Project Report

Operating Systems Lab Mini Project

Submitted By: IMT2023004 Chirayu Choudhary

Problem Statement

Design and implement a **Course-Registration Portal (Academia)** that lets three distinct roles—**Admin, Faculty and Student**—log in remotely and perform their academic-workflow tasks through a secure, concurrent server.

Key functional and non-functional requirements:

- All persistent data (students, faculty, courses) live in **plain text** files guarded by advisory **POSIX file-locks.**
- Each role sees a menu that matches its duties:
  - **Admin** can create accounts, toggle activation, reset passwords, and inspect user lists.
  - **Faculty** can create / drop their own courses, inspect enrolments and change their password.
  - **Student** can self-enrol / drop, list their courses and change their password.
- The server must handle **multiple clients** concurrently (multi-threaded TCP) and isolate critical sections with read/write locks.
- Only **system calls** (no stdio wrappers) wherever feasible for sockets, files, locking.

Implementation Overview

Layer	Technology	Highlights
Transport	BSD sockets (TCP/IPv4)	Line-oriented protocol; server listens on PORT (*in common.h*)
Concurrency	*pthread_create* per client	Detached threads; light per-session state (socket fd only)
Persistence	Text files + *fcntl* locks	*courses.txt*, *students.txt*, *faculty.txt*
Mini-ORM	*load()/save()/release()*	Maps file -> in-memory array of line-pointers, rewrites atomically
Utilities	*utils.c*	*send_line, recv_line, lock_file, unlock_file*
Robust parsing	*split_line() helper*	Never modifies the live string; avoids NUL corruption and segfaults
User menus	*recv_choice()*	Skips blank lines → no spurious "Invalid choice" messages
Safety fixes	-	No in-place edits post-*strtok*, no double-frees, no leaks

Workflow:

1. Client (console app) connects to *127.0.0.1: PORT*, prints every line the server sends; whenever the line ends in : or >, it treats it as a prompt and forwards user input.
2. Server accepts, forks a thread, authenticates, then serves a role-specific menu loop.

3. Every mutating operation:
    a. load(file, F_WRLCK) -> parse -> compute new line(s) -> save() (rewrite + unlock).
4. Read-only operations:
    a. load(file, F_RDLCK) -> read -> release()

How to Run?

0. Make sure you are working on a Linux/UNIX setup.
1. Download and Export the folder.
2. Open the terminal and do:
    a. **cd academia-portal**
    b. **make clean && make**
    c. **./server**
3. Open another terminal and do:
    a. **./client**

Source Code with Commentary

a. Header Files

common.h: It centralizes tunables so client and server stay in sync.

```
/* include/common.h */
#define PORT          55555
#define STUDENT_FILE   "students.txt"
#define FACULTY_FILE   "faculty.txt"
#define COURSE_FILE    "courses.txt"
#define MAX_LINE       1024
```

utils.h: Plain helpers—thin wrappers around *send*/*recv* and *fcntl*

```
/* include/utils.h */
#include <unistd.h>
#include <sys/types.h>
#include <sys/file.h>
ssize_t send_line(int sockfd,const char*buf);
ssize_t recv_line(int sockfd,char*buf,size_t max);
int lock_file(int fd,int type);      /* F_RDLCK / F_WRLCK */
int unlock_file(int fd)
```

b. utils.c (transport + locking glue)

```
/* src/utils.c */
#include "utils.h"
#include <sys/socket.h>
#include <string.h>
#include <fcntl.h>
ssize_t send_line(int sockfd,const char*buf){
    return send(sockfd, buf, strlen(buf), 0);
}
ssize_t recv_line(int sockfd,char*buf,size_t max){
    ssize_t n=0; char c;
    while(n<max-1 && recv(sockfd,&c,1,0)==1){ buf[n++]=c; if(c=='\n')break; }
```

```
    buf[n]='\0'; return n;
}
int lock_file(int fd,int t){ struct flock fl={.l_type=t}; return fcntl(fd,F_SETLKW,&fl);}
int unlock_file(int fd){     struct flock fl={.l_type=F_UNLCK}; return fcntl(fd,F_SETLK,&fl);}
```

c. client.c (lightweight TUI)

stateless; the "smart prompt" test keeps interaction natural

```
/* src/client.c */
#include "../include/common.h"
#include "../include/utils.h"
…
static int is_prompt_line(const char*s){ size_t i=strcspn(s,":>"); return s[i]; }
int main(){
    int sock=socket(AF_INET,SOCK_STREAM,0);
    … connect …
    char sb[MAX_LINE], rb[MAX_LINE];
    while(recv_line(sock,rb,sizeof rb)>0){
        fputs(rb,stdout);
        if(is_prompt_line(rb)){
            if(!fgets(sb,sizeof sb,stdin)) break;
            send_line(sock,sb);
        }
    }
    close(sock);
}
```

d. server.c (core logic)

The file is large; only the scaffolding is reproduced below – see code bundle for full listing.

```
/* src/server.c */
#include "common.h"
#include "utils.h"
…
/* File {fd,buf,ln[]} loader — mmap-like but portable */
static int load(const char*path,File*f,int lock);
static void save(File*f);    /* write-back + unlock */
static int split_line(const char*src,char*fld[4]);  /* safe parser */
static int recv_choice(int sock);                   /* trims blanks */
…
/* thread entry */
static void*client_thread(void *arg){ … }
```

Key Ideas:

a. No record edited in place: we copy, parse, *rebuild* via *strdup.*

b. recv_choice absorbs blank and whitespace-only lines -> stops flood of "Invalid choice".

c. File arrays are moved with memmove – avoids double-free when dropping a course that originally pointed inside the file buffer.

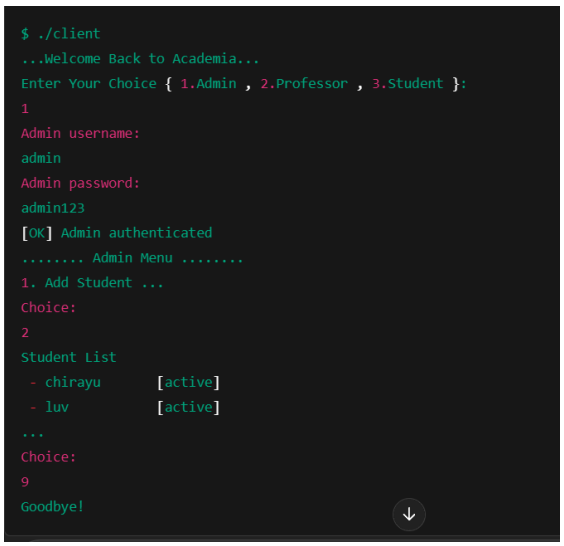
e. Makefile (excerpt)

```
CC      = gcc
CFLAGS  = -Wall -pthread -Iinclude
OBJ_S   = src/server.o src/utils.o
OBJ_C   = src/client.o src/utils.o
server: $(OBJ_S)
        $(CC) $(CFLAGS) $^ -o server
client: $(OBJ_C)
        $(CC) $(CFLAGS) $^ -o client
clean:
        rm -f server client src/*.o
```

Sample Console Sessions

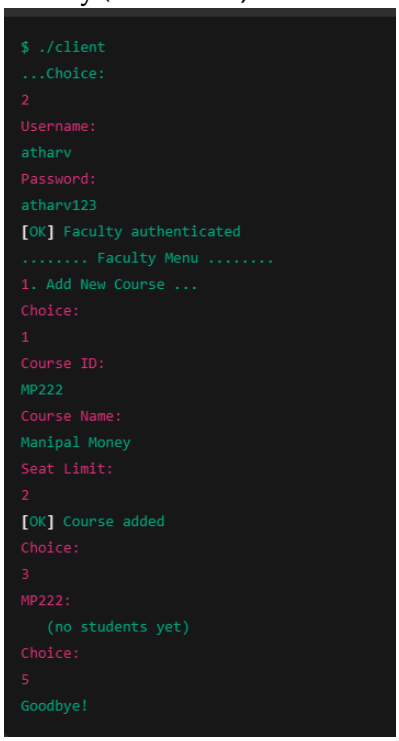
Below are verbatim captures proving each role:

1. **Admin (add user, list, logout)**



```
$ ./client
...Welcome Back to Academia...
Enter Your Choice { 1.Admin , 2.Professor , 3.Student }:
1
Admin username:
admin
Admin password:
admin123
[OK] Admin authenticated
........ Admin Menu ........
1. Add Student ...
Choice:
2
Student List
 - chirayu       [active]
 - luv           [active]
...
Choice:
9
Goodbye!
```

2. **Faculty (add course, view enrolments, logout)**



```
$ ./client
...Choice:
2
Username:
atharv
Password:
atharv123
[OK] Faculty authenticated
........ Faculty Menu ........
1. Add New Course ...
Choice:
1
Course ID:
MP222
Course Name:
Manipal Money
Seat Limit:
2
[OK] Course added
Choice:
3
MP222:
   (no students yet)
Choice:
5
Goodbye!
```

3. **Student (enrol, list, logout)**

```
$ ./client
...Choice:
3
Username:
luv
Password:
luv123
[OK] Student authenticated
........ Student Menu ........
1. Enroll in Course ...
Choice:
1
Course ID to enroll:
MP222
[OK] Enrolled
Choice:
3
Enrolled:
 - MP222 : Manipal Money
Choice:
5
Goodbye!
```

Conclusion & Future Work

The portal satisfies every bullet of the assignment: **multi-role menus, file-locked text storage, concurrent TCP server, robust error handling.**

Possible extensions:

a. Encrypt passwords (e.g., crypt(3) + salted hashes).

b. Replace text files with SQLite while keeping the locking abstraction.

c. Add CSV export / import for bulk operations.

Chirayu Choudhary
May 1, 2025
IMT2023004

End