Revision exercises

Exer 1: Consider the following two schedules of concurrent execution:

\[ S_1 = r_1(A); w_4(B); w_1(C); w_3(D); r_2(C); w_3(A); w_1(A); r_3(B); w_3(B); r_2(D); w_2(A) \]

\[ S_2 = r_1(A); w_4(B); w_1(C); w_3(D); r_2(C); w_1(A); w_3(A); r_3(B); w_3(B); r_2(D); w_2(A) \]

α) Draw the precedence graph for each one of them

β) Is these two schedules conflict serializable? If yes, give an equivalent serial schedule

Exer 2: Consider the following pairs of transactions. Use 2-phase locking and insert the appropriate locks for each transaction, assuming that the scheduler performs only exclusive locking. In case of concurrent execution, are there deadlocks?

α) \( T_1 = r_1(A); r_1(B); w_1(C); w_1(B) \)

\( T_2 = r_2(D); r_2(B); w_2(B); w_2(D) \)

β) \( T_1 = r_1(C); r_1(B); w_1(C); w_1(B) \)

\( T_2 = r_2(B); r_2(C); w_2(B); w_2(C) \)
## Upgrading locks (Αναβάθμιση φραγμών)

- A transaction that wants to read and write an element, may first obtain a shared lock on the item for reading (being “friendly” toward other transactions) and then upgrade it to an exclusive lock for writing.

- Example:

  T1: SL1(A); R1(A); SL1(B); R1(B); XL1(B); W1(B); U1(A); U1(B)
  T2: SL2(A); R2(A); SL2(B); R2(B); U2(A); U2(B)

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>SL1(A); R1(A); SL1(B); R1(B); XL1(B); W1(B); U1(A); U1(B)</td>
</tr>
<tr>
<td>T2</td>
<td>SL2(A); R2(A); SL2(B); R2(B); U2(A); U2(B)</td>
</tr>
</tbody>
</table>

If it asked for a XL1(B) from the beginning, then it would have to wait and not even be able to read B.
Upgrading locks (Αναβάθμιση φραγμών)

• Unfortunately, the use of upgrading introduces a new and potentially serious source of deadlocks.

• Example:

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL1(A); R1(A);</td>
<td>SL2(A); R2(A);</td>
<td></td>
</tr>
<tr>
<td>XL1(A); <strong>wait</strong></td>
<td>XL2(A); <strong>wait</strong></td>
<td></td>
</tr>
</tbody>
</table>

• Both T1 and T2 try to upgrade to x-lock, but the scheduler forces each to wait because the other has a s-lock on A.
Update locks (Ενημερώσιμοι φραγμοί)

• We can avoid the deadlock problems caused by lock upgrade by using **update locks**:
  – An update lock ULi(X) allows transaction Ti to read X but not to write X
  – Only an update lock can be upgraded to a write lock (a read lock can **not** be upgraded)
  – An update lock can be granted on X when there are already shared locks on X
  – Once there is an update lock on X, **no** additional locks are allowed on X (otherwise such a lock would never be upgraded to exclusive because there are always other locks)
### Update locks (Ενημερώσιμοι φραγμοί)

- **Example:**

  T1: UL1(A); R1(A); XL1(A); W1(A); U1(A);
  T2: UL2(A); R2(A); XL2(A); W2(A); U2(A);

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>UL1(A); R1(A);</td>
<td>UL2(A); wait</td>
</tr>
<tr>
<td>XL1(A); W1(A); U1(A);</td>
<td>UL2(A); R2(A);</td>
</tr>
<tr>
<td></td>
<td>XL2(A); W2(A); U2(A);</td>
</tr>
</tbody>
</table>
Update locks (Ενημερώσιμοι φραγμοί)

• Compatibility matrix

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>X</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>X</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>U</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
Increment locks (Φραγμοί αύξησης)

• Many transactions operate on the DB only by incrementing or decrementing stored values.
  – e.g. consider a transaction that transfers money from one bank account to another, ticket booking etc.
• The useful property of increment actions is that they can be interchanged with each other and their relative order doesn’t matter
• However, they don’t interchange with transactions that read or write
• Transactions may include operations of the form INC(A,c), meaning that constant c is to be added to DB element A
• INC(A,c) stands for: Read(A,t); t:=t+c; Write(A,t);
• Increment actions need increment locks: ILi(X)
Increment locks (Φραγμοί αύξησης)

• Increment actions and locks requires us to make several modifications to the definitions of consistent transactions, conflicts, and legal schedules:
  – A consistent transaction can only have an increment action INC\(i(X)\) if it holds an increment lock on X at the time. An increment lock does not enable either read or write actions, however.
  – In a legal schedule, any number of transactions can hold an increment lock on item X at a time. If a transaction has an increment lock on X, no other transaction can have a shared or exclusive lock on X at the same time.
  – INC\(i(X)\) conflicts with R\(j(X)\) and W\(j(X)\) for \(j \neq i\)
  – INC\(i(X)\) does not conflict with INC\(j(X)\)
**Increment locks (Φραγμοί αύξησης)**

- **Example:**

  \[
  \begin{align*}
  T1: & \text{ SL1}(A); \text{ R1}(A); \text{ IL1}(B); \text{ INC1}(B); \text{ U1}(A); \text{ U1}(B); \\
  T2: & \text{ SL2}(A); \text{ R2}(A); \text{ IL2}(B); \text{ INC2}(B); \text{ U2}(A); \text{ U2}(B);
  \end{align*}
  \]

\[
\begin{array}{cccc}
\text{T1} & & \text{T2} \\
\hline
\text{SL1}(A); \text{ R1}(A); & & \text{SL2}(A); \text{ R2}(A); \text{ IL2}(B); \text{ INC2}(B); \\
\text{IL1}(B); \text{ INC1}(B); & & \text{U2}(A); \text{ U2}(B); \\
\text{U1}(A); \text{ U1}(B); & & \text{U2}(A); \text{ U2}(B);
\end{array}
\]
### Increment locks (Φραγμοί αύξησης)

- **Compatibility matrix**

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>X</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>X</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>I</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>
Lock table (Πίνακας φραγμών)

- The scheduler maintains a lock table

Handling Lock Requests: First-come-first-served or Priority to S locks or Priority to U locks ...
Locks With Multiple Granularity
What kind of objects do we lock?

- Locking works well in any case, but should we lock small or large objects?
  - If large objects are locked (e.g., relations)
    - Need few locks
    - Low concurrency (χαμηλή συγχρονικότητα)
  - If small objects are locked (e.g., tuples, attributes)
    - Need more locks
    - More concurrency

- We can do both.
  - The bathroom metaphor:
Locking on Hierarchies of DB Elements

• Assume there is a tree structure to the data
  – Hierarchy of lockable elements (relations, tuples, attributes)
  – Data organized in a tree (e.g. B+-tree)
• Locking schemes seen so far perform poorly in such cases
• 3 levels of DB elements:
  – Relations are the largest lockable elements
  – Each relation comprises one or more blocks
  – Each block contains one or more tuples
• E.g. bank (block- or tuple-level locking, but sometimes accounts relation), document management DB
• Need a new type of lock, called a warning lock
Warning locks (Προειδοποιητικοί φραγμοί)

- **Warning locks** denote the intention (πρόθεση) to obtain a lock
  - **IS**: intention to obtain a shared lock
  - **IX**: intention to obtain an exclusive lock

- **Rules:**
  - To place a S or X lock, start at the root of the hierarchy
  - If we are at the element that we want to lock, request S or X lock
  - If the element is further down the hierarchy, place a warning of the appropriate kind at the current node (IS or IX)
  - Proceed to the appropriate child node and repeat until the desired node is reached
Warning locks (Προειδοποιητικοί φραγμοί)

• In order to decide whether or not one of these locks can be granted, we use the compatibility matrix.

<table>
<thead>
<tr>
<th></th>
<th>IS</th>
<th>IX</th>
<th>S</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>IX</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>S</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>X</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
Warning locks (Προειδοποιητικοί φραγμοί)

• Incompatibilities:
  – An IS on a node N is only incompatible with an X lock on N
  – An IX on a node N is incompatible with S and X on N
  – Potential conflicts that may arise with IS/IS, IS/IX, IX/IS and IX/IX will be resolved at a lower level
  – An S on N is compatible with IS and S
  – An X on N is incompatible with every other type of lock
• Can only lock existing items, but not items that might later be inserted.
• To handle insertions / deletions:
  – Get exclusive lock on A before deleting A
  – At insert A operation, the transaction is given exclusive lock on A