#include <iostream>

#include <thread>

#include <mutex>

#include <condition\_variable>

#include <map>

#include <string>

#include <chrono>

using namespace std;

// Structs for Patient and Appointment

struct Patient {

int id;

string name;

string record;

};

struct Appointment {

int patientId;

string date;

string details;

};

// Global maps

map<int, Patient> patients;

map<int, Appointment> appointments;

// Mutexes

mutex patientMutex;

mutex recordMutex;

mutex appointmentMutex;

mutex cvMutex;

// Condition variable

condition\_variable appointmentCV;

bool appointmentNotification = false;

// Lock status flags

bool isPatientLocked = false;

bool isRecordLocked = false;

bool isAppointmentLocked = false;

// Operation count for fairness

int opCount[12] = {0};

// ID generator

int generateId() {

static int id = 0;

return ++id;

}

// Register new patient

void registerPatient() {

opCount[1]++;

isPatientLocked = true;

lock\_guard<mutex> lock(patientMutex);

Patient p;

p.id = generateId();

cout << "Enter patient name: ";

cin.ignore();

getline(cin, p.name);

p.record = "No records yet.";

patients[p.id] = p;

cout << "Patient registered with ID: " << p.id << endl;

isPatientLocked = false;

}

// Update patient

void updatePatient() {

opCount[2]++;

isPatientLocked = true;

lock\_guard<mutex> lock(patientMutex);

int id;

cout << "Enter patient ID to update: ";

cin >> id;

if (patients.find(id) != patients.end()) {

cout << "Enter new name: ";

cin.ignore();

getline(cin, patients[id].name);

cout << "Updated successfully.\n";

} else {

cout << "Patient not found.\n";

}

isPatientLocked = false;

}

// Remove patient

void removePatient() {

opCount[3]++;

isPatientLocked = true;

lock\_guard<mutex> lock(patientMutex);

int id;

cout << "Enter patient ID to remove: ";

cin >> id;

if (patients.erase(id)) {

appointments.erase(id);

cout << "Patient removed.\n";

} else {

cout << "Patient not found.\n";

}

isPatientLocked = false;

}

// Schedule appointment

void scheduleAppointment() {

opCount[4]++;

if (appointmentMutex.try\_lock()) {

isAppointmentLocked = true;

int id;

Appointment a;

cout << "Enter patient ID: ";

cin >> id;

if (patients.find(id) != patients.end()) {

a.patientId = id;

cout << "Enter date: ";

cin.ignore();

getline(cin, a.date);

cout << "Enter details: ";

getline(cin, a.details);

appointments[id] = a;

{

lock\_guard<mutex> lk(cvMutex);

appointmentNotification = true;

}

appointmentCV.notify\_all();

cout << "Appointment scheduled.\n";

} else {

cout << "Patient not found.\n";

}

appointmentMutex.unlock();

isAppointmentLocked = false;

} else {

cout << "Another appointment is being scheduled. Please wait.\n";

}

}

// Update appointment

void updateAppointment() {

opCount[5]++;

isAppointmentLocked = true;

lock\_guard<mutex> lock(appointmentMutex);

int id;

cout << "Enter patient ID: ";

cin >> id;

if (appointments.find(id) != appointments.end()) {

cout << "Enter new date: ";

cin.ignore();

getline(cin, appointments[id].date);

cout << "Enter new details: ";

getline(cin, appointments[id].details);

cout << "Appointment updated.\n";

} else {

cout << "Appointment not found.\n";

}

isAppointmentLocked = false;

}

// Cancel appointment

void cancelAppointment() {

opCount[6]++;

isAppointmentLocked = true;

lock\_guard<mutex> lock(appointmentMutex);

int id;

cout << "Enter patient ID to cancel appointment: ";

cin >> id;

if (appointments.erase(id)) {

cout << "Appointment canceled.\n";

} else {

cout << "Appointment not found.\n";

}

isAppointmentLocked = false;

}

// Update record

void updateRecord() {

opCount[7]++;

isRecordLocked = true;

lock\_guard<mutex> lock(recordMutex);

int id;

cout << "Enter patient ID to update record: ";

cin >> id;

if (patients.find(id) != patients.end()) {

cout << "Enter new record: ";

cin.ignore();

getline(cin, patients[id].record);

cout << "Record updated.\n";

} else {

cout << "Patient not found.\n";

}

isRecordLocked = false;

}

// View record

void viewRecord() {

opCount[8]++;

isRecordLocked = true;

lock\_guard<mutex> lock(recordMutex);

int id;

cout << "Enter patient ID to view record: ";

cin >> id;

if (patients.find(id) != patients.end()) {

Patient& p = patients[id];

cout << "\n--- Patient Record ---\n";

cout << "ID : " << p.id << endl;

cout << "Name : " << p.name << endl;

cout << "Record : " << p.record << endl;

if (appointments.find(id) != appointments.end()) {

Appointment& appt = appointments[id];

cout << "Pending Appointment: YES\n";

cout << "Date: " << appt.date << endl;

cout << "Details: " << appt.details << endl;

} else {

cout << "Pending Appointment: NO\n";

}

} else {

cout << "Patient not found.\n";

}

isRecordLocked = false;

}

// Display lock status

void displayLockStatus() {

opCount[9]++;

cout << "\n--- Lock Status ---\n";

cout << "Patient Data Lock : " << (isPatientLocked ? "LOCKED" : "FREE") << endl;

cout << "Record Data Lock : " << (isRecordLocked ? "LOCKED" : "FREE") << endl;

cout << "Appointment Lock : " << (isAppointmentLocked ? "LOCKED" : "FREE") << endl;

}

// Deadlock check

void checkDeadlocks() {

opCount[10]++;

cout << "\n--- Deadlock Check ---\n";

if (isPatientLocked && isRecordLocked && isAppointmentLocked) {

cout << "\u26A0\uFE0F Potential deadlock: all resources are locked!\n";

} else {

cout << "No deadlocks detected.\n";

}

}

// Fairness report

void ensureFairness() {

opCount[11]++;

cout << "\n--- Operation Fairness Report ---\n";

for (int i = 1; i <= 11; i++) {

cout << "Option " << i << ": " << opCount[i] << " times\n";

}

}

// Background notification thread

void notifyAppointments() {

while (true) {

unique\_lock<mutex> lk(cvMutex);

appointmentCV.wait(lk, []() { return appointmentNotification; });

cout << "[Notification] New appointment scheduled.\n";

appointmentNotification = false;

}

}

// Menu display

void showMenu() {

cout << "\n--- Hospital Management System ---\n";

cout << "1. Register Patient\n";

cout << "2. Update Patient\n";

cout << "3. Remove Patient\n";

cout << "4. Schedule Appointment\n";

cout << "5. Update Appointment\n";

cout << "6. Cancel Appointment\n";

cout << "7. Update Record\n";

cout << "8. View Record\n";

cout << "9. Display Lock Status\n";

cout << "10. Check Deadlocks\n";

cout << "11. Ensure Fairness\n";

cout << "0. Exit\n";

cout << "Choose an option: ";

}

// Main loop

int main() {

thread notifier(notifyAppointments);

notifier.detach();

while (true) {

showMenu();

int choice;

cin >> choice;

switch (choice) {

case 1: registerPatient(); break;

case 2: updatePatient(); break;

case 3: removePatient(); break;

case 4: scheduleAppointment(); break;

case 5: updateAppointment(); break;

case 6: cancelAppointment(); break;

case 7: updateRecord(); break;

case 8: viewRecord(); break;

case 9: displayLockStatus(); break;

case 10: checkDeadlocks(); break;

case 11: ensureFairness(); break;

case 0:

cout << "Exiting system...\n";

return 0;

default:

cout << "Invalid choice.\n";

}

this\_thread::sleep\_for(chrono::milliseconds(300));

}

return 0;

}

























