OS MINI PROJECT

Course Registration Portal (Academia)

Problem Statement

Design and implement a **Course Registration Portal** that simulates a real-world academic registration system with different user roles: **Student, Faculty, and Admin**. The system should allow:

- **Student** to register and enroll in available courses.
- **Faculty** to manage and view course assignments.
- **Admin** to initialize and manage records.

This system is implemented using client-server architecture via TCP socket programming in C. The server is responsible for data storage and client requests, while clients interact with the system based on their roles.

Implementation Details

Technologies Used

- **Language**: C
- **Communication**: TCP Sockets
- **Storage**: File-based (binary/text)
- **Design**: Modular programming using header files

System Modules and Files

File/Folder	Description
`client.c`	Client-side logic — user input, menu interaction, request sending
`server.c`	Server-side logic — handles multiple clients, data processing
`set_admin.c`	Initializes admin credentials and account
`set_record.c`	Sets up initial records (courses, users, etc.)
`Records/`	Directory to store persistent data (student/course/faculty files)

Header Files and Their Functions:

Header File	Purpose
`functionsd.h`	Utility/helper functions
`ReadWrite.h`	File I/O for reading/writing data structures
`record_set.h`	For record management
`recordg_s.h`	Getter/setter for various user and course records
`structs.h`	Defines core data structures (Student, Course, Faculty, etc.)
`hstudent.h`	Student-specific operations (course registration)
`hfaculty.h`	Faculty-specific operations (assignments, views)
`hcourse.h`	Course-specific operations
`hlogin.h`	Authentication logic for user login
`hmenu.h`	Logic for displaying and handling menus
`facultyg_s.h`	Faculty-related getter/setter functions
`studentg_s.h`	Student-related getter/setter functions
`coursegs.h`	Course getter/setter and file writing

How to Compile and Run:

Step 1: Setup admin account

```
gcc -o set_admin set_admin.c
./set_admin
```

Step 2: Initialize records (students, courses, faculty)

```
gcc -o set_record set_record.c
./set_record
```

Step 3: Start the server

```
gcc -o server server.c
./server
```

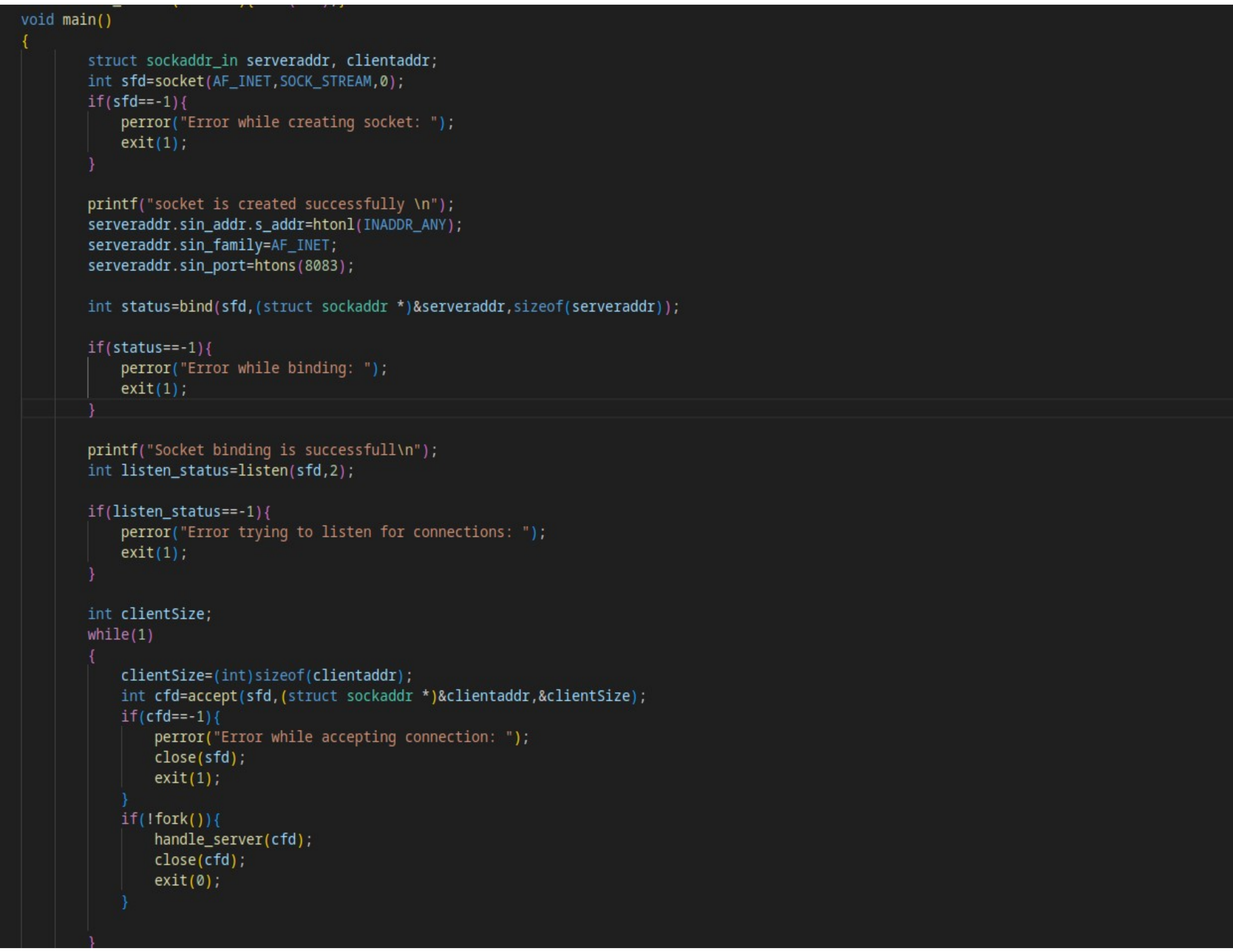
Step 4: Launch the client (in a new terminal)

```
gcc -o client client.c
./client
```

Source Code with Explanation:

Server.c:

- **Creates TCP Socket**: `socket(AF_INET, SOCK_STREAM, 0)`.
- **Binds to Port 8083**: Binds to all interfaces using `INADDR_ANY`.
- **Listens for Clients**: Starts listening with a backlog of 2.
- **Accepts Clients in Loop**: Uses `accept()` to handle incoming connections.
- **Forks for Each Client**: `fork()` creates a new process per client.
- **Handles Communication**: Child process calls `menu(cfd)` via `handle_server()`.
- **Closes Sockets**: Child closes `cfd`; parent closes `sfd` on shutdown.

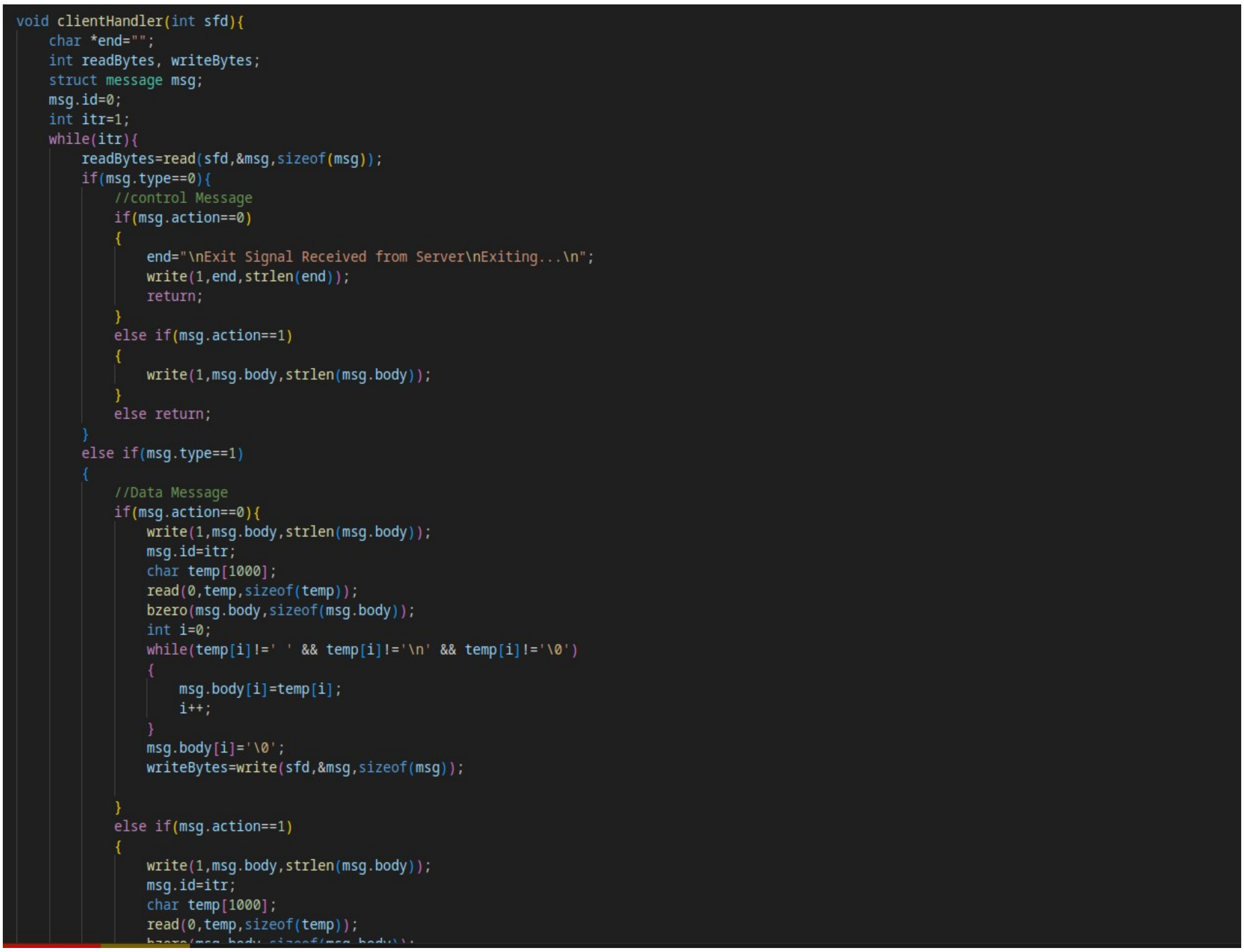


```c
void main()
{
        struct sockaddr_in serveraddr, clientaddr;
        int sfd=socket(AF_INET,SOCK_STREAM,0);
        if(sfd==-1){
            perror("Error while creating socket: ");
            exit(1);
        }

        printf("socket is created successfully \n");
        serveraddr.sin_addr.s_addr=htonl(INADDR_ANY);
        serveraddr.sin_family=AF_INET;
        serveraddr.sin_port=htons(8083);

        int status=bind(sfd,(struct sockaddr *)&serveraddr,sizeof(serveraddr));

        if(status==-1){
            perror("Error while binding: ");
            exit(1);
        }

        printf("Socket binding is successfull\n");
        int listen_status=listen(sfd,2);

        if(listen_status==-1){
            perror("Error trying to listen for connections: ");
            exit(1);
        }

        int clientSize;
        while(1)
        {
            clientSize=(int)sizeof(clientaddr);
            int cfd=accept(sfd,(struct sockaddr *)&clientaddr,&clientSize);
            if(cfd==-1){
                perror("Error while accepting connection: ");
                close(sfd);
                exit(1);
            }
            if(!fork()){
                handle_server(cfd);
                close(cfd);
                exit(0);
            }

        }
```

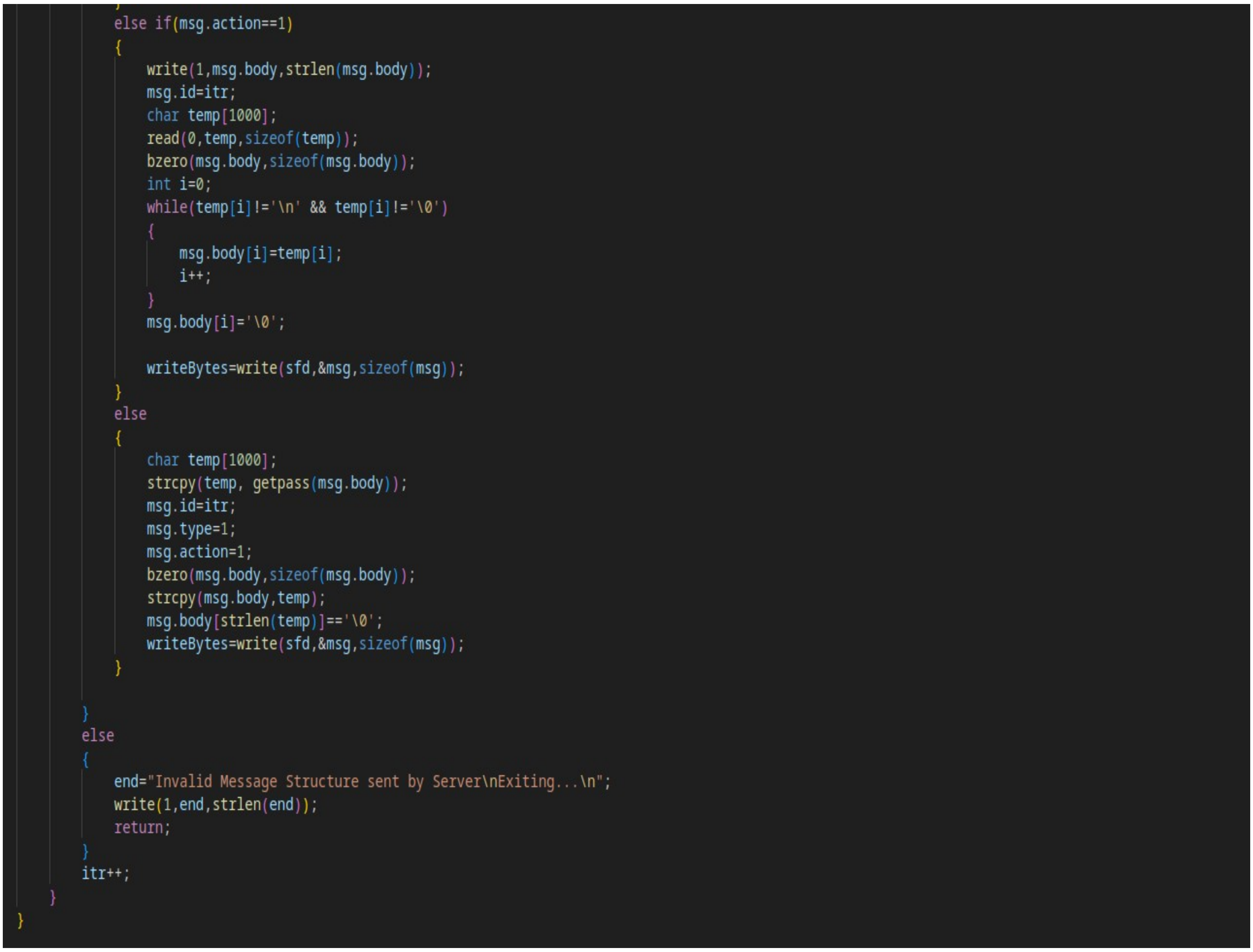
Client.c:

Creates TCP Socket: `socket(AF_INET, SOCK_STREAM, 0)` creates the client socket.

- **Connects to Server**: Connects to `localhost` on port `8083` using `connect()`.
- **Handles Server Communication**: Calls `my_client_handle(sfd)` to begin interaction.
- **Processes Messages from Server**:
- **Control Messages**: For actions like exit or display prompts.
- **Data Messages**: For user input (single word, string, or password).
- **Reads Input & Sends to Server**: Captures user input and sends responses back.
- **Closes Socket**: Gracefully closes connection after interaction.



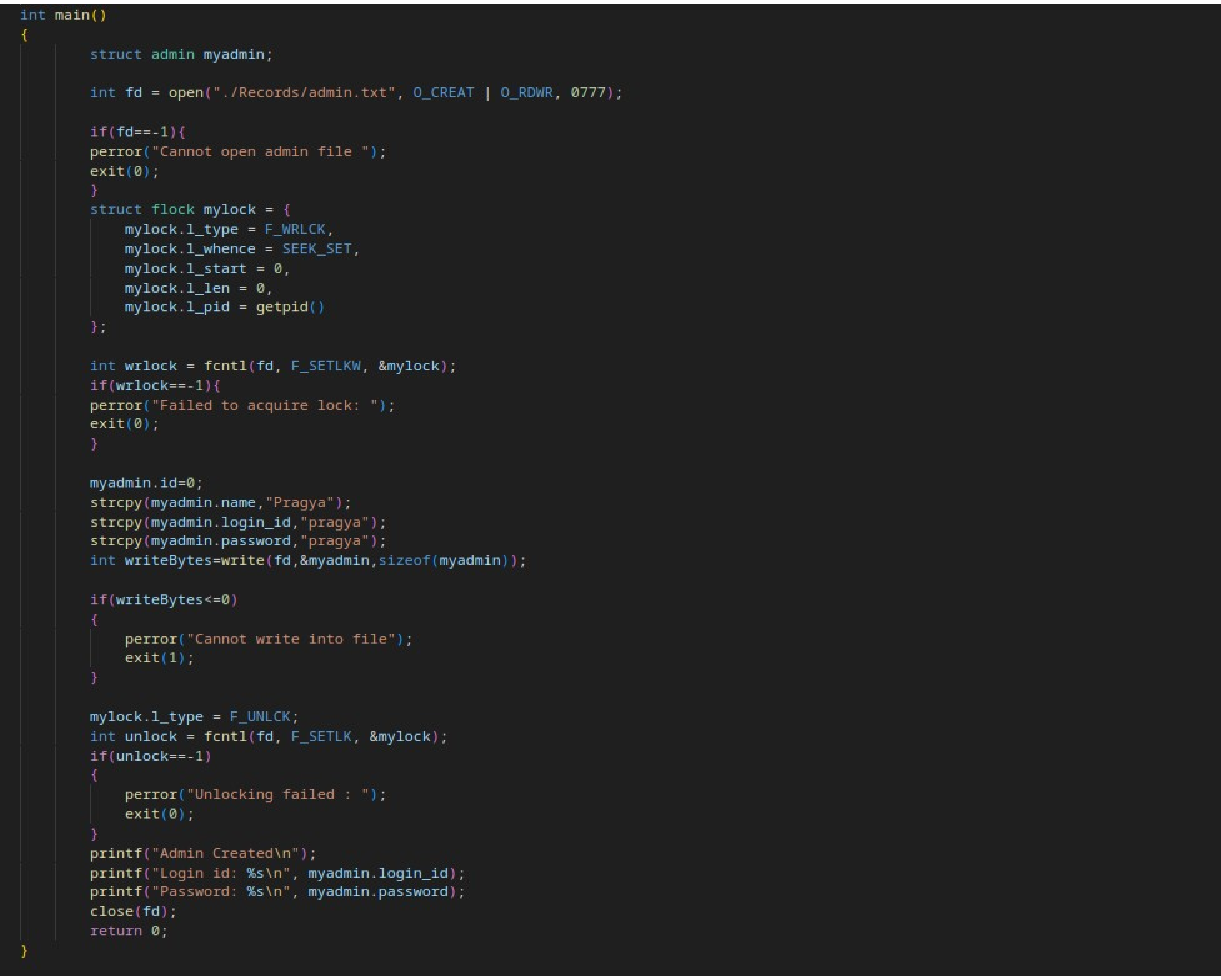
```c
void clientHandler(int sfd){
    char *end="";
    int readBytes, writeBytes;
    struct message msg;
    msg.id=0;
    int itr=1;
    while(itr){
        readBytes=read(sfd,&msg,sizeof(msg));
        if(msg.type==0){
            //control Message
            if(msg.action==0)
            {
                end="\nExit Signal Received from Server\nExiting...\n";
                write(1,end,strlen(end));
                return;
            }
            else if(msg.action==1)
            {
                write(1,msg.body,strlen(msg.body));
            }
            else return;
        }
        else if(msg.type==1)
        {
            //Data Message
            if(msg.action==0){
                write(1,msg.body,strlen(msg.body));
                msg.id=itr;
                char temp[1000];
                read(0,temp,sizeof(temp));
                bzero(msg.body,sizeof(msg.body));
                int i=0;
                while(temp[i]!=' ' && temp[i]!='\n' && temp[i]!='\0')
                {
                    msg.body[i]=temp[i];
                    i++;
                }
                msg.body[i]='\0';
                writeBytes=write(sfd,&msg,sizeof(msg));

            }
            else if(msg.action==1)
            {
                write(1,msg.body,strlen(msg.body));
                msg.id=itr;
                char temp[1000];
                read(0,temp,sizeof(temp));
```



```
            }
            else if(msg.action==1)
            {
                write(1,msg.body,strlen(msg.body));
                msg.id=itr;
                char temp[1000];
                read(0,temp,sizeof(temp));
                bzero(msg.body,sizeof(msg.body));
                int i=0;
                while(temp[i]!='\n' && temp[i]!='\0')
                {
                    msg.body[i]=temp[i];
                    i++;
                }
                msg.body[i]='\0';

                writeBytes=write(sfd,&msg,sizeof(msg));
            }
            else
            {
                char temp[1000];
                strcpy(temp, getpass(msg.body));
                msg.id=itr;
                msg.type=1;
                msg.action=1;
                bzero(msg.body,sizeof(msg.body));
                strcpy(msg.body,temp);
                msg.body[strlen(temp)]=='\0';
                writeBytes=write(sfd,&msg,sizeof(msg));
            }

        }
        else
        {
            end="Invalid Message Structure sent by Server\nExiting...\n";
            write(1,end,strlen(end));
            return;
        }
        itr++;
    }
}
```

SetAdmin.c:

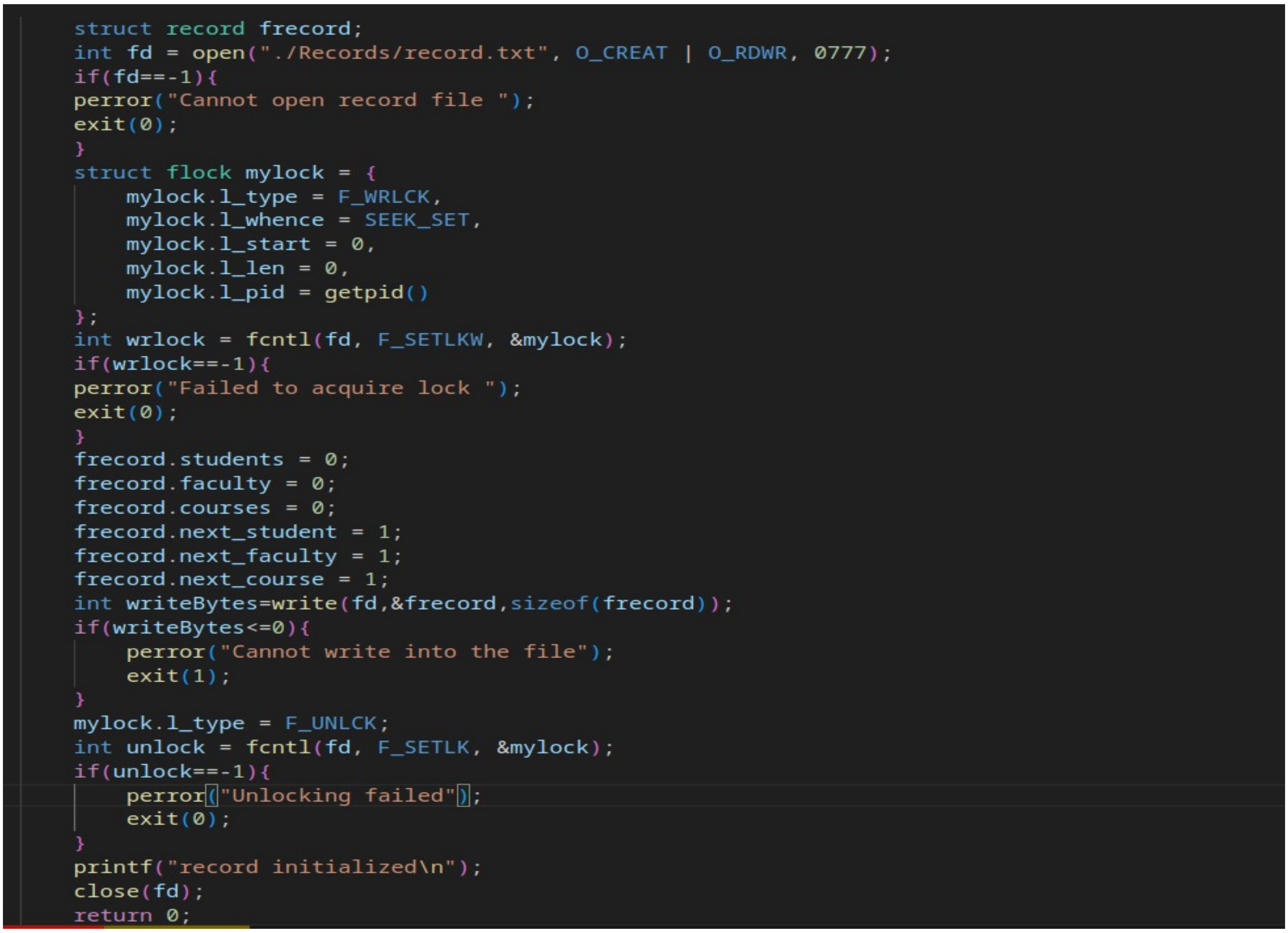
- Opens (or creates) `admin.txt` with read/write access and locks the file using `fcntl()` to prevent concurrent writes.
- Fills an `admin` struct with predefined details.
- Writes the struct to the file.
- Unlocks the file and prints login credentials



```c
int main()
{
        struct admin myadmin;

        int fd = open("./Records/admin.txt", O_CREAT | O_RDWR, 0777);

        if(fd==-1){
        perror("Cannot open admin file ");
        exit(0);
        }
        struct flock mylock = {
            mylock.l_type = F_WRLCK,
            mylock.l_whence = SEEK_SET,
            mylock.l_start = 0,
            mylock.l_len = 0,
            mylock.l_pid = getpid()
        };

        int wrlock = fcntl(fd, F_SETLKW, &mylock);
        if(wrlock==-1){
        perror("Failed to acquire lock: ");
        exit(0);
        }

        myadmin.id=0;
        strcpy(myadmin.name,"Pragya");
        strcpy(myadmin.login_id,"pragya");
        strcpy(myadmin.password,"pragya");
        int writeBytes=write(fd,&myadmin,sizeof(myadmin));

        if(writeBytes<=0)
        {
            perror("Cannot write into file");
            exit(1);
        }

        mylock.l_type = F_UNLCK;
        int unlock = fcntl(fd, F_SETLK, &mylock);
        if(unlock==-1)
        {
            perror("Unlocking failed : ");
            exit(0);
        }
        printf("Admin Created\n");
        printf("Login id: %s\n", myadmin.login_id);
        printf("Password: %s\n", myadmin.password);
        close(fd);
        return 0;
}
```

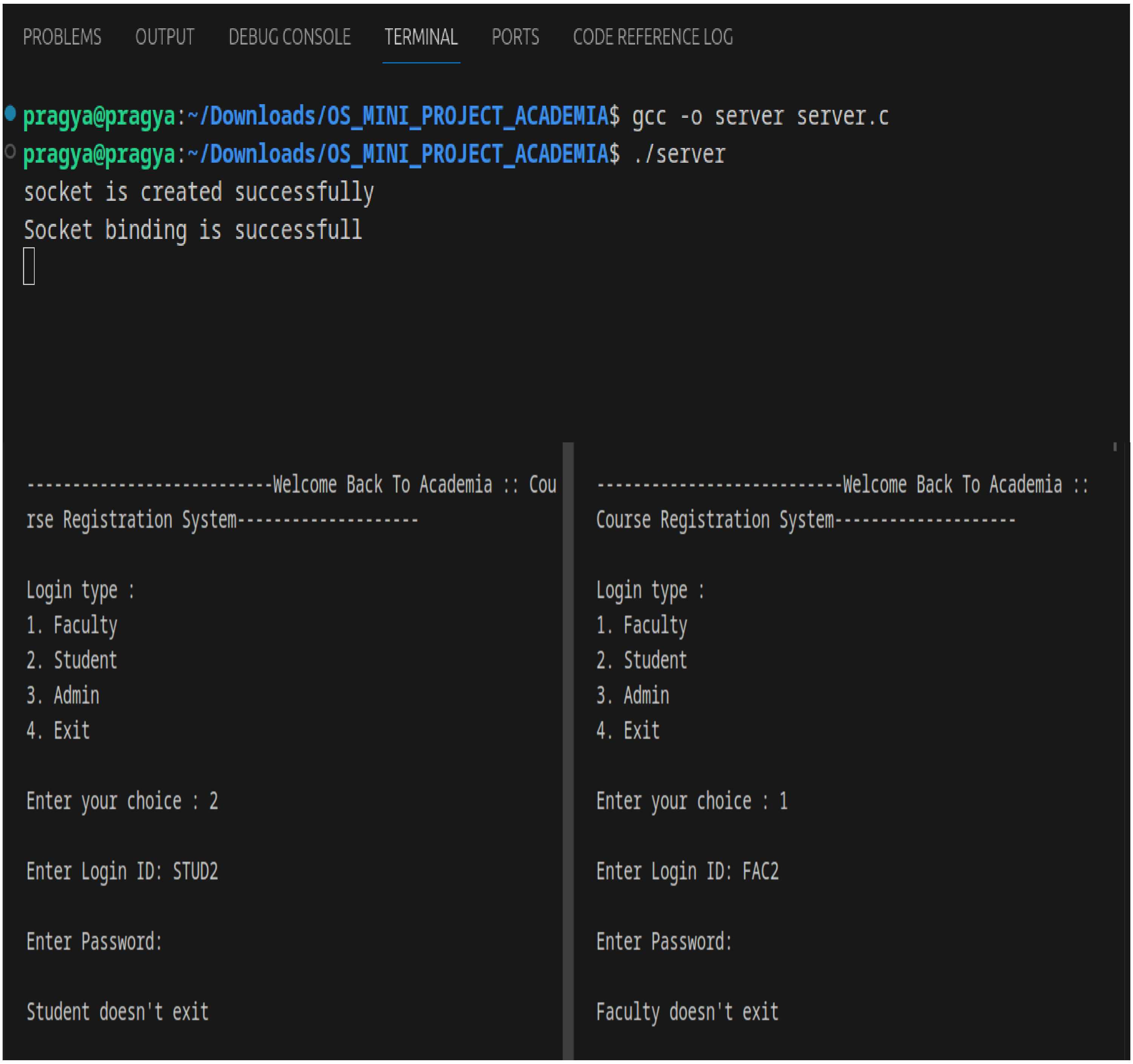
Set_Record.c:

- Opens (or creates) `record.txt` with read/write access and locks the file for writing using `fcntl()`.
- Initializes all record fields (students, faculty, courses) to 0, and their `next_*` IDs to 1.
- Writes the struct to the file.



```
    struct record frecord;
    int fd = open("./Records/record.txt", O_CREAT | O_RDWR, 0777);
    if(fd==-1){
    perror("Cannot open record file ");
    exit(0);
    }
    struct flock mylock = {
        mylock.l_type = F_WRLCK,
        mylock.l_whence = SEEK_SET,
        mylock.l_start = 0,
        mylock.l_len = 0,
        mylock.l_pid = getpid()
    };
    int wrlock = fcntl(fd, F_SETLKW, &mylock);
    if(wrlock==-1){
    perror("Failed to acquire lock ");
    exit(0);
    }
    frecord.students = 0;
    frecord.faculty = 0;
    frecord.courses = 0;
    frecord.next_student = 1;
    frecord.next_faculty = 1;
    frecord.next_course = 1;
    int writeBytes=write(fd,&frecord,sizeof(frecord));
    if(writeBytes<=0){
        perror("Cannot write into the file");
        exit(1);
    }
    mylock.l_type = F_UNLCK;
    int unlock = fcntl(fd, F_SETLK, &mylock);
    if(unlock==-1){
        perror("Unlocking failed");
        exit(0);
    }
    printf("record initialized\n");
    close(fd);
    return 0;
```

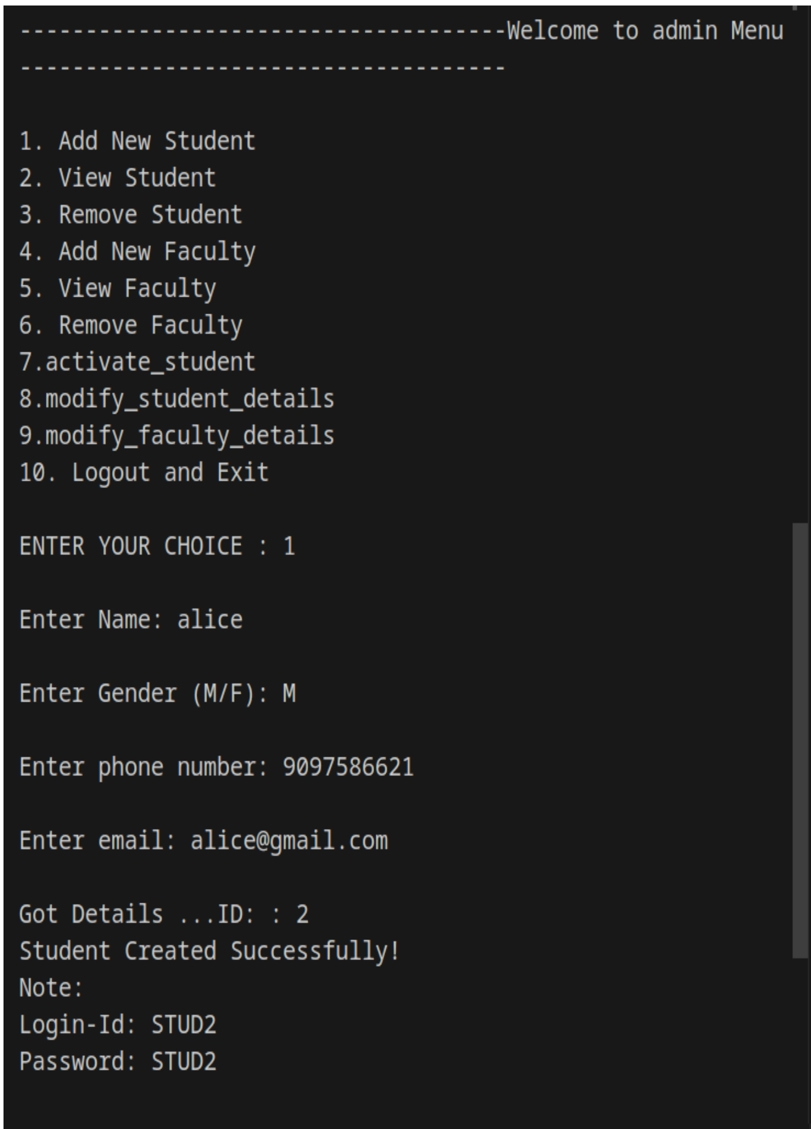
Sample Screenshots:



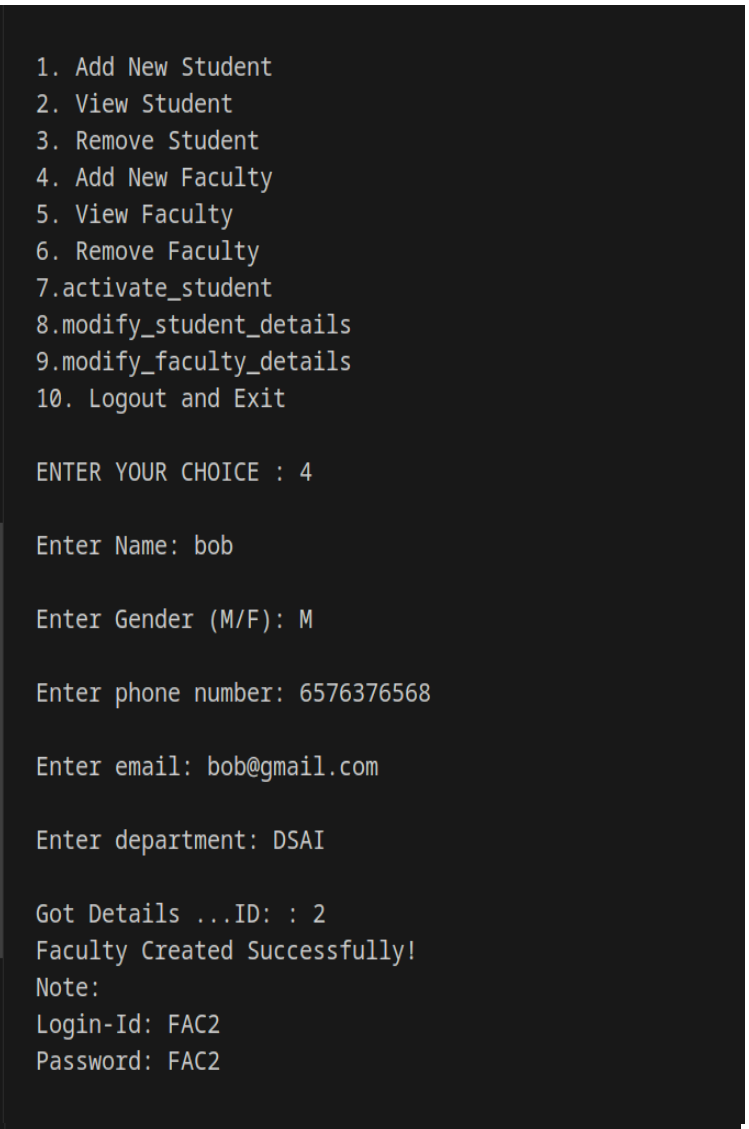
```
pragya@pragya:~/Downloads/OS_MINI_PROJECT_ACADEMIA$ gcc -o server server.c
pragya@pragya:~/Downloads/OS_MINI_PROJECT_ACADEMIA$ ./server
socket is created successfully
Socket binding is successfull
```

```
-------------------------Welcome Back To Academia :: Cou
rse Registration System------------------

Login type :
1. Faculty
2. Student
3. Admin
4. Exit

Enter your choice : 2

Enter Login ID: STUD2

Enter Password:

Student doesn't exit
```

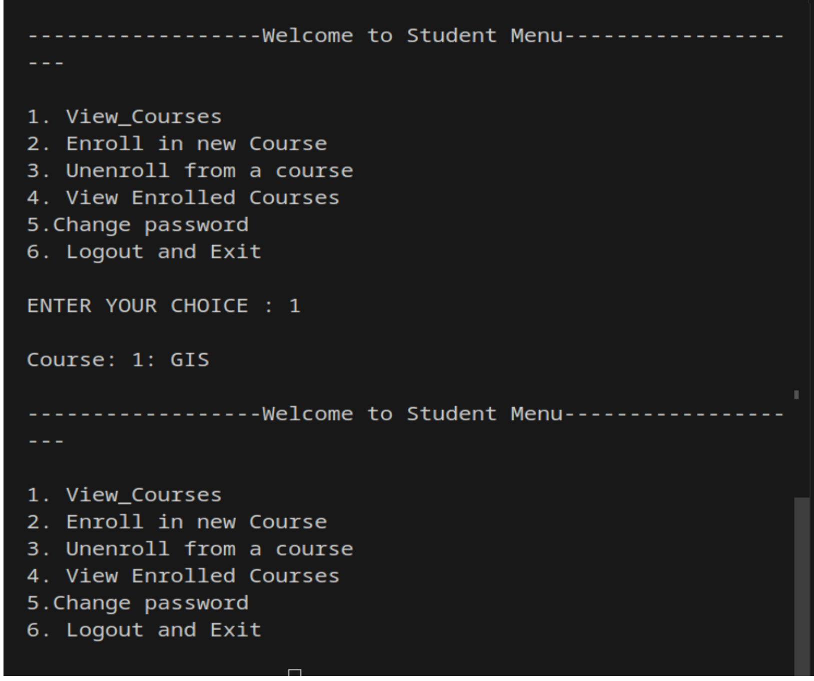
```
-------------------------Welcome Back To Academia ::
Course Registration System------------------

Login type :
1. Faculty
2. Student
3. Admin
4. Exit

Enter your choice : 1

Enter Login ID: FAC2

Enter Password:

Faculty doesn't exit
```



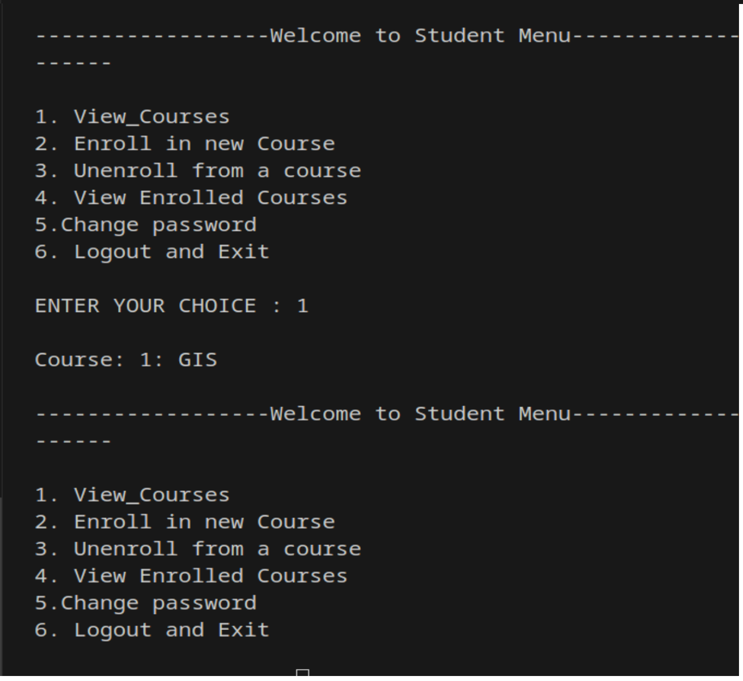
```
-----------------------------------Welcome to admin Menu
-----------------------------------


1. Add New Student
2. View Student
3. Remove Student
4. Add New Faculty
5. View Faculty
6. Remove Faculty
7.activate_student
8.modify_student_details
9.modify_faculty_details
10. Logout and Exit

ENTER YOUR CHOICE : 1

Enter Name: alice

Enter Gender (M/F): M

Enter phone number: 9097586621

Enter email: alice@gmail.com

Got Details ...ID: : 2
Student Created Successfully!
Note:
Login-Id: STUD2
Password: STUD2
```



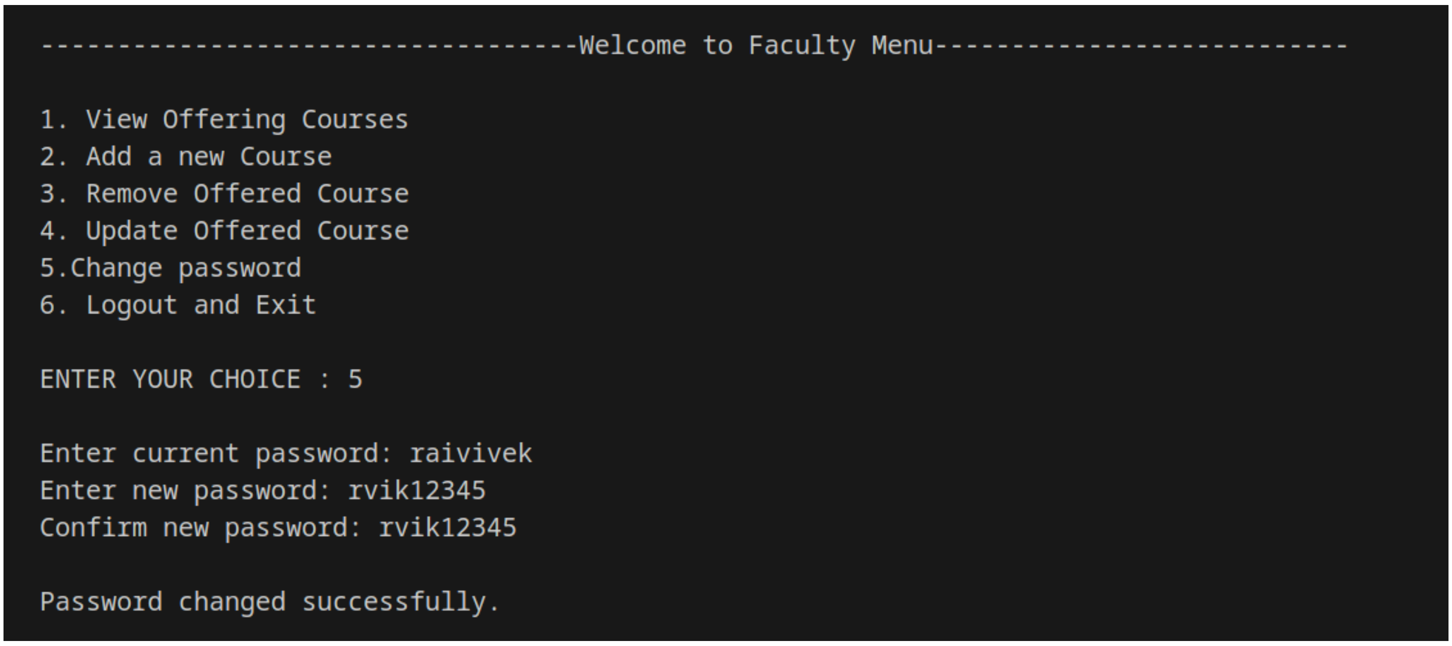
```
1. Add New Student
2. View Student
3. Remove Student
4. Add New Faculty
5. View Faculty
6. Remove Faculty
7.activate_student
8.modify_student_details
9.modify_faculty_details
10. Logout and Exit

ENTER YOUR CHOICE : 4

Enter Name: bob

Enter Gender (M/F): M

Enter phone number: 6576376568

Enter email: bob@gmail.com

Enter department: DSAI

Got Details ...ID: : 2
Faculty Created Successfully!
Note:
Login-Id: FAC2
Password: FAC2
```



```
-----------------Welcome to Student Menu----------------
---

1. View_Courses
2. Enroll in new Course
3. Unenroll from a course
4. View Enrolled Courses
5.Change password
6. Logout and Exit

ENTER YOUR CHOICE : 1

Course: 1: GIS

-----------------Welcome to Student Menu----------------
---

1. View_Courses
2. Enroll in new Course
3. Unenroll from a course
4. View Enrolled Courses
5.Change password
6. Logout and Exit
```



```
-----------------Welcome to Student Menu-------------
------

1. View_Courses
2. Enroll in new Course
3. Unenroll from a course
4. View Enrolled Courses
5.Change password
6. Logout and Exit

ENTER YOUR CHOICE : 1

Course: 1: GIS

-----------------Welcome to Student Menu-------------
------

1. View_Courses
2. Enroll in new Course
3. Unenroll from a course
4. View Enrolled Courses
5.Change password
6. Logout and Exit
```



```
----------------------------------Welcome to Faculty Menu--------------------------

1. View Offering Courses
2. Add a new Course
3. Remove Offered Course
4. Update Offered Course
5.Change password
6. Logout and Exit

ENTER YOUR CHOICE : 5

Enter current password: raivivek
Enter new password: rvik12345
Confirm new password: rvik12345

Password changed successfully.
```