Universität Freiburg
Institut für Informatik
Prof. Dr. P. Fischer

Georges-Köhler Allee, Geb. 51
D-79110 Freiburg
peter.fischer@informatik.uni-freiburg.de

Exercises
Distributed Systemes: Part 2
Summer Term 2014
3.7.2014
Solution Proposal

4. Exercise sheet: Refresh Concurrency Control and Recovery

Exercise 1
Consider the following schedules.

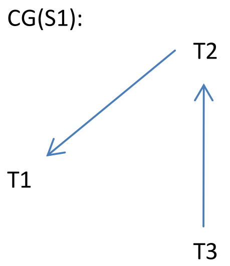
S_1: $R_3X\ R_2Y\ W_2Y\ R_1Y\ W_1Y\ R_2X\ W_2X\ R_1X\ W_1X\ W_3Z.$

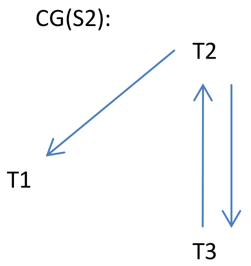
S_2: $R_3X\ R_2Y\ W_2Y\ R_1Y\ W_1Y\ R_2X\ W_2X\ R_1X\ W_1X\ W_3Y.$

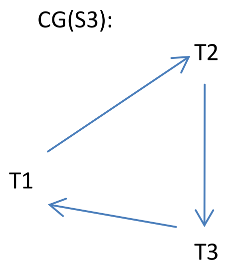
S_3: $R_1Y\ W_1Y\ R_2Y\ W_2Y\ R_2X\ W_2X\ R_3Z\ W_3X\ R_1X\ W_1X.$

For each schedule give its conflict graph. Which schedules are serializable, which are not?

Solution:







S_1 is serializable, S_2, S_3 are not serializable.

Exercise 2
Assume on a database three transactions are being executed.

a) The transactions are of the form:

$$\begin{array}{lll} T_1: & RA & WA \\ T_2: & RA & WA \\ T_3: & RA & WA \end{array}$$

(i) How many serial schedules do exist for T_1, T_2, T_3? Give the reasons!

(ii) How many serializable schedules do exist for T_1, T_2, T_3, which are not serial ones? Give the reasons!

b) The transactions are of the form:

$$\begin{array}{lll} T_1: & RA & WC \\ T_2: & RB & WA \\ T_3: & RC & WD \end{array}$$

(i) How many schedules do exist for T_1, T_2, T_3, which are not serializable? Give the reasons!

(ii) Applying 2-phase-locking, is it possible that all serializable schedules of T_1, T_2, T_3 may occur? Give the reasons!

Solution:

(ai) 6 - all possible permutations

(aii) none - every non-serial schedule either contains $R_i...R_j...W_i...W_j...$ or $R_i...R_j...W_j...W_i...$ where $i \neq j$. Both groups of actions lead to cycles in the conflict graph

bi) None - There only a conflict between T_1 and T_2 as well T_1 and T_3, respectively, but no conflict between T_2 and T_3. As a result, no schedule could ever generate a cycle in the conflict graph.

(bii) No - the following schedule provides a counterexample:

$$S = R_1A\ R_2B\ W_2A\ R_3C\ W_3D\ W_1C$$

In order to generate the prefix $R_1A\ R_2B\ W_2A$ of the schedule S within the constraints of 2PL, U_1A has to occur before W_2A. Because of W_1C and the constraints of 2P L_1C has to precede U_1A. Furthermore U_1C has to follow W_1C. This yields (without restricting generality) the following order of Lock- and Unlock operations in the schedule

$$S = L_1A\ R_1A\ L_2B\ R_2B\ L_1C\ U_1A\ W_2A\ R_3C\ W_3D\ W_1C\ U_1C$$

With this order under 2PL, T_3 cannot be executed.

Exercise 3

Consider the schedule S:

$T_1:$			$R(X)$		$W(Y)$	
$T_2:$				$R(Y)$		$W(Y)$
$T_3:$	$R(Z)$	$W(Y)$				

(a) Demonstrate that S is not (conflict-) serializable.
(b) We call two schedules equivalent, whenever (i) they are built out of the same transactions, (ii) in both schedules the transactions read the same values, and (iii) both schedules produce the same final state of the database. Demonstrate that the serial schedule $T_1\ T_3\ T_2$ and schedule S are equivalent.

Solution:

(a) The conflict graph has a cycle: $T_3 \to T_2 \to T_1 \to T_2$, with conflicts on Y

(b) We need to show that (i) - (iii) hold. (i) holds by definition, since the same three transactions are executed. For th(iii) holds, since For (ii) and (iii) , X and Z do not need to be considered, since these objects are never written to, which means that the same reads we will be performed and they have no effect on the final state. We can therefore focus on Y. To show (ii), we need to explicitly express which "version"/"state" of Y has been read, which can be modelled in the same way as dependency graph (similar to the conflicts). For S, we see $T_3 \to T_2$, i.e., T_2 reads the value of Y written by T_3 before. These are the same read dependencies of $T_1T_3T_2$. For (iii), we observe that the final state of Y is determined by T_2, since its write is executed last. The same effect is cause by $T_1T_3T_2$. Since (i) - (iii), the two schedules are equivalent.

Exercise 4

(a) Give an example of three transactions, which obey 2PL and have the following properties: (i) When being executed a deadlock may occur. (ii) For each pair of the three trnsactions and for any execution of such a pair, no deadlock can occur.
(b) Make suggestions for deadlock-free variants of the 2PL-protocol.

Solution:

(a) One possible set of transactions may look like this: $T_1 = W_1A\ W_1B\ T_2 = W_2B\ W_2C\ T_3 = W_3C\ W_3A$. After executing the first step for each of these three transactions, locks are held for A, B and C. As no transaction can release a lock (due to 2PL) and none can be acquired, a deadlock has occurred. For each pair, a "free" object is available after the first step (e.g. for T_1 and T_2, C is not yet locked) so that one transaction can get all the locks and complete. After that, the other transaction can request the previously locked resource and also complete.

(b) All locks for a transaction could be pre-claimed at the beginning of a transaction in a single request

Exercise 5

Consider the following schedule.

```
                                                    System-
                                                    failure
T1  LA RA       WA                                     |
T2       LB RB           LD RD WB      CO UD,B         |
T3                               LC RC          WC     |
```

Assume that actions W_1A, W_2B are not materialized in the database, however action W_3C is.

(i) Give the state of the database, the ram/buffer manager and the log file when the system failure occurs.
(ii) Describe the operations done when executing the restart algorithm and give the resulting state of the database.

Solution:

(a)
- **Database Buffer/RAM:** A_1, B_2, C_3, where X_i denotes that object X was modified by transaction i.
- **Disk:** A_0,B_0,C_3
- **Log:**
  <1,BOT,1,-,...>
  <2,BOT,2,-,...>
  <3,UPD,1,1,$A := A_1$,$A := A_0$>
  <4,UPD,2,2,B_2,B_0>
  <5,BOT,3,-,...>
  <6,COM,2,4,...>
  <7,UPD,3,5,C_3,C_0>

BOT (begin of transaction) is not strictly necessary, but is shown to make actions more explicit. We use a simplified log format containing:
<sequence no,operation,transaction,previous page,Redo,Undo>

(b)
a) Determining the non-committed transactions (losers) by scanning the log in a forward manner. This yields (T_1, T_3).
b) Redo of all transactions by scanning the log in a forward manner and applying the *Redo* step of each UPD entry, i.e. entries with sequence numbers 3, 4 and 6. The RAM and Disk will contain A_1, B_2, C_3.
c) Undo of Losers by scanning the log backwards and applying the undo entries for losers. As a result, RAM and disk will contain A_0,B_0,C_0