**基于Linux的多用户航班购买系统**

项目背景

现在人们的日常出行都会搭乘交通工具，其中就包括火车、高铁、飞机等等，搭乘这些交通工具就需要先买票才能乘坐，所以就需要一个能够让用户能够自助买票的系统。这个多用户航班购买系统可以实现不同用户同时进行登陆购票，改签，退票以及取票之类的相关基础操作，还能让用户之间进行一些互动，比如用户之间可以进行聊天。

管理员使用管理员账号登录时可以进行航班的查询，录入和删除操作。

目录

1、环境要求…………………………………………………………………………………………………………（4）

2、项目功能…………………………………………………………………………………………………………（4）

3、项目技术点概述…………………………………………………………………………………………… （6）

4、需求分析…………………………………………………………………………………………………………（6）

5、项目实施平台以及技术分析 ……………………………………………………………………… （6）

6、系统框架…………………………………………………………………………………………………………（8）

7、项目细节分析…………………………………………………………………………………………………（9）

**1、环境要求**

项目所需软件：虚拟机、Linux操作系统、vscode

**2、项目功能**

项目的功能主要可以分成两个部分：用户部分和管理员部分。

用户部分：用户可以选择登录、注册、找回密码功能。当用户登录成功后可以使用：

功能1：航班列表，可以查看所有航班；

功能2：查询航班，通过匹配目的地、出发日期、机型、价格查询符合条件的航班；

功能3：快速查询，通过输入航班号快速查询该航班；

功能4：订购航班，用户可以购买多趟航班，也可以给好友购票，但同一航班只能购买一张票；

功能5：已购航班，显示用户已购买的航班；

功能6：退票改签，退票时需要扣除相应的手续费，改签则不需要；

功能7：查询余额，查询账户的余额；

功能8：充值账户，有相应的充值优惠活动；

功能9：修改信息，可以修改用户的相关信息；

功能10：聊天功能，用户可以添加和删除好友，可以进入聊天室聊天，已在聊天室的会收到好友进入聊天室的提示

功能11：取票服务，打印用户已购买的航班，若该航班已被删除，则提示用户去办理退票或改签业务；

功能12：退出登录，用户退出登录，并将用户的信息保存至本地文件中。

用户注册就需要用户输入相应的个人信息来完成注册，如果用户忘记密码需要找回密码就需要输入一些个人信息来进行验证，验证成功后就可以修改密码。

管理员部分：管理员只有一个账号为：root，密码为：123456。并且管理员账号不能被修改

管理员登录成功后可以使用：

功能1：航班列表，查看所有航班；

功能2：查询航班，通过匹配目的地、出发日期、机型、价格查询符合条件的航班；

功能3：快速查询，通过输入航班号快速查询该航班；

功能4：录入航班，输入航班的相关信息，判断是否为已存在的航班，如果该航班已存在，则会修改该航班信息，如果该航班不存在，则新添加航班，并进行新航班信息的录入。完成修改/录入后，将新数据保存至文件中；

功能5：删除航班，删除对应的航班，并将剩余的航班信息保存至文件中，同时会对拥有该次航班并且在线的用户进行广播通知；

功能6：退出登录

可以在服务器端输入quit来主动退出服务器进行维护，服务器主动退出时会保存数据到文件并释放内部的资源。

**3、项目技术点概述**

该项目运用到的技术主要有：字符串的相关操作，如：控制字符串的输出格式，字符串的拼接，字符串的比较；文件IO的相关操作，如：文件的创建，文件的打开和关闭，文件的读写，Json格式的文件的读写操作；链表的相关操作，如：初始化头节点，初始化嵌套链表头节点，节点数据域和指针域的赋值，节点的插入，节点的删除，链表的销毁；线程的相关操作，如：线程的创建和退出；互斥锁的使用；网络编程的相关操作，如：套接字的创建，绑定IP地址、端口号、协议到套接字上，设置监听套接字，接收和发送数据；

**4、需求分析**

可以实现普通用户和管理员登录；普通用户可以查看所有航班信息、通过指定信息查询航班信息、购票、查看已购买的票、退票、改签、查询余额、充值、修改个人信息、取票、用户之间聊天等功能；管理员可以查看所有航班信息、通过指定信息查询航班信息、录入航班、删除航班等功能。

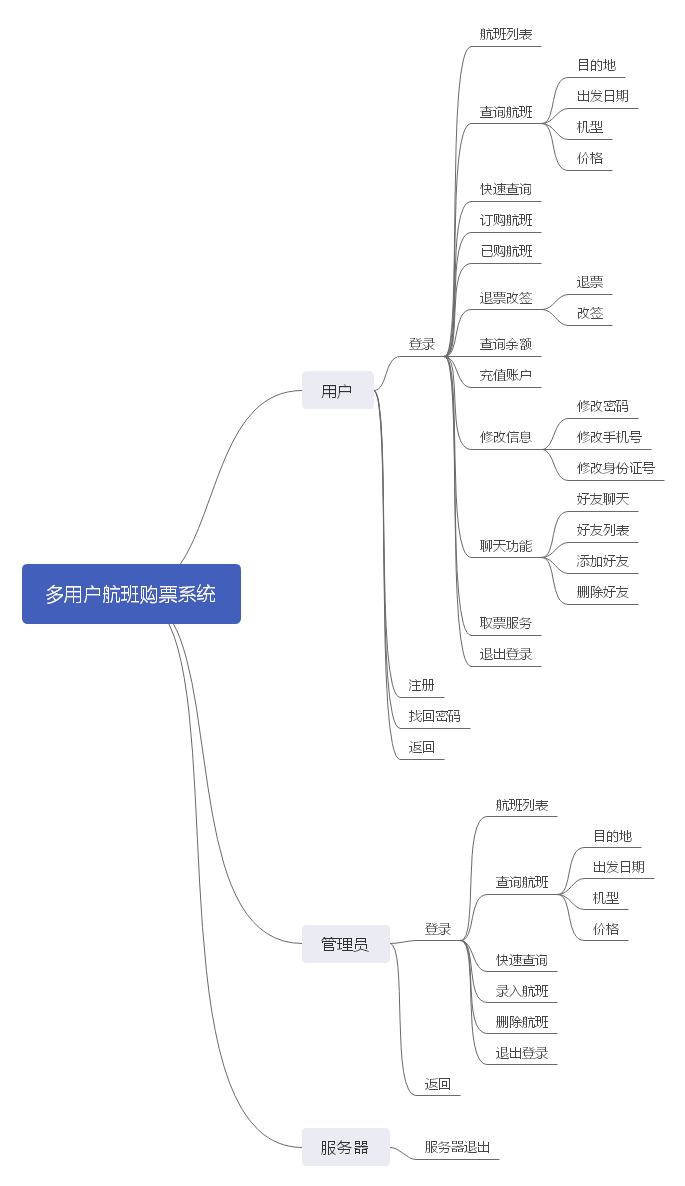
**5、项目实施平台以及技术分析**

该项目是在Linux平台下，基于C语言实现的，一开始服务器先运行时，会先初始化链表的头节点，然后使用文件IO的相关操作，将存放着Json格式数据的文件中的数据读取出来，再通过Json库函数处理读取到的Json格式数据，将其存入链表的新节点中，之后将新节点插入到链表尾。

客户端可以不断向服务器发送数据，服务器端需要不断接收多个客户端的信息，因此就需要用到多任务并发，多任务并发有多线程和多进程，由于多线程消耗的资源比多进程要少得多，所以本系统使用的的多线程，通过一个线程对应一个客户端来接收客户端的信息并做出相应的操作。同时由于是服务器与客户端之间进行通信就需要用到网络编程的相关操作。

该项目所有客户端接收到的界面都是由服务器来发送的，服务器是通过读取本地文件中的界面文件与通过字符串的拼接来获取要发送的界面并发送给客户端的。这里就需要使用到网络编程和字符串操作及文件IO相关操作来进行实现。

接收到的数据需要需要进行分析处理，就需要用到字符串的比较。执行用户的命令，如果是查询航班则需要用到链表的遍历，买票则是节点的插入，退票则是节点的删除，并且访问临界资源前需要用互斥锁先上锁，等操作完成之后再解锁。在用户退出的时候会进行用户信息的文件数据更新。在服务器主动退出时，也会进行服务器内数据信息的文件存储更新操作，这时候就需要用到文件IO的相关操作来实现文件内数据的更新。

**6、系统框架**

**7、项目细节分析**

// 用户信息

typedef struct user\_info

{

    // 数据域

    // 套接字（初始值为-1）

    int connfd;

    // 聊天室状态（初始值为0）

    int istalk;

    // 身份证

    char id[ID\_MAX\_LEN + 1];

    // 是否为管理员

    int isroot;

    // 用户名

    char name[NAME\_MAX\_LEN + 1];

    // 密码

    char pwd[PWD\_MAX\_LEN + 1];

    // 手机号

    char tel[TEL\_MAX\_LEN + 1];

    // 朋友列表使用链表存取

    pt\_friend myFriend;

    // 航班列表使用链表存取

    pt\_myflg myflg;

    // 余额

    int cash;

    // 是否为vip

int vip;

    // 指针域

    struct list\_head list; //（前后关系）

} st\_uinfo, \*pt\_uinfo;

上图是本项目的用户信息结构体，其中name用于存储用户的名字（用户添加好友和登录注册是用到，因为用户名唯一），pwd用于存储用户的密码，id用于存储用户的身份证号码（用户购票时需要用到，因为身份证号码是唯一的），tel用于存储用户的电话号码（用户找回密码是需要用到，因为电话号码是唯一的），isroot用于存储用户是否为管理员，istalk用于存储用户是否在聊天室中，cash用于存储用户的账户余额，vip用于表示该用户的vip等级（0级到9级），connfd用于存储客户端连接到服务器时的套接字（在用户每次成功登录账号时都会重新赋值，在每次用户退出登录时都会赋值为-1，表示该用户不在线）。

myFriend为用户信息链表内的用于存放好友的链表，内部存放了好友的用户名

// 用户好友列表

typedef struct my\_friend

{

    // 数据域

    char my\_friend[32];

    // 指针域

    struct my\_friend \*next;

} st\_friend, \*pt\_friend;

myflg为用户购买的航班的信息链表，内部存放的信息为航班号（用于查询航班信息），购买的航班的原价（作为改签航班的依据），购买该航班所使用的价格（作为退票的依据）。

// 用户购买的航班

typedef struct my\_flight

{

    // 数据域

    char flight[20]; // 航班

    int o\_price; // 购买原价

    int price; // 购买价格

    // 指针域

    struct my\_flight \*next;

} st\_myflg, \*pt\_myflg;

// 航班信息

typedef struct flight\_list

{

    // 数据域

    // 航班号 唯一

    char code[20];

    // 余票

    int num;

    // 金额

    int cash;

    // 飞机类型

    char type[5];

    // 起飞时间

    char time[9];

    // 班期

    char date[13];

    // 起点站

    char s\_plane[3];

    // 终点站

    char e\_plane[3];

    // 指针域

    struct list\_head list; //（前后关系）

} st\_flg, \*pt\_flg;

上图是本项目的航班信息结构体，其中code用于存储航班号（唯一），num用于存储该航班的剩余票数，cash用于存储该航班的价格，type用于存储航班的机型，time用于存储航班的起飞时间，date用于存储航班的班期，s\_plane和e\_plane分别用来存储航班的起点站和终点站。