Dept. of Computer Science and Engineering,

CS69011: Computing Lab 1

Assignment 6

Maximum Marks: 100

Access to the reservation database of Indian Railways is a typical example of a critical section. There may be several concurrent processes that wish to access the database for checking the reservation status and for booking reservations. We may broadly classify these processes into two types, namely:

- *Readers:* These processes simply read the status of the reservations
- *Writers:* These processes change the reservation status by making a reservation

For each train, we may have a *read lock* and a *write lock*. A reader must acquire a *read lock* in order to study the reservation status, while a writer must acquire a *write lock* to write the database. When a process does not get its desired lock, it must wait for it. We are given the following constraints on the acquisition of locks:

1. Write locks are exclusive -- a process can obtain a write lock only when no process has a read or write lock.
2. Read locks are shared -- one or more processes can obtain read locks when no process has a write lock.
3. Writers have higher priority -- new readers are not granted as long as there are waiting writers.
4. Locks are non-preemptive

We will assume that each process holds at most one lock at a time. Suppose the reservation database has the following structure:

```
struct train{
        int train-id;
        int AC2, AC3, SC;  // No. of available berths
        struct reservation *rlist;
}

struct reservation{
        int pnr;
        char passenger-name[20], age[3], sex;
        char class[4];  // AC2, AC3, or SC
        flag status; // waitlisted or confirmed
}

struct train *rail-data;
```

Each train has 3 classes, namely 2AC, 3AC, and SC (sleeper class). *rlist* is a dynamically allocated array of reservations. *rail-data* is a dynamically allocated array of train records. We assume that all this is for a given date only. The possible transactions are as follows:

- Check availability, given the train-id and class
- Make a reservation -- if not available, then it goes into waiting list. The position on the waiting list is given by the position in rlist.
- Cancel a reservation

Task-1:

Write the following functions:

- get-read-lock(train-id) // *The calling process blocks until it gets a read lock on the given train*
- get-write-lock(train-id) // *The calling process blocks until it gets a write lock on the given train*
- release-read-lock(train-id)
- release-write-lock(train-id)

Create a system of 4 child processes, P1, P2, P3, P4, which respectively reads input files in1, in2, in3, in4. Each input file contains a list of instructions of the form:

```
reserve ⟨name⟩ ⟨age⟩ ⟨sex⟩ ⟨trainid⟩ ⟨class⟩
cancel ⟨pnr⟩
```

The reservation database is assumed to be in shared memory. Each reservation request must first check the availability using a read-lock and then obtain a write-lock for making the reservation, if available. If no reservation is available, then the passenger will be waitlisted. If a passenger cancels a reservation, the first waitlisted passenger will be confirmed. The processes sleep for 1 second after processing each reservation / cancellation request. The processes print the outcome of their reservation / cancellation after processing each request.

For simplicity assume that there are 3 trains, and 10 berths of each class in each train.

Task-2:

In continuation with Task 1, we look at the Railways database as a 3-tier hierarchical database.

- At the highest tier, we have the whole database, containing the reservation status of all trains.
- At the middle tier, we have individual trains.
- At the lowest tier, we have the reservation classes (such as 1AC, 2AC, CC, SC, etc) in a given train.

Objects of the higher tiers are collections of objects of the lower tier (for example, the reservation status of a train in a given date is the collection of the reservation status of the different classes in the train).

The object hierarchy may therefore be represented by a tree with the root representing the whole database, nodes at depth 1 representing trains, and nodes at depth 2 representing reservation classes in a train. Processes can acquire the following types of locks on the nodes of the tree:

- *Shared locks* (S): An S-lock at node n is equivalent to read locks on all nodes in the sub-tree rooted at n.
- *Exclusive locks* (X): An X-lock at node n is equivalent to write locks on all nodes in the sub-tree rooted at n.
- *Intention shared* (IS): A process acquires an IS-lock at node n if it requires a S-lock on one or more nodes in the sub-tree rooted at n.
- *Intention exclusive* (IX): A process acquires an IX-lock at node n if it requires a X-lock on one or more nodes in the sub-tree rooted at n.
- *Shared intention exclusive* (SIX): An SIX-lock at node n is equivalent to a S-lock at n plus a IX-lock at n.

In order to lock a node, a process must acquire the appropriate locks on all nodes in the path from the root to that node. The lock compatibility matrix is as follows.

	IS	IX	S	SIX	X
IS	true	true	true	true	false
IX	true	true	false	false	false
S	true	false	true	false	false
SIX	true	false	false	false	false
X	false	false	false	false	false

Incompatible locks on the same object is not permitted at any given point of time.

The priority among locks is as follows: X > SIX > S > IX > IS. // *X has highest priority, IS has lowest priority*

1. Reconstruct your data structure for the Railways database to accommodate the three levels of granularity.
2. Write functions for get-<lock-type>-lock(train-id) and release-<lock-type>-lock(train-id), where <lock-type> can be S, X, IS, IX, SIX.
3. Write functions for the following types of transactions:

    - Making a reservation in a given train and class
    - Checking availability in one or more classes in a given train and booking if available
    - Checking the total availability in a given class among one or more trains with a given source and destination
    - Making reservations in two connecting trains only when both connections have availability
    - Adding an extra coach in a train (amounts to adding to the availability of berths)
    - Adding an extra train

Create a suitable interface to demonstrate the working of the reservation system in accordance with the above system.