not they offer most of the times is they do not need to communicate between themselves, they do not need to contact between themselves like that.

Now why do we have this cooperating processes? One is the information sharing like a number of processes they may like to share information like maybe they are so, communication coming from different places and all those information need to be gathered and put on to some centralized server. So, as a result there is some information sharing. Competition speed up multiple processes running in parallel so, maybe it is like that we just; we have got say ten sensors sensing the temperature of the room and updating some database.

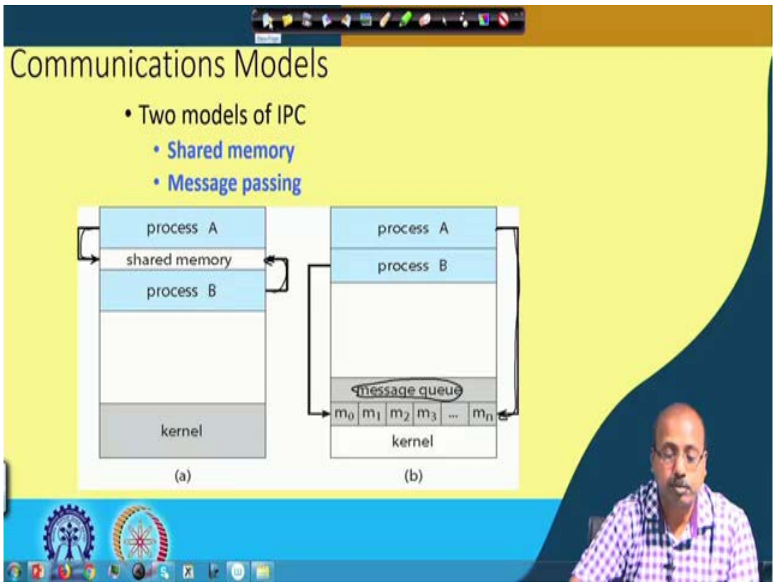
So, maybe all this ten sensors these are; there can be ten different programs which are doing the operation. And as a result this overall update operation will be done faster. So, this is the computational speed up may be there or if you can have some complex computation so, that computation may be divided into several parts. Like for example, if I have got say a system of simultaneous equations like say $a11x1 + a121x2 + a13x3 =b1$. So, this is one equation then $a21x1 + a22x2 + a23x3 = b2$.

Similarly, $a31x1 + a32x2 + a33x3 = b3$. Now if this set of equations so, if we want to solve this simultaneous equations so, there are some methods, iterative methods for doing that, in which each equation we use for one particular variable update. So, if we reserve the first equation for updating x 1, second equation for updating x 2 and third equation for updating x 3. Then these three updates may be given to three different processes that can execute concurrently. And that way we can have speed up but the updated values of 1 x 2 x 3 needs to be communicated to the other processes as well. Like, the first process is updating x 1 and that updated value must be communicated to process 2 and process 3; similarly process 2 updating x 2 must communicate to process 1 and process 3 the value of x 2 so, like that.

So, we have got computational speed up we have got information sharing, we have got modularity also like maybe the portions which are relatively independent accepting this information sharing. So, they may be divided into different modules and then these individual modules may be developed by different groups. So, we have got the modularity. And of course, the convenience, the way that we think about solving a problem so, if we can think about some its some independent processes; independent cooperating processes, then definitely that will help us in developing the overall program.

So, this type of situation this cooperating processes need inter process communication. So, this is given a broad heading of IPC Inter Process Communication and operating system must provide some mechanism by which this inter process communication can take place efficiently. So, we will be looking into some such techniques followed in general operating systems this inter process communication.

(Refer Slide Time: 14:59)



Communications Models

- Two models of IPC
  - Shared memory
  - Message passing

So, there are two different models of this inter process communication; one is based on the shared memory, other is based on message passing. So, shared memory is like a board ok; so, for example, I can say like if I write something on a board, then some other process can see that and it can understand it is like that. Now, one possibility is that if I write on a board then that is visible to all other processes or say everybody in the class can see if we write something on to the board.

Or if we can have some permission ok so, it may so, happen that process A has written something on to the shared memory, but to read the content of the shared memory some permission is necessary. So, if that permission is granted to process B, then process B can only see that data or similarly if process B write something on to the shared memory, so, we can have process A can just read it from there.

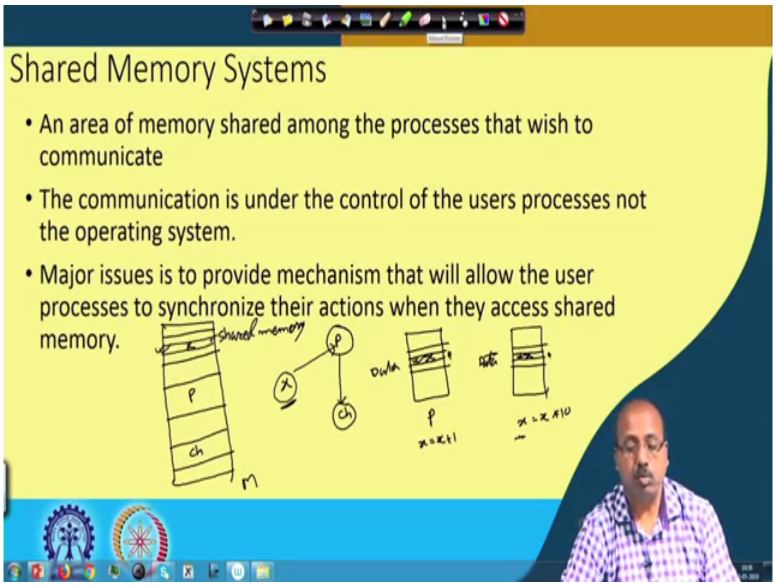
So, of course, what type of access we have; so, that is a question. Like in this diagram so, as you can see here so, process A can write something on to the shared memory, similarly process B can also write something on to the shared memory. How to they read it so that is not depicted here. So, it is assume that whenever it is written onto the shared memory. So, both the processes are ready to see it.

So, this is one type of sharing, another type of sharing that we can have is by means of message passing. So, this process A it sends message to process; send message to the message queue and that is meant for process B, and then process B can get the message from this message queue. So, this kernel; so, we can have a message queue that has got all the messages coming from different processes and process B can read that message queue one by one.

So, if it is meant for that particular process so, every process will have got a corresponding message queue and then any process that wants to send a message to the another process. So, it must write on to the queue of that particular message or it can of that process or it can say it should send the message to the corresponding queue only. And then the receiving process when it executes some received type of system call so, it will be reading from the message queue and then it can take a decision that what type of who is the sender of this message and accordingly take some action.

So, we have got this two message; two type of communication mechanism inter process communication mechanism, one is based on shared memory another is based on message passing.

(Refer Slide Time: 17:51)



Shared Memory Systems

- An area of memory shared among the processes that wish to communicate
- The communication is under the control of the users processes not the operating system.
- Major issues is to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory.

Now, shared memory systems so, an area of memory shared among the processes that wish to communicate. Now this shared memory type of communication. So, as we can understand that even if I have got a parent process and that creates a child process and this parent process has got code, data and stack. Suppose this parent process has got some variable x init so, accordingly in the data segment of this parent process space is allocated for x. So, this is given for x.

Now when the child process is created so, this is for the parent this child process is created. So, child process gets a copy of the data segment. Even if it has not executed the exec call so, it will get a copy of the data segment and it will get a copy of the variable x here. But you should understand that this x and this x they are totally different. So, if the parent executes a statement x equal to x plus 1, so that will update this x and if the child executes and a statement like x equal to x into 10 so, that will update this x ok; so, they are not same.

So, basically whenever make creative fork, the data segment that are created at different though one of the; initially they are same because they are copy of the parent only, but after that whatever modification is done to the data segment. So, they are for individual processes. So, they are not common between the processes. So, if you really want something is to be shared, that is some variable has to be accessible to both the processes then I need some special area of memory which will be used as the shared memory. So, if this is my complete memory that I have maybe in this part I have kept the code data and stack segment for this parent process p and here I have created everything for the child process ch.

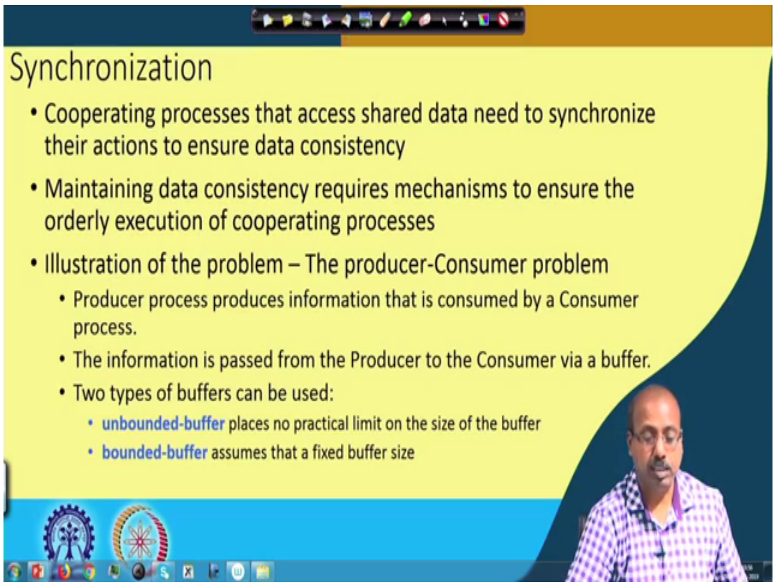
Now, if I say; if I want that this value of x will be visible to both this parent and child, then I should create this x in a region of memory which is called the shared memory. I should create it into shared memory the variable x here. And somehow ensure that both p and c h can see this shared memory location x and can manipulator modified.

So, this is the job of the shared memory. So, how this shared memory is implemented that is operating system specific, but this has to be there. So, an area of memory shared among the processes that wish to communicate that should be there. The communication is under the control of the user's processes not the operating system. So, this is a facility that is provided by the operating system, but which processes will coordinate and how they will coordinate.

In the sense that it may so, happened that this parent process so, it will just; it will write on to this x and the child process it will only be able to read from this location x, but child process will not be allowed to write on to x. So, this way there can be different types of read write make a permissions that can be there.

So, this control of this communication should be in the hand of the user processes and not by the operating system. And major issue is to provide mechanism that will allow the user processes to synchronies their actions when they access shared memory. So, major issue is that may be when this parent process is updating this x, this child process is trying to also read the content of this, but that way it will get some inconsistent data and that will create some difficulty. So, somehow the we have to have a mechanism by which will be taking care of that.

(Refer Slide Time: 22:01)

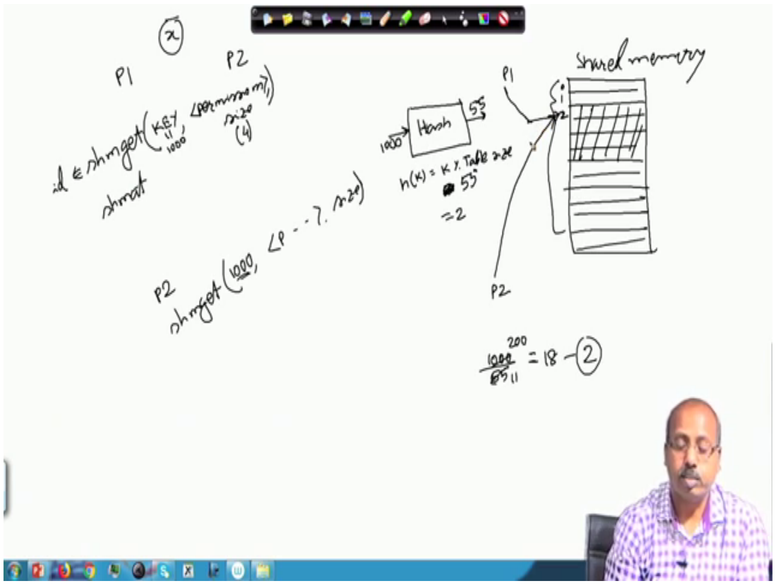


Synchronization

- 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, before coming to this, so, let me clarify something with respect to UNIX operating system how this is the shared memory is done. So, in case of UNIX operating system what happens is that.

(Refer Slide Time: 22:15)



As I said that there is a shared memory location, suppose this is the shared memory space of the full memory. So, I will try to explain like how two processes, they can make sure that they are accessing the same shared memory location. So, we have got this shared memory divided into say suppose I have got a number of locations in the shared memory and each location is accessed by some key value ok. So, suppose I have got process P 1 and process P 2 and they want to share the variable x between them. So, what this P 1 will do? So, P 1 will be executing a statement which is called shared memory get.

So, it will be executing a system called shared memory get and here it will mention some key value, this key value is nothing, but some integer value and then it will also mention some permissions, some read write execute permissions that should be there for the shared memory key for the shared memory. And of course, the size of the shared memory so, how many bytes that you want to have.

So, suppose I mention this key value equal to 100, what the operating system will do is that it will use a hash function; it will use a hash function where input is this 100 and accordingly it will generate an index in this range ok. So, it will just suppose if generates and index of say 55. So, as you know the hash function is nothing, but say one possible hash function h of key maybe key percent table size.

So, key percent table size so, in this case suppose I give a value of 1000 and my table size is say 50, then it will be a fully divide it. So, say table size is same a 55; so, it is 1000 divided by 55 so, this will be 11 this is 200. So, this is 1, 18 and reminder is 2. So, it will come to these two reminder will come so, this value will be equal to 2.

So, if you give it 1000 so, this is location 0 1 2. So, it will be pointing to this one. So, this is the key value giving a value of 1000 will ultimately mean that I am talking about the shared memory location 2. And this permission I will give the read write execute permissions like how the other processes will see this particular location and the size like how many bytes that you want. So, maybe we want say 4byte; so, this 4 bytes will be created; so, these are the 4 bytes that are there.

Now for P 2 also it will be using a shm get with the same key values. So, it will also mention key value as 1000 and permissions and this size. So, other parameters will be there as I said. So, since both the processes they are using the same key values. So, for both of them they will point to the same location.

So, P 1 will also point to this and P 2 will also point to this and depending upon the permission that we have set. So, they are may be read write execute permissions. So, as a result they may be allow to modify read or write etcetera. So, this is how this in case of

UNIX operating system, we make the processes to share some memory location which are not part of their data segment. So, this x variable which is not part of the data segment; so, this actually returns some id and then there are a pair of other calls like shmat and all. So, by this you can attach a particular variable to this particular id so, that can be done. So, I am not going in to that so, for more detail you can consult UNIX operating system inter process communication primitives.

But what is required of course, is the synchronization between this between this process is like one process is say modifying the value of x and another process is trying to read it or write the data. So, as a result it can lead to data inconsistency. So, you have to somehow ensure that the data is consistent. So, maintaining data consistency it requires mechanism to ensure and orderly execution of the cooperating processes.

So, if it is half done, half update is done then naturally the content of the data will be inconsistent. In this direction so, will be looking into one particular problem which is known as producer consumer problem, where we have got some producer process that produces some information that is consumed by a consumer processor. Typical example may be an user is pressing keys of the keyboard, now that key has when the keys are being pressed so, that is there is a keyboard process which reads the values, and there is a another process another program to which the values are sent.

So, that way we have got this read this keyboard process and this other program that is executing so, there is a communication between them; so, the producer consumer process. A keyboard process is the producer process and the reader process is the consumer process. So, we have to have some sort of buffer for communicating between them and then the buffer maybe unbounded buffer or bounded buffer in nature. So, we will continue with this in the next class.