Source Code:

```
// Multi-Threaded Library Management System
// A simple, thread-safe library system supporting admin and user roles.
// Admins can manage the book catalog; users can borrow/return books and
check availability.

#include <iostream>
#include <string>
#include <vector>
#include <mutex>
#include <condition_variable>
#include <sstream>
#include <limits>
#include <cstdlib>
#include <cctype>
#include <iomanip>
#include <algorithm>

using namespace std;

// Global mutex to synchronize console I/O across threads
static mutex io_mutex;

// Clear the terminal screen (cross-platform)
inline void clearScreen()
{
#ifdef _WIN32
    system("cls");
#else
    cout << "\033[2J\033[H" << flush;
#endif
}

// Read an integer choice from user, reprompt on invalid
int getMenuChoice()
{
    int    choice;
    string line;

    while (true)
    {
        {
            lock_guard<mutex> io(io_mutex);
            cout << "Choice: " << flush;
        }

        if (!getline(cin, line))
```

```
            return -1;

        stringstream ss(line);
        if (ss >> choice)
            return choice;

        {
            lock_guard<mutex> io(io_mutex);
            cout << "Invalid input. Please enter a number.\n" << flush;
        }
    }
}

// Reader-writer lock with writer preference
class RWLock
{
public:
    void lockRead();
    void unlockRead();
    void lockWrite();
    void unlockWrite();
    bool tryLockWrite();

private:
    mutex              mtx;
    condition_variable cv;
    int                activeReaders  = 0;
    int                waitingWriters = 0;
    bool               writerActive   = false;
};

void RWLock::lockRead()
{
    unique_lock<mutex> lk(mtx);
    cv.wait(lk, [&]() { return !writerActive && waitingWriters == 0; });
    ++activeReaders;
}

void RWLock::unlockRead()
{
    unique_lock<mutex> lk(mtx);
    if (--activeReaders == 0)
        cv.notify_all();
}

void RWLock::lockWrite()
{
    unique_lock<mutex> lk(mtx);
    ++waitingWriters;
```

```
    cv.wait(lk, [&]() { return !writerActive && activeReaders == 0; });
    --waitingWriters;
    writerActive = true;
}

void RWLock::unlockWrite()
{
    unique_lock<mutex> lk(mtx);
    writerActive = false;
    cv.notify_all();
}

bool RWLock::tryLockWrite()
{
    unique_lock<mutex> lk(mtx, try_to_lock);
    if (!lk.owns_lock() || writerActive || activeReaders > 0)
        return false;
    writerActive = true;
    return true;
}

// Book record
struct Book
{
    string title;
    string author;
    int    count;
    int    id;
};

// User account record
struct Account
{
    string        username;
    string        firstName;
    string        middleName;
    string        lastName;
    string        password;
    int           id;
    bool          loggedIn;
    bool          isAdmin;

    // NEW: which book IDs this user currently has borrowed
    vector<int>   borrowedBookIds;
};

// Main library class
class Library
{
```

```
public:
    Library();
    void registerUser();
    int  loginUser();
    void userSession(int idx);

private:
    void listAllBooks();
    void addBook();
    void updateBook();
    void removeBook();
    void borrowBook();
    void returnBook();
    void checkAvailability();
    void displayLockStatus();
    void detectDeadlocks();
    void ensureFairness();

    int   findBookIndex(const string &title);
    bool  validPassword(const string &pwd);

    vector<Book>    books;
    vector<Account> accounts;

    // track which account is currently active
    int             currentUserIdx = -1;

    RWLock            booksLock;
    recursive_mutex   updateMutex;
    mutex             cvMutex;
    condition_variable bookCv;
    mutex             accountMutex;
};

// Default admin account information
Library::Library()
{
    accounts.push_back({
        "admin",
        "Administrator",
        "",
        "",
        "password",
        1,
        false,
        true,
        {}              // borrowedBookIds empty
    });
}
```

```
// Find book by title
int Library::findBookIndex(const string &title)
{
    for (size_t i = 0; i < books.size(); ++i)
        if (books[i].title == title)
            return static_cast<int>(i);
    return -1;
}

// Password strength check
bool Library::validPassword(const string &pwd)
{
    if (pwd.size() < 8) return false;

    bool upper   = false;
    bool lower   = false;
    bool digit   = false;
    bool special = false;

    for (char c : pwd)
    {
        upper   |= isupper((unsigned char)c);
        lower   |= islower((unsigned char)c);
        digit   |= isdigit((unsigned char)c);
        special |= ispunct((unsigned char)c);
    }

    return upper && lower && digit && special;
}

// Register a new user
void Library::registerUser()
{
    lock_guard<mutex> lk(accountMutex);

    string first, middle, last, pwd, confirm;
    {
        lock_guard<mutex> io(io_mutex);
        cout << "First Name: " << flush;
    }
    getline(cin, first);

    {
        lock_guard<mutex> io(io_mutex);
        cout << "Middle Name: " << flush;
    }
    getline(cin, middle);
```

```
    {
        lock_guard<mutex> io(io_mutex);
        cout << "Last Name: " << flush;
    }
    getline(cin, last);

    string uname = first.substr(0,1) + middle.substr(0,1) + last;
    {
        lock_guard<mutex> io(io_mutex);
        cout << "Your username: " << uname << '\n' << flush;
    }

    while (true)
    {
        {
            lock_guard<mutex> io(io_mutex);
            cout << "Password (Minimum of 8 chars, must include
upper/lower/digit/special): " << flush;
        }
        getline(cin, pwd);

        if (!validPassword(pwd))
        {
            lock_guard<mutex> io(io_mutex);
            cout << "Weak password.\n" << flush;
            continue;
        }

        {
            lock_guard<mutex> io(io_mutex);
            cout << "Confirm password: " << flush;
        }
        getline(cin, confirm);

        if (pwd != confirm)
        {
            lock_guard<mutex> io(io_mutex);
            cout << "Passwords do not match.\n" << flush;
        }
        else
        {
            break;
        }
    }

    accounts.push_back({
        uname,
        first,
        middle,
```

```
        last,
        pwd,
        static_cast<int>(accounts.size()) + 1,
        false,
        false,
        {}   // start with no borrowed books
    });

    {
        lock_guard<mutex> io(io_mutex);
        cout << "User registered with ID: " << accounts.back().id << '\n'
<< flush;
    }
}

// Login user
int Library::loginUser()
{
    string uname, pwd;

    {
        lock_guard<mutex> io(io_mutex);
        cout << "Username: " << flush;
    }
    getline(cin, uname);

    {
        lock_guard<mutex> io(io_mutex);
        cout << "Password: " << flush;
    }
    getline(cin, pwd);

    lock_guard<mutex> lk(accountMutex);
    for (auto &acct : accounts)
    {
        if (acct.username == uname && acct.password == pwd)
        {
            acct.loggedIn = true;
            lock_guard<mutex> io(io_mutex);
            cout << "Welcome, " << acct.firstName << "!\n" << flush;
            return acct.id - 1;
        }
    }

    {
        lock_guard<mutex> io(io_mutex);
        cout << "Invalid credentials.\n" << flush;
    }
    return -1;
```

```
}

// List all books
void Library::listAllBooks()
{
    lock_guard<mutex> io(io_mutex);
    booksLock.lockRead();

    if (books.empty())
    {
        cout << "No books in the library.\n";
    }
    else
    {
        constexpr int ID_W = 4, T_W = 40, A_W = 30, C_W = 6;
        cout << left
             << setw(ID_W) << "ID"
             << setw(T_W) << "Title"
             << setw(A_W) << "Author"
             << setw(C_W) << "Count" << "\n";
        cout << string(ID_W + T_W + A_W + C_W, '-') << "\n";

        for (auto &b : books)
        {
            cout << left
                 << setw(ID_W) << b.id
                 << setw(T_W) << b.title
                 << setw(A_W) << b.author
                 << setw(C_W) << b.count << "\n";
        }
    }

    booksLock.unlockRead();
}

// Add book
void Library::addBook()
{
    lock_guard<mutex> io(io_mutex);

    cout << "Book title: " << flush;
    string t; getline(cin, t);

    cout << "Author: " << flush;
    string a; getline(cin, a);

    cout << "Quantity: " << flush;
    int c; cin >> c;
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
```

```
    booksLock.lockWrite();
    books.push_back({t, a, c, static_cast<int>(books.size()) + 1});
    booksLock.unlockWrite();

    cout << "Added '" << t << "'.\n" << flush;
}

// Update book
void Library::updateBook()
{
    lock_guard<recursive_mutex> rec(updateMutex);
    lock_guard<mutex>            io(io_mutex);

    cout << "Title to update: " << flush;
    string t; getline(cin, t);

    booksLock.lockWrite();
    int idx = findBookIndex(t);
    if (idx < 0)
    {
        cout << "Book not found.\n" << flush;
        booksLock.unlockWrite();
        return;
    }

    cout << "New title: " << flush;  getline(cin, books[idx].title);
    cout << "New author: " << flush; getline(cin, books[idx].author);
    cout << "New qty: " << flush;    cin >> books[idx].count;
    cin.ignore(numeric_limits<streamsize>::max(), '\n');

    booksLock.unlockWrite();
    cout << "Book updated.\n" << flush;
}

// Remove book
void Library::removeBook()
{
    lock_guard<mutex> io(io_mutex);

    cout << "Title to remove: " << flush;
    string t; getline(cin, t);

    booksLock.lockWrite();
    int idx = findBookIndex(t);
    if (idx < 0)
    {
        cout << "Book not found.\n" << flush;
        booksLock.unlockWrite();
```

```
        return;
    }

    books.erase(books.begin() + idx);
    booksLock.unlockWrite();

    cout << "Book removed.\n" << flush;
}

// Borrow book
void Library::borrowBook()
{
    lock_guard<mutex> io(io_mutex);

    // remember who’s borrowing
    int uid = currentUserIdx;

    cout << "Title to borrow: " << flush;
    string t; getline(cin, t);

    if (!booksLock.tryLockWrite())
    {
        cout << "Library busy. Try later.\n" << flush;
        return;
    }

    int idx = findBookIndex(t);
    if (idx < 0)
    {
        cout << "Book not found.\n" << flush;
        booksLock.unlockWrite();
        return;
    }

    if (books[idx].count == 0)
    {
        cout << "Out of stock. Waiting...\n" << flush;
        booksLock.unlockWrite();

        unique_lock<mutex> lk(cvMutex);
        bookCv.wait(lk, [&]() {
            booksLock.lockRead();
            bool ok = (findBookIndex(t) >= 0 &&
books[findBookIndex(t)].count > 0);
            booksLock.unlockRead();
            return ok;
        });

        booksLock.lockWrite();
```

```
        idx = findBookIndex(t);
    }

    if (idx >= 0 && books[idx].count > 0)
    {
        --books[idx].count;

        // record it on the user’s account
        accounts[uid].borrowedBookIds.push_back(books[idx].id);

        cout << "Borrowed '" << t << "'. Remaining: " << books[idx].count
<< "\n" << flush;
    }
    else
    {
        cout << "Still unavailable.\n" << flush;
    }

    booksLock.unlockWrite();
}

// Return book
void Library::returnBook()
{
    lock_guard<mutex> io(io_mutex);

    // remember who’s returning
    int uid = currentUserIdx;

    cout << "Title to return: " << flush;
    string t; getline(cin, t);

    booksLock.lockWrite();
    int idx = findBookIndex(t);

    if (idx < 0)
    {
        cout << "Book not found.\n" << flush;
        booksLock.unlockWrite();
        return;
    }

    int bookId = books[idx].id;
    auto &loaned = accounts[uid].borrowedBookIds;
    auto it = find(loaned.begin(), loaned.end(), bookId);

    if (it != loaned.end())
    {
        // user did borrow it: accept the return
```

```
        ++books[idx].count;
        loaned.erase(it);

        cout << "Returned '" << t << "'. Now: " << books[idx].count << "\n"
<< flush;
    }
    else
    {
        // user never borrowed that title
        cout << "You did not borrow that book, so it cannot be returned.\n"
<< flush;
    }

    booksLock.unlockWrite();
    bookCv.notify_all();
}

// Check availability
void Library::checkAvailability()
{
    lock_guard<mutex> io(io_mutex);

    cout << "Title to check: " << flush;
    string t; getline(cin, t);

    booksLock.lockRead();
    int idx = findBookIndex(t);

    if (idx >= 0)
        cout << books[idx].count << " copies available.\n" << flush;
    else
        cout << "Book not found.\n" << flush;

    booksLock.unlockRead();
}

// Lock status
void Library::displayLockStatus()
{
    if (booksLock.tryLockWrite())
    {
        booksLock.unlockWrite();
        lock_guard<mutex> io(io_mutex);
        cout << "Write lock is free.\n" << flush;
    }
    else
    {
        lock_guard<mutex> io(io_mutex);
        cout << "Write lock is held.\n" << flush;
```

```
    }
}

// Deadlock stub
void Library::detectDeadlocks()
{
    lock_guard<mutex> io(io_mutex);
    cout << "No deadlocks detected.\n" << flush;
}

// Fairness stub
void Library::ensureFairness()
{
    lock_guard<mutex> io(io_mutex);
    cout << "Fairness ensured (no starvation).\n" << flush;
}

// User session loop
void Library::userSession(int idx)
{
    // record who’s active
    currentUserIdx = idx;

    while (accounts[idx].loggedIn)
    {
        clearScreen();

        if (accounts[idx].isAdmin)
        {
            {
                lock_guard<mutex> io(io_mutex);
                cout << "\nAdmin Menu:\n"
                     << "1) Add Book\n"
                     << "2) Update Book\n"
                     << "3) Remove Book\n"
                     << "4) List All Books\n"
                     << "5) Lock Status\n"
                     << "6) Deadlock Info\n"
                     << "7) Fairness Info\n"
                     << "8) Logout\n";
            }

            int choice = getMenuChoice();
            switch (choice)
            {
                case 1:
                {
                    addBook();
                    break;
```

```
            }
            case 2:
            {
                updateBook();
                break;
            }
            case 3:
            {
                removeBook();
                break;
            }
            case 4:
            {
                listAllBooks();
                break;
            }
            case 5:
            {
                displayLockStatus();
                break;
            }
            case 6:
            {
                detectDeadlocks();
                break;
            }
            case 7:
            {
                ensureFairness();
                break;
            }
            case 8:
            {
                accounts[idx].loggedIn = false;
                {
                    lock_guard<mutex> io2(io_mutex);
                    cout << "Logged out.\n" << flush;
                }
                break;
            }
            default:
            {
                {
                    lock_guard<mutex> io2(io_mutex);
                    cout << "Invalid option.\n" << flush;
                }
                break;
            }
        }
```

```
        }
        else
        {
            {
                lock_guard<mutex> io(io_mutex);
                cout << "\nUser Menu:\n"
                     << "1) Borrow Book\n"
                     << "2) Return Book\n"
                     << "3) Check Availability\n"
                     << "4) Logout\n";
            }

            int choice = getMenuChoice();
            switch (choice)
            {
                case 1:
                {
                    borrowBook();
                    break;
                }
                case 2:
                {
                    returnBook();
                    break;
                }
                case 3:
                {
                    checkAvailability();
                    break;
                }
                case 4:
                {
                    accounts[idx].loggedIn = false;
                    {
                        lock_guard<mutex> io2(io_mutex);
                        cout << "Logged out.\n" << flush;
                    }
                    break;
                }
                default:
                {
                    {
                        lock_guard<mutex> io2(io_mutex);
                        cout << "Invalid option.\n" << flush;
                    }
                    break;
                }
            }
        }
```

```
        // Pause before clearing
        {
            lock_guard<mutex> io(io_mutex);
            cout << "Press Enter to continue..." << flush;
        }
        cin.get();
    }
}

// Main Program
int main()
{
    Library lib;

    while (true)
    {
        clearScreen();

        {
            lock_guard<mutex> io(io_mutex);
            cout << "\nMenu:\n"
                 << "1) Register\n"
                 << "2) Login\n"
                 << "3) Exit\n";
        }

        int choice = getMenuChoice();

        if (choice == 1)
        {
            lib.registerUser();
        }
        else if (choice == 2)
        {
            int idx = lib.loginUser();
            if (idx >= 0)
                lib.userSession(idx);
        }
        else if (choice == 3)
        {
            break;
        }
        else
        {
            lock_guard<mutex> io(io_mutex);
            cout << "Invalid choice.\n" << flush;
        }
```

```
        {
            lock_guard<mutex> io(io_mutex);
            cout << "Press Enter to continue..." << flush;
        }
        cin.get();
    }

    lock_guard<mutex> io(io_mutex);
    cout << "Shutting down...\n" << flush;
    return 0;
}
```

TEST CASE 1: VALID USER REGISTRATIO

```
Menu:
1) Register
2) Login
3) Exit
Choice: 1
First Name: Justine Jhigz
Middle Name: Digal
Last Name: Vizco
Your username: JDVizco
Password (Minimum of 8 chars, must include upper/lower/digit/special): Justine123!
Confirm password: Justine123!
User registered with ID: 2
Press Enter to continue...
```

TEST CASE 2: INVALID USER REGISTRATION

```
Menu:
1) Register
2) Login
3) Exit
Choice: 1
First Name: Justine Jhigz
Middle Name: Digal
Last Name: Vizco
Your username: JDVizco
Password (Minimum of 8 characters, must include  upper, lower, digit, special):
Weak password: must be at least 8 characters long,  must include at least:
  - one uppercase letter
  - one lowercase letter
  - one digit
  - one special character (e.g. !@#$%)
Password (Minimum of 8 characters, must include  upper, lower, digit, special):
```

```
Menu:
1) Register
2) Login
3) Exit
Choice: 1
First Name: Justine Jhigz
Middle Name: Digal
Last Name: Vizco
Your username: JDVizco
Password (Minimum of 8 characters, must include  upper, lower, digit, special): wasd
Weak password: must be at least 8 characters long,  must include at least:
  - one uppercase letter
  - one digit
  - one special character (e.g. !@#$%)

Password (Minimum of 8 characters, must include  upper, lower, digit, special): Wasd
Weak password: must be at least 8 characters long,  must include at least:
  - one digit
  - one special character (e.g. !@#$%)

Password (Minimum of 8 characters, must include  upper, lower, digit, special): Wasd1
Weak password: must be at least 8 characters long,  must include at least:
  - one special character (e.g. !@#$%)

Password (Minimum of 8 characters, must include  upper, lower, digit, special): Wasd1!
Weak password: must be at least 8 characters long,
Password (Minimum of 8 characters, must include  upper, lower, digit, special): Justine123!
Confirm password: Wasd1!
Passwords do not match.
```

TEST CASE 3: VALID USER REGISTRATION

```
Menu:
1) Register
2) Login
3) Exit
Choice: 1
First Name: Justine Jhigz
Middle Name: Digal
Last Name: Vizco
Your username: JDVizco
Password (Minimum of 8 characters, must include  upper, lower, digit, special): Justine123!
Confirm password: Justine123!
User registered with ID: 2
Press Enter to continue...
```

TEST CASE 4: INVALID USER LOGIN

```
Menu:
1) Register
2) Login
3) Exit
Choice: 2
Username: admin
Password: admin
Invalid credentials.
Press Enter to continue...
```

TEST CASE 5: VALID ADMIN LOGIN

```
Menu:
1) Register
2) Login
3) Exit
Choice: 2
Username: admin
Password: password
```

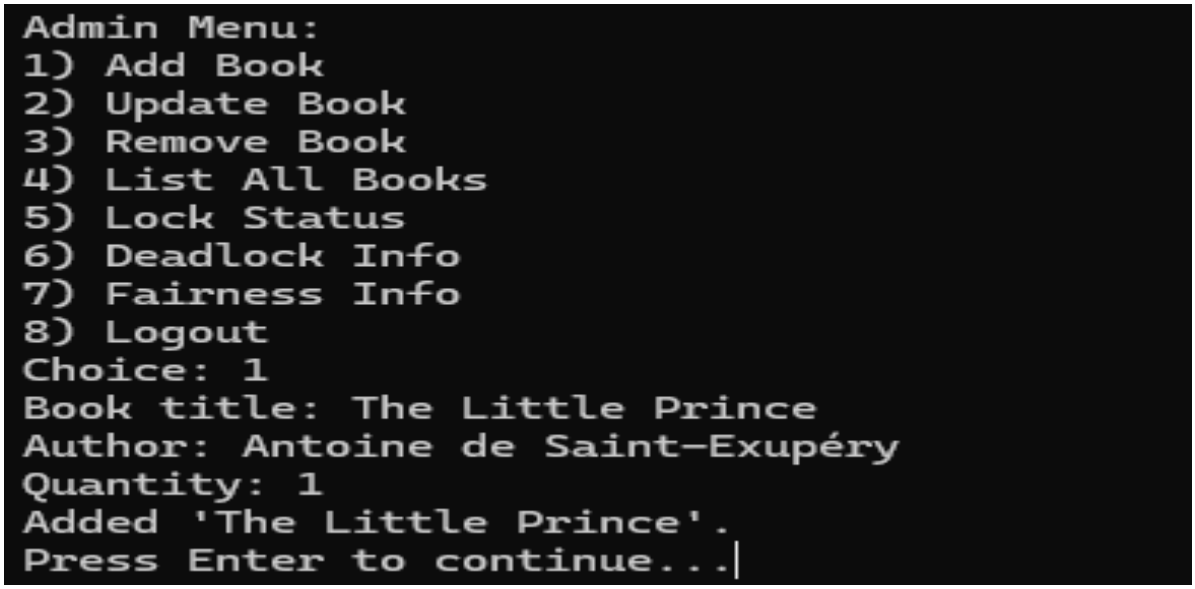
```
Welcome, Administrator!

Admin Menu:
1) Add Book
2) Update Book
3) Remove Book
4) List All Books
5) Lock Status
6) Deadlock Info
7) Fairness Info
8) Logout
Choice:
```

TEST CASE 6: ADD BOOK

```
Welcome, Administrator!

Admin Menu:
1) Add Book
2) Update Book
3) Remove Book
4) List All Books
5) Lock Status
6) Deadlock Info
7) Fairness Info
8) Logout
Choice: 1
Book title: The Lord of the Rings
Author: J.R.R. Tolkien
Quantity: 1
Added 'The Lord of the Rings'.
Press Enter to continue...
```



```
Admin Menu:
1) Add Book
2) Update Book
3) Remove Book
4) List All Books
5) Lock Status
6) Deadlock Info
7) Fairness Info
8) Logout
Choice: 1
Book title: Harry Potter and the Sorcerer's Stone
Author: J.K. Rowling
Quantity: 1
Added 'Harry Potter and the Sorcerer's Stone'.
Press Enter to continue...
```

```
Admin Menu:
1) Add Book
2) Update Book
3) Remove Book
4) List All Books
5) Lock Status
6) Deadlock Info
7) Fairness Info
8) Logout
Choice: 4
ID  Title                                Author                      Count
-------------------------------------------------------------------------
1   The Lord of the Rings                J.R.R. Tolkien              1
2   The Little Prince                    Antoine de Saint-Exupéry    1
3   Harry Potter and the Sorcerer's Stone J.K. Rowling               1
Press Enter to continue...
```

TEST CASE 7: INVALID UPDATE BOOK

```
Admin Menu:
1) Add Book
2) Update Book
3) Remove Book
4) List All Books
5) Lock Status
6) Deadlock Info
7) Fairness Info
8) Logout
Choice: 2
Title to update: abc
Book not found.
Press Enter to continue...
```

TEST CASE 8: VALID UPDATE BOOK

```
Admin Menu:
1) Add Book
2) Update Book
3) Remove Book
4) List All Books
5) Lock Status
6) Deadlock Info
7) Fairness Info
8) Logout
Choice: 2
Title to update: Harry Potter and the Sorcerer's Stone
New title: Harry Potter
New author: Justine
New qty: 1
Book updated.
Press Enter to continue...

Admin Menu:
1) Add Book
2) Update Book
3) Remove Book
4) List All Books
5) Lock Status
6) Deadlock Info
7) Fairness Info
8) Logout
Choice: 4
ID  Title                                Author                         Count
--------------------------------------------------------------------------------
1   The Lord of the Rings                J.R.R. Tolkien                 1
2   The Little Prince                    Antoine de Saint-Exupéry       1
3   Harry Potter                         Justine                        1
Press Enter to continue...
```

TEST CASE 9: LIST ALL BOOKS

```
Admin Menu:
1) Add Book
2) Update Book
3) Remove Book
4) List All Books
5) Lock Status
6) Deadlock Info
7) Fairness Info
8) Logout
Choice: 4
ID  Title                                Author                         Count
--------------------------------------------------------------------------------
1   The Lord of the Rings                J.R.R. Tolkien                 1
2   The Little Prince                    Antoine de Saint-Exupéry       1
3   Harry Potter                         Justine                        1
Press Enter to continue...
```

TEST CASE 10: LOGOUT AS ADMIN

```
Admin Menu:
1) Add Book
2) Update Book
3) Remove Book
4) List All Books
5) Lock Status
6) Deadlock Info
7) Fairness Info
8) Logout
Choice: 8
Logged out.
Press Enter to continue...
```

```
Menu:
1) Register
2) Login
3) Exit
Choice:
```

TEST CASE 11: INVALID BORROW BOOK AS USER

```
Welcome, Justine!

User Menu:
1) Borrow Book
2) Return Book
3) Check Availability
4) Logout
Choice: 1
Title to borrow: Abcd
Book not found.
Press Enter to continue...
```

TEST CASE 12: VALID BORROW BOOK AS USER

```
Welcome, Justine!

User Menu:
1) Borrow Book
2) Return Book
3) Check Availability
4) Logout
Choice: 1
Title to borrow: The Little Prince
Borrowed 'The Little Prince'. Remaining: 0
Press Enter to continue...
```

TEST CASE 13: INVALID RETURN BOOK AS USER

```
Welcome, Justine!

User Menu:
1) Borrow Book
2) Return Book
3) Check Availability
4) Logout
Choice: 2
Title to return: Harry Potter and the Sorcerer's Stone
You did not borrow that book, so it cannot be returned.
Press Enter to continue...
```

TEST CASE 14: VALID RETURN BOOK AS USER

```
Welcome, Justine Jhigz!

User Menu:
1) Borrow Book
2) Return Book
3) Check Availability
4) Logout
Choice: 2
Title to return: The Little Prince
Returned 'The Little Prince'. Now: 1
Press Enter to continue...
```

TEST CASE 15: CHECK AVAILABILITY

```
Welcome, Justine Jhigz!

User Menu:
1) Borrow Book
2) Return Book
3) Check Availability
4) Logout
Choice: 3
Title to check: abc
Book not found.
Press Enter to continue...
```

```
User Menu:
1) Borrow Book
2) Return Book
3) Check Availability
4) Logout
Choice: 3
Title to check: The Little Prince
1 copies available.
Press Enter to continue...
```

```
User Menu:
1) Borrow Book
2) Return Book
3) Check Availability
4) Logout
Choice: Invalid input. Please enter a number.
Choice: 1
Title to borrow: The Lord of the Rings
Borrowed 'The Lord of the Rings'. Remaining: 0
Press Enter to continue...

User Menu:
1) Borrow Book
2) Return Book
3) Check Availability
4) Logout
Choice: 3
Title to check: The Lord of the Rings
0 copies available.
Press Enter to continue...
```

MACHINE PROBLEM 2 - PROBLEM 2

HOSPITAL MANAGEMENT SYSTEM

```cpp
#include <condition_variable>
#include <iostream>
#include <thread>
#include <mutex>
#include <map>
#include <shared_mutex>
#include <string>
#include <vector>
#include <atomic>
#include <set>
#include <chrono>
using namespace std;

// ============================================
//                    CLASSES

struct Appointment {
   int id;
   int patientId;
   string datetime;
   string reason;
};

struct Patient {
   int id;
   string name;
   int age;
};

struct Record {
   int patientId;
   string patientName;
   int patientAge;
   vector<string> entries;
};

// CHECK DEADLOCKS
class LockMonitor {
public:
   atomic<bool> patientLock{false};
   atomic<bool> appointmentLock{false};
   atomic<bool> recordLock{false};

   // Show current lock status for each resource
   void displayLockStatus() {
       cout << "\n--- Lock Status ---\n";
       cout << "Patient Lock: " << (patientLock ? "LOCKED" : "UNLOCKED") << "\n";
       cout << "Appointment Lock: " << (appointmentLock ? "LOCKED" : "UNLOCKED") << "\n";
       cout << "Record Lock: " << (recordLock ? "LOCKED" : "UNLOCKED") << "\n";
   }

   // Naive check to simulate potential deadlock situations
   void checkDeadlocks() {
       cout << "\n--- Deadlock Check ---\n";
       if (patientLock && appointmentLock && recordLock) {
           cout << "⚠️  Potential deadlock: all resources are locked!\n";
       } else {
           cout << "No deadlocks detected.\n";
```

```
        }
    }
};

LockMonitor lockMonitor; // global lockMonitor

// PATIENT MANAGER
class PatientManager {
private:
    map<int, Patient> patients;
    shared_mutex patientMutex;
    int nextPatientId = 0;

public:
    // Register a new patient
    void registerPatient(const string& name, int age) {
        lockMonitor.patientLock = true;
        unique_lock lock(patientMutex);
        int id = ++nextPatientId;
        patients[id] = {id, name, age};
        cout << "Patient registered with ID " << id << ": " << name << "\n";
        lockMonitor.patientLock = false;
    }

    // Update EXISTING patient
    void updatePatient(int id, const string& name, int age) {
        if (patientMutex.try_lock()) {
            if (patients.find(id) != patients.end()) {
                patients[id] = {id, name, age};
                cout << "Patient updated: " << name << "\n";
            } else {
                cout << "Patient not found.\n";
            }
            patientMutex.unlock();
        } else {
            cout << "Patient database is busy. Try again later.\n";
        }
    }

    // Remove an EXISTING/registered patient(s)
    void removePatient(int id) {
        lockMonitor.patientLock = true;
        unique_lock lock(patientMutex);
        if (patients.erase(id)) {
            cout << "Patient removed.\n";
        } else {
            cout << "Patient not found.\n";
        }
        lockMonitor.patientLock = false;
    }

    // List all EXISTING/registered patient(s)
    void listPatient() {
        lockMonitor.patientLock = true;
        shared_lock lock(patientMutex);
        for (const auto& [id, patient] : patients) {
            cout << "ID: " << id << ", Name: " << patient.name << ", Age: " << patient.age << "\n";
        }
        lockMonitor.patientLock = false;
    }
};

// APPOINTMENT MANAGER
class AppointmentManager {
private:
    map<int, Appointment> appointments;
    mutex appMutex;
```

```
    condition_variable_any appointmentNotif;
    int nextAppointmentId = 0;

public:
    // Schedule appointments
    void scheduleAppointment(int patientId, const string& datetime, const string& reason) {
        lockMonitor.appointmentLock = true;
        unique_lock lock(appMutex);
        int id = ++nextAppointmentId;
        appointments[id] = {id, patientId, datetime, reason};
        cout << "Appointment scheduled with ID " << id << ".\n";
        appointmentNotif.notify_all();  // Notifies the system if there are waiting threads
        lockMonitor.appointmentLock = false;
    }

    // Update EXISTING appointment
    // Emphasis on existing
    void updateAppointment(int id, const string& newDatetime, const string& newReason) {
        if (appMutex.try_lock()) {
            if (appointments.find(id) != appointments.end()) {
                appointments[id].datetime = newDatetime;
                appointments[id].reason = newReason;
                cout << "Appointment updated.\n";
            } else {
                cout << "Appointment not found.\n";
            }
            appMutex.unlock();
        } else {
            cout << "Appointments are currently being updated. Try again later.\n";
        }
    }

    // Cancel/Remove Existing Appointment by ID
    void cancelAppointment(int id) {
        lockMonitor.appointmentLock = true;
        unique_lock lock(appMutex);
        if (appointments.erase(id)) {
            cout << "Appointment canceled.\n";
        } else {
            cout << "Appointment not found.\n";
        }
        lockMonitor.appointmentLock = false;
    }

    // List all EXISTING/scheduled appointments
    void listAppointments() {
        lockMonitor.appointmentLock = true;
        unique_lock lock(appMutex);
        for (const auto& [id, appt] : appointments) {
            cout << "ID: " << id << ", Patient ID: " << appt.patientId
                 << ", DateTime: " << appt.datetime << ", Reason: " << appt.reason << "\n";
        }
        lockMonitor.appointmentLock = false;
    }
};

// RECORD MANAGER
class RecordManager {
private:
    map<int, Record> records;
    mutex recordMutex;

public:
    // Add new patient record
    void addRecord(int patientId, const string& name, int age) {
        lockMonitor.recordLock = true;
        unique_lock lock(recordMutex);
```

```
        if (records.find(patientId) == records.end()) {
            records[patientId] = {patientId, name, age, {}};
            cout << "Record created for Patient ID " << patientId << ".\n";
        } else {
            cout << "Record already exists for this patient.\n";
        }
        lockMonitor.recordLock = false;
    }

    // Update EXISTING record
    void updateRecord(int patientId, const string& entry) {
        if (recordMutex.try_lock()) {
            if (records.find(patientId) != records.end()) {
                records[patientId].entries.push_back(entry);
                cout << "Medical record updated for Patient ID " << patientId << ".\n";
            } else {
                cout << "No record found. Add one first.\n";
            }
            recordMutex.unlock();
        } else {
            cout << "Record system is busy. Try again later.\n";
        }
    }

    // View EXISTING patient record by ID
    void viewRecord(int patientId) {
        lockMonitor.recordLock = true;
        unique_lock lock(recordMutex);
        if (records.find(patientId) != records.end()) {
            const auto& r = records[patientId];
            cout << "Record for Patient ID " << patientId << ":\n";
            cout << "Name: " << r.patientName << ", Age: " << r.patientAge << "\n";
            cout << "Entries:\n";
            for (const auto& entry : r.entries) {
                cout << "- " << entry << "\n";
            }
        } else {
            cout << "No records found for this patient.\n";
        }
        lockMonitor.recordLock = false;
    }
};
// ============================================

// ============================================
//                      MENU's

// PATIENT MENU
void patientMenu() {
    cout << "\n=== Patient Management Menu ===\n";
    cout << "1. Register Patient\n";
    cout << "2. Update Patient\n";
    cout << "3. Remove Patient\n";
    cout << "4. List Patients\n";
    cout << "0. Back to Main Menu\n";
    cout << "Choose an option: ";
}

// APPOINTMENT MENU
void appointmentMenu() {
    cout << "\n--- Appointment Management Menu ---\n";
    cout << "1. Schedule Appointment\n";
    cout << "2. Update Existing Appointment\n";
    cout << "3. Remove Existing Appointment\n";
    cout << "4. List Appointments\n";
    cout << "0. Back to Main Menu\n";
    cout << "Choose an option: ";
```

```
}

// RECORD MENU
void recordMenu() {
   cout << "\n--- Recording Management Menu ---\n";
   cout << "1. Add Record\n";
   cout << "2. Update Record\n";
   cout << "3. View Records\n";
   cout << "0. Back to main menu.\n";
   cout << "Choose an option: ";
}

// MAIN MENU
void menu() {
   cout << "\n--- Hospital Management Menu ---\n";
   cout << "1. Patient Management\n";
   cout << "2. Appointment Management\n";
   cout << "3. Record Management\n";
   cout << "4. Concurrency Control\n";
   cout << "5. Check Deadlocks\n";
   cout << "0. Exit\n";
   cout << "Choose an option: ";
}
// ============================================

int main() {
   // Create instances of the three system managers
   PatientManager pm;
   AppointmentManager am;
   RecordManager rm;
   int mainChoice = -1; // Set to run at least once

   while (mainChoice != 0) {
       menu();
       cin >> mainChoice;

       if (mainChoice == 1) { // Patient Management
           int patientChoice = -1;
           while (patientChoice != 0) {
               patientMenu();
               cin >> patientChoice;

               int id, age;
               string name;

               if (patientChoice == 1) { // Register Patient
                   cout << "Enter Name: ";
                   cin.ignore();
                   getline(cin, name);
                   cout << "Enter Age: ";
                   cin >> age;
                   pm.registerPatient(name, age);
               } else if (patientChoice == 2) { // Update Patient Information
                   cout << "Enter ID, New Name, New Age: ";
                   cin >> id >> name >> age;
                   pm.updatePatient(id, name, age);
               } else if (patientChoice == 3) { // Remove EXISTING Patient(s)
                   cout << "Enter ID to remove: ";
                   cin >> id;
                   pm.removePatient(id);
               } else if (patientChoice == 4) { // List ALL EXISTING patients
                   pm.listPatient();
               } else if (patientChoice == 0) {
                   cout << "Returning to main menu...\n";
               } else {
                   cout << "Invalid choice.\n";
               }
```

```
        }
    } else if (mainChoice == 2) { // Appointment Management
        int appointmentChoice = -1;
        while (appointmentChoice != 0) {
            appointmentMenu();
            cin >> appointmentChoice;

            if (appointmentChoice == 1) { // Schedule new appointment
                int patientId;
                string date, reason;
                cout << "Enter Patient ID: ";
                cin >> patientId;
                cin.ignore();
                cout << "Enter Appointment Date: ";
                getline(cin, date);
                cout << "Enter Reason: ";
                getline(cin, reason);
                am.scheduleAppointment(patientId, date, reason);
            } else if (appointmentChoice == 2) { // Update EXISTING appointment
                int id;
                string newDate, newReason;
                cout << "Enter Appointment ID: ";
                cin >> id;
                cin.ignore();
                cout << "Enter New Date: ";
                getline(cin, newDate);
                cout << "Enter New Reason: ";
                getline(cin, newReason);
                am.updateAppointment(id, newDate, newReason);
            } else if (appointmentChoice == 3) { // Cancel/Remove EXISTING appointment
                int id;
                cout << "Enter Appointment ID to cancel: ";
                cin >> id;
                am.cancelAppointment(id);
            } else if (appointmentChoice == 4) { // List ALL EXISTING appointments
                am.listAppointments();
            } else if (appointmentChoice == 0) {
                cout << "Returning to main menu...\n";
            } else {
                cout << "Invalid choice.\n";
            }
        }
    } else if (mainChoice == 3) { // Record Management
        int recordChoice = -1;
        while (recordChoice != 0) {
            recordMenu();
            cin >> recordChoice;
            cin.ignore();

            if (recordChoice == 1) { // Add new record by ID
                int id, age;
                string name;
                cout << "Enter Patient ID: ";
                cin >> id; // Uses PatientManager's patient ID to add records to REGISTERED patients
                cin.ignore();
                cout << "Enter Name: ";
                getline(cin, name);
                cout << "Enter Age: ";
                cin >> age;
                cin.ignore();
                rm.addRecord(id, name, age);
            } else if (recordChoice == 2) { // Update EXISTING patient's record(s)
                int id;
                string entry;
                cout << "Enter Patient ID: ";
                cin >> id;
                cin.ignore();
```

```
                cout << "Enter new record entry (e.g., '2025-05-25: Follow-up for BP'): ";
                getline(cin, entry);
                rm.updateRecord(id, entry);
            } else if (recordChoice == 3) { // View EXISTING patient's record(s)
                int id;
                cout << "Enter Patient ID: ";
                cin >> id;
                rm.viewRecord(id);
            } else if (recordChoice == 0) {
                cout << "Returning to main menu...\n";
            } else {
                cout << "Invalid choice.\n";
            }
        }
    } else if (mainChoice == 4) { // View current lock status
        lockMonitor.displayLockStatus();
    } else if (mainChoice == 5) { // Check for deadlocks
        lockMonitor.checkDeadlocks();
    } else if (mainChoice == 0) { // Exit
        cout << "Terminating program...\n";
    } else {
        cout << "Invalid choice.\n";
    }
}
// Simulate concurrency with threads
cout << "\n--- Simulating concurrent operations ---\n";

// Thread 1: Register multiple patients
auto patientThread = [&]() {
    for (int i = 0; i < 5; ++i) {
        pm.registerPatient("Patient_" + to_string(i), 20 + i);
        this_thread::sleep_for(chrono::milliseconds(100));
    }
};

// Thread 2: Schedule appointments
auto appointmentThread = [&]() {
    for (int i = 1; i <= 5; ++i) {
        am.scheduleAppointment(i, "2025-06-" + to_string(10 + i), "Checkup");
        this_thread::sleep_for(chrono::milliseconds(80));
    }
};

// Thread 3: Add record entries
auto recordThread = [&]() {
    for (int i = 1; i <= 5; ++i) {
        rm.addRecord(i, "Patient_" + to_string(i), 20 + i);
        rm.updateRecord(i, "Initial visit - all clear");
        this_thread::sleep_for(chrono::milliseconds(90));
    }
};

// Launch threads
thread t1(patientThread);
thread t2(appointmentThread);
thread t3(recordThread);

// Join threads
t1.join();
t2.join();
t3.join();

cout << "\n--- Concurrent operations finished ---\n";

// Optional: Display lock status and check for deadlocks
lockMonitor.displayLockStatus();
lockMonitor.checkDeadlocks();
```

```
    return 0;
}
```

TEST CASE 1: VALID MAIN MENU INPUT

```
--- Hospital Management Menu ---
1. Patient Management
2. Appointment Management
3. Record Management
4. Concurrency Control
5. Check Deadlocks
0. Exit
Choose an option:1


=== Patient Management Menu ===
1. Register Patient
2. Update Patient
3. Remove Patient
4. List Patients
0. Back to Main Menu
Choose an option:
```

TEST CASE 2: INVALID MAIN MENU INPUT

```
--- Hospital Management Menu ---
1. Patient Management
2. Appointment Management
3. Record Management
4. Concurrency Control
5. Check Deadlocks
0. Exit
Choose an option:13

Invalid choice.

--- Hospital Management Menu ---
1. Patient Management
2. Appointment Management
3. Record Management
4. Concurrency Control
5. Check Deadlocks
0. Exit
Choose an option:
```

TEST CASE 3: VALID REGISTER PATIENT INPUT

```
=== Patient Management Menu ===
1. Register Patient
2. Update Patient
3. Remove Patient
4. List Patients
0. Back to Main Menu
Choose an option:1

Enter Name:John Wick

Enter Age:52

Patient registered with ID 1: John Wick

=== Patient Management Menu ===
1. Register Patient
2. Update Patient
3. Remove Patient
4. List Patients
0. Back to Main Menu
Choose an option:
```

TEST CASE 4: INVALID AGE INPUT IN REGISTER PATIENT

```
--- Hospital Management Menu ---
1. Patient Management
2. Appointment Management
3. Record Management
4. Concurrency Control
5. Check Deadlocks
0. Exit
Choose an option:1


=== Patient Management Menu ===
1. Register Patient
2. Update Patient
3. Remove Patient
4. List Patients
0. Back to Main Menu
Choose an option:1

Enter Name:Jan

Enter Age:awidhaihdw

Invalid. Please enter a valid age.
Enter Age:
```

TEST CASE 5: VALID UPDATE PATIENT DETAILS INPUT

```
=== Patient Management Menu ===
1. Register Patient
2. Update Patient
3. Remove Patient
4. List Patients
0. Back to Main Menu
Choose an option:2

Enter ID, New Name, New Age:1 Johnny 55

Patient updated: Johnny

=== Patient Management Menu ===
1. Register Patient
2. Update Patient
3. Remove Patient
4. List Patients
0. Back to Main Menu
Choose an option:
```

TEST CASE 6: INVALID UPDATE PATIENT DETAILS INPUT

```
=== Patient Management Menu ===
1. Register Patient
2. Update Patient
3. Remove Patient
4. List Patients
0. Back to Main Menu
Choose an option:1

Enter Name:JOHN WICK

Enter Age:54

Patient registered with ID 1: JOHN WICK

=== Patient Management Menu ===
1. Register Patient
2. Update Patient
3. Remove Patient
4. List Patients
0. Back to Main Menu
Choose an option:2

Enter ID:E

Invalid ID. Please enter a number:
```

TEST CASE 7: VALID REMOVE EXISTING PATIENT INPUT

```
=== Patient Management Menu ===
1. Register Patient
2. Update Patient
3. Remove Patient
4. List Patients
0. Back to Main Menu
Choose an option:3

Enter ID to remove:1

Patient removed.

=== Patient Management Menu ===
1. Register Patient
2. Update Patient
3. Remove Patient
4. List Patients
0. Back to Main Menu
Choose an option:
```

TEST CASE 8: INVALID REMOVE PATIENT INPUT

```
=== Patient Management Menu ===
1. Register Patient
2. Update Patient
3. Remove Patient
4. List Patients
0. Back to Main Menu
Choose an option:1

Enter Name:JOHN WICK

Enter Age:54

Patient registered with ID 1: JOHN WICK

=== Patient Management Menu ===
1. Register Patient
2. Update Patient
3. Remove Patient
4. List Patients
0. Back to Main Menu
Choose an option:3

Enter ID:PANCAKES

Invalid. Please enter a valid ID.
Enter ID:
```

TEST CASE 9: VIEW LIST OF PATIENTS

```
=== Patient Management Menu ===
1. Register Patient
2. Update Patient
3. Remove Patient
4. List Patients
0. Back to Main Menu
Choose an option:4

ID: 1, Name: JOHN WICK, Age: 54
ID: 2, Name: JAN, Age: 21

=== Patient Management Menu ===
1. Register Patient
2. Update Patient
3. Remove Patient
4. List Patients
0. Back to Main Menu
Choose an option:
```

TEST CASE 10: SCHEDULE A NEW APPOINTMENT

```
--- Appointment Management Menu ---
1. Schedule Appointment
2. Update Existing Appointment
3. Remove Existing Appointment
4. List Appointments
0. Back to Main Menu
Choose an option:1

Enter Patient ID:1

Enter Appointment Date:2025-06-13

Enter Reason:CHECKUP

Appointment scheduled with ID 1.

--- Appointment Management Menu ---
1. Schedule Appointment
2. Update Existing Appointment
3. Remove Existing Appointment
4. List Appointments
0. Back to Main Menu
Choose an option:
```

TEST CASE 11: UPDATE EXISTING APPOINTMENT

```
--- Appointment Management Menu ---
1. Schedule Appointment
2. Update Existing Appointment
3. Remove Existing Appointment
4. List Appointments
0. Back to Main Menu
Choose an option:2

Enter Appointment ID:1

Enter New Date:2025-05-31

Enter New Reason:OPERATION

Appointment updated.

--- Appointment Management Menu ---
1. Schedule Appointment
2. Update Existing Appointment
3. Remove Existing Appointment
4. List Appointments
0. Back to Main Menu
Choose an option:
```

TEST CASE 12: REMOVE SCHEDULED APPOINTMENT

```
--- Appointment Management Menu ---
1. Schedule Appointment
2. Update Existing Appointment
3. Remove Existing Appointment
4. List Appointments
0. Back to Main Menu
Choose an option:3

Enter Appointment ID to cancel:1

Appointment canceled.

--- Appointment Management Menu ---
1. Schedule Appointment
2. Update Existing Appointment
3. Remove Existing Appointment
4. List Appointments
0. Back to Main Menu
Choose an option:
```

TEST CASE 13: ADD NEW PATIENT RECORD

```
--- Recording Management Menu ---
1. Add Record
2. Update Record
3. View Records
0. Back to main menu.
Choose an option:1

Enter Patient ID:1

Enter Name:JOHN WICK

Enter Age:54

Record created for Patient ID 1.

--- Recording Management Menu ---
1. Add Record
2. Update Record
3. View Records
0. Back to main menu.
Choose an option:
```

TEST CASE 14: UPDATE EXISTING PATIENT RECORD

```
--- Recording Management Menu ---
1. Add Record
2. Update Record
3. View Records
0. Back to main menu.
Choose an option:2

Enter Patient ID:1

Enter new record entry (e.g., '2025-05-25: Follow-up for BP'):2025-07-01: BLOOD PRESSURE NORMAL

Medical record updated for Patient ID 1.

--- Recording Management Menu ---
1. Add Record
2. Update Record
3. View Records
0. Back to main menu.
Choose an option:
```

TEST CASE 15: VIEW EXISTING PATIENT RECORD

```
--- Recording Management Menu ---
1. Add Record
2. Update Record
3. View Records
0. Back to main menu.
Choose an option:3

Enter Patient ID:1

Record for Patient ID 1:
Name: JOHN WICK, Age: 54
Entries:
- 2025-07-01

--- Recording Management Menu ---
1. Add Record
2. Update Record
3. View Records
0. Back to main menu.
Choose an option:
```

```
--- Recording Management Menu ---
1. Add Record
2. Update Record
3. View Records
0. Back to main menu.
Choose an option:3

Enter Patient ID:1

Record for Patient ID 1:
Name: JOHN WICK, Age: 54
Entries:
- 2025-07-01: BLOOD PRESSURE NORMAL

--- Recording Management Menu ---
1. Add Record
2. Update Record
3. View Records
0. Back to main menu.
Choose an option:
```

TEST CASE 16: VIEW CONCURRENCY CONTROL

```
--- Hospital Management Menu ---
1. Patient Management
2. Appointment Management
3. Record Management
4. Concurrency Control
5. Check Deadlocks
0. Exit
Choose an option:4


--- Lock Status ---
Patient Lock: UNLOCKED
Appointment Lock: UNLOCKED
Record Lock: UNLOCKED

--- Hospital Management Menu ---
1. Patient Management
2. Appointment Management
3. Record Management
4. Concurrency Control
5. Check Deadlocks
0. Exit
Choose an option:
```

TEST CASE 17: CHECK FOR POSSIBLE DEADLOCKS

```
--- Hospital Management Menu ---
1. Patient Management
2. Appointment Management
3. Record Management
4. Concurrency Control
5. Check Deadlocks
0. Exit
Choose an option:5


--- Deadlock Check ---
No deadlocks detected.

--- Hospital Management Menu ---
1. Patient Management
2. Appointment Management
3. Record Management
4. Concurrency Control
5. Check Deadlocks
0. Exit
Choose an option:
```