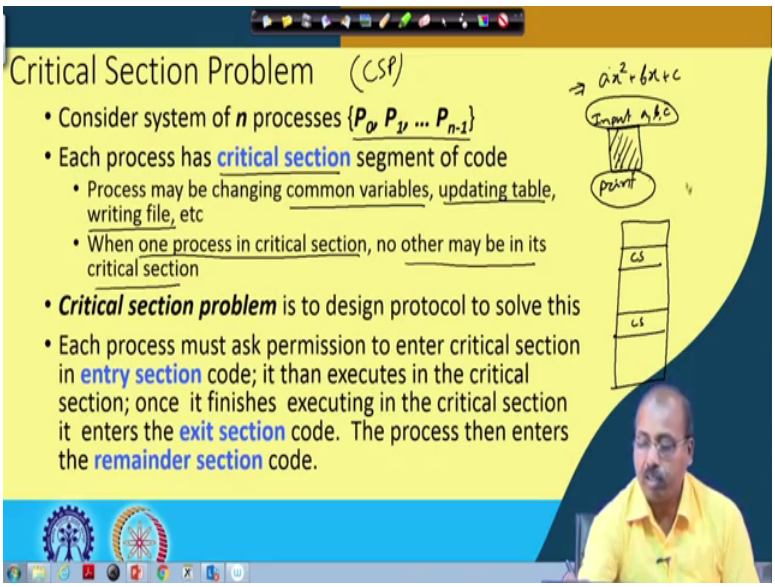
**Operating System Fundamentals**
**Prof. Santanu Chattopadhyay**
**Department of Electronics and Electrical Communication Engineering**
**Indian Institute of Technology, Kharagpur**

**Lecture- 30**
**Process Synchronization (Contd.)**

(Refer Slide Time: 00:29)




So, critical section problem which is a often in short known as a CSP. So, which are often known as CSP. So, this CSP, so, they are actually very common in a system and if I have got say n processes, then this n processes say P 0 P 1 up to P n minus 1 and each process has critical section code. So, what is a critical section? So, in the critical section; so, process may be changing common variables updating table writing file etcetera.

So, whenever a process accesses some shared resource is it becomes a critical section. So, typically well; so, if I have got if I have written a piece of program, then apparently it seems even suppose the is the good old example of solving a quadratic equation and getting the roots of it. So, this also has got critical section because it is a at some point of time it is trying to input the values of a b and c and those abc values they may be coming from a different files or different input devices and that input device or that particular file may be used by some other process also.

And now for example, I have given some input from the keyboard, now there are three processes who are waiting for getting the input from the keyboard. Now if this if these

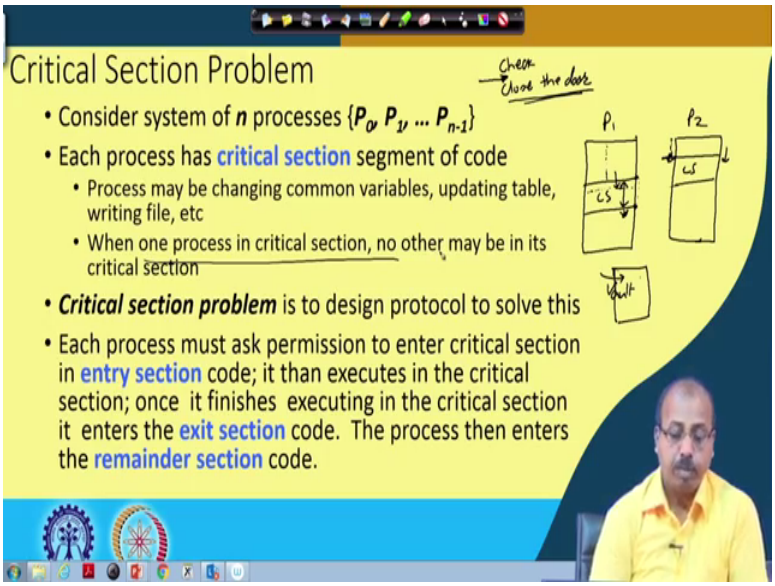
three processes they are actually racing between each other to get the input from the keyboard. Now which process should get the key the value of the key? So, that is an issue to be solved. So, that is basically a critical section of code where a process is trying to read the input.

Similarly, after some time this process will be doing some print operation or some output operation and then it is trying to access the screen or the printer or some file. So, there also we have got risk condition because a number of processes they may try to modify the output device or try to modify the file. So, the critical section though apparently it seems that this is only true for some very complex system programs, but that is not the case even the simplest program will have critical section.

So, critical section means a portion of the code where it is accessing the shared resource. So, in this particular program so, after it has read the values of a b and c. So, it is going into calculating the root. So, if in this part of the code if we assume that it is not going to do any file access or input output device access ok. So, then I can say that in this part of the code is not critical, but this is this part is critical and this part is critical. So, in general I can say that if this is a piece of total execution of a process, then we can have some portions of it which are critical.

So, this may be a critical section. So, this may be a critical section and it is not very difficult to identify the critical sections because you can always find out like the portion of the code where it is trying to access some shared resources and whenever it is trying to do that. So, that is a critical section. So, whenever a process is changing some common variable updating some table which is shared across a number of processes or it is writing onto a file so, this may be a critical section. I the requirement is that when one process is in critical section no other process should be in its critical section. So, contention comes when I have got say two different programs. So, they are trying to access the same file or the both of them are trying to access the printer.

(Refer Slide Time: 04:16)




So, if I have got; if I have got two processes. So, one process is like this. So, the in this part of the code of its code is a critical section and for another process; process 2. So, suppose we have got this part of the code as the critical section. Now when process 1 is executing in this portion of the code, so, process 2 should not be allowed to enter into this. So, how do we do this thing and why this is necessary? This can happen because when process 1 is executing. So, in a time shared system maybe it has executed for some duration and then when it was executing the critical section. So, it got descheduled.

And then process 2 came so, process to started from this point now some. So, it should not be allowed beyond this point. So, at this point there should be some check by which the system will prevent this process to from entering into the critical section. So, before entering the critical section the process 2 should make some sort of check that whether I am allowed to enter into the critical section and if it is allowed, then only it should enter into the critical section.

Similarly, the same check was done by P 1 at this point also, but at that time since P 2 was not in its critical section. So, it was allowed to be through after that it might have raised a flag and that after raising that flag. So, when it finished the operation then it comes here then it tells that now I am out of the section. So, this can be; this can be the this signal can be made available and then this P 2 now can be allowed to enter into the critical section.

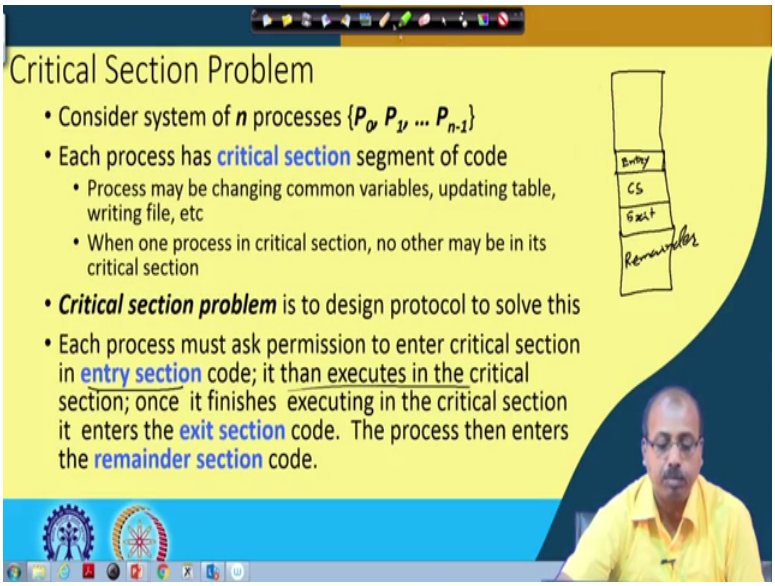
So, it is something like this, suppose there is a room there is a say in the bank there is a vault. So, in the vault so, we want that at one point one point of time only one person should enter into the vault ok. So, if we think that way then what happens is that. So, maybe the first person comes finds that nobody is there in the vault, so, enters into the vault.

Now, after entering into the vault, the person should immediately lock it because no other person should be able to enter. Now if the person is a bit slow then what will happen is that in between some other process other person will come and find the door to be open and that person will also enter. Now this is the question of synchronization like as soon as the person enters the door should automatically get locked. So, similarly in terms of processes you can try to understand that when a process is entering into the critical section, then this entry check and if the check is if the entry is permitted then that entry being stopped just after this check. So, this is very important and this has to be done in an atomic fashion.

So, in between if I say that first I check whether I can enter or not and then after that I close the door. So, if there is; so, if they are done by different instructions in the computer, then in between the process may get descheduled the process may get descheduled. So, before closing the door maybe the process get descheduled so another process comes it checks and it also finds a door to be open and enters into the critical section.

So, this is very important that though it is obvious that when one process in critical section no other process should be there, but ensuring it becomes a bit tricky. So, we have to ensure that this check and permission should be done in an atomic fashion. So, if you look into the piece of code that we have so, critical section problem is to design protocol to solve this problem solve this particular issue, each process must ask permission to enter into the critical section in entry section.

(Refer Slide Time: 08:18)

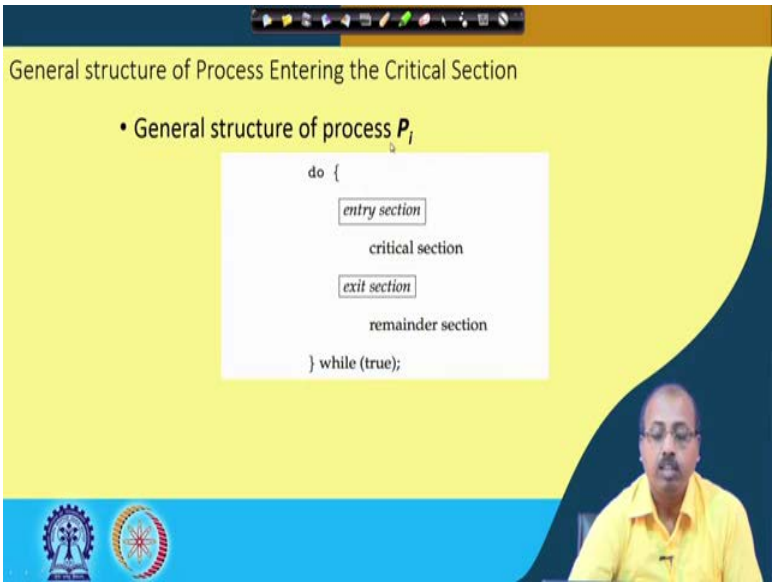



So, if in the code; so, if this is the portion of the code which is the critical section then before that there should be a portion which is known as the entry section. So, if this is the total process execution totals process that we have. So, there is before critical section. So, there is an entry section; in the entry section which the process checks whether I am allowed to enter into the critical section or not. So, maybe it will take some flag and all and based on that it will come to a decision whether it is permitted to enter into the critical section. So, that is the critical section. So, after the entry section, it executes in the critical section and once this execution is over then this comes out of the critical section and comes to another section which we call exit section.

So, in the exit section. So, it will do some activities so, that some other process who is trying to enter into the critical section may be told that you can enter now. So, that way we have got this thing and after this the process after coming out from the critical section may be live for quite some time. So, this part is the reminder remainder section. So, a process initially may be doing something, after that when it is trying to enter into the critical section it goes to an entry section where it checks for some flags etcetera.

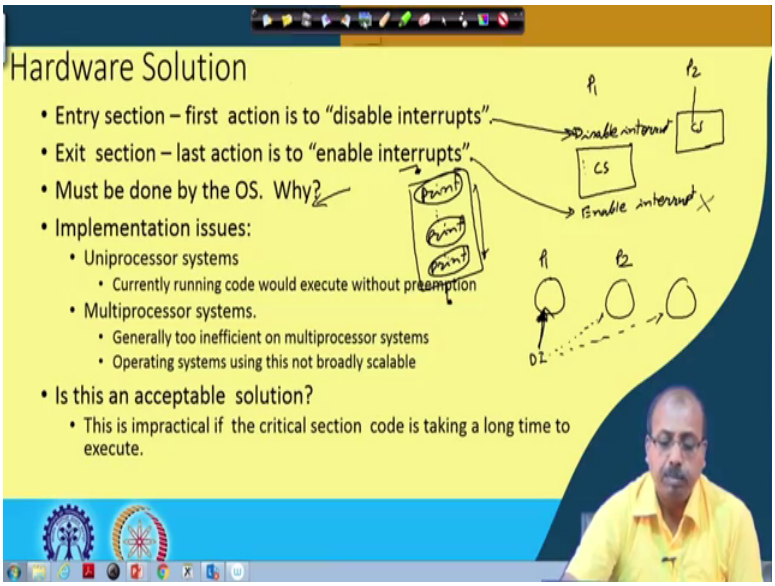
Then it enters into the critical section, and then it comes out of the critical section goes to the exit section via v in which it does resetting of those flags and all and after that it goes to the remainder section. So, this is the flow that we have in the critical section problem. So, we will see how this individual sections can be designed in a computer system?

(Refer Slide Time: 10:11)




So, this is the general structure of process entering into the critical section. So, we purposefully put it in a do infinite loop do while loop telling that a process can enter into the critical section several times ok. So, this is the, but every time it enters into the critical section before that it is in the entry section where it checks whether something whether it is permitted to enter into the critical section, then enters into it then exits and then the remainder section and that for the most general case a process may be running in an infinite loop entered into the critical section simultaneously. So, we have to consider the processes of this generic structure for solving the critical section for solving the critical section problem.

(Refer Slide Time: 10:59)




Now, hardware based solutions that we can have is at the entry section we do a disable interrupts and the exit section we do an enable interrupt. So, what we say is like this that. So, this is the if this is the critical section now if we try to answer the question, why another process may be allowed to enter into the critical section when first process is in critical section? And the answer that comes is this is because of the interrupts because when the process is executing in this maybe the timer interrupts came and the time slice of the process has expired.

So, the timer interrupts comes and then this process gets descheduled and another process which has got critical section somewhere here so, it gets scheduled. So, that way it can enter into the critical section. Or maybe some I O event has occurred and then the process is interrupted or some higher priority processes arrived so, that is why if this is interrupted its execution and it is taken out of execution. Now to stop all these things so, the basic philosophy that we can have is that we disable all interrupts. So, we can have some disable interrupt instruction.

So, most of the processors we have got this disabled interrupt facility. So, it is disabled here and after the critical section is over so, we do enable interrupts. So, what happens is that since in between the interrupt remains the disabled. So, no other no interrupts can come so, the process cannot be disabled. So, process cannot be taken back from the CPU. So, this entry section is simply disabled instruction disabling interrupt instruction and

exit section is enable interrupt instruction and it must be done by the operating system why?

Because if it is left to the user program, then some user may be may have done this disable interrupt, but forgot to do this enable intern. So, this is not there in the core piece of code. So, naturally all the interrupts in the system will remain disabled and the system will not be able to respond to any activity only way out may be to do a non mask able interrupt via the reset line and that the whole system will get reset.

So, that is why this disabled and disabling and enabling interrupts must be done by the operating system and not left to the individual user programs. Implementation issues like uniprocessor system currently running code would execute without preemption, but for multiprocessor system it is generally too inefficient on multiprocessors because in a uniprocessor system if the processor executes a DI instruction or disable interrupt instructions.

So, interrupts are disabled, but if there are multiple processors and say suppose I have got three processors and these processor executes a disable interrupt, but it is not mandatory that the two processes that we have P 1 and P 2 they are having common critical section. So, P 1 may be on this processor and P 2 may be on this processor. So, disabling interrupts of the first processor does not solve the purpose because P 2 is on a separate processor so, that interrupt is not disabled.

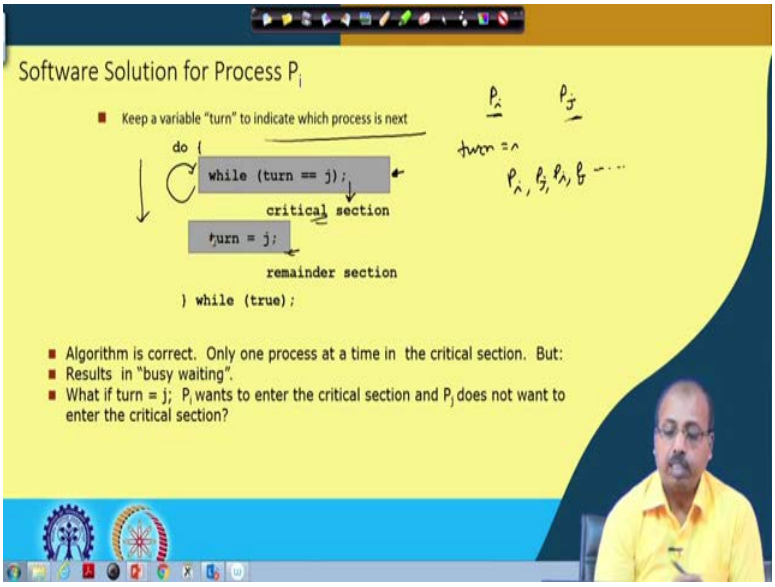
So, naturally p P 2 may run on the second processor. So, it is inefficient for multiprocessor systems. So, multiprocessor system if P 1 for the first processor you want to do a disable interrupt, then that information must go to other processors as well some sort of message passing should be there by which these this will be broadcasted to all the processors that we have in the system. So, operating system will be doing a broadcasting that this is all this interrupts all the interrupts are disabled so, that has to be taken care of.

So, multiprocessor systems it becomes difficult. So, is this an acceptable solution? So, this is impractical if the critical signal section code is taking a long time to execute. So, this is another issue because ultimately this critical section is written by some user. So, and if the user is not very much careful then what can happen is that the critical section code may become very large. So, one user may find that in my program I am trying to do this print operation at several places.

So, somewhere here, then somewhere here then again somewhere at some point later. So, what the user can do is that he may put this entire thing as a critical section, as a result the interrupt is disabled at this point and it is enabled at this point in between for a long duration interrupt remains disabled. However, looking it into more detailed fashion you can understand these are actually the critical sections not the others, but since it is up to the user program to do this i and DI. So, to declare the critical section so, it becomes difficult.

So, user might have demarcated long code as the critical section and sometimes it may be necessary that the long code really makes the critical section, but for most of the situations. So, it is not that. So, it may be taken care of, but if the critical section is too large then this becomes impractical. So, we have to be look into this enable disable interrupt based resolution, but that is not always very efficient.

(Refer Slide Time: 16:40)




So, next well be looking into some other solution some software solution. The first software solution that we look into is by means of a turn variable. So, we assume that this turn variable. So, this turn variable it keeps a indicate like which process is next. So, what is happening is that suppose that this process is coded like this. So, it checks while turn equal to equal to j. So, this is the critical section. Suppose I have got two processes for example. So, say suppose I have got two processes P i and P j and what this P i

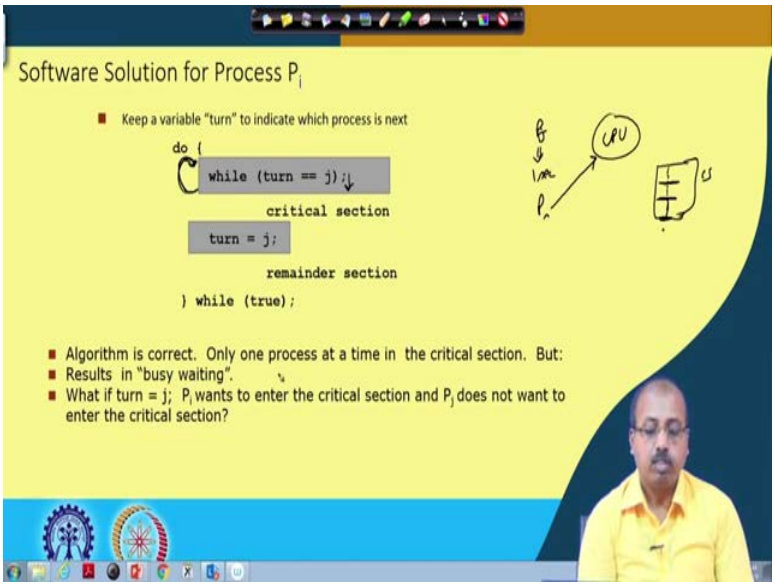
process is checking is it is checking whether the turn is equal to j or not if turn is equal to j then it knows that now it is P j is turned to execute. So, P i gracefully waits in this loop.

So, it is waiting in this loop. So, after some time P j will be P j will execute and its it will be over and. So, this P j the turn will become equal to i and at that time when turn becomes equal to i, so, this check will fail at that point. So, it will find turn is not equal to j in that case it will be through this entry section will come to the critical section. So, it will execute it and then it will set turn to be equal to j.

So, this turn is set equal to j. So, it is the turn of the other process now. So, initially if I assume that turn value is equal to i then what happens? This check fails it enters into the critical section and now it tells that the next turn is of j. So, what is happening is that the once P i is executed then P j is executed then again another instance of P i executed another loop of this P i executes then another P j executes. So, that way I can have a solution and since at any point of time turn can be either equal to i or j it cannot have both the value simultaneously.

So, at one point of time only one process will be in the critical section because to the turn is equal to i then P i will be in critical section, if turn equal to j then P j is in critical section and vice versa. If P i is in critical section then the turn value must be equal to i and if P j is in critical section then turn value must be equal to j. So, algorithm is correct. So, it ensures that two processes they cannot be in their critical section simultaneously and they are not in their critical section simultaneously, but it results in some sort of busy waiting.

(Refer Slide Time: 19:42)



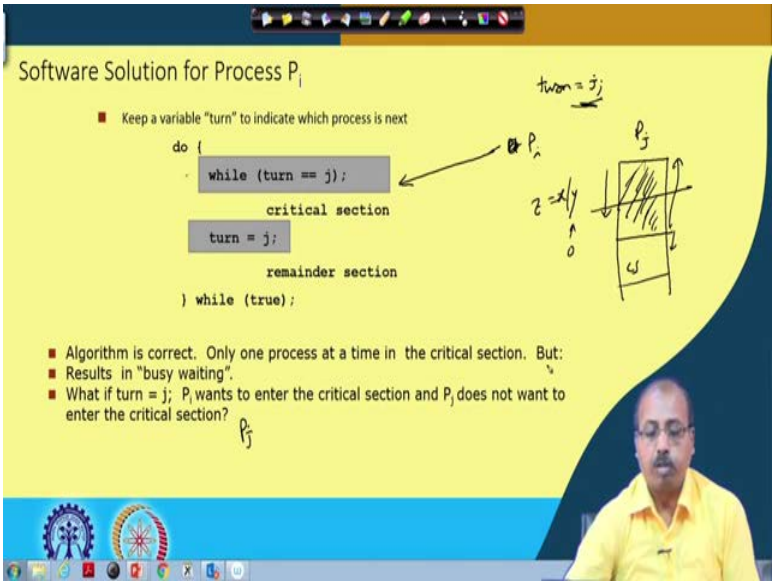

So, busy waiting at this point. So, the turn is equal to j. So, this P i is looping here. So, what happens is that. So, P i is looping here. So, maybe at present P j is executing and P j is executing on the CPU now. So, it is given a time slice of say 1 second. So, it executes for 1 second and it was in the critical section. So, it was in the critical section. So, it is in 1 second it has progressed up to this much and then it got descheduled then P i will get a schedule.

So, P i comes to CPU, but it executes for 1 second only this while loop only this while loop. So, it goes on checking and unnecessarily wastes 1 second of CPU time after that P i will be digital and P j will again come back. So, P j will start from this point again execute for 1 second and again it will be descheduled at this point. So, again P i will come. So, P i will again start waiting at this loop for 1 second; for 1 second it will be executing that while loop unnecessarily and after that P i will get descheduled. And again P j will come and this time suppose it finishes this thing and then it will make turn to be equal to i. So, next time P i get scheduled. So, it will be through this while loop and enter into the critical section.

So, this is some sort of busy waiting. So, this P i is waiting, but it is waiting in a busy loop it is keeping the CPU busy checking the value of the turn variable. So, what if and so, this is one problem with this particular algorithm another problem is serious which it

says that turn equal to j P i wants to enter critical section and P j does not want to enter critical section. So, it may so, happen that we have made a code like this.

(Refer Slide Time: 21:29)



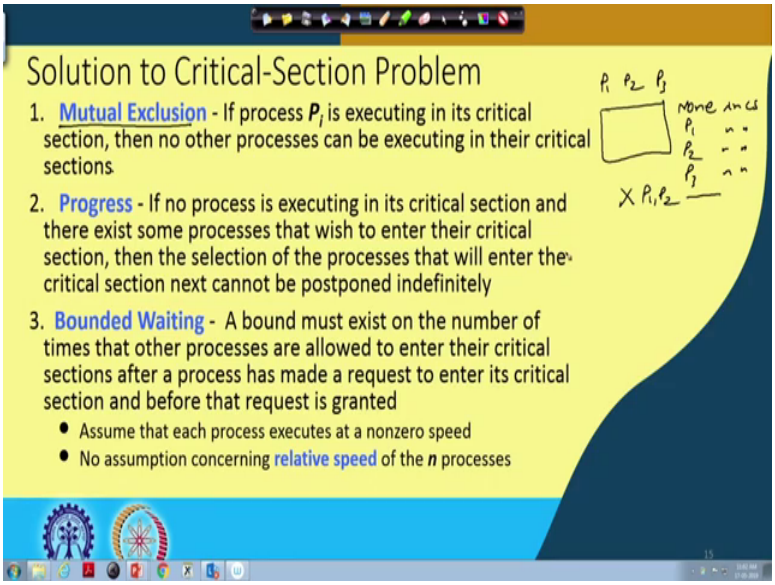

So, initially globally we have made a turn to be equal to j it can be randomly made equal to i or j suppose somebody has written the programs and in such a fashion that turn equal to j, now the process ones code is process P i is code is like this and process p js code is something like this. So, it; so, process P j has got lot of initial things to do and then here I have got the critical section for P j; now P i wants to. So, as soon as P i gets initiated it starts executing this code, but it finds that turn is equal to j. So, it cannot enter into the critical section. So, it goes on waiting here on the other end P j it is not entering into the critical section, it is doing something else in the non critical section.

So, for this much time P i will be made to it and after that P j will come. So, P j will enter into the critical section. So, unnecessarily we are making P i to wait though P j was not entering into the critical section. So, we are making pi 2 wait. So, this is the third point that we have that turn is equal to j and P i wants to enter critical section and P j does not want to enter critical section, then in that case also P i has to wait for P j to come enter into the critical section, go out say turn to be equal to i and then only P i will proceed. A more serious condition that will come suppose the process P j dies in between.

So, while executing say this part of the code. So, there was an there is there is a divide by 0 error. So, z equal to x by y and y value is equal to 0. So, this process dies at this point

then what will happen? Then P i will also be stopped permanently at this check because a turn will remain equal to j this job is not there. So, there is nobody to set turn back to i. So, as a result P i will also be stopped. So, P i will also go on waiting in the busy wait loop. So, this particular solution based on turn variable has got this problem. So, what can be other solution so, that we have to see.

(Refer Slide Time: 23:43)

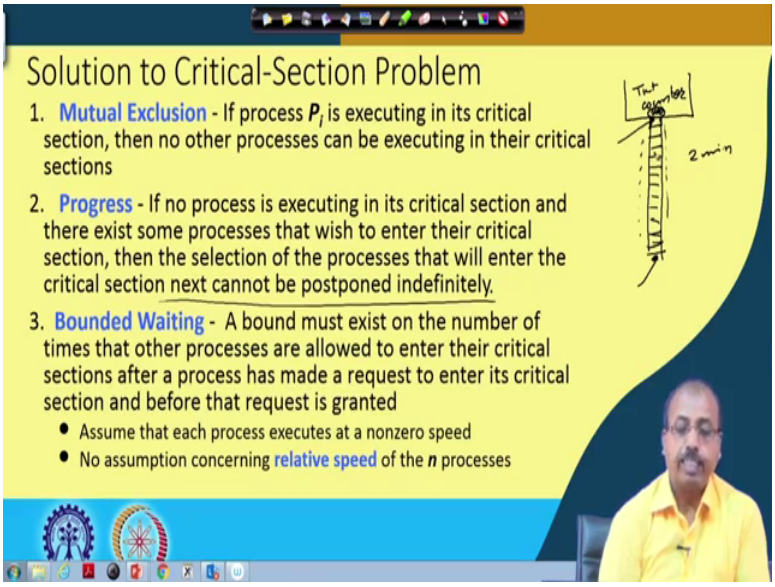



So, will look into some solutions, but before going to those solutions. So, we have to look into some properties of solutions that are given for critical section problem. Like given a critical section problem so, suppose I come up with a new solution mechanism. Now how do I judge whether my solution is correct or not? So, solution is correct if it satisfies these three conditions; the first condition is the mutual exclusion condition, it says that if process P i is executing in its critical section, then no other processes can be executing in their critical section.

So, this is the vital one that if I have got a number of processes P 1 P 2 P 3 etcetera and all of them want all of them have their critical sections then at any point of time only one of these three processes can be at most one of these three processes can in the critical section. So, at if you look into the system at any point of time, you can have a situation that none of them are in critical section none in the critical section or you can have P 1 in critical section or P 2 in critical section or P 3 in critical section.

So, you cannot have the situation that the P 1 and P 2 are in critical section. So, this cannot happen. So, we have to ensure that if there are a number of processes in the to enter into the critical section that only one of them is in the critical section. So, that is the mutual exclusion condition. Then the second condition says its a progress. So, it says that if no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter into the critical section next cannot be postponed indefinitely.

(Refer Slide Time: 25:54)




So, it is like this that suppose in a real life example maybe I have a situation like this. So, suppose I have got a ticket counter ok. So, this is a ticket counter and there is a queue; there is a queue in front of the ticket counter where people are standing in a queue. So, if we come and join the queue at this point so, I see how many people are there in the queue and assuming that, but each person takes a 2 minute time. So, I can calculate like how much time I will need to get my ticket ok.

So, that is a fair enough decision. So, at one point of time, so, this is a problem of critical section. So, this counter access is a critical section problem at one point of time only 1 person can buy ticket ok. So, 2 person requests cannot be processed simultaneously. So, that way we have got this thing. So, now; so, this decision cannot be postponed indefinitely like if this fellow at the beginning of the at the beginning of the counter if suppose this fellow has got a mobile phone and this is continually getting requests over

mobile phone and those requests they are giving to the counter telling that give me this particular ticket also.

So, that way you really do not know like how many tickets will be bought by each of these persons. So, I cannot say that this will. So, at some point of time I will get my ticket because there may be a continuous flow of messages to the mobiles of these people and they can go on buying tickets and then I will not be able to do an estimate at this point.

So, this is this postponement. So, the then the selection of the processes that will enter into the critical section next so, we cannot postpone that thing indefinitely. So, that selection process like who gets next. So, that cannot be postponed indefinitely or maybe we have a situation like this that the person standing here then this person before going. So, they call somebody else and then does it. And then we have got bounded wait there must be a bound on the that on the number of times the other processes are allowed to enter into the critical sections after a process has made a request to enter into the critical section and before that request is grant request is granted.

So, each process; so, they; so, it is there is a bound on the number of times other process that this fellow has come. So, this fellow has joined the queue and telling that I want to get the ticket and there must be a bound on the number of times other processes are allowed to enter into the critical section. So, how many times other processes will go into the critical section before this request can be granted. So, there that is a bounded wait. So, we should have progress that is the all processes that we have that the decision like who next enters into the critical section that decision cannot be postponed and then indefinitely and then who will be waiting next, who will be when will I get that ticket. So, that decision cannot also be postponed.

So, we can say the progress is this maybe this at the ticket counter after getting one ticket, then the person may say there may be two thing that this counter will call the next person for buying the ticket now when this call comes. So, if that is not fixed that there is no bound then this progress may be violated. So, there are so, many people who are waiting for buying the ticket and this fellow the this person that is sitting at the ticket counter so, is not calling anybody to buy ticket. So, that is basically violation of the progress requirement and this the third case that we had when this the calls are coming

over mobile phones and then they purchasing ticket like that so, that is violating the bounded wait.

So, these three conditions must be satisfied mutual exclusion progress and bounded wait and if all of them are satisfied then we can say that this solution is a correct solution to the critical section problem. So, here each process is assumed to execute to execute for a non zero speed and as some no assumption concerning the relative speed of the processes. So, there we do not say like how much at what speed which process will execute. So, they can take arbitrary amount of time. So, under all these assumptions this these three conditions are to be satisfied. So, we look into this in more detail in the next class.