Exercise 1 (5 points) Under the 2PL protocol it is possible for transactions to “starve” in the following sense: A transaction gets involved in a deadlock, is chosen as the victim and aborted. After a restart, it again gets involved in a deadlock, is chosen as the victim and aborted, and so on. Provide a concrete example for such a situation, and describe how 2PL could be extended in order to avoid starvation of transactions.

Solution (Thanks, Benedikt!)

In 2PL it depends on the strategy which transaction is chosen as victim. For example, if the victim is chosen randomly it might reduce the probability of the same transaction being chosen as victim several times.

In this example we could pick the transaction is the youngest.

\[ s = w_1(x) w_2(y) w_3(z) \ldots w_1(y) w_2(x) w_3(y) \ldots \]

\( t_1 \) and \( t_2 \) will run into a deadlock since each of them has the lock for \( x \) or \( y \) and since \( t_2 \) is younger than \( t_1 \) it is chosen as victim and killed. All following transactions already started before the deadlock was detected and thus each of them is running longer than the restarted \( t_2 \) and all of them will cause deadlock with \( t_2 \) which is then repeatedly killed.

This could be prevented by proclaiming all required locks at the beginning of the transaction which will prevent deadlocks by design. However, this might cause performance issues.

Exercise 2 (5 points) Show that the wait-die and wound-wait approaches to deadlock prevention both guarantee an acyclic WFG at any point in time.

Solution
Using induction (Thanks, Philipp!)

Proof by structural induction on the edges of the waiting graph $WFG$, denoted as $|WFG|$

**Base Case:** $|WFG| = 0$
Since we have no edges, the graph is trivially acyclic. And the wait-die / wound-wait invariants are satisfied.

**Hypothesis:** $|WFG| = n$ is acyclic. And:
Wait-die invariant: $t_i \rightarrow t_j \Rightarrow i < j$ Wound-wait invariant: $t_i \rightarrow t_j \Rightarrow j < i$

**Step:** $|WFG| = n + 1$
The means, we have a subgraph $|WFG''| = n$, which is acyclic by the hypothesis and an additional edge $e$ on this graph.

**Case wait-die:** The hypothesis holds: $t_i \rightarrow t_j \Rightarrow i < j$. And we can differentiate on the direction of $e = t_k \rightarrow t_l$, i.e. $t_k$ was just blocked by $t_l$:
If $t_k$ was started before $t_l$ ($k < l$). Which means the invariant still holds and the “waits for” relation is a partial order, thus cycle-free.
If $t_l$ was started before $t_k$, the deadlock prevention strategy aborts $t_l$. This causes the edge to never be inserted and in the even smaller graph $WFG''$, the hypothesis still holds.

**Case wound-wait:** The hypothesis holds: $t_i \rightarrow t_j \Rightarrow j < i$. Again, we differentiate on the direction of $e = t_k \rightarrow t_l$:
If $t_k$ was started before $t_l$, then the deadlock prevention algorithm aborts $t_l$ and in the resulting smaller waits for graph all invariants still hold.
If $t_l$ was started before $t_k$, we can also partially order all transactions by the starting time, which means the graph is circle free and the invariant holds.
Direct proofs and proofs by contradiction (Thanks, Christian T.!

Transaction Systems Assignment 5

Exercise 2:

Wait-die:

Direct proof:

~"Old" TAs wait for "young" TAs to finish/release lock.
in case of a conflict. Otherwise abort.

Therefore in any Wait-For-Graph (WFG)

there can only be an edge from an "older"
TA to a "younger" TA. This means the it
is ordered by time and the WFG is acyclic.

Contradiction assume a cyclic WFG. This would mean

there is an edge from a "younger" TA to
an "older" TA (i.e. young waits for old TAs).

But this is in contradiction to the definition of
would wait (because there are only edges from
old to young TAs).

Direct proof:

Wound-wait:

~Only young TAs wait for "old" TAs to release locks.

This means in any WFG there can only be
Edges from "young" TAs to "older" TAs. Again
this means the directed Edges are ordered by
time. This means there cannot be a cycle in WFG.

Contradiction assume a Cycle in the WFG. This would mean

there is an edge from an "older" TA to a "younger"
TA. But according to the definition, this is
not possible.
Exercise 3 (10 points) Consider the following input schedules to the O2PL protocol:

\[ s_1 = w_1(x) \quad r_2(x) \quad c_2 \quad r_3(y) \quad c_3 \quad w_1(y) \quad c_1 \]

\[ s_2 = w_1(x) \quad r_2(x) \quad r_3(y) \quad c_3 \quad r_2(z) \quad c_2 \quad w_1(y) \quad c_1 \]

Which are the corresponding output schedules produced by O2PL? For each of the two schedules, give the details about when locks are requested, granted, attempted to be released, and eventually released.

Solution We define:

- \( o_i \in \{r_i, w_i\} \)
- \( ro_i(x) \) - transaction \( i \) requests lock \( o \)
- \( go_i(x) \) - transaction \( i \) gets lock \( o \)
- \( ao_i(x) \) - transaction \( i \) attempts to release lock \( o \)
- \( uo_i(x) \) - transaction \( i \) eventually releases lock \( o \)

**s_1**:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_1 )</td>
<td>( T_2 )</td>
<td>( T_3 )</td>
</tr>
<tr>
<td>( rw_1(x) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( gw_1(x) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( w_1(x) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( rr_2(x) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( gr_2(x) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( r_2(x) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( ar_2(x) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( rr_3(y) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( gr_3(y) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( r_3(y) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( ar_3(y) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( ur_3(y) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( c_3 )</td>
</tr>
<tr>
<td>( aw_1(y) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( gw_1(y) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( w_1(y) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( aw_1(y) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( uw_1(y) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( ur_2(x) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( c_2 )</td>
</tr>
<tr>
<td>( c_1 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exercise 4 (5 points) Show that O2PL is susceptible to deadlocks (i.e., it is not deadlock free).

Solution We give a counterexample, which leads to a deadlock.

\[ s = r_1(x)w_2(x)w_2(y)c_2w_1(y)c_1 \]

At \( c_2 \) \( T_2 \) cannot commit because it is order dependent on \( T_1 \). But when \( w_1(y) \) would be executed the schedule would not be in CSR. Therefore \( T_1 \) has to wait until \( T_2 \) releases the lock, but this does not happen until \( T_1 \) releases at least one lock. Therefore we have a deadlock here.