

122022010 Shrayank Jai Mistry

Description: Write a program to simulate 2 phase locking protocol (rigorous 2 phase locking) Wound – Die Method For Deadlock Prevention.

Programming Language: Python

```
1 transactions = {}
2 lock_table = {}
3 rollback_transactions = {}
4 waiting_transactions = {}
5 active_transactions = list()
6
7 class lock:
8     def __init__(self, transaction_id, type):
9         self.transaction_id = transaction_id
10        self.type = type
11
12 class transaction:
13     time_stamp_cnt = 0
14     def __init__(self, transaction_id):
15         self.transaction_id = transaction_id
16         self.time_stamp = None
17         self.instructions = list()
18         self.waiting_instructions = list()
19
20 def transaction_rollback(transaction):
21     locks_freed = list()
22     /* Traversing the lock table to give away all locks */
23     for key, value in lock_table.items():
24         if value == None: continue
25         lock = value[0]
26         if lock.transaction_id == transaction.transaction_id:
27             locks_freed.append(key)
28             lock_table[key] = None
29     return locks_freed
```

```

1  def start_schedule(instrucs_cnt, instructions):
2      for i in range(instrucs_cnt):
3          ins = instructions[i]
4
5          /* FOR ROLLEDBACK TRANSACTIONS */
6          for t, _ in rollback_transactions.items():
7              rollback_transactions[t] -= 1
8
9          /* Complete the rolledback transaction */
10         if rollback_transactions[t] == 0:
11             current_instructions = transactions[t].instructions
12             rollback_transactions.clear()
13             start_schedule(len(current_instructions), current_instructions)
14             break
15
16         /* Creating New Transactions */
17         if ins.find('BEGIN') != -1:
18             transaction.time_stamp_cnt += 1
19             transaction_id = ins[ins.find('(') + 1:ins.find(')')]
20             new_transaction = transaction(transaction_id)
21             new_transaction.time_stamp = transaction.time_stamp_cnt
22             transactions[transaction_id] = new_transaction
23
24             print("TRANSACTION (" + transaction_id + ") " + " HAS STARTED.")
25             continue

```

```

1  if ins.find('COMMIT') != -1:
2      transaction_id = ins[ins.find('(') + 1:ins.find(')')]
3      transactions[transaction_id].instructions.append(ins)
4      locks_freed = transaction_rollback(transactions[transaction_id])
5
6      print("TRANSACTION (" + transaction_id + ") " + " HAS COMMITED.")
7
8      ts_made_active = list()
9      for d in locks_freed:
10         for t, data in waiting_transactions.items():
11             if data == d:
12                 active_transactions.append(t)
13                 ts_made_active.append(t)
14         for t in ts_made_active:
15             waiting_transactions.pop(t)
16
17         if len(active_transactions) > 0:
18             print("TRANSACTION " + str(active_transactions) + " STATUS CHANGED FROM WAITING TO ACTIVE.")
19
20             #####
21             TODO Completing remaining instructions of waiting transactions
22
23             active_instructions = transactions[active_transactions[0]]
24             current_instructions = active_instructions.waiting_instructions
25             start_schedule(len(current_instructions), current_instructions)
26             active_transactions.clear()
27             #####
28             continue

```

```

1  /* Storing individual instructions for transactions
2      transaction_id = ins[ins.find('_') + 1:]
3      transactions[transaction_id].instructions.append(ins)
4
5      if waiting_transactions.get(transaction_id) != None:
6          transactions[transaction_id].waiting_instructions.append(ins)
7          continue
8
9      if rollback_transactions.get(transaction_id) != None:
10         continue
11
12     if ins.find('READ') != -1 or ins.find('WRITE') != -1:
13         operation = ins[0:ins.find('(')]
14         data_item = ins[ins.find('(') + 1:]
15         print("TRANSACTION (" + transaction_id + ") IS PERFORMING " + operation + " OPERATION ON DATA [" + data_item + "
16     ]")
17
18     if ins.find('LOCK') != -1:
19         type = ins[ins.find('-') + 1:]
20         data_item = ins[ins.find('(') + 1:]
21
22         /* Creating a new data item in LOCK TABLE
23         if lock_table.get(data_item) == None:
24             lock_table[data_item] = list()
25
26         if len(lock_table[data_item]) == 0:             /* No Locks on data item
27             new_lock = lock(transaction_id, type)
28             lock_table[data_item].append(new_lock)
29             print("LOCK ACQUIRED ON DATA ITEM " + data_item + " BY TRANSACTION (" + transaction_id + ").")
30         else:                                           /* Locks are present
31             locked_t = transactions[lock_table[data_item][0].transaction_id]
32             current_t = transactions[transaction_id]
33
34             if locked_t.time_stamp < current_t.time_stamp: /* ROLLBACK CURRENT
35                 rollback_transactions[current_t.transaction_id] = 4 /* SIMULATION PURPOSE VALUE 4
36                 # rollback_transactions.append([current_t.transaction_id, 3])
37                 transaction_rollback(current_t)
38                 print("TRANSACTION (" + current_t.transaction_id + ") IS BEING ROLLEDBACK.")
39             else:
40                 waiting_transactions[current_t.transaction_id] = data_item
41                 transactions[current_t.transaction_id].waiting_instructions.append(ins)
42                 print("TRANSACTION (" + current_t.transaction_id + ") IS BEING WAITING.")

```

```

1  if __name__ == "__main__":
2      file = open("transaction_input.txt", "r")
3      data = []
4      for line in file:
5          data.append(line.strip())
6
7      instrucs_cnt = int(data[0])
8      instructions = data[1:]
9
10     start_schedule(instrucs_cnt, instructions)


```

INPUT FILE:



```
1  21
2  BEGIN(T1)
3  LOCK-X(A)_T1
4  READ(A)_T1
5  WRITE(A)_T1
6  BEGIN(T2)
7  LOCK-X(B)_T2
8  WRITE(B)_T2
9  LOCK-X(B)_T1
10 READ(A)_T1
11 READ(B)_T1
12 COMMIT(T2)
13 COMMIT(T1)
14 BEGIN(T4)
15 LOCK-X(C)_T4
16 BEGIN(T3)
17 LOCK-S(D)_T3
18 LOCK-S(C)_T3
19 LOCK-S(D)_T4
20 READ(C)_T3
21 COMMIT(T4)
22 COMMIT(T3)
```

OUTPUT :



```
1 TRANSACTION (T1) HAS STARTED.
2 LOCK ACQUIRED ON DATA ITEM A BY TRANSACTION (T1).
3 TRANSACTION (T1) IS PERFORMING READ OPERATION ON DATA [A]
4 TRANSACTION (T1) IS PERFORMING WRITE OPERATION ON DATA [A]
5 TRANSACTION (T2) HAS STARTED.
6 LOCK ACQUIRED ON DATA ITEM B BY TRANSACTION (T2).
7 TRANSACTION (T2) IS PERFORMING WRITE OPERATION ON DATA [B]
8 TRANSACTION (T1) IS BEING WAITING.
9 TRANSACTION (T2) HAS COMMITTED.
10 TRANSACTION ['T1'] STATUS CHANGED FROM WAITING TO ACTIVE.
11 LOCK ACQUIRED ON DATA ITEM B BY TRANSACTION (T1).
12 TRANSACTION (T1) IS PERFORMING READ OPERATION ON DATA [A]
13 TRANSACTION (T1) IS PERFORMING READ OPERATION ON DATA [B]
14 TRANSACTION (T1) HAS COMMITTED.
15 TRANSACTION (T4) HAS STARTED.
16 LOCK ACQUIRED ON DATA ITEM C BY TRANSACTION (T4).
17 TRANSACTION (T3) HAS STARTED.
18 LOCK ACQUIRED ON DATA ITEM D BY TRANSACTION (T3).
19 TRANSACTION (T3) IS BEING ROLLEDBACK.
20 LOCK ACQUIRED ON DATA ITEM D BY TRANSACTION (T4).
21 TRANSACTION (T4) HAS COMMITTED.
22 LOCK ACQUIRED ON DATA ITEM D BY TRANSACTION (T3).
23 LOCK ACQUIRED ON DATA ITEM C BY TRANSACTION (T3).
24 TRANSACTION (T3) IS PERFORMING READ OPERATION ON DATA [C]
25 TRANSACTION (T3) HAS COMMITTED.
```