Course: Design and Analysis of Algorithms

Section: 15

Professor: Hasan Mahmood Aminul Islam (HMI)

Project Title: NSU RDS

Group Name: NSUers the Great

Group Members:

- Md. Tanvir Islam Shikdar 2021543042
- Kawsar Hossain 2132211642

Project Overview

The North South University Management System is designed to manage the records of students, faculty members, and courses efficiently. This system leverages modern C++ features and efficient data structures & algorithms to ensure quick access, modification, and maintenance of records. The header file `university_management.h` defines the core components of this system, providing a robust and scalable solution.

Key Features

1. **Student Management**: Handles the addition, enrollment, and retrieval of student records.
2. **Faculty Management**: Manages faculty member records, including course assignments.
3. **Course Management**: Manages course records, including the list of enrolled students and assigned faculty members.

Explanation of Chosen Data Structures and Algorithms

- **Hash Tables (std::unordered_map)**:
  - **Usage**: For storing and retrieving records of students, faculty members, and courses.
  - **Complexity**: Average O(1) for insertions, deletions, and look-ups.
  - **Reason**: Ensures efficient management of large datasets, providing fast access to records.

- **Sets (std::unordered_set)**:
  - **Usage**: For managing course enrollments and faculty course assignments.
  - **Complexity**: Average O(1) for insertions, deletions, and look-ups.
  - **Reason**: Guarantees uniqueness of entries and fast access.

- **Smart Pointers (std::shared_ptr)**:
  - **Usage**: For dynamic memory management of records.
  - **Reason**: Automatically manages memory, preventing leaks and simplifying resource management.

- **Concurrency (std::shared_mutex)**:
  - **Usage**: For thread-safe access to shared data structures.
  - **Reason**: Allows multiple threads to read data simultaneously while ensuring exclusive access for write operations, improving performance in multi-threaded environments.

Detailed Description of `university_management.h`

The header file `university_management.h` defines the structures and classes necessary for managing the university's records. Below is a breakdown of its key components:

Student Structure:

```
struct Student {
  int student_id;              ///< Unique identifier for the student
  std::string name;            ///< Name of the student
  std::unordered_set<int> courses; ///< Set of course IDs the student is enrolled in
};
```

Faculty Structure:

```
struct Faculty {
  int faculty_id;              ///< Unique identifier for the faculty member
  std::string name;            ///< Name of the faculty member
  std::unordered_set<int> courses; ///< Set of course IDs the faculty member is
teaching
};
```

Course Structure:

```
struct Course {
  int course_id;               ///< Unique identifier for the course
  std::string name;            ///< Name of the course
  int faculty_id;              ///< Faculty member ID who teaches the course
  std::unordered_set<int> students; ///< Set of student IDs enrolled in the course
};
```

StudentManager Class:

```
class StudentManager {
public:
  void addStudent(int student_id, const std::string &name);
  void enrollInCourse(int student_id, int course_id);
  std::unordered_set<int> getStudentCourses(int student_id) const;
private:
  std::unordered_map<int, std::shared_ptr<Student>> student_records; ///< Hash
table for student records
  mutable std::shared_mutex mtx; ///< Shared mutex for thread safety
};
```

FacultyManager Class:

```
class FacultyManager {
public:
  void addFaculty(int faculty_id, const std::string &name);
  void assignCourse(int faculty_id, int course_id);
  std::unordered_set<int> getFacultyCourses(int faculty_id) const;
private:
  std::unordered_map<int, std::shared_ptr<Faculty>> faculty_records; ///< Hash
table for faculty records
  mutable std::shared_mutex mtx; ///< Shared mutex for thread safety
};
```

CourseManager Class:

```
class CourseManager {
public:
  void addCourse(int course_id, const std::string &name, int faculty_id);
  void enrollStudent(int course_id, int student_id);
  std::unordered_set<int> getCourseStudents(int course_id) const;
private:
  std::unordered_map<int, std::shared_ptr<Course>> course_records; ///< Hash
table for course records
  mutable std::shared_mutex mtx; ///< Shared mutex for thread safety
};
```

UniversityManager Class:

```
class UniversityManager {
public:
  void addStudent(int student_id, const std::string &name);
  void enrollInCourse(int student_id, int course_id);
  std::unordered_set<int> getStudentCourses(int student_id) const;
```

```
    void addFaculty(int faculty_id, const std::string &name);
    void assignCourse(int faculty_id, int course_id);
    std::unordered_set<int> getFacultyCourses(int faculty_id) const;

    void addCourse(int course_id, const std::string &name, int faculty_id);
    std::unordered_set<int> getCourseStudents(int course_id) const;
private:
    StudentManager student_manager; ///< Manager for student records
    FacultyManager faculty_manager; ///< Manager for faculty records
    CourseManager course_manager;   ///< Manager for course records
};
```

Conclusion

The North South University Management System is designed to efficiently manage the records of students, faculty, and courses using modern C++ features and data structures. The use of `std::unordered_map`, `std::unordered_set`, smart pointers, and shared mutexes ensures that the system is both efficient and thread-safe. This design allows the system to handle the complex relationships and large datasets typical of a university environment effectively.

By leveraging these advanced data structures and algorithms, the University Management System provides a robust, scalable, and maintainable solution for managing university records.