1. Multiple ATMs are accessing and updating a shared bank account. With out proper synchronization, two withdrawals might result in an overdraft. Write a multi-threaded program simulating ATM withdrawals using a mutex to protect the shared account balance from race conditions.

```
#include <stdio.h>
#include <pthread.h>

int account_balance = 1000;
pthread_mutex_t account_mutex;

void* perform_withdrawal(void* arg) {
   int withdrawal_amount = *((int*)arg);
   pthread_mutex_lock(&account_mutex);

   if (account_balance >= withdrawal_amount) {
      printf("Attempting to withdraw %d...\n", withdrawal_amount);
      account_balance -= withdrawal_amount;
      printf("Withdrawal successful. New balance: %d\n", account_balance);
   } else {
      printf("Failed to withdraw %d. Insufficient funds. Current balance: %d\n", withdrawal_amount,
account_balance);
   }

   pthread_mutex_unlock(&account_mutex);
   return NULL;
}

int main() {
   pthread_t atm1, atm2;
   int amount1 = 700, amount2 = 500;

   pthread_mutex_init(&account_mutex, NULL);

   pthread_create(&atm1, NULL, perform_withdrawal, &amount1);
   pthread_create(&atm2, NULL, perform_withdrawal, &amount2);

   pthread_join(atm1, NULL);
   pthread_join(atm2, NULL);

   pthread_mutex_destroy(&account_mutex);

   return 0;
}
```

```
Compiled Successfully. memory: 1664 time: 0 exit code: 0

Attempting to withdraw 700...
Withdrawal successful. New balance: 300
Failed to withdraw 500. Insufficient funds.
 Current balance: 300
```

2. Multiple users try to book the same seat in a theater simultaneously. Only one should succeed; the rest should get a ”seat already booked” message. Design a thread-safe seat reservation system where each seat can be booked by only one thread. Use a mutex to prevent double booking.

```
#include <stdio.h>
#include <pthread.h>
#include <stdbool.h>

bool isSeatBooked = false;
pthread_mutex_t seatMutex;
```

```
void* tryToBookSeat(void* arg) {
    int userId = *((int*)arg);

    pthread_mutex_lock(&seatMutex);
    if (!isSeatBooked) {
        isSeatBooked = true;
        printf("User %d booked the seat successfully.\n", userId);
    } else {
        printf("User %d failed: Seat already booked.\n", userId);
    }
    pthread_mutex_unlock(&seatMutex);

    return NULL;
}

int main() {
    pthread_t userThreads[5];
    int userIds[5] = {1, 2, 3, 4, 5};

    pthread_mutex_init(&seatMutex, NULL);

    for (int i = 0; i < 5; i++) {
        pthread_create(&userThreads[i], NULL, tryToBookSeat, &userIds[i]);
    }

    for (int i = 0; i < 5; i++) {
        pthread_join(userThreads[i], NULL);
    }

    pthread_mutex_destroy(&seatMutex);

    return 0;
}
```

```
Compiled Successfully. memory: 1792 time: 0 exit code: 0
User 1 booked the seat successfully.
User 3 failed: Seat already booked.
User 2 failed: Seat already booked.
User 4 failed: Seat already booked.
User 5 failed: Seat already booked.
```

3. A group of users wants to book 3 consecutive seats together (e.g., For a family). Multiple such group threads attempt to book, and they should only succeed if 3 adjacent seats are available. Design a thread-safe booking system where each group (thread) checks for and books 3 consecutive available seats. Use mutexes to ensure that no race condition occurs.

```
#include <stdio.h>
#include <pthread.h>
#include <stdbool.h>

#define TOTAL_SEATS 10
bool seatAvailable[TOTAL_SEATS] = {false};
pthread_mutex_t seatLock;

void* reserveSeats(void* arg) {
    int teamID = *((int*)arg);

    pthread_mutex_lock(&seatLock);
    for (int i = 0; i <= TOTAL_SEATS - 3; i++) {
        if (!seatAvailable[i] && !seatAvailable[i+1] && !seatAvailable[i+2]) {
            seatAvailable[i] = seatAvailable[i+1] = seatAvailable[i+2] = true;
            printf("Team %d booked seats %d, %d, %d\n", teamID, i, i+1, i+2);
            pthread_mutex_unlock(&seatLock);
            return NULL;
```

```
Compiled Successfully. memory: 1664 time: 0 exit code: 0
Team 1 booked seats 0, 1, 2
Team 2 booked seats 3, 4, 5
Team 3 booked seats 6, 7, 8
Team 4 failed to book 3 consecutive seats.
```

```
        }
    }
    printf("Team %d failed to book 3 consecutive seats.\n", teamID);
    pthread_mutex_unlock(&seatLock);

    return NULL;
}

int main() {
    pthread_t teamThreads[4];
    int teamIDs[4] = {1, 2, 3, 4};

    pthread_mutex_init(&seatLock, NULL);

    for (int i = 0; i < 4; i++) {
        pthread_create(&teamThreads[i], NULL, reserveSeats, &teamIDs[i]);
    }

    for (int i = 0; i < 4; i++) {
        pthread_join(teamThreads[i], NULL);
    }

    pthread_mutex_destroy(&seatLock);

    return 0;
}
```