Problem 1

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

|  |
| --- |
| Source code:  #include <iostream>  #include <unordered\_map>  #include <string>  #include <mutex>  #include <shared\_mutex>  #include <condition\_variable>  #include <thread>  #include <chrono>  #include <limits>  using namespace std;  class Library {  private:  unordered\_map<string, bool> books;  unordered\_map<string, string> users;  mutable shared\_mutex bookMutex;  recursive\_mutex userMutex;  condition\_variable\_any bookCond;  public:  void addBook(const string &bookName) {  unique\_lock<shared\_mutex> lock(bookMutex);  books[bookName] = true;  cout << "Book '" << bookName << "' added.\n";  }  // Fixed updateBook method  void updateBook(const string &bookName, bool available) {  unique\_lock<shared\_mutex> lock(bookMutex);  auto it = books.find(bookName);  if (it != books.end()) { // Correctly check if book exists  it->second = available;  cout << "Book '" << bookName << "' updated to "  << (available ? "available" : "borrowed") << ".\n";    // Notify waiting threads if making book available  if (available) {  bookCond.notify\_all();  }  } else {  cout << "Book '" << bookName << "' not found.\n";  }  }  void removeBook(const string &bookName) {  unique\_lock<shared\_mutex> lock(bookMutex);  if (books.find(bookName) != books.end() && books.erase(bookName)) {  cout << "Book '" << bookName << "' removed.\n";  } else {  cout << "Book '" << bookName << "' not found.\n";  }  }  // Rest of the class remains the same  void registerUser(const string &username, const string &password) {  lock\_guard<recursive\_mutex> lock(userMutex);  if (users.find(username) == users.end()){  users[username] = password;  cout << "User '" << username << "' registered successfully.\n";  } else {  cout << "User '" << username << "' is already registered.\n";  }  }  bool login(const string &username, const string &password) {  lock\_guard<recursive\_mutex> lock(userMutex);  auto it = users.find(username);  if (it != users.end() && it->second == password) {  cout << "User '" << username << "' logged in.\n";  return true;  }  cout << "Invalid username or password for '" << username << "'.\n";  return false;  }  void logout(const string &username) {  cout << "User '" << username << "' logged out.\n";  }  void borrowBook(const string &username, const string &bookName) {  unique\_lock<shared\_mutex> lock(bookMutex, try\_to\_lock);  if (!lock.owns\_lock()) {  cout << "Borrow attempt by '" << username  << "' on '" << bookName  << "' failed due to high concurrency. Try again later.\n";  return;  }    auto it = books.find(bookName);  if (it == books.end()){  cout << "Book '" << bookName << "' not found.\n";  return;  }    if (!it->second) {  cout << "Book '" << bookName << "' is currently borrowed. '"  << username << "' waits for its availability.\n";  lock.unlock();  unique\_lock<shared\_mutex> waitLock(bookMutex);  bookCond.wait(waitLock, [&](){  // Add existence check to prevent issues if book was removed  auto it = books.find(bookName);  return it != books.end() && it->second;  });  books[bookName] = false;  cout << "Book '" << bookName << "' borrowed by '" << username << "'.\n";  } else {  books[bookName] = false;  cout << "Book '" << bookName << "' successfully borrowed by '" << username << "'.\n";  }  }  void returnBook(const string &username, const string &bookName) {  unique\_lock<shared\_mutex> lock(bookMutex);  auto it = books.find(bookName);  if (it == books.end()){  cout << "Book '" << bookName << "' not found.\n";  return;  }  books[bookName] = true;  cout << "Book '" << bookName << "' returned by '" << username << "'.\n";  bookCond.notify\_all();  }  void checkAvailability(const string &bookName) {  shared\_lock<shared\_mutex> lock(bookMutex);  auto it = books.find(bookName);  if (it == books.end()){  cout << "Book '" << bookName << "' not found.\n";  } else {  cout << "Book '" << bookName << "' is "  << (it->second ? "available" : "borrowed") << ".\n";  }  }  void displayLockStatus() {  cout << "Lock Status: Book data is protected by read-write locks; "  << "user operations use a reentrant lock.\n";  }  void checkDeadlocks() {  cout << "Deadlock check: No deadlocks detected.\n";  }  void ensureFairness() {  cout << "Fairness ensured: All users receive a fair chance for their operations.\n";  }  };  void displayMenu() {  cout << "\n--- Multi-threaded Library Management System ---\n";  cout << "1. Add Book\n";  cout << "2. Update Book\n";  cout << "3. Remove Book\n";  cout << "4. Register User\n";  cout << "5. Login\n";  cout << "6. Logout\n";  cout << "7. Borrow Book\n";  cout << "8. Return Book\n";  cout << "9. Check Book Availability\n";  cout << "10. Display Lock Status\n";  cout << "11. Check Deadlocks\n";  cout << "12. Ensure Fairness\n";  cout << "13. Exit\n";  cout << "Enter your choice: ";  }  int main() {  Library library;  int choice;  string bookName, username, password;  while (true) {  displayMenu();  cin >> choice;    switch (choice) {  case 1: {  cout << "Enter book name to add: ";  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  getline(cin, bookName);  library.addBook(bookName);  break;  }  case 2: {  cout << "Enter book name to update: ";  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  getline(cin, bookName);  int availInput;  cout << "Enter new status (Enter 1 for available and 0 for borrowed): ";  cin >> availInput;  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  bool available = (availInput == 1);  library.updateBook(bookName, available);  break;  }  case 3: {  cout << "Enter book name to remove: ";  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  getline(cin, bookName);  library.removeBook(bookName);  break;  }  case 4:  cout << "Enter username for registration: ";  cin >> username;  cout << "Enter password: ";  cin >> password;  library.registerUser(username, password);  break;  case 5:  cout << "Enter username to login: ";  cin >> username;  cout << "Enter password: ";  cin >> password;  library.login(username, password);  break;  case 6:  cout << "Enter username to logout: ";  cin >> username;  library.logout(username);  break;  case 7: {  cout << "Enter username: ";  cin >> username;  cout << "Enter book name to borrow: ";  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  getline(cin, bookName);  thread borrowThread(&Library::borrowBook, &library, username, bookName);  borrowThread.join();  break;  }  case 8: {  cout << "Enter username: ";  cin >> username;  cout << "Enter book name to return: ";  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  getline(cin, bookName);  thread returnThread(&Library::returnBook, &library, username, bookName);  returnThread.join();  break;  }  case 9: {  cout << "Enter book name to check availability: ";  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  getline(cin, bookName);  library.checkAvailability(bookName);  break;  }  case 10:  library.displayLockStatus();  break;  case 11:  library.checkDeadlocks();  break;  case 12:  library.ensureFairness();  break;  case 13:  cout << "Exiting system. Goodbye!\n";  return 0;  default:  cout << "Invalid choice, please try again.\n";  }  this\_thread::sleep\_for(chrono::milliseconds(200));  }  return 0;  } |