Course Registration Portal (Academia) Report

SYED NAVEED MOHAMMED
IMT2023119

1. Problem Statement

The problem addressed by this project is the development of a multi-user, client-server application for a course registration system. The system manages information for three distinct user roles: Administrator, Faculty, and Students. Each role has specific functionalities:

- **Administrator:** Manage student and faculty accounts (add, view, activate, block, modify).
- **Faculty:** Manage the courses they offer (view, add, remove, update details) and change their password.
- **Students:** Manage their course enrollments (view all courses, enroll, drop, view enrolled courses) and change their password.

The system handles concurrent access to data files securely and provides a reliable communication channel between the client and the server.

2. Implementation Details

The system is implemented using a client-server architecture in C.

- **Server (server.c):**
  - Acts as the central hub, listening for incoming client connections on a defined port (PORT).
  - Uses multi-threading (pthread) to handle multiple clients concurrently. Each client connection is handled in a separate thread.
  - Implements a signal handler (signal_handler) to gracefully shut down the server upon receiving a SIGINT signal (e.g., Ctrl+C).
  - Utilizes binary semaphores (semaphore.h) to protect shared data files (students.dat, faculty.dat, courses.dat) from race conditions during read and write operations. Separate semaphores are used for each data file.

- Authenticates users based on their role (Admin, Faculty, Student) and credentials using the authenticate function from utils.c.
- Directs authenticated users to their respective menu functions (admin_menu, faculty_menu, student_menu).

- **Client (Client.c):**
  - Connects to the server using TCP sockets.
  - Provides a command-line interface for user interaction.
  - Sends user input to the server and receives responses.
  - Implements a custom messaging protocol using markers ("END", "EXPECT", "EXIT") to delineate messages received from the server, handling potentially fragmented messages.
  - Includes a signal handler (handle_sigint) for graceful client disconnection.

- **User Role Modules (admin.c, faculty.c, student.c):**
  - These files contain the specific menu and functionality implementations for each user role.
  - They interact with the data files (.dat) to perform operations relevant to their role (e.g., adding/viewing users, managing courses, enrolling/dropping courses).
  - File operations within these modules use system calls like open, read, write, close, unlink, and rename. Temporary files are used for modifying records to ensure data integrity.
  - Access to data files is guarded by the semaphores defined and initialized in server.c.

- **Utility Functions (utils.c, utils.h):**
  - Provide common functions used across different modules.
  - Includes the authenticate function for user login.
  - Includes the change_password function for users to update their credentials.
  - Includes functions for sending and receiving messages over the socket (send_message, send_message_expect, recv_message), incorporating the custom message markers.
  - Includes functions for exiting the client (exit_client, exit_client_logged_in).

- **Common Definitions (common.h):**
  - Defines constants used throughout the project (e.g., PORT, BUFFER_SIZE, MAX_CLIENTS, MAX_COURSES, MAX_SEATS).
  - Defines the structures for Student, Faculty, and Course data.

- Declares the global semaphores as extern.
- Declares the main menu function prototypes for use in server.c.

Data is persisted in binary files (students.dat, faculty.dat, courses.dat). Each record in these files corresponds to a structure defined in common.h. File is locked using semaphores for preventing data corruption when multiple threads access the same file. Additionally, before writing to the actual file, a temporary file is created which is written to and if any checks fail then it is deleted ensuring atomicity of the transaction (writing).

3. Source Code (Function prototypes and explanations)

1) Server.c

```
void handle_client(int client_socket);
void signal_handler(int sig);
int main();
```

The `main` function is the entry point of the server application. It initializes the semaphores used for controlling access to data files, sets up the network socket for listening for incoming connections, binds the socket to the specified port, and starts listening. It then enters a loop to accept new client connections and creates a new thread (`handle_client`) for each connected client to manage their interactions concurrently.

The `signal_handler` function is designed to catch the SIGINT signal (typically generated by pressing Ctrl+C). When this signal is received, the handler cleans up the semaphores before the server exits gracefully.

The `handle_client` function is executed in a separate thread for each connected client. It manages the communication with that specific client, including presenting the initial login menu, authenticating the user based on their chosen role, and then directing them to the appropriate menu function (admin, faculty, or student) based on successful authentication. It continues to handle the client's requests within their respective menu until the client logs out or disconnects.

2) Client.c

```c
void handle_sigint(int sig);
int recv_message(int socket, char *buffer);
void send_message(int socket, const char *message);
int main(int argc, char *argv[]);
```

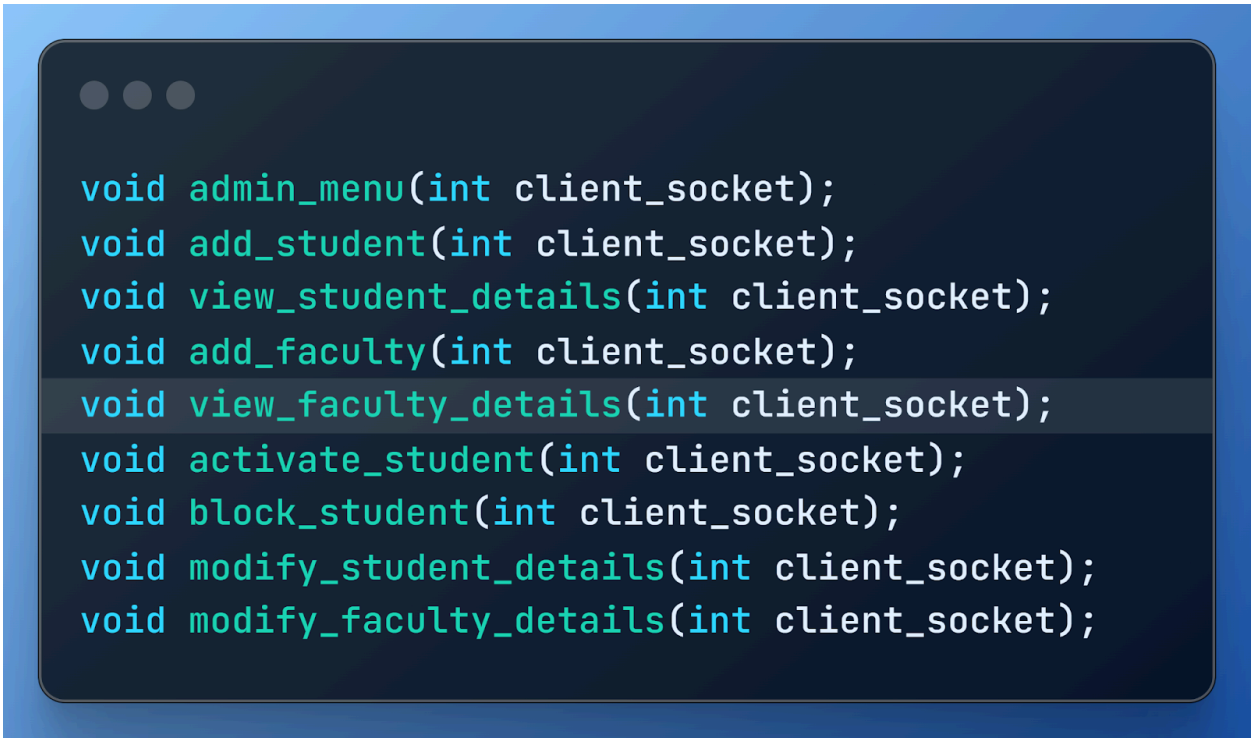
`handle_sigint`: A function to handle the interrupt signal (like Ctrl+C) to ensure a clean disconnection from the server.

`recv_message`: A function that receives data from the server, handling potentially fragmented messages by buffering and looking for specific markers ("END", "EXPECT", "EXIT") to determine the end and type of the message.

`send_message`: A simple function to send messages to the server, appending an "END" marker.

`main`: The primary function that sets up the client socket, connects to the server (allowing for an optional IP address argument), and then enters a loop to continuously receive messages from the server, display them, and send user input back to the server until a disconnection occurs.

3) admin.c



```c
void admin_menu(int client_socket);
void add_student(int client_socket);
void view_student_details(int client_socket);
void add_faculty(int client_socket);
void view_faculty_details(int client_socket);
void activate_student(int client_socket);
void block_student(int client_socket);
void modify_student_details(int client_socket);
void modify_faculty_details(int client_socket);
```

`admin_menu(int client_socket)`: This function presents the main menu options to the administrator client connected via `client_socket`. It receives the administrator's choice and calls the corresponding function to perform the requested action. It loops until the administrator chooses to log out and exit.

`add_student(int client_socket)`: This function handles the process of adding a new student to the system. It prompts the administrator for the student's name and password, checks for duplicate usernames, assigns a unique ID, and saves the student's information to the `students.dat` file, using a semaphore to ensure exclusive access during the write operation.

`view_student_details(int client_socket)`: This function retrieves and displays the details of all students from the `students.dat` file to the administrator. It shows the student ID, name, and their active/blocked status. A semaphore is used to protect the file during reading.

`add_faculty(int client_socket)`: Similar to `add_student`, this function handles adding a new faculty member. It prompts for the faculty's name and password, checks for duplicates, assigns an ID, and saves the information to `faculty.dat`, using a semaphore for file locking.

`view_faculty_details(int client_socket)`: This function retrieves and displays the details of all faculty members from the `faculty.dat` file to the administrator, showing their ID and name. It uses a semaphore for file access.

`activate_student(int client_socket)`: This function allows the administrator to change a student's status to 'active'. It prompts for the student ID, finds the student record in `students.dat`, updates their `active` status to 1, and saves the modified record. It uses a temporary file and rename operation for safe updating and is protected by a semaphore.

`block_student(int client_socket)`: This function allows the administrator to change a student's status to 'blocked'. It prompts for the student ID, finds the record, updates their `active` status to 0, and saves the modified record using a temporary file and rename, protected by a semaphore.

`modify_student_details(int client_socket)`: This function enables the administrator to modify a student's name and/or password. It prompts for the student ID and then the new name and password (allowing fields to be left blank to keep current values). It updates the student record in `students.dat` using a temporary file and rename, with semaphore protection. It also checks for duplicate usernames if the name is being changed.

`modify_faculty_details(int client_socket)`: Similar to `modify_student_details`, this function allows the administrator to modify a faculty member's name and/or password in the `faculty.dat` file, using a temporary file, rename, and semaphore protection. It also checks for duplicate names if the name is changed.

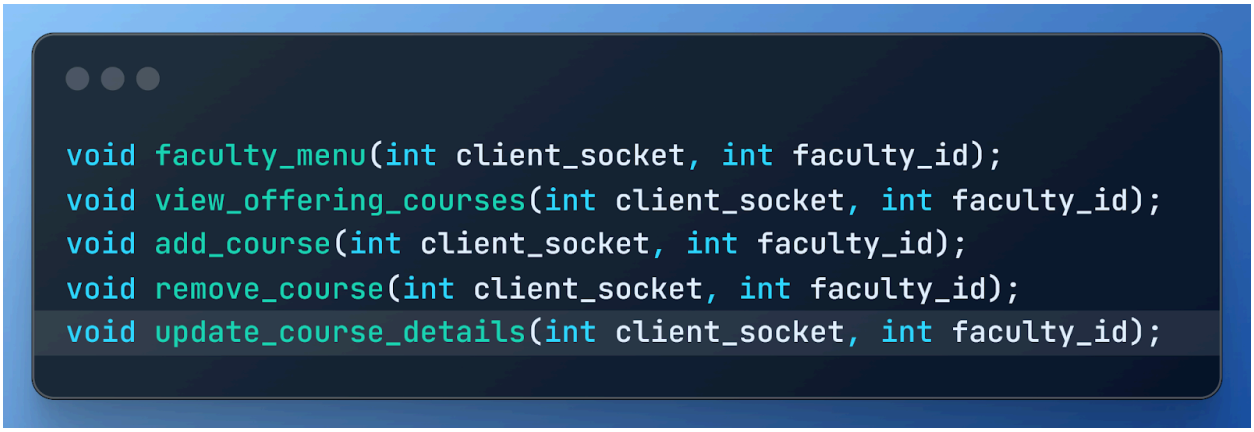
4) common.h

```c
#ifndef COMMON_H
#define COMMON_H
#include <semaphore.h>
#define PORT 8080
#define MAX_CLIENTS 100
#define BUFFER_SIZE 1024
#define MAX_COURSES 50
#define MAX_SEATS 10
typedef struct {
    int id;
    char name[50];
    char password[50];
    int active; // 1 for active, 0 for inactive
} Student;

typedef struct {
    int id;
    char name[50];
    char password[50];
} Faculty;
typedef struct {
    int id;
    char name[100];
    int faculty_id;
    int total_seats;
    int available_seats;
    int enrolled_students[MAX_SEATS];
    int enrollment_count;
} Course;
extern sem_t student_sem, faculty_sem, course_sem;
void admin_menu(int client_socket);
void faculty_menu(int client_socket, int faculty_id);
void student_menu(int client_socket, int student_id);
#endif
```

This file defines the structs, semaphores and constants as macros which are going to be used across all the files and is a header included in them.

5) faculty.c



```
void faculty_menu(int client_socket, int faculty_id);
void view_offering_courses(int client_socket, int faculty_id);
void add_course(int client_socket, int faculty_id);
void remove_course(int client_socket, int faculty_id);
void update_course_details(int client_socket, int faculty_id);
```

`faculty_menu(int client_socket, int faculty_id)`: This function serves as the main menu interface for a logged-in faculty member. It displays a list of options related to course management and password change. It receives the faculty member's choice and calls the appropriate function to handle the selected action. The menu is displayed repeatedly until the faculty member chooses to log out.

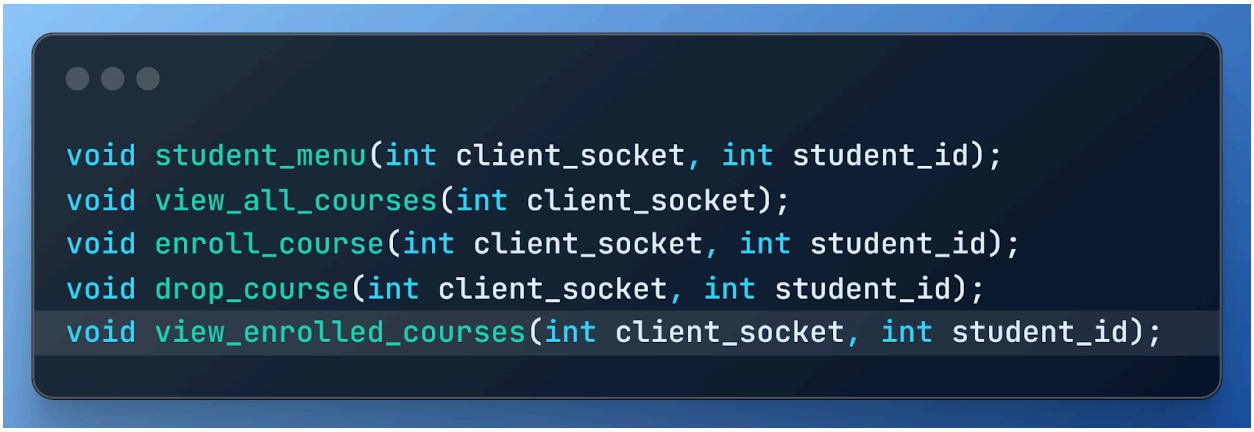
`view_offering_courses(int client_socket, int faculty_id)`: This function retrieves and sends a list of all courses currently being offered by the faculty member identified by `faculty_id` to the connected client. It reads course data from the `courses.dat` file, filters for courses taught by this faculty member, and formats the output to show course ID, name, and enrollment details. It uses a semaphore (`course_sem`) to control access to the course data file.

`add_course(int client_socket, int faculty_id)`: This function allows a faculty member to add a new course to the system. It prompts the faculty member for the course name and total seats. It then creates a new `Course` record with a unique ID, associates it with the faculty member's ID, initializes enrollment details, and writes the new course record to the `courses.dat` file. A semaphore is used to ensure safe writing to the file.

`remove_course(int client_socket, int faculty_id)`: This function enables a faculty member to remove one of their offered courses from the catalog. It first displays the faculty's courses using `view_offering_courses` and then prompts for the course ID to remove. It checks if the faculty member is authorized to remove the course and if there are any enrolled students (courses with enrolled students cannot be removed). If eligible, it creates a temporary file, copies all courses *except* the one to be removed, and then replaces the original `courses.dat` file with the temporary one, all while using a semaphore for file access control.

`update_course_details(int client_socket, int faculty_id)`: This function allows a faculty member to modify the details of a course they offer. It first displays the faculty's courses and prompts for the course ID to update. It then prompts for a new course name and/or total seats (allowing the user to leave fields blank to keep the current value). It updates the corresponding course record in the `courses.dat` file using a temporary file and rename operation, ensuring data integrity with a semaphore. It also includes a check to prevent setting the total seats below the current enrollment count**.**

6) student.c



```
void student_menu(int client_socket, int student_id);
void view_all_courses(int client_socket);
void enroll_course(int client_socket, int student_id);
void drop_course(int client_socket, int student_id);
void view_enrolled_courses(int client_socket, int student_id);
```

`student_menu(int client_socket, int student_id)`: This function displays the main menu options specifically for a logged-in student. It prompts the student to choose from actions like viewing all courses, enrolling in a new course, dropping a course, viewing their enrolled courses, changing their password, or logging out. Based on the student's input, it calls the corresponding function to perform the requested action.

`view_all_courses(int client_socket)`: This function retrieves and sends a list of all available courses from the `courses.dat` file to the connected student client. It displays the course ID, name, and the number of available seats for each course. It uses a semaphore (`course_sem`) to ensure safe reading of the course data.

`enroll_course(int client_socket, int student_id)`: This function allows a student to enroll in a course. It first displays all available courses using `view_all_courses` and then prompts the student for the ID of the course they want to enroll in. It checks if the course exists, if the student is already enrolled, and if there are available seats. If the enrollment is valid, it updates the course record in `courses.dat` by adding the student's ID to the enrolled students list, incrementing the enrollment count, and decrementing the available seats. It uses a temporary file and rename operation, protected by a semaphore, for safe updating.

`drop_course(int client_socket, int student_id)`: This function allows a student to drop an enrolled course. It first displays the courses the student is currently enrolled in and then prompts for the course ID to drop. It verifies that the student is indeed enrolled in the specified course. If so, it updates the course record in `courses.dat` by removing the student's ID from the enrolled students list, decrementing the enrollment count, and incrementing the available seats. This update is done using a temporary file and rename operation, protected by a semaphore.

`view_enrolled_courses(int client_socket, int student_id)`: This function retrieves and sends a list of all courses that the student identified by `student_id` is currently enrolled in to the connected client. It reads through the `courses.dat` file and checks the enrolled students list for each course to find matches for the student's ID. It then displays the ID, name, and the faculty ID for each enrolled course. A semaphore is used for safe access to the course data.

7) utils.c

```c
int authenticate(int client_socket, int role);
void change_password(int client_socket, int id, int role);
void exit_client(int client_socket);
void exit_client_logged_in(int client_socket); // Note: Missing parameter type in prototype
void send_message_expect(int socket, const char *message);
void send_message(int socket, const char *message);
int recv_message(int socket, char *buffer);
```

`authenticate(int client_socket, int role)`: This function handles the user authentication process. It takes the client socket and the user's chosen role (Admin, Faculty, or Student) as input. It prompts the client for a username and password, reads the corresponding data file (`students.dat` or `faculty.dat`, or checks hardcoded admin credentials), verifies the credentials, and returns the authenticated user's ID (or 1 for admin) if successful, or -1 if authentication fails or the student account is deactivated. It uses semaphores to protect access to the data files.

`change_password(int client_socket, int id, int role)`: This function allows a logged-in user (Faculty or Student) to change their password. It takes the client socket, the user's ID, and their role as input. It prompts for the current password and the new password. It then reads the appropriate data file, verifies the current password, updates the password in the user's record, and saves the changes using a temporary file and rename operation for safety. Semaphores are used to protect the data files during this process.

`exit_client(int client_socket)`: This function sends a final "Goodbye" message including an "EXIT" marker to the client before the client disconnects completely from the initial login menu.

`exit_client_logged_in(client_socket)`: This function sends a "Goodbye" message to a client who is logging out from one of the role-specific menus (Admin, Faculty, or Student).

`send_message_expect(int socket, const char *message)`: This function sends a message to the client and appends an "EXPECT" marker. This marker signals to the receiving client that it should expect user input after displaying the message.

`send_message(int socket, const char *message)`: This function sends a general message to the client and appends an "END" marker. This marker indicates the end of a standard message that does not require immediate user input.

`recv_message(int socket, char *buffer)`: This function receives data from the client socket. It is designed to handle messages that might be split across multiple network packets by using a static buffer. It reads data until it encounters one of the defined markers ("END", "EXPECT", "EXIT") to determine the end and type of the received message. It returns a value indicating which marker was found (0 for "END", 1 for "EXPECT", 2 for "EXIT"), or -1 in case of an error.

3. Output Screenshots

1) server.c

```
naveed09@LAPTOP-IVOT94I5:/mnt/c/Users/SYED NAVEED/Downloads/IMT2023119_Mini_Project/Final$ make
gcc -Wall -Wextra -pthread -g -c server.c -o server.o
gcc -Wall -Wextra -pthread -g -c utils.c -o utils.o
gcc -Wall -Wextra -pthread -g -c admin.c -o admin.o
gcc -Wall -Wextra -pthread -g -c faculty.c -o faculty.o
gcc -Wall -Wextra -pthread -g -c student.c -o student.o
gcc -Wall -Wextra -pthread -g server.o utils.o admin.o faculty.o student.o -o server
gcc -Wall -Wextra -pthread -g -c Client.c -o Client.o
gcc -Wall -Wextra -pthread -g Client.o -o client
touch students.dat faculty.dat courses.dat
naveed09@LAPTOP-IVOT94I5:/mnt/c/Users/SYED NAVEED/Downloads/IMT2023119_Mini_Project/Final$ ./server
Server started. Waiting for connections...
```

2) Client.c

```
gcc -Wall -Wextra -pthread -g -c student.c -o student.o
gcc -Wall -Wextra -pthread -g server.o utils.o admin.o fa
culty.o student.o -o server
gcc -Wall -Wextra -pthread -g -c Client.c -o Client.o
gcc -Wall -Wextra -pthread -g Client.o -o client
touch students.dat faculty.dat courses.dat
naveed09@LAPTOP-IVOT94I5:/mnt/c/Users/SYED NAVEED/Downloa
ds/IMT2023119_Mini_Project/Final$ ./server
Server started. Waiting for connections...
New client connected: 127.0.0.1:33222

naveed09@LAPTOP-IVOT94I5:/mnt/c/Users/SYED NAVEED/Downloa
ds/IMT2023119_Mini_Project/Final$ ./server
Server started. Waiting for connections...
New client connected: 127.0.0.1:43044
```

```
naveed09@LAPTOP-IVOT94I5:/mnt/c/Users/SYED NAVEED/Downloa
ds/IMT2023119_Mini_Project/Final$ ./client
Connecting to server...
Connected to server.


.....Welcome Back to Academia :: Course Re
gistration.....
Login Type
Enter Your Choice { 1. Admin , 2. Professor, 3. Student,
4. Exit } :
```

3) admin.c

```
...................Welcome Back to Academia :: Course Re
gistration...................
Login Type
Enter Your Choice { 1. Admin , 2. Professor, 3. Student,
4. Exit } : 1
Enter username: admin
Enter password: admin123
Authentication successful!

........ Welcome to Admin Menu ........
1. Add Student
2. View Student Details
3. Add Faculty
4. View Faculty Details
5. Activate Student
6. Block Student
7. Modify Student details
8. Modify Faculty Details
9. Logout and Exit
Enter your choice:
```

4) student.c

```
Login Type
Enter Your Choice { 1. Admin , 2. P
Enter username: syed
Enter password: naveed
Authentication successful!

--- Student Menu ---
1. View All Courses
2. Enroll (pick) new course
3. Drop Course
4. View Enrolled Course Details
5. Change Password
6. Logout and Exit

Enter your choice:
```

5) Adding new student

```
........ Welcome to Admin Menu ........
1. Add Student
2. View Student Details
3. Add Faculty
4. View Faculty Details
5. Activate Student
6. Block Student
7. Modify Student details
8. Modify Faculty Details
9. Logout and Exit
Enter your choice: 1

Enter student name: syed
Enter student password: naveed
Student added successfully! ID: 1
```

6) **Faculty login**

```
.....................Welcome Back to Academia :: Course Registration....
Login Type
Enter Your Choice { 1. Admin , 2. Professor, 3. Student, 4. Exit } : 2
Enter username: naveed
Enter password: naveed
Authentication successful!

........ Welcome to Faculty Menu ........
1. View Offering Courses
2. Add New Course
3. Remove Course from Catalog
4. Update Course Details
5. Change Password
6. Logout and Exit
Enter your choice:
```

7) **Adding new courses**

```
........ Welcome to Faculty Menu ........
1. View Offering Courses
2. Add New Course
3. Remove Course from Catalog
4. Update Course Details
5. Change Password
6. Logout and Exit
Enter your choice: 2
Enter course name: dsa
Enter total seats: 100
Course added successfully! ID: 1
```

8) **Enrolling in courses**

```
--- Student Menu ---
1. View All Courses
2. Enroll (pick) new course
3. Drop Course
4. View Enrolled Course Details
5. Change Password
6. Logout and Exit

Enter your choice: 2
Available courses:
ID: 1, Name: dsa, Available Seats: 100
Enter course ID to enroll (0 to cancel): 1
Enrolled successfully!
```

9) **View offered courses (student enrolled above)**

```
........ Welcome to Faculty Menu ........
1. View Offering Courses
2. Add New Course
3. Remove Course from Catalog
4. Update Course Details
5. Change Password
6. Logout and Exit
Enter your choice: 1
Your offered courses:
ID: 1, Name: dsa, Enrolled: 1/100
```

10) **Drop courses**

```
--- Student Menu ---
1. View All Courses
2. Enroll (pick) new course
3. Drop Course
4. View Enrolled Course Details
5. Change Password
6. Logout and Exit

Enter your choice: 3
Your enrolled courses:
ID: 1, Name: dsa
Enter course ID to drop (0 to cancel): 1
Course dropped successfully!
```

11) Remove course from catalog

```
........ Welcome to Faculty Menu ........
1. View Offering Courses
2. Add New Course
3. Remove Course from Catalog
4. Update Course Details
5. Change Password
6. Logout and Exit
Enter your choice: 3
Your offered courses:
ID: 1, Name: dsa, Enrolled: 0/100
ID: 2, Name: daa, Enrolled: 0/20
Enter course ID to remove (0 to cancel): 2
Course removed successfully.

........ Welcome to Faculty Menu ........
1. View Offering Courses
2. Add New Course
3. Remove Course from Catalog
4. Update Course Details
5. Change Password
6. Logout and Exit
Enter your choice: 1
Your offered courses:
ID: 1, Name: dsa, Enrolled: 0/100
```

12) Update course details

```
........ Welcome to Faculty Menu ........
1. View Offering Courses
2. Add New Course
3. Remove Course from Catalog
4. Update Course Details
5. Change Password
6. Logout and Exit
Enter your choice: 4
Your offered courses:
ID: 1, Name: dsa, Enrolled: 0/100
Enter course ID to update (0 to cancel): 1
Current course details:
Name: dsa
Total Seats: 100
Available Seats: 100
Enter new course name (or press Enter to keep current): daa
Enter new total seats (or 0 to keep current):
Course updated successfully.
```

13) **Block student**

```
........ Welcome to Admin Menu ........
1. Add Student
2. View Student Details
3. Add Faculty
4. View Faculty Details
5. Activate Student
6. Block Student
7. Modify Student details
8. Modify Faculty Details
9. Logout and Exit
Enter your choice: 6

Enter student ID to block: 1
Student ID 1 blocked successfully.
```

14) **Activate student**

```
Enter student ID to block: 1
Student ID 1 blocked successfully.

........ Welcome to Admin Menu ........
1. Add Student
2. View Student Details
3. Add Faculty
4. View Faculty Details
5. Activate Student
6. Block Student
7. Modify Student details
8. Modify Faculty Details
9. Logout and Exit
Enter your choice: 5

Enter student ID to activate: 1
Student ID 1 activated successfully.
```

15) **Modify Student details**

```
........ Welcome to Admin Menu ........
1. Add Student
2. View Student Details
3. Add Faculty
4. View Faculty Details
5. Activate Student
6. Block Student
7. Modify Student details
8. Modify Faculty Details
9. Logout and Exit
Enter your choice: 7

Enter Student ID to modify: 1
Enter new name (or leave blank to keep current): hello
Enter new password (or leave blank to keep current): hell
o
Student details updated successfully.
```

16) **Logging out and disconnecting from server**

```
........ Welcome to Admin Menu ........
1. Add Student
2. View Student Details
3. Add Faculty
4. View Faculty Details
5. Activate Student
6. Block Student
7. Modify Student details
8. Modify Faculty Details
9. Logout and Exit
Enter your choice: 9

Logging out...
Goodbye! Thank you for using Academia Portal.

.....................Welcome Back to Academia :: Course Re
gistration.....................
Login Type
Enter Your Choice { 1. Admin , 2. Professor, 3. Student,
4. Exit } : 4
Server disconnected.
```