Hello And

welcome.

To yet another module of TBSC Electronics Semester. 5.

Operating systems Paper code ELC105 Unit 3 concurrency, mutual exclusion and synchronization module name semaphores message passing part one module #16 presented by Mr Girish Abhyankar

Associate professor Dnyanprassarak Mandal's college and

Research Center. Assagao Goa.

Outline.

Will be studying about semaphores, the fundamental

principle, mutual exclusion using semaphores, message

passing and synchronization.

In message passing.

Learning outcomes students will be able to explain the

principles of semaphore. Explain The mutual exclusion using

semaphores. Explain the principles of message passing and

explain the synchronization issue with respect to message

passing. Semaphores.

Two or more processes can cooperate by means of simple

signals such that a process can be forced to stop at a specified

place until it has received a

specific signal. Signals can be structured appropriately to

satisfy any complex coordinate

requirement. For signaling special variables, called

semaphores are used.

To transmit a signal via

semaphore s, A process executes the primitive signal (S) signal.

to receive a signal via semaphore S.

A process executes the primitive

wait. If the corresponding signal has not yet been

transmitted, the process is suspended until the transmission

takes place. So what is semaphore? semaphore is

basically a variable that has an integer value upon which three

operations are defined. Semaphore may be initialized to

a non negative value.

The wait operation Decrement The semaphore value.

If the value becomes negative,then the process executing the

wait is blocked.

The signal operation increments.The semaphore value will look at

this in details in the examples.

If the value is not positive,then a process blocked by a wait

operation is unblocked.

OK, and very important.

The wait and signal operations are assumed to be atomic, that is, they cannot be interrupted and each routine is treated as an indivisible step.

So here are the definitions ofthe semaphore primitive. So here we define a struct named semaphore which has two members. One is account variable count of integer type and a queue of Type queue. This is a wait function or wait operation. It is taking one value semaphore S as an input it gives nothing at the output what t it does.? It decrements the count by 1. OK, and if the count dot count is less than zero, then that process is placed in the queue OK and blocked. It is blocked. Similarly, look at this signal. The signal semaphore S. It increments the value of the count by one OK and if S dot count is less than or equal to 0, remove a process from the S dot Q which was previously blocked and place the process P on the ready list. There's Another definition of binary semaphore. If you carefully seeboth, there is now this value. The count was int. Now there is a variable name value. It can take only two values, zero and one because it is a binary semaphore. OK, again we have await signal or wait operation.

Wait B we have called it. It Takes a binary semaphore S.

If S dot value is equal to 1, it is made zero. Else if it's

already 0, this process is placed in the queue and the

process is blocked. Similarly,here we are defining a signal,

it also takes a binary

semaphore s. If S dot queue is empty, then S dot values made as

one else. If it is not empty,process is removed from S dot Q

and placed on the ready list, so these are the definitions of

semaphores OK. So how to achieve mutual exclusion using semaphore?

So here this is a main program.OK, and here is the function P.

So let there be n processes if identified in the array of p i ,

in each process a waitt is executive just before

entering a critical section as shown here, there is a wait

operation OK before entering the critical section. If S becomes

negative, the process is suspended. We have seen that in

the previous slides if becomes 0

after decrementing. The process immediately enters its critical

section, 'cause now S is no longer positive and no other

process will be able to enter its critical section. After this

particular process is already entered, the critical section.

So this is a small code which explains that. So if P1 comes

here and enters the critical section, any other process which tries to enter the critical section will not be able to do so unless the first process which has entered critical comes out an issues up signal procedure because that will increment. semaphore by one. OKso for both semaphores and binary semaphore so Q is used to hold processes waiting on the semaphore. Question is the order in which processes are removed from the queue support. There Are three processes waiting in the queue which will come first which will get unblocked first. So first input first output is the fairest policy, but now in this what happened the process that has been blocked the longest he's released from the queue first, and a semaphore whose definition include this policy is called as a strong semaphorer. Semaphore that does not specify the order in which processes are removed from the block. Q is a weak semaphore OK So here is an example of strong semaphore. You can see this pink color blocks showing up processor look at figure one. We can say that P is a process which is being executed OK. Semaphore already is 1. Now These are Q. And why this PQ and why these are the processes and they depend upon the answer generated by the process X. Letus say like that. OK, so already

there is 1 result generated fromX. So p executing it uses

that result. It gives a wait signal and it gets the signal so

it gets the result and

therefore. It immediately inFigure 2 it has joined back the

Ready queue and the first in the queue Here the process Q has

now come to the processor in theFigure 2 and look at the value

of semaphore S. It was one. Now That he has given a wait signal

so it has gone to zero. OK, now what happens next in the queue

is not able to get the semaphore, not the answer of

process X is what they're looking out for. So Q is no more

getting answer from X because X Is not longer executed after the

first answer for x has been used by P. OK, so Q now enters

the. Block State from figure toFigure 3. Q has come to the

blocks at the next one in the Q has come to the processor,

and Q executes wait semaphore become minus one OK. Q

Exit because he has a executorwhere South became one to zero.

Now Q has executive operations that as becomes minus one. Now

See X has come therefore in the step number in the figure,

number 4X generates output,therefore becomes 0 and now the

Q which was blocked for the result from X has now been moved

to the. Ridiculous and look at the figure 5. Now the y

Process in the Figure 4, which was the first in the waiting

list in the ready queue, has come to the processor and S is

of course 0 then PQ and this access again come and the

process will continue. This is how the semaphore execution is

done. Here is another example,can see that are P Q & R. There are

three processes. C Critical Section is shown by this

a particular dark line. Green line shows normal execution and

blackline shows blocked on the semaphore. OK, so here is a

Q for semaphore lock..

OK, so let us assume these are the dotted lines showing that

time reference. So let us say that this app displays P,

executes a wait on the lock, so it's a value of Ssemaphore which

is a lock already. Lock was one,so lock is decremented by one. So

the value of lock becomes zero and P directly enters the

critical section process. OK now, after some time, the Q also

wants the lock, so it again increments the value of lock by

one, so zero becomes a -- 1. But Since it is minus one, the queue

is. Placed in the blocked queue.OK, it is blocked and it enters

the blocked on the semaphore.The dark black line here shows

that it is being blocked. after another time interval. Say R

also executes a wait for the lock and the lock becomes minus two OK

till then. Now the process is in the critical section. Let

us at this time now so therefore they see when Q&R also executive

both of them executes the wait semaphore value goes from zero

to minus one 2 -- 2 and both of them are in the block queue.

Now P executes signal. So the value of semaphore is

incremented by one, so minus two becomes minus one and the first

process which was blocked is been removed from the block and

now we get Q process is able to enter the critical section then

subsequently Q also completes the critical section executes

signal semaphore from minus one goes to zero and the R which was

blocked is now allowed to go into the critical section. This

is how the shared data protected by semaphore

processes are accessing the shared data protected by the

Semaphore OK. Next issue is message passing. So when the

process interacts with one another, two fundamental

requirements must be satisfied.Synchronization and

communication processes need to be synchronized to enforce

mutual exclusion. Cooperating Processes may need to exchange

information. One approach to providing both these functions

is message passing, so message passing as an advantage that it

lends itself to the implementation in distributed

systems as well as in a shared memory multiprocessor in uniprocessor systems. So what is actually the primitives which are used. These are the primitives send. DestinationComma Message received source comma message so this is a minimum set of operation needed for process engaged in the message passing. So process sends information in the form of message to another process whose address is given as a destination and also a process exhibits receive or news received signal. It tells from which source it is expecting the signal. OK, so issue is related to that which will see in this module is synchronization. So communication of a message between two processes implies some level of synchronization.

Quiet between the two receiver cannot receive a message until a sender is send the message OK.In addition, we need to specify what happens to process after it issues are send or receive primitive. So if you consider the send primitive first. So when a send primitive is executed in the process, there are two possibilities. Either The sending process is blocked until the message sent is received or it is not blocked.

OK, so similarly when the process is user receive it is

trying to receive some messages from the other. Again there are two possibilities. If a message has previously been sent by some source. The message is received.An execution continues if there is no waiting message, then two possibilities. Maybe there one the process is blocked until a message is received or arrives, or the process continues to execute. Abandoning the attempt to receive. Thus both sender and receiver can be blocking or not blocking. So there are three combinations. OK, one is blocking send blocking receiveOK, so the three combinations are there. But in the practical system we may have one or two combination. One is blocking send blocking receive both the sender and receiver are blocked.

That message is delivered. This Is rendezvous OK. This combination allows for tight synchronization between processes, nonblocking send blocking receive. Although sender we continue on it because it's non blocking send, the receiver is blocked until the requested message arrives, so it allows a process to send one or more messages to a variety of destinations as quickly as possible. A process that will receive a message before it can do useful work needs to be blocked until such a message arrives. An example is a server

process. That exists to provide service or resource to other process. Non blockingsend nonblocking receivet is the last of the third category. Neitherparty is required to wait so the non blocking sending the most natural for many concurrent programming tasks. For example,if it is used to request an output operations such as printing, it allows the requesting process to issue the request in the form of a message and carry on OK. One potential danger of course is there in the run blocking set. Is that an error could lead to a situation In which a process repeatedly generate messages consuming system resources, including the processor time in the buffer space OK. Also, the non blocking send places a burden on the programmer to determine that a message has been received. Processes must employ reply messages to acknowledge receipt of message for receive primitive. The blocking version appears to be the most natural.

Generally, a process that requires a message will need the expected information before proceeding. However, if a message is lossed, which can happen in the distributed system, or if a process fails before it sends an anticipated message, receiving process could be blocked indefinitely. This

problem can be solved using non blocking receive. OK, so other possible approaches are to allow the process to test whether a messages waiting before issuing received an allowance process to specify more than one source enough. Receive primitive in. This Second approach is useful if a process is waiting for a message for more than one source and can proceed if any of these messages. OK, so this reference. This material was with respect to referring to these references. OK, thank you.