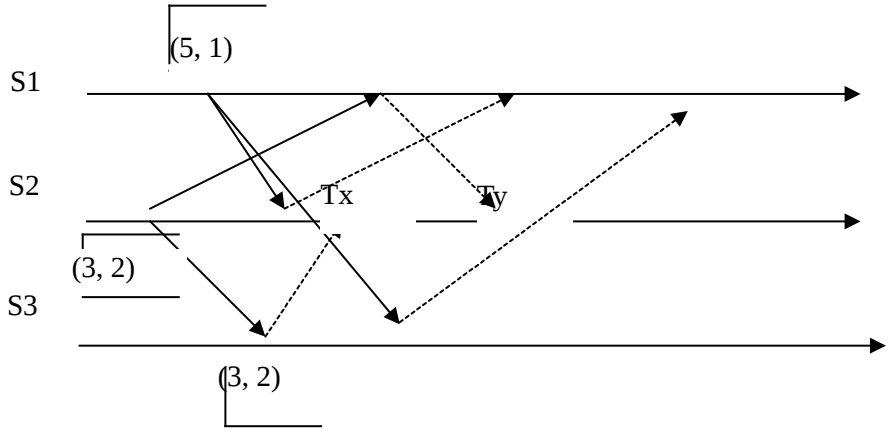
http://www.utdallas.edu/~praba/f03/hw3f02.sol
HOMEWORK 3

1. A student came to my office with the following space-time diagram for Lamport's distributed mutual exclusion algorithm. The student argued that the Site S2 can enter into Critical Section (CS) at Tx (instead of Ty), even though:

i. S2 has not received a REPLY message from S1 at Tx.
ii. S2's request is not at the top of S1's request_queue at Tx.

Give me precise points to counter or accept the student's argument. (If the answer is "Yes", you should say why there would not be a problem in entering into CS without S1's REPLY message and without S2's request being at the top of S1's queue. If the answer is "No", you should explain which aspect of Lamport's algorithm will prevent S2's entry into CS at Tx).



What would be your arguments if the system follows Ricart-Agrawala's algorithm instead of Lamport's?

Solution:
Yes. S2 will enter into CS at Tx by Lamport's algorithm. This is because, according to Lamport, a site can enter into CS if the following two conditions are satisfied:

(1) Site Si's request is at the top of its request queue.
(2) Site Si has received a request from all other sites with timestamps greater than Si's.

In this case, S1's timestamp is larger than S2's timestamp. S2 knows about this based on S1's request. Hence, even though S1's reply has not arrived, S2 can enter CS.
The answer is "No" in the case of Ricart-Agarwala. Because, the condition in Ricart-Agarwala is, only if the site gets reply from all other sites, the requesting site can enter CS. Hence, S2 can enter CS only at Ty.

2. Expand on the Singhal's example discussed in the class as follows: show that after a site executes CS, there may be more than 1 site requesting CS. (Idea is to show the need for a fair selection of sites).

Solution:

• Assume there are 3 sites in the system. Initially:
Site 1: SV1[1] = H, SV1[2] = N, SV1[3] = N. SN1[1], SN1[2], SN1[3] are 0.
Site 2: SV2[1] = R, SV2[2] = N, SV2[3] = N. SNs are 0.
Site 3: SV3[1] = R, SV3[2] = R, SV3[3] = N. SNs are 0.
Token: TSVs are N. TSNs are 0.

• Assume site 2 is requesting token.
S2 sets SV2[2] = R, SN2[2] = 1.
S2 sends REQUEST (2,1) to S1 (since only S1 is set to R in SV[2])
• S1 receives the REQUEST. Accepts the REQUEST since SN1[2] is smaller than the message sequence number.
Since SV1[1] is H: SV1[2] = R, TSV[2] = R, TSN[2] = 1, SV1[1] = N.
Send token to S2

• S2 receives the token. SV2[2] = E.

• Assume that both S1 and S3 request the CS now.
First, S1 sets SV1[1]=R, SN1[1]=1
S1 sends REQUEST(1,1) to S2 (since only S2 is set to R in SV1)

• S2 receives the REQUEST. Accepts the REQUEST since SN2[1] is smaller than the message sequence number. Since SV2[2] = E, SV2[1]=R

• Next S3 requests CS
S3 sets SV3[3] = R, SN3[3] = 1.
S3 sends REQUEST(3,1) to S1 and S2 (since S1 and S2 are set to R in SV3)

• S1 receives the REQUEST. Accepts the REQUEST since SN1[3] is smaller than the message sequence number.
Since SV1[1] = R and SV1[3] = N , set SV1[3] = R and send REQUEST(1,1) to S3.

• S2 receives the REQUEST. Accepts the REQUEST since SN2[3] is smaller than the message sequence number.
Since SV2[2]=E, SV2[3]=R

• After exiting the CS, SV2[2] = TSV[2] = N.
Updates SN, SV, TSN, TSV.
Since both S1 and S3 are currently requesting the CS, S2 can send the token to anyone of them.
In this case, S3 might receive the token even though S1 had requested it first.

3. Assume a distributed system with 9 sites using Maekawa's algorithm. Enumerate the elements (i.e., the site ids) in each subset Ri. Construct an example scenario where the above system run into deadlock (when Maekawa's algorithm is applied) and explain how the deadlock will be resolved.

Solution:
The 9 sites can be grouped into 9 sets. Following the rule N = K(K-1)+1, K will be 4 (if K=3, N will have to be 7 following the stated rule). Members of the sets Ri are as follows: (note that many such sets are possible).

R1: 1,2,3,4
R2: 2,5,8,9
R3: 3,1,6,8
R4: 4,2,5,6
R5: 5,6,1,7
R6: 3,6,7,9
R7: 1,4,7,8
R8: 2,7,8,9
R9: 3,4,5,9

Site 1 and site 9(in R1 and R9) wish to enter the critical section. Site 1 requests all the members in its set (i.e., sites 1,2,3,4). It obtains locks from everybody except site 3. This is because site 3 has given its lock to site 9. Site 9 obtains lock from all the members of its set except site 4. This is because site 4 has given its lock to site 1. So neither site 9 nor site 1 can enter critical section. This is clearly a deadlock situation.
Resolution: Lets assume the timestamp of 9 (t1) is strictly less that timestamp of 1 (t2). In the above scenario, site 4 would receive the request from site 9 after it has granted its lock to site 1. Site 9 being the higher priority site (since t1 < t2), site 4 will send a INQUIRE message to site 1, asking whether it was successful in obtaining the locks of other members. Since site 1 was not successful in obtaining the lock of site 3, it would say a ‘no’ and YIELD its lock to site 4. Site 9 will now obtain the lock of site 4, and will enter CS. Thus the deadlock is resolved.

4. We saw that Singhal's heuristic algorithm may not follow the order of requests to critical section. I feel that the same problem may exist with Suzuki-Kasami's algorithms, i.e., the algorithms may not follow the order of CS requests as well (though the degree of the problem may considerably be lower). Can you construct an example scenario for Suzuki-Kasami algorithm to show this?

Solution:
Suzuki-Kasami Algorithm

Consider the following example:

There are 4 sites (named 1,2,3,4) and let site 1 hold the token currently. Let site 2 request the token at time t2 and let site 3 request the token at time t1, such that t2 > t1.
The LN and RN arrays at site 1 look like this:

	RN Array	LN Array(in the token)	
	---	---	
Ids	1 2 3 4	1 2 3 4	Request queue is empty
Content	7 6 6 5	7 6 6 5	

Now it so happens that site 1 receives 2's request first and then 3's request. After receiving these requests, the queues will look like

	RN Array	LN Array(in the token)	
	---	---	
Ids	1 2 3 4	1 2 3 4	Request queue: 2 3
Content	7 7 7 5	7 6 6 5	

Since 2's request is ahead in the queue and since RN(2) = LN(2) + 1, site 2 is awarded the token. Even though 3's request was made earlier than 2's request, 2 is awarded the token first.

5. Consider Singhal's heuristic algorithm. Is it possible that a token holder does not receive a site's token request message but still passes the token to that site? Explain your answer.

Solution:
Yes it is possible.
Let's imagine that there are 8 sites in the system. Site 4 currently holds the token. Site 3 and site 7 wish to enter the CS. Let's say site 3 sends the request to a subset of servers (i.e., 1,4 and 5). Similarly, site 7 also sends the request to a subset of servers (i.e., 2,4 and 6). Let's say site 4 receives the request of 3 first and then 7. So after completing CS it puts the requests of 3 and 7 in the token status variable (TSV). Then it passes the token to site 3, since it encounters 3 first in TSV. After site 3 completes its CS it passes the token to site 7, since site 7's request is present in the TSV. Note that, even though token holder 3 did not receive 7's request, it still passes the token to it, because of the presence of site 7's request in the TSV.

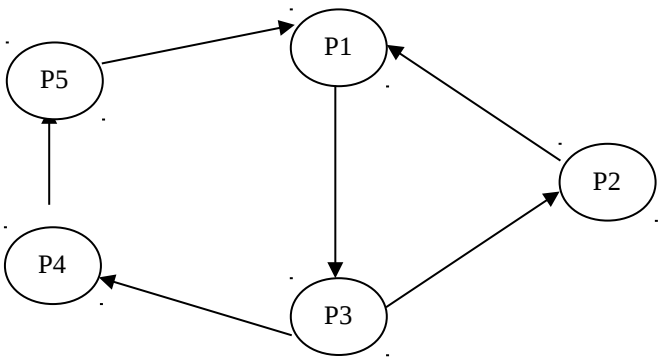
6. Consider the WFG in a distributed system shown in (Figure 7.1 of Singhal's book). Assume that the system follows the OR request model. Apply the diffusion computation based algorithm. Show the flow of messages and determine whether a deadlock is present in the system.

Solution:
Diffusion Algorithm
---

1. P1 sends query (1,1,2) to P2.
2. Since this is an engaging query, P2 forwards this to all the sites in its dependent set DS2. num2(1) = 1, and wait2(1) = true.
3. P3 forwards query to P4[query (1,3,4)]
4. P4 forwards to P5 and P5 to P6 and P7. P7 to P10.
5. Since P10 is not blocked, it discards the query. But P6 forwards to P8, P8 to P9 and P9 to P1.
6. P1 receives (1,9,1), decides that it is not an engaging query and sends a reply to P4.
7. Since num5(1) != 0 in P5, it does not send a reply to P4.
8. P4, P3 and P2 behave in the same manner. So num1(1) != 0 in P1. Therefore deadlock is not present in the system.

7. Consider the following Wait For Graph (WFG) in a distributed system following the AND resource request model. Let us assume that the edge-chasing algorithm is initiated by P5 and P2. Both P5 and P2 detect cycles, and hence detect 2 deadlocks in the system. Now, P5 identifies that *rolling back* or *terminating* P3 (i.e., make P3 release the resources involved in the deadlock) will help in resolving the deadlock (on the left hand side of the figure). Incidentally, rolling back or terminating P3 also helps in resolving the other deadlock (on the right hand side of the figure). However, P2 being unaware of this fact, wants to rollback or terminate P1 to resolve the deadlock (on the right hand side of the figure). Note: you can assume all processes have the same priority.

As you can see, the above case of 2 roll backs or terminations is not needed. Can you suggest some ways to overcome the extra roll back or termination initiated by P2? Try to make your suggestion in such a way that it can work in a general scenario as well, i.e., do not make your suggestion very specific to the given example WFG. Also, your suggestion need NOT have any relation to the edge-chasing algorithm.



Solution:
Main issue is to identify the overlapping process in deadlocks. This can be done by the initiators of deadlock detection algorithm, by exchanging the WFG and determining the overlapping processes. The question that arises is: how to identify the initiators? One possibility is that if a process (e.g. P1 or P3 in the above figure) receives deadlock detection request/probe from more that 1 initiator, the process can inform the initiators about multiple detections being carried out. For example, P1 or P3 can inform P5 and P2 about the 2 simultaneous detections being carried out. So P2 and P5 get to know each other as "initiators". After the 2 deadlocks are detected, P2 and P5 can exchange the detected WFGs and identify the overlapping processes, and reach an agreement on the process to be terminated.

8. Show whether Byzantine agreement can be reached among 5 processors if 2 of them are faulty.

Solution:
According to the Byzantine rule, in order for the processors to reach an agreement, the faulty processors, m, should not exceed: trunc[(n-1)/3] where n is the total number of processors.

Here m = 2 and n = 5.
Since m exceeds trunc ((n-1)/3), Byzantine agreement cannot be reached.