Design and Development of Course Registration Portal (Academia)

EGC 301P – Operating Systems Lab – Mini Project

SUBMITTED BY: NAVANEETH D (IMT2023095)

Problem Statement

Existing academic management systems often lack customization, security, and efficient concurrent access tailored to institutional needs. There is a need for a user-friendly, role-based course registration system that allows students, faculty, and administrators to securely interact with course data. The system must ensure data integrity, concurrent access control, and secure authentication, while handling multiple clients through a server-based architecture.

Implementation Details

For the implementation of this project, a file-based storage system was used to maintain all student, faculty, and course data. The server was developed using socket programming to handle connections from multiple clients concurrently. To support multiple users simultaneously, threading was employed, allowing each connected user (Admin, Faculty, or Student) to interact with the system independently. Critical sections involving course enrolment and modification were protected using mutex locks to ensure data consistency and prevent race conditions. The project was implemented entirely in C, utilizing system-level programming concepts such as system calls, process and file management, thread handling, and synchronization primitives for robust and efficient operation.

Source Code Explanation

Server

This code implements the server-side logic for a role-based multi-client system where users (Admins, Students, and Faculty) can log in and perform role-specific operations. It uses socket programming and multithreading (pthreads) to handle concurrent client connections and file-based data storage with mutex-based synchronization.

Key Components and Flow:

1. **Global Variables & Structs**:
    - sockfd, serv_addr are used for network setup.
    - A Command structure holds information about the user and the command sent by the client.
2. **tryLogin() Function**:
    - Handles user login for Admin, Student, and Faculty roles.
    - Depending on the role, it loads the corresponding file (ADMIN_FILE, STUDENT_FILE, or FACULTY_FILE), locks it with a mutex, and searches for matching credentials.
    - If a matching Admin is not found, the user is added to the file.
    - For Students/Faculty, additional fields like course_count, isActive, department, etc. are updated upon successful login.
3. **Thread Function handle_client()**:
    - Reads the command from the connected client.
    - If the client is not logged in, only login (LOGIN) commands are accepted.
    - On successful login, the role is stored and specific operations are allowed.
4. **Admin Operations**:
    - ADD_FACULTY, ADD_STUDENT: Add new users.
    - VIEW_STUDENT, VIEW_FACULTY: View lists of users.
    - ACTIVATE_STUDENT, BLOCK_STUDENT: Change student activeness status.
    - MODIFY_STUDENT, MODIFY_FACULTY: Modify user information.
5. **Faculty Operations**:
    - ADD_COURSE: Add a new course.
    - VIEW_COURSE: View courses assigned to the faculty.
6. **General Features**:
    - LOGOUT command resets the session.

- Appropriate status codes like SUCCESS, FAILED, INVALID_CREDENTIALS, NOT_FOUND are sent back to the client.
- Memory and file descriptors are carefully managed and cleaned up.

This modular and thread-safe design allows multiple users with different roles to interact with the server simultaneously, ensuring consistent file access using mutexes.

Client

The code implements a command-line interface (CLI) for managing courses, users, and passwords for an academic system with roles like **Admin**, **Faculty**, and **Student**. Here's a brief description of how the system works:

1. **Main Function:**
    - The main function handles the initial login process, where users are prompted to enter their role (Admin, Faculty, or Student) and credentials (username and password).
    - Based on the role, the appropriate user structure (Admin, Faculty, or Student) is allocated and populated with data received from the server upon successful login.
2. **Signal Handling:**
    - The program is set up to gracefully handle various signals (e.g., SIGINT, SIGTERM, SIGSEGV, etc.), ensuring proper cleanup (freeing memory and closing the socket) before exiting.
3. **User Interaction:**
    - After login, the program continuously presents a menu for the logged-in user to interact with, based on their role:
        - **Admin**: Admin commands allow the admin to add/remove courses, modify courses, view courses, change passwords, and log out.
        - **Faculty**: Faculty commands let faculty members add/remove courses, modify courses, view courses, change passwords, and log out.
        - **Student**: Students can view courses, enroll in courses, drop courses, change their password, and log out.
4. **Functionality for Adding/Removing Courses (Faculty Only):**

- Faculty can add or remove courses from their list. The system checks for existing courses before adding, and it ensures that courses can only be removed if no students are enrolled.

5. **Enrolling and Dropping Courses (Student Only):**
    - Students can enrol or drop courses. The system prevents students from enrolling in the same course twice or dropping a course they're not enrolled in.
6. **Password Change:**
    - Both students and faculty can change their passwords. The system updates the password both locally and on the server.
7. **Communication with Server:**
    - The sendToServer function sends commands (such as course management or login details) to the server and handles the response.
    - Server responses (e.g., SUCCESS, NOT_FOUND, INVALID_CREDENTIALS) dictate the next course of action, including whether an operation was successful or if an error occurred.
8. **Socket Communication:**
    - The code sets up a TCP connection to the server and uses the socket to send and receive data (like user credentials, course information, etc.).

The code integrates user roles and various course management features into a single interface with the server handling persistent data and command execution.

Output Screenshots

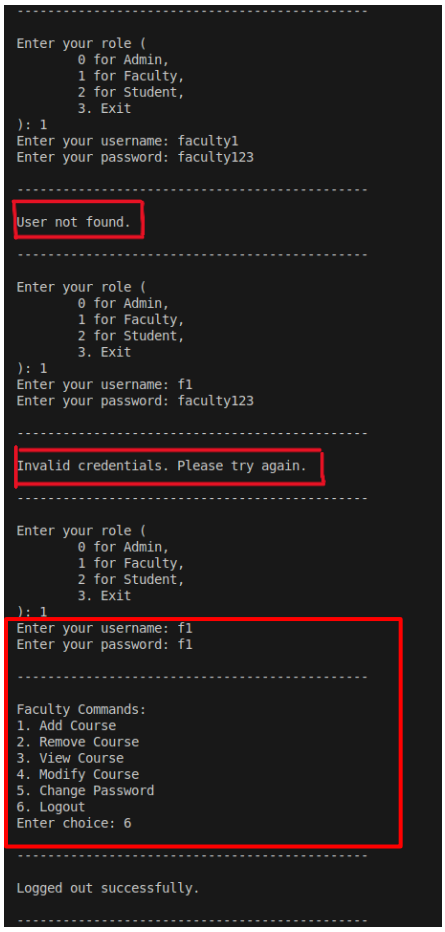

FIGURE 1: AUTHENTICATION

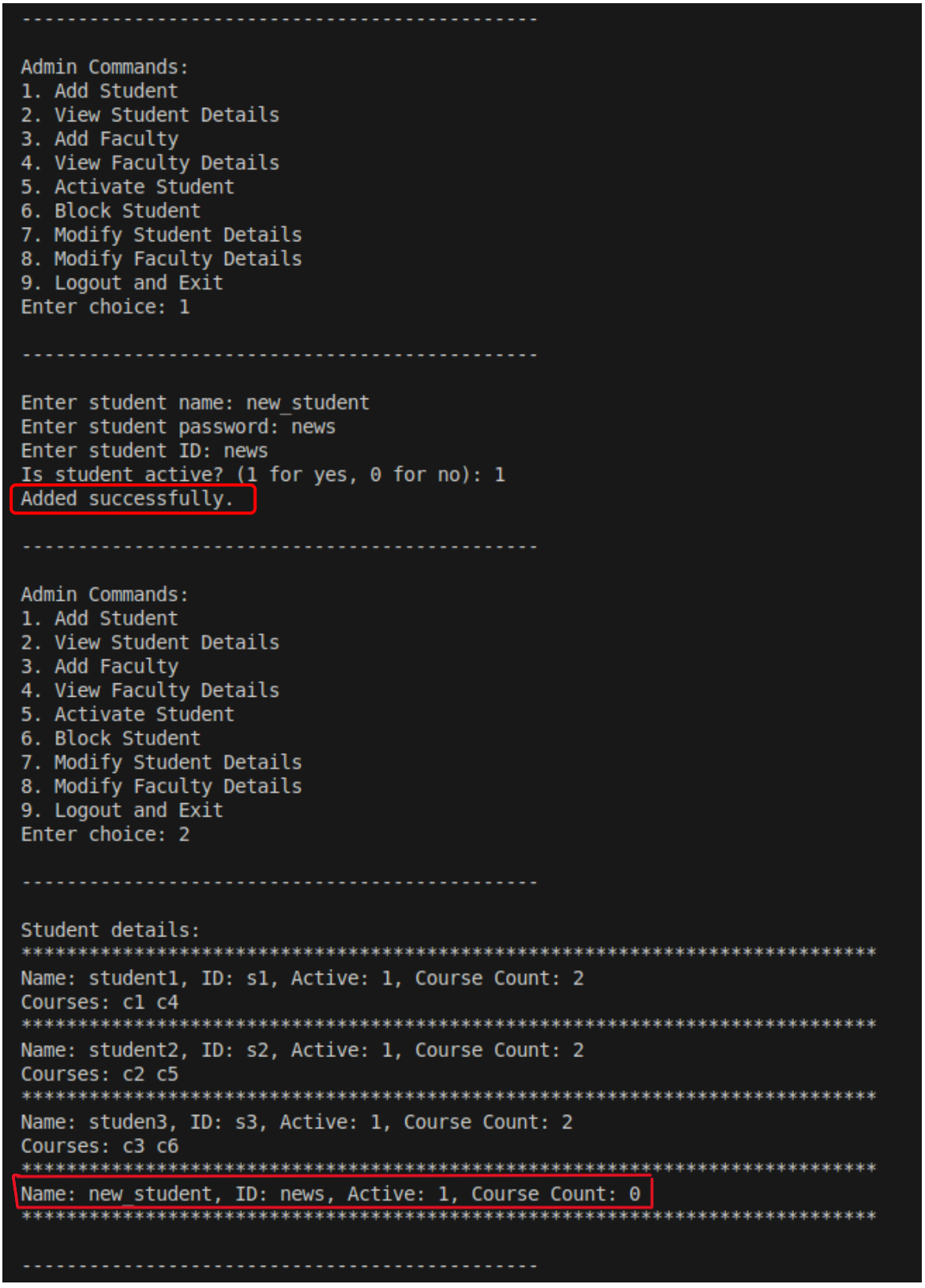

FIGURE 2: ADD STUDENTS AND VIEW STUDENT DETAILS

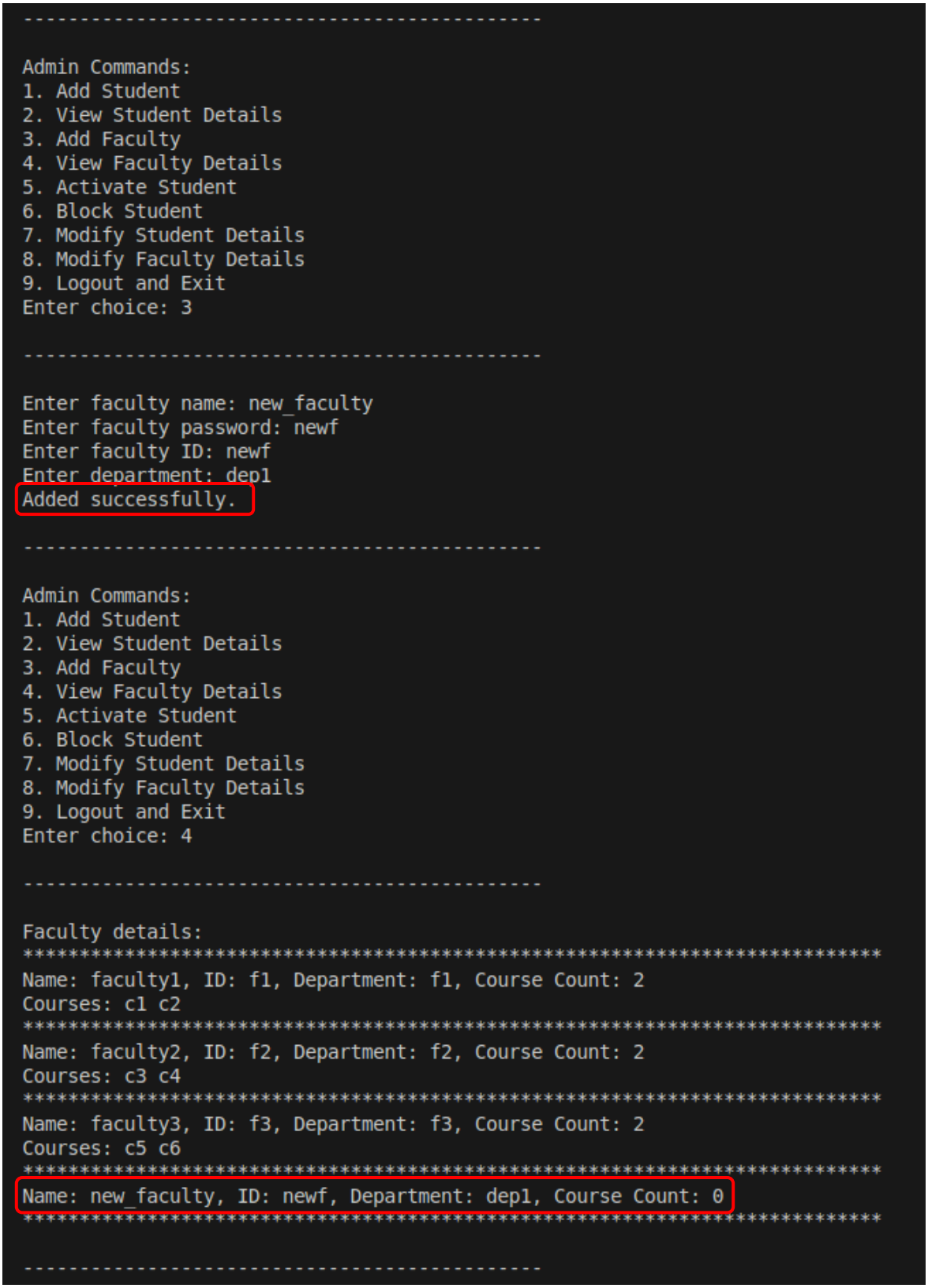

FIGURE 3: ADD FACULTY AND VIEW FACULTY DETAILS

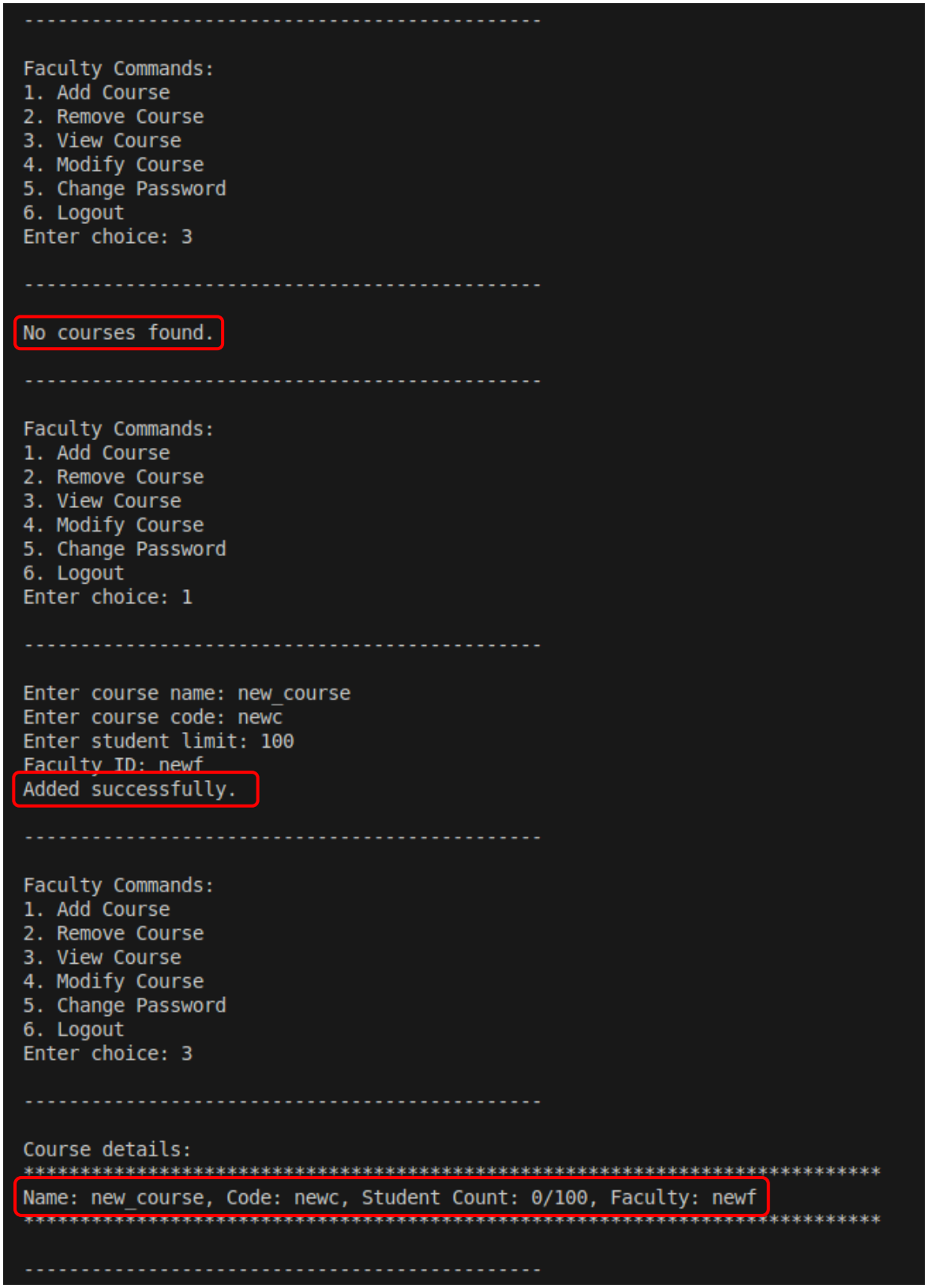

FIGURE 4: FACULTY CAN ADD NEW COURSE

FIGURE 5: FACULTY CAN REMOVE COURSES

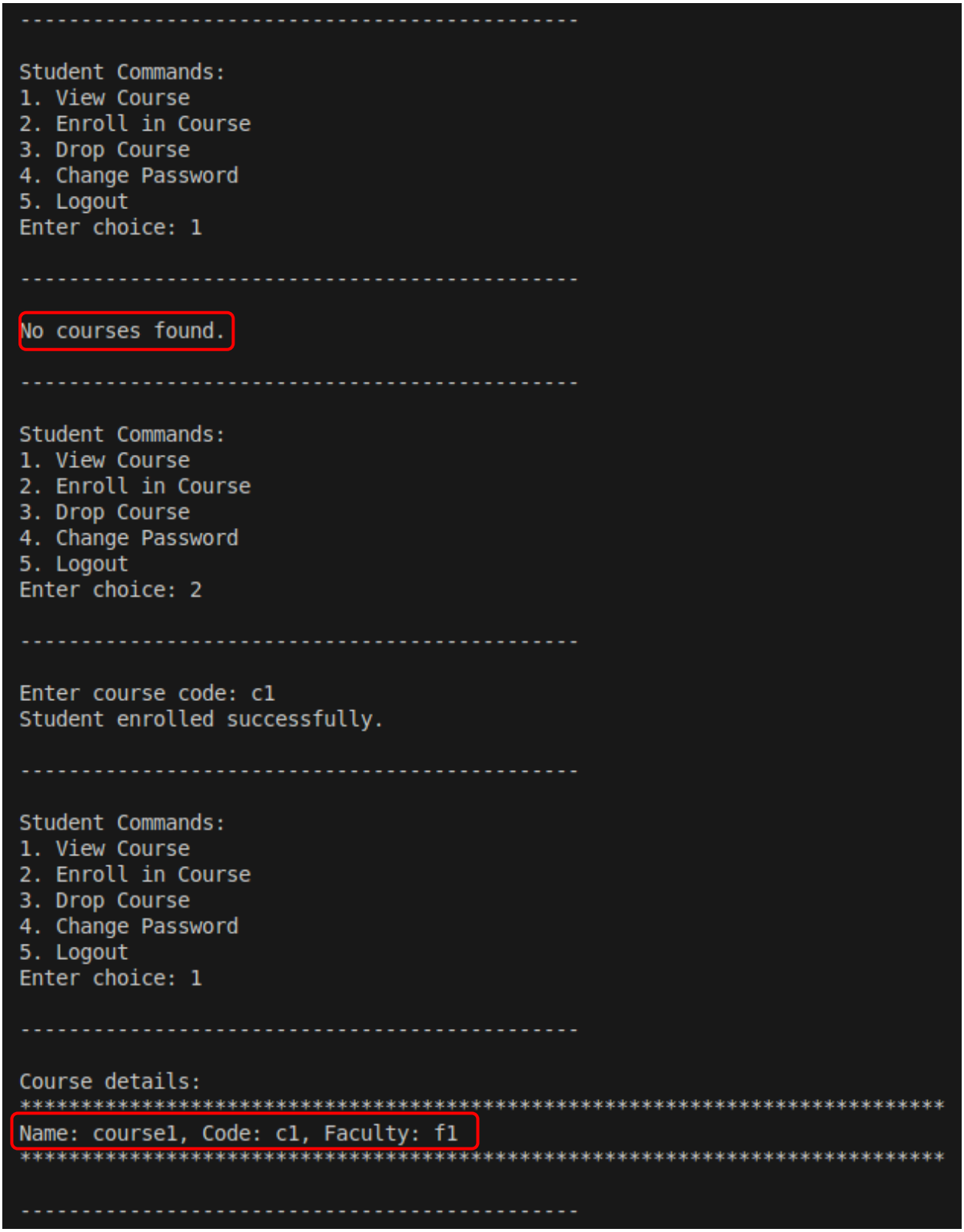

FIGURE 6: STUDENTS CAN ENROL IN COURSES

```
Enter your role (
        0 for Admin,
        1 for Faculty,
        2 for Student
): 1
Enter your username: f1
Enter your password: f1

-------------------------------------------

Faculty Commands:
1. Add Course
2. Remove Course
3. View Course
4. Modify Course
5. Change Password
6. Logout
Enter choice: 3

-------------------------------------------

Course details:
*********************************************************************************
Name: course1, Code: c1, Student Count: 2/100, Faculty: f1
Students: s1 news
*********************************************************************************
Name: course2, Code: c2, Student Count: 1/100, Faculty: f1
Students: s2
*********************************************************************************

-------------------------------------------
```

FIGURE 7: FACULTY DATA IS ALSO UPDATED WHEN STUDENT ENROLS

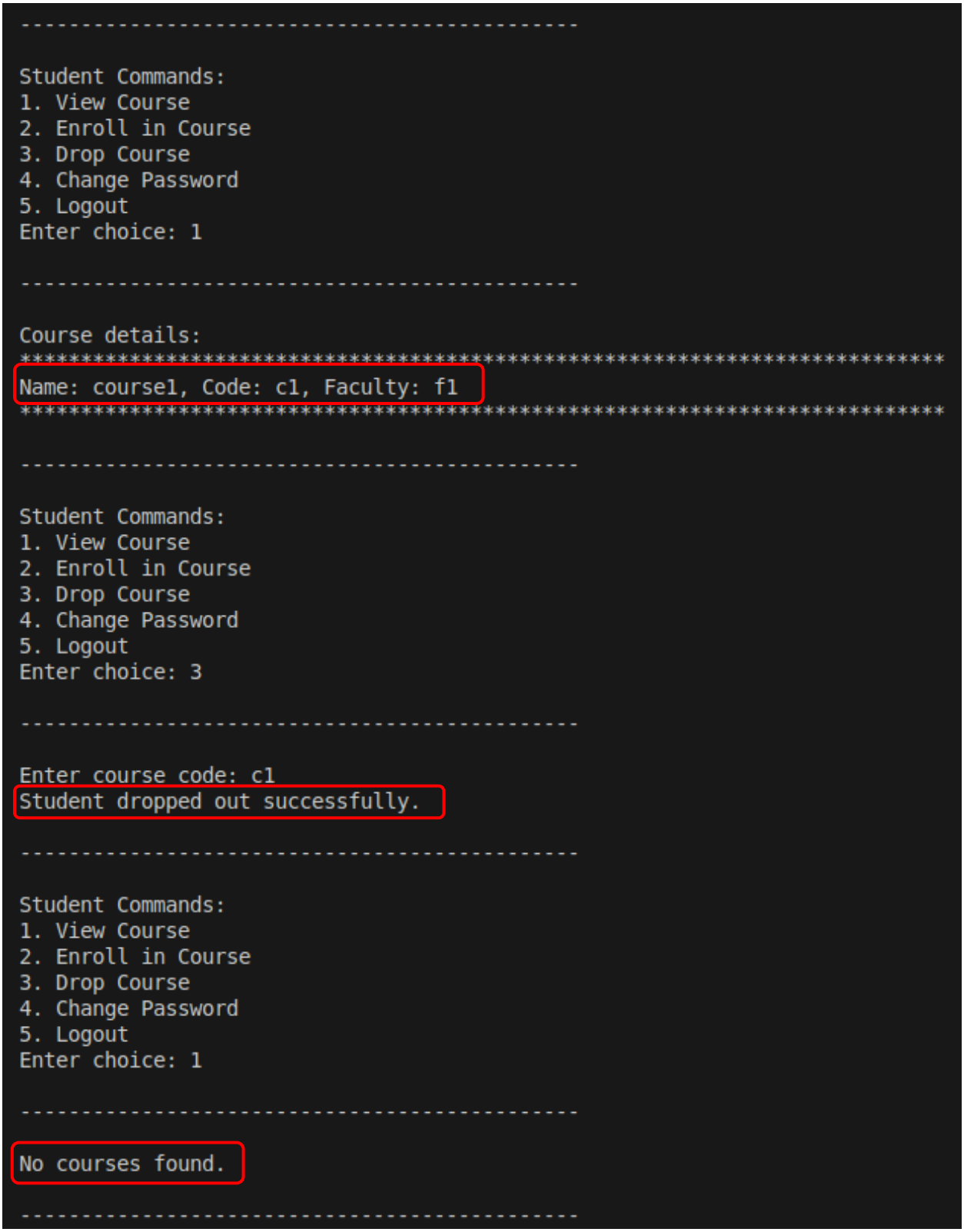

FIGURE 8: STUDENTS CAN DROP COURSES