In this assignment, you write a multi-threaded ticket-reservation system for a single day. Suppose that you have t trains. Each train has a capacity c of passengers. Queries made to the reservation system are of three types:

1) inquire the number of available seats in a train,
2) book k tickets in a train, and
3) cancel a booked ticket.

In order that the reservation system is not overloaded, there is a limit – call it MAX – on the maximum number of active queries at any instant. Moreover, in order to insure consistency of the database, different threads reading/modifying the reservation for the same train must go through a mechanism of mutual exclusion. You are asked to use the *pthread* API calls in order to implement a simulation of this reservation system.

The main (master) thread creates a list of t trains, and initializes the number of available seats in each train to c. The master thread then creates s threads that run concurrently in a loop, and make automatically generated random queries periodically. Each query is of one of the three types mentioned above. The type of the query, the train number (for queries of type 1 or 2), the number k of seats to book (type 2), and the ticket to cancel (type 3) are generated randomly. During each query (that is, between the beginning and the (successful) completion of a query), a thread sleeps for a random short interval (this may, for example, simulate bank transaction time for booking and cancellation queries). Moreover, between making two consecutive queries, a thread sleeps for a random short interval. Let us now see how the limit MAX on the number of active queries, and the mutual exclusion are to be handled.

At any point of time, at most MAX queries can be active. Any new query ((MAX + 1)-st or (MAX + 2)-nd or so on) must wait until one or more of the active queries finish. Use appropriate condition variable(s) to enforce this restriction. Note that this wait is to be interpreted as *blocking*, that is, a thread waiting for the server load to reduce must block until signaled by another thread during the completion of an active query.

A write query (type 2 or 3) requires care. Two or more threads are not allowed to write the data concurrently for the same train. Moreover, a write cannot run concurrently when a read query is active on the same train. However, writing the data for two different trains may proceed concurrently. Concurrent reads for the same train are also allowed. One possibility is to create a mutex for each of the t trains. But this may be impractical particularly if t is large. In fact, there can be at most MAX active queries at any time. So a better approach is to maintain a shared table with MAX entries. Each entry in the shared table is a triple consisting of a train number, the query type, and the number of the thread which has made this query. A blank entry may be represented by the train number –1. A read query can proceed provided that the corresponding train is not in the shared table in write mode. Likewise, a write query can proceed provided that the corresponding train is not in the shared table in read/write mode. Access to the shared table should be guarded by an appropriate mutex. Whenever a thread succeeds in making a query, it populates a blank entry in the shared table with the appropriate triple. When the query completes, that particular entry is deleted from the shared table by the thread. Notice that when a query fails in order to insure database consistency, the thread is *not blocked*. It instead proceeds to make the next query (after a short sleep) in the loop.

After all the threads run for a predetermined amount of time T, the server needs to shut down. At this time, all worker threads exit one by one. After this, the master thread prints the current reservation status for all the trains, and exits. Use an appropriate barrier in order to synchronize the master thread with the termination of all the worker threads.

Typical values of the parameters that your program should handle are:

t	= the number of trains	= 100
c	= the capacity of each train	= 500
k	= the number of tickets booked in a query of type 2	= A random integer in the range 5–10
s	= the number of worker threads	= 20
MAX	= the maximum number of concurrent active queries	= 5
T	= the total running time	= 1 – 10 minute(s)

Each worker thread may maintain a private (non-shared) list of bookings made by it. A cancellation request is chosen randomly from this private list. The threads should print appropriate diagnostic messages (like query inputs and outputs along with thread numbers, waiting and signaling on a query, time out, and so on).