# Course Management System using Socket Programming in C

## 1 Problem Statement

Design and implementation a role-based Course Management System using socket programming in C. The system supports three user roles: Admin, Faculty, and Student. Users must authenticate using credentials stored in a file. Admins can manage users, faculty can manage courses, and students can enroll/unenroll from courses. The system must support concurrent clients and ensure data consistency using file locking.

## 2 Implementation Details

### 2.1 Technologies Used

- Language: C (with POSIX sockets)
- Interprocess Communication: TCP sockets
- Concurrency: Pthreads on server side
- File management: Plain text files with advisory file locking (fcntl)

### 2.2 File Structure

- `data/users.txt`: Stores user records in format: `username|role|password|active`
- `data/courses.txt`: Stores course records: `courseid|coursename|faculty|seats|enrolled`
- `src/`: Has all the course files
- `Makefile`: Run make to compile all

### 2.3 Modules and Their Purpose in src/

- **server.c**: Accepts client connections, handles authentication, and routes user operations based on role.
- **client.c**: Prompts for user input and interacts with the server.

- **net_utils.c/h**: Provides socket communication helpers like send_message and recv_message.
- **auth_utils.c/h**: Handles user authentication from the user file.
- **admin_ops.c/h**: Admin can add users, toggle user activity, and update passwords.
- **faculty_ops.c/h**: Faculty can add/remove courses and view enrolled students.
- **student_ops.c/h**: Students can enroll/unenroll in courses and view enrolled ones.
- **password.c/h**: Shared function to change user passwords.

### 2.4 File Locking

- **Read locks** (`F_RDLCK`) are used while authenticating or viewing data.
- **Write locks** (`F_WRLCK`) are used when modifying users or courses.

### 2.5 Logging

Server prints log messages with timestamps indicating connection events, login attempts, and menu actions.

## 3 Source Code Snippets

#### Socket Communication Helper (`net_utils.c`)

```
void send_message(int sock, const char *msg) {
    send(sock, msg, strlen(msg), 0);
}

void recv_message(int sock, char *buf, size_t buflen) {
    memset(buf, 0, buflen);
    int n = recv(sock, buf, buflen - 1, 0);
    if (n > 0) {
        buf[n] = '\0';
        buf[strcspn(buf, "\r\n")] = 0; // Remove newline
    }
}
```

#### Authentication Check (`auth_utils.c`)

```
int authenticate(const char *role, const char *username, const char *
   password) {
    int fd = open(USER_FILE, O_RDONLY);
    if (fd < 0) return 0;

    struct flock lock = {F_RDLCK, SEEK_SET, 0, 0};
    fcntl(fd, F_SETLKW, &lock);
```

```c
    FILE *fp = fdopen(fd, "r");
    char line[256];
    int found = 0;

    while (fgets(line, sizeof(line), fp)) {
        char frole[16], funame[64], fpass[64];
        int active;
        if (sscanf(line, "%63[^|]|%15[^|]|%63[^|]|%d", funame, frole,
           fpass, &active) == 4) {
            if (strcmp(frole, role) == 0 &&
                strcmp(funame, username) == 0 &&
                strcmp(fpass, password) == 0 &&
                active == 1) {
                found = 1;
                break;
            }
        }
    }

    lock.l_type = F_UNLCK;
    fcntl(fd, F_SETLK, &lock);
    fclose(fp);
    return found;
}
```

**Server Logging** (`server.c`)

```c
void log_event(const char *username, const char *action) {
    time_t now = time(NULL);
    struct tm *t = localtime(&now);

    if (username)
        printf("[%02d:%02d:%02d] [%s] %s\n", t->tm_hour, t->tm_min, t->
           tm_sec, username, action);
    else
        printf("[%02d:%02d:%02d] %s\n", t->tm_hour, t->tm_min, t->tm_sec,
           action);
}
```

**Admin: Add New User** (`admin_ops.c`)

```c
void admin_add_user(const char *role, int client_sock) {
    ...
    dprintf(fd, "%s|%s|%s|1\n", username, role, password);
    ...
}
```

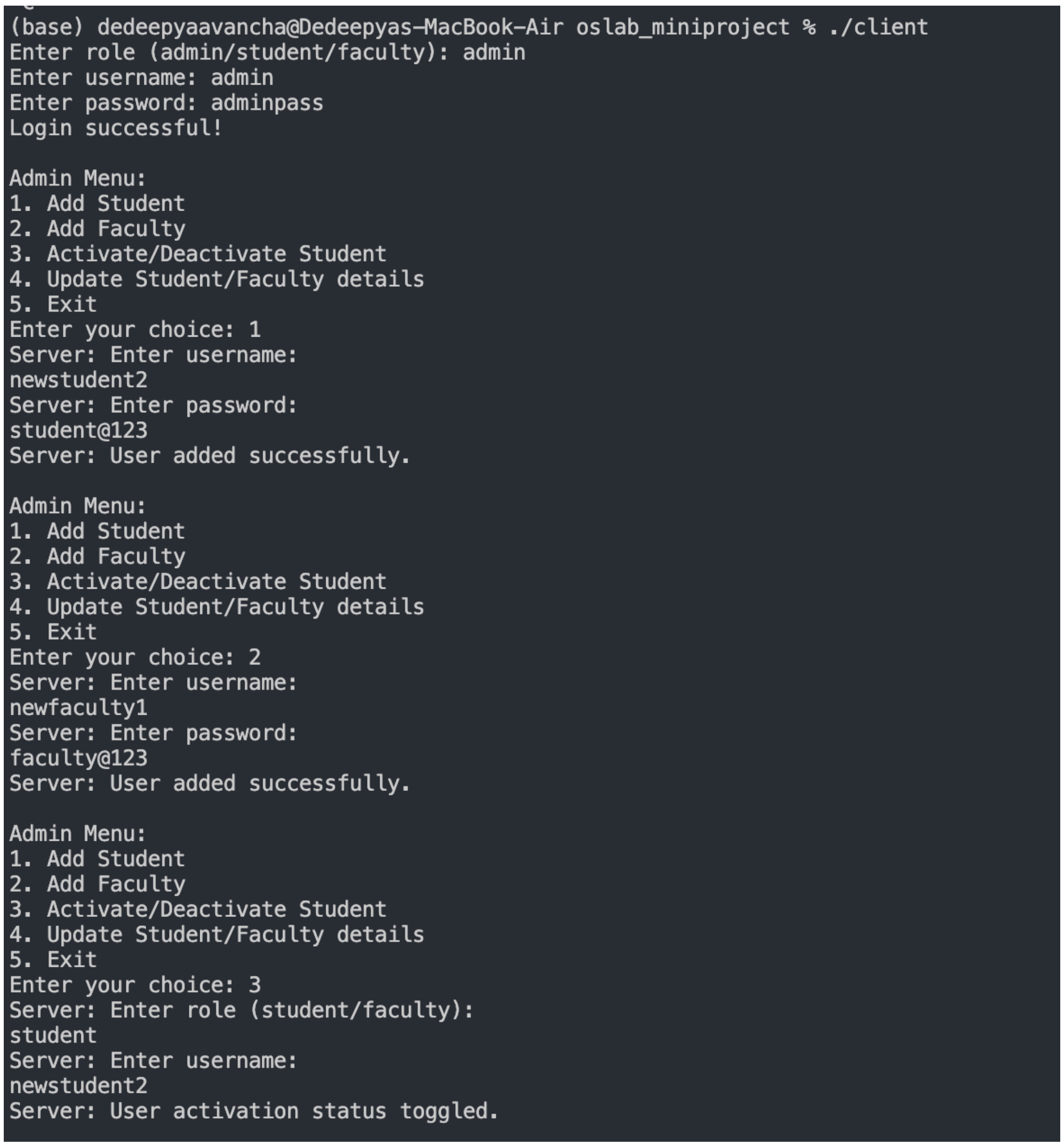
**Faculty: Add Course** (`faculty_ops.c`)

```
void add_course(int client_sock, const char *faculty) {
    ...
    dprintf(fd, "%s|%s|%s|%s|\n", course_id, course_name, faculty, seats);
    ...
}
```

**Student: Enroll in Course** (`student_ops.c`)

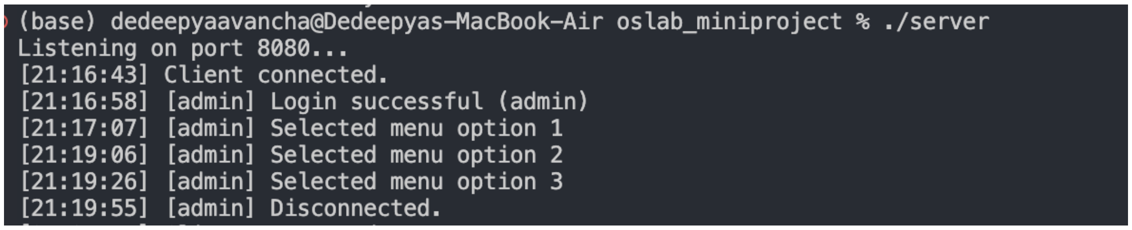
```
void enroll_course(int client_sock, const char *student) {
    ...
    if (enrolled >= seats) {
        send_message(client_sock, "No seats available.");
        return;
    }
    ...
}
```

## 4 Screenshots



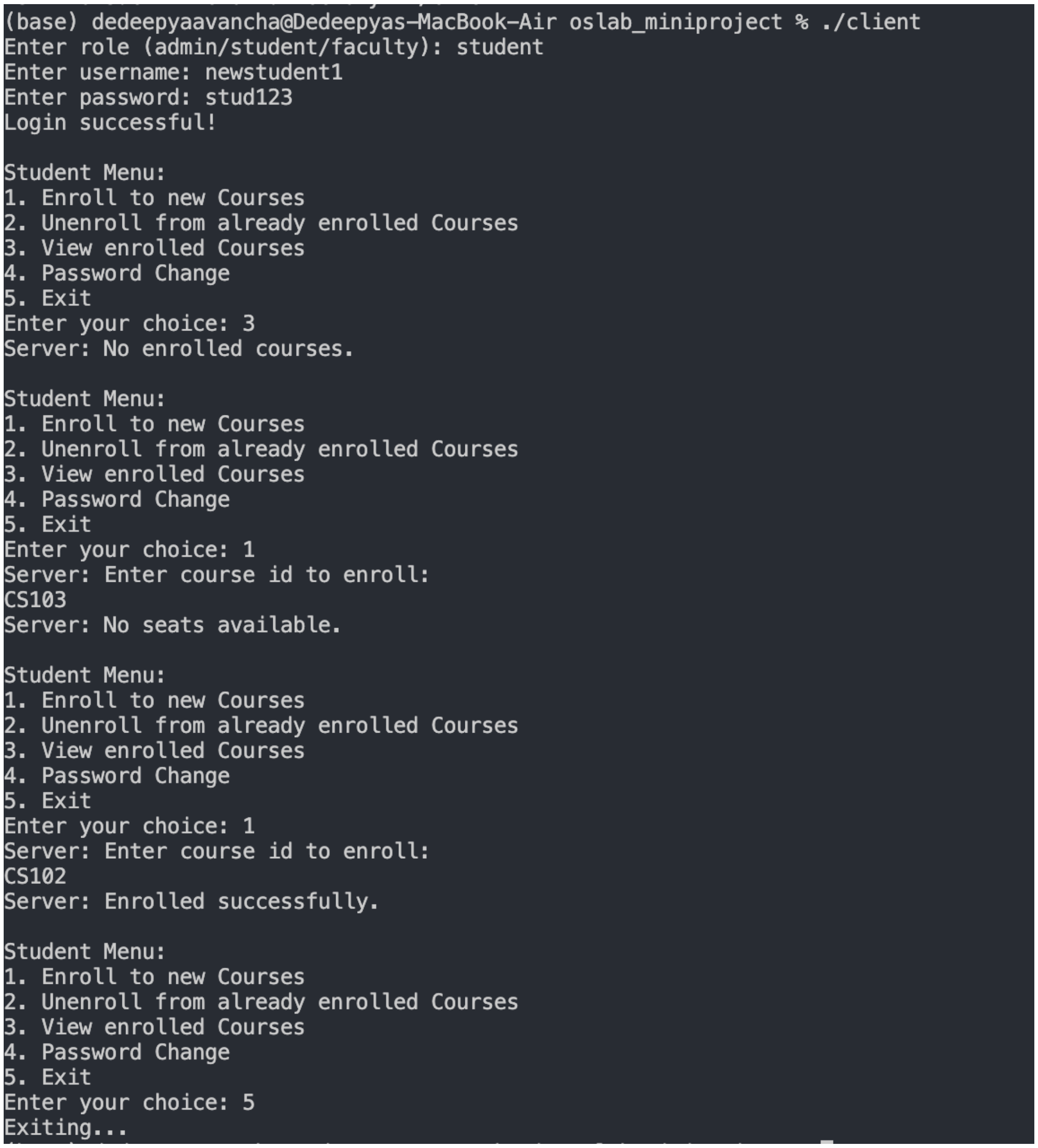
```
(base) dedeepyaavancha@Dedeepyas-MacBook-Air oslab_miniproject % ./client
Enter role (admin/student/faculty): admin
Enter username: admin
Enter password: adminpass
Login successful!

Admin Menu:
1. Add Student
2. Add Faculty
3. Activate/Deactivate Student
4. Update Student/Faculty details
5. Exit
Enter your choice: 1
Server: Enter username:
newstudent2
Server: Enter password:
student@123
Server: User added successfully.

Admin Menu:
1. Add Student
2. Add Faculty
3. Activate/Deactivate Student
4. Update Student/Faculty details
5. Exit
Enter your choice: 2
Server: Enter username:
newfaculty1
Server: Enter password:
faculty@123
Server: User added successfully.

Admin Menu:
1. Add Student
2. Add Faculty
3. Activate/Deactivate Student
4. Update Student/Faculty details
5. Exit
Enter your choice: 3
Server: Enter role (student/faculty):
student
Server: Enter username:
newstudent2
Server: User activation status toggled.
```

Figure 1: Admin



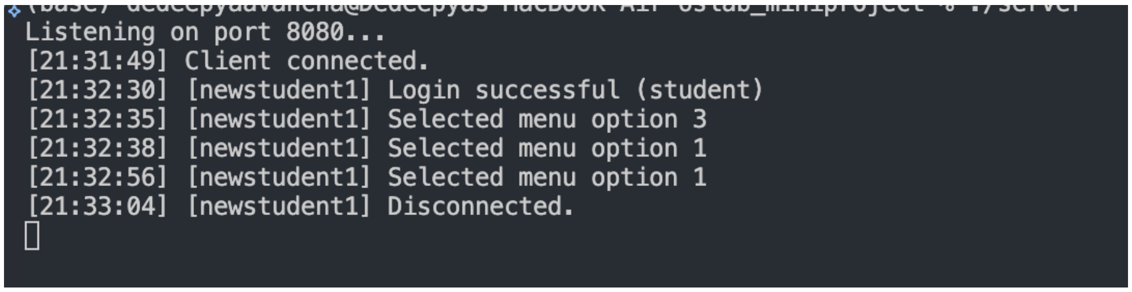
```
(base) dedeepyaavancha@Dedeepyas-MacBook-Air oslab_miniproject % ./server
Listening on port 8080...
[21:16:43] Client connected.
[21:16:58] [admin] Login successful (admin)
[21:17:07] [admin] Selected menu option 1
[21:19:06] [admin] Selected menu option 2
[21:19:26] [admin] Selected menu option 3
[21:19:55] [admin] Disconnected.
```

Figure 2: Admin ops, server side



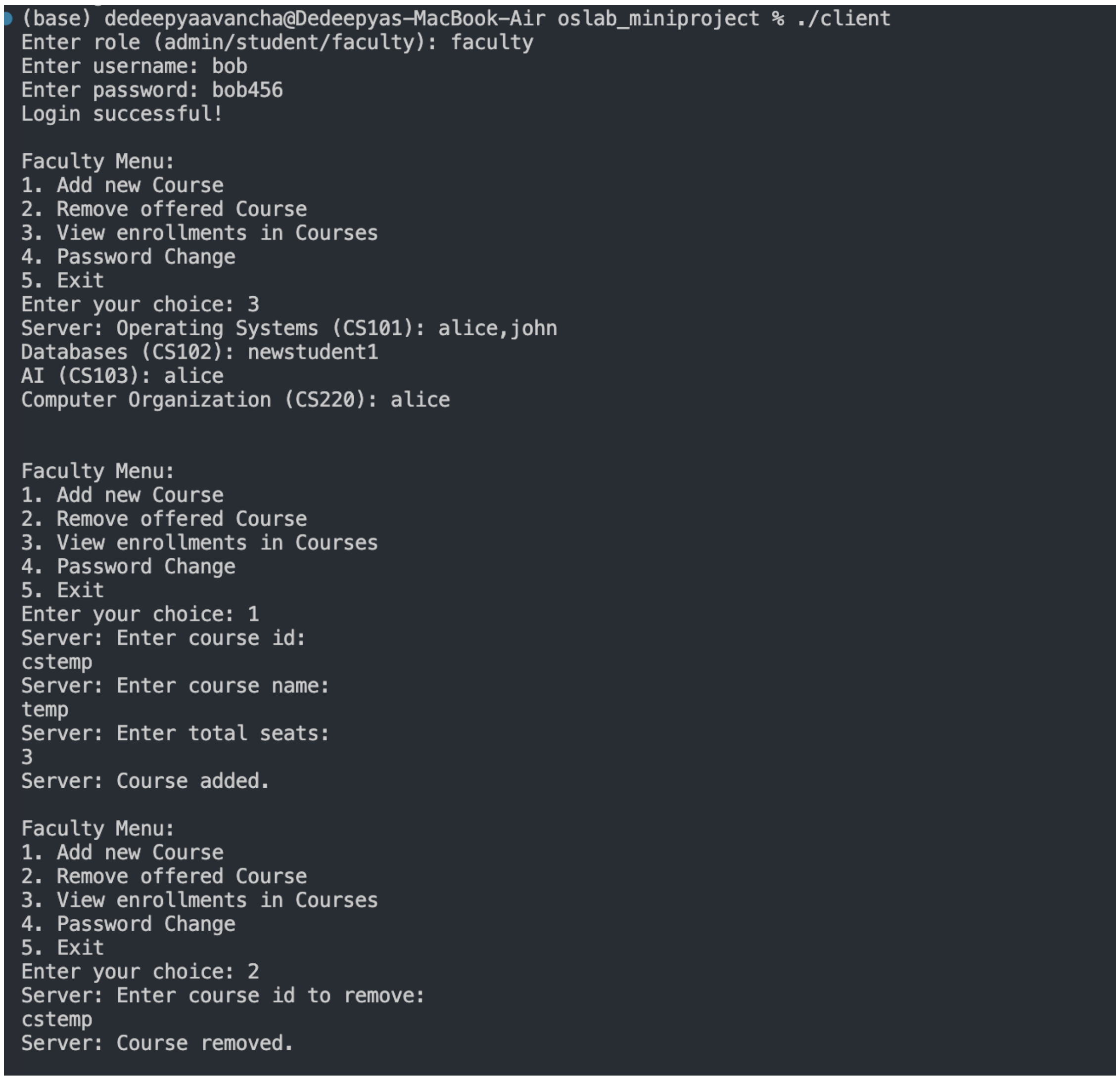
```
(base) dedeepyaavancha@Dedeepyas-MacBook-Air oslab_miniproject % ./client
Enter role (admin/student/faculty): student
Enter username: newstudent1
Enter password: stud123
Login successful!

Student Menu:
1. Enroll to new Courses
2. Unenroll from already enrolled Courses
3. View enrolled Courses
4. Password Change
5. Exit
Enter your choice: 3
Server: No enrolled courses.

Student Menu:
1. Enroll to new Courses
2. Unenroll from already enrolled Courses
3. View enrolled Courses
4. Password Change
5. Exit
Enter your choice: 1
Server: Enter course id to enroll:
CS103
Server: No seats available.

Student Menu:
1. Enroll to new Courses
2. Unenroll from already enrolled Courses
3. View enrolled Courses
4. Password Change
5. Exit
Enter your choice: 1
Server: Enter course id to enroll:
CS102
Server: Enrolled successfully.

Student Menu:
1. Enroll to new Courses
2. Unenroll from already enrolled Courses
3. View enrolled Courses
4. Password Change
5. Exit
Enter your choice: 5
Exiting...
```

Figure 3: Student



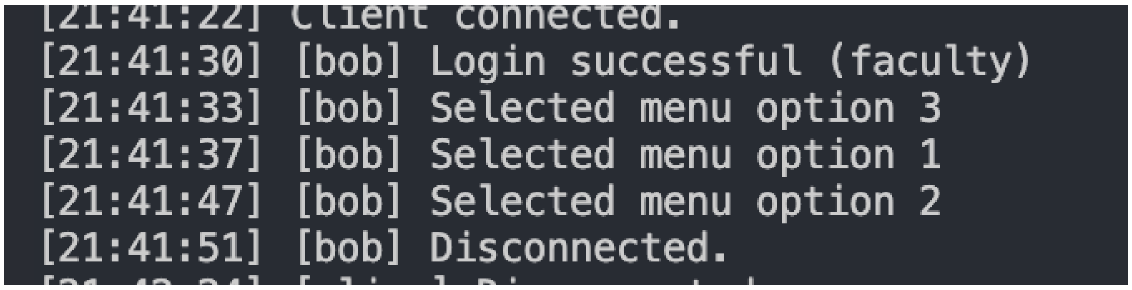
```
Listening on port 8080...
[21:31:49] Client connected.
[21:32:30] [newstudent1] Login successful (student)
[21:32:35] [newstudent1] Selected menu option 3
[21:32:38] [newstudent1] Selected menu option 1
[21:32:56] [newstudent1] Selected menu option 1
[21:33:04] [newstudent1] Disconnected.
```

Figure 4: Student ops server side



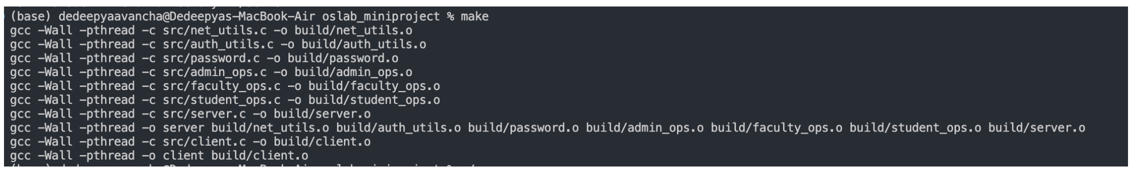
```
(base) dedeepyaavancha@Dedeepyas-MacBook-Air oslab_miniproject % ./client
Enter role (admin/student/faculty): faculty
Enter username: bob
Enter password: bob456
Login successful!

Faculty Menu:
1. Add new Course
2. Remove offered Course
3. View enrollments in Courses
4. Password Change
5. Exit
Enter your choice: 3
Server: Operating Systems (CS101): alice,john
Databases (CS102): newstudent1
AI (CS103): alice
Computer Organization (CS220): alice


Faculty Menu:
1. Add new Course
2. Remove offered Course
3. View enrollments in Courses
4. Password Change
5. Exit
Enter your choice: 1
Server: Enter course id:
cstemp
Server: Enter course name:
temp
Server: Enter total seats:
3
Server: Course added.

Faculty Menu:
1. Add new Course
2. Remove offered Course
3. View enrollments in Courses
4. Password Change
5. Exit
Enter your choice: 2
Server: Enter course id to remove:
cstemp
Server: Course removed.
```

Figure 5: Faculty



```
[21:41:22] Client connected.
[21:41:30] [bob] Login successful (faculty)
[21:41:33] [bob] Selected menu option 3
[21:41:37] [bob] Selected menu option 1
[21:41:47] [bob] Selected menu option 2
[21:41:51] [bob] Disconnected.
```

Figure 6: Faculty Server-side



```
(base) dedeepyaavancha@Dedeepyas-MacBook-Air oslab_miniproject % make
gcc -Wall -pthread -c src/net_utils.c -o build/net_utils.o
gcc -Wall -pthread -c src/auth_utils.c -o build/auth_utils.o
gcc -Wall -pthread -c src/password.c -o build/password.o
gcc -Wall -pthread -c src/admin_ops.c -o build/admin_ops.o
gcc -Wall -pthread -c src/faculty_ops.c -o build/faculty_ops.o
gcc -Wall -pthread -c src/student_ops.c -o build/student_ops.o
gcc -Wall -pthread -c src/server.c -o build/server.o
gcc -Wall -pthread -o server build/net_utils.o build/auth_utils.o build/password.o build/admin_ops.o build/faculty_ops.o build/student_ops.o build/server.o
gcc -Wall -pthread -c src/client.c -o build/client.o
gcc -Wall -pthread -o client build/client.o
```

Figure 7: make

## 5 Conclusion

This course management system demonstrates multi-role, concurrent, file-safe communication using TCP sockets in C. File locking is used for concurrency. Future improvements could include replacing text files with a database or developing a GUI for client interaction.