Designated Exercise Set 3

Answers

15.4

In the central server algorithm for mutual exclusion, describe a situation in which two requests are not processed in happened-before order.
Network latencies might affect the order of the token requests the server receives.
Process a might request the token before process b, but if the latency of the connection of process a is high, then the request of process b might reach the centralized server before the request of process a.

(Page 636)

15.6

Give an example execution of the ring-based algorithm to show that processes are not necessarily granted entry to the critical section in happened-before order.
The processes in the ring can communicate independently of the ring.
Imagine we have a clockwise ring of process A, B and C, where process A currently has the token. Process C requests the token. Process B then requests the token.

Process B would then in this case get the token before process C, even though process C requested the token before B. This means that the processes aren't granted entry in happened-before order.

(Page 637)

15.23

Show that Byzantine agreement can be reached for three generals, with one of them faulty, if the generals digitally sign their messages.
If the receiver of a transaction knows who the originator is, then it is possible to find the Byzantine "traitor" (faulty/incorrect).
If the Byzantine commander sends out three different values to three different lieutenants, then because of the signed message, they can determine that the commander is the traitor. They can verify the messages between each other, and can see that the differing messages also come from the commander.
If a lieutenant is a traitor, then the other lieutenants will be able to see the original order, and see who sent them the incorrect order. The lieutenant that sent the incorrect order is the traitor.

(Page 665, Wikipedia)

17.4

Give an example of the interleaving, of two transactions that is serially equivalent at each server but is not serially equivalent globally.
Server 1:
T:Read(A) Write(A)
U:Read(A) Write(A)
These are serially equivalent with T finishing before U

Server 2:
U:Read(B) Write(B)
T:Read(B) Write(B)
These are not serially equivalent globally since the interleavings aren't serially equivalent, due to the cycle T->U->T.
(See page 740)

17.10

A centralized global deadlock detector holds the union of local wait-for graphs. Give an example to explain how a phantom deadlock could be detected if a waiting transaction in a deadlock cycle aborts during the deadlock detection procedure.

If the deadlock detection procedure has "passed" a transaction that then releases its lock (by aborting), then the deadlock no longer exists.
A phantom deadlock will therefore be registered by the deadlock detection algorithm until it runs again.
(Page 745)

17.11

Consider the edge-chasing algorithm (without priorities). Give examples to show that it could detect phantom deadlocks.
By checking the entire wait-for graph, any deadlocks are found and any phantom deadlocks are "cleared up".
If the probe visits every process in the cycle, then any phantom deadlocks will be found. (Shitty explanation, but phantom deadlocks can't exist with edge-chasing.)
(See page 746)

17.12

A server manages the objects a1, a2, ... an.
It provides two operations for its clients:

```
read(i)                 returns the value of ai
write(i, Value)         assigns Value to ai
```

The transactions T, U and V are defined as follows:

```
T: x = read(i); write(j, 44);
U: write(i, 55); write(j, 66);
V: write(k, 77); write(k, 88);
```

1. **Describe the information written to the log file on behalf of these three transactions if strict two-phase locking is in use and U acquires ai and aj before T.**
   As mentioned above, U preceeeds T, while T proceeds U and finally V can transact its entries.
   P0: Initial, P1: WriteData: i=55, P2: WriteData: j=66, P3: PrepareTransaction P1(i) & P2(j), P4: Transact U - P3 (Due to Two-Phase locking)
   P5: WriteData: j=44, P6: PrepareTransaction P5(j), P7: Transact T - P6
   P8: WriteData: k=77, P9: WriteData k=88, P10: PrepareTransaction P9(k), P11: Transact U - P10

2. **Describe how the recovery manager would use this information to recover the effects of T, U and V when the server is replaced after a crash.**
   The recovery manager would upon recovery start at the back of the transaction log(P11), and trace back the intentions of the steps and restore the data items from the given information.
   For example it sees that V has committed, it goes to P10, sees that P10's intentions was to Transact P9(k), goes to P9 and restores ak as 88, goes to P8 and restores ak as 77.
   Then it goes to the P7's commit, then proceeds to the intention of transaction for P6, P5(j), and recovers aj as 44. Then it goes to P4, and moves on to P3 to see the intended transactions of P3. This was P1(i) & P2(j), but as aj was already restored, it skips that and restores ai as 55 and we are now at the intial state of the transaction log(in this example).

3. **What is the significance of the order of the commit entries in the log file?**
   The order of the commit entries in the log file mirror the order of the transaction commits.
   The most recent transactions are the ones who are restored first as they are written to the log last. This could be seen in the abovementioned question, where aj in P2 was skipped as it had already been restored in P5, and we end up with the value aj=44 and not aj=66.

(See pages 753–754)