Computer Science and Application 计算机科学与应用, 2021, 11(10), 2478-2495
Published Online October 2021 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2021.1110252

Hans汉斯

基于Linux的图形使用者界面火车售票系统

廖文雄[1]，庄智颖[1]，林明灶[2]

[1]惠州学院电子信息与电气工程学院，广东 惠州
[2]东莞理工学院，广东 东莞

收稿日期：2021年8月12日；录用日期：2021年10月7日；发布日期：2021年10月14日

摘 要

本文阐述了一款基于Linux操作系统与Android相结合的火车售票系统。在Linux上使用C语言进行服务端的编写，通过GCC、Makefile对代码进行编译，并利用GDB和tcpdump调试工具进行调试程序，同时使用Linux系统上的pycharm软件进行python爬虫的编写，对12306网站的车票进行爬取。此外，使用Windows系统上的Android studio软件对安卓APP进行编程，将Linux服务端通过socket通信与安卓APP进行通信。最后实现了一个服务端、多个客户端的火车售票系统，其具有爬取12306网站的车票信息、登录/注册账号、用户买票/退票/改签、管理员修改车票/用户信息等功能。

关键词

Linux系统，Socket网络编程，Android，TCP/IP，python爬虫

GUI Train Ticket System Based on Linux System

Wenxiong Liao[1], Chihying Chuang[1], Mingtzau Lin[2]

[1]Department of Electronic Information and Electrical Engineering, Huizhou University, Huizhou Guangdong
[2]Dongguan University of Technology, Dongguan Guangdong

Received: Aug. 12th, 2021; accepted: Oct. 7th, 2021; published: Oct. 14th, 2021

Abstract

This paper describes a GUI train ticket system based on Linux system via combining Linux with Android. The related program of the server is written on Linux through C language, the code is

文章引用: 廖文雄, 庄智颖, 林明灶. 基于 Linux 的图形使用者界面火车售票系统[J]. 计算机科学与应用, 2021, 11(10): 2478-2495. DOI: 10.12677/csa.2021.1110252

compiled through GCC and Makefile, and the program is debugged by GDB and tcpdump. Meanwhile, the software named pycharm from Linux is utilized to compile python in order to scrap the ticket information of the 12306 website. Besides, an APP, compiled by Android Studio from Windows, communicates with Linux server through socket. Therefore, a GUI train ticket system with a server and multiple clients is realized. It has functions that include crawling the ticket information of the 12306 website, logging in/registering an account, buying/refunding/changing tickets, and modifying the ticket/user information by the administrator.

Keywords

Linux System, Socket, Android, TCP/IP, Web Scraping with Python

Copyright © 2021 by author(s) and Hans Publishers Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/
Open Access

1. 引言

随着科技的不断发展以及互联网的进步，许多在线下才能完成的事情，已可以通过互联网的帮助，在手机或者电脑上完成，而为人们节省下许多时间。早期购买火车票时，都需要到各地的火车站排队，但是在网上购票系统出现后，人们便能通过轻松地点击手机或鼠标，就将自己所需的车票购买下来，如此提高了购票效率，为生活带来很大的便利。

由于 Linux 系统具有免费和开源等特性，以及拥有多任务和丰富的网络接口(如 TCP 和 UDP)和更加安全的用户及文件权限策略，再加上 Android 是如今主流的手机系统，所以使用 Linux 和 Android studio 进行开发与编程，可以提供良好和稳定的服务端，也能实现一个操作便捷、随时使用的 GUI (图形操作界面)的客户端，以便用户能随时随地进行操作。

本设计基于 Linux 系统下来实现一个具有图形使用者界面的火车售票系统，其附有：用户买票/退票/改签、用户注册/登录、管理员管理用户/车票信息、通过 python 爬虫的方式将 12306 中的车票载入本地、操作简捷的安卓 APP 等功能。根据以上的需求，拟定整体的设计思路与开发流程之后，确定整个设计分为服务端和客户端两个部分：服务端负责接收和处理客户端传递过来的信息，车票、用户信息的存储，以及大部分功能的实现；客户端主要负责信息的显示，以及获取用户输入的信息，通过 socket 通信将信息传递给服务端进行处理。整体设计框架如下：

1) Linux 服务端有多个线程，分别是主线程负责接收客户端的连接，然后将连接的客户端标志写入连接状态表中；分线程 1 负责监听连接状态表中的客户端是否有发送消息过来，如果有就将消息写入数据缓冲区；还有 n 个分线程 2 负责监听数据缓冲区是否具有数据，如果有数据就将数据进行对应的处理。

2) 安卓客户端大体是分为不同的活动，以及一个为了长时间连接客户端，以及服务端能检测到客户端保持在线，而持续发送的心跳包的服务。服务是在运行 APP 初就开始持续执行的，活动的运行则是在主活动运行之后，根据用户不同的操作，切换到不同的活动中。

2. 程序开发环境介绍

本设计进行开发所使用的程序，简单的概述如下：

1) GCC (GNU 编译器套件)

GCC (GNU Compiler Collection，GNU 编译器套件)整体流程分为：预处理、编译、汇编、链接四个步骤，首先调用 cpp 进行预处理，对.c 文件中的包含和宏定义进行分析，生成.i 文件。第二步调用 cc1 进行编译，生成.s 的汇编文件。第三步调用 as 进行汇编，生成.o 文件。最后调用 ld 将.o 文件进行链接，生成可执行文件[1]。

2) Makefile 编译

使用 Makefile 进行自动化编译，写好 Makefile 的文件，输入一个 make，进行自动编译。Makefile 执行多条 GCC 编译语句，通过不同的语句进行复杂的一些操作，同时还可以加入 shell 脚本，执行一些操作系统的命令[2]。

3) GDB 调试工具

Linux 工具在 GCC 编译时加入-g 这个编译选项，通过 GDB 调试，接着在 Linux 终端输入 gdb，进入 GDB 的工作界面，通过一些基础的命令来调试程序。

4) tcpdump 调试工具

tcpdump 根据使用者的需求对网络上的数据包进行截获的工具，是 Linux 系统中的一个调试工具。

5) pycharm

pycharm 使用 python 语言进行高效开发的工具，具有调试、代码跳转、智能提示、版本控制等功能。本设计创建好 project 和文件，将爬虫的代码写入文件，编译器自动地进行编译。

6) Android studio

Android studio 前端布局的实时预览、自带的安卓模拟器、第三方的 sdk 和开源项目、gradle 构建系统，可以通过拖动控件就将其加入布局中等功能[3]。

3. 程序技术要点原理

本设计使用的程序技术概述如下：

1) socket 网络通信编程

本设计在 Linux 部分使用了 TCP 协议[4]，主要依序调用 socket()、bind()、listen()、int accept()等函数，连接上客户端之后，调用 sendto()、sendmsg()、send()之类的函数就能发送数据到客户端，也可以通过调用 recvfrom()、recvmsg()、recv()这类的函数接收客户端的数据。最后，在不需要传输数据后，通过调用 int close(int sockfd)关闭 socket 对象。

2) 多线程的使用

使用多线程技术，就可以给 accept()函数另外创建一条线程，能够一直保持等待接收客户端的连接请求，不会影响到与已连接的客户端进行通信和对数据进行处理[5]。

3) 多线程并发处理与多线程同步

使用复杂点的多线程难免会遇到并发的问题，需要对线程进行同步的处理。

4) IO 多路复用服务端模型

图 1 展示了 IO 多路复用服务端模型。使用 IO 多路复用服务端模型的好处在于相比于单进程或者是单线程的服务端，实现多个客户端连接和收发数据；可以一个线程负责接收客户端请求连接，一个线程负责接收数据，将两个不同的功能分开，减小耦合[6]。

5) 活动(Activity)

安卓的活动内容有活动的创建、在 AndroidManifest.xml 中注册活动、活动之间的跳转、以及活动的生命周期，图 2 展示了活动的生命周期[7]。

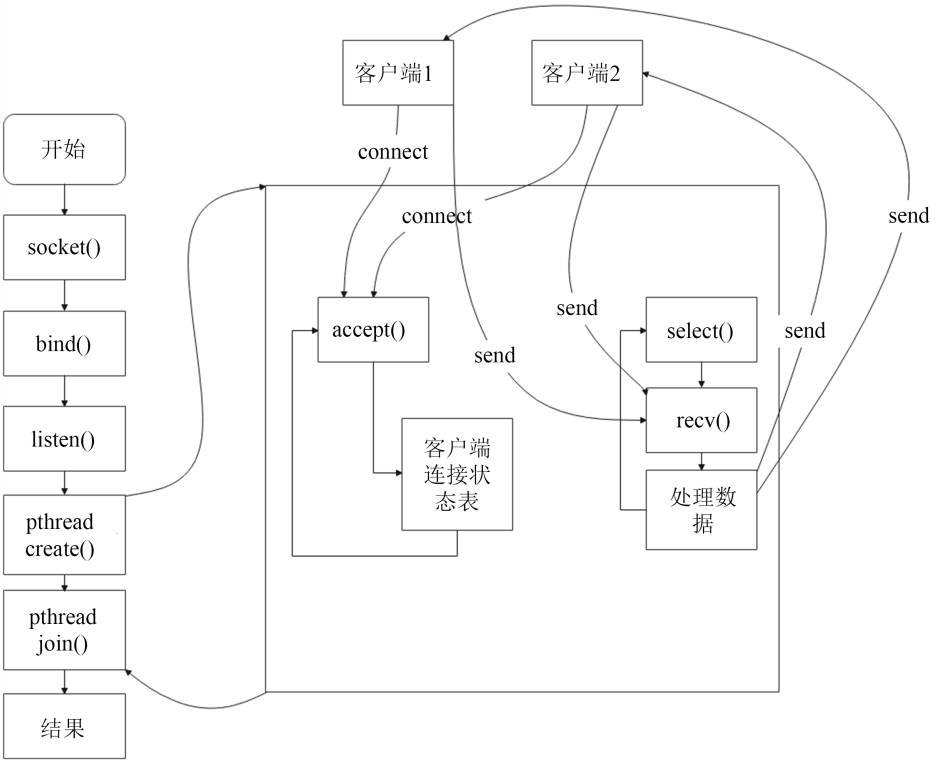

Figure 1. IO multiplexing server model
图 1. IO 多路复用服务端模型图

6) 线性布局(LinearLayout)和相对布局(RelativeLayout)

通过 android:orientation 属性来指定是水平排列还是竖直排列。当其属性设置为 android:orientation="horizontal"则为水平排列，若 android:orientation="vertical"则为竖直排列。相对布局用顶部、底部、中心、右边、左边或者父布局或者其他控件等等为基准来设置位置[8]。

7) 服务

Android 的服务用于后台用户切换到另一个 APP，APP 不被系统杀死，服务不会中断[8]。服务和活动需要在 AndroidManifest.xml 文件中注册之后才能使用。

8) 心跳包

安卓 APP 在后台被直接杀死，无法正常地断开 socket 的连接，为了与 Linux 服务端保持长连接，同时也让 Linux 服务端能知道安卓 APP 是否还在保持连接状态，需要加入心跳包的功能。安卓客户端需要定时发送一个数据包给 Linux 服务端，Linux 服务端接收到心跳包就判定安卓客户端还在保持连接。如果超过一定时间 Linux 服务端没有收到心跳包的话，判定安卓客户端已经断开了，将 socket 的连接给断开。

9) python 爬虫

requests 模块是 python 中原生的一款基于网络请求的模块，用于模拟浏览器发请求，可将请求后得到的数据进行进一步分析来得到想要的数据[9]。UA (User-Agent)是用户代理，不使用 UA 伪装的话，默认的 UA 很容易会被网站监测到是爬虫，然后请求不到数据，所以需要将爬虫通过 UA 伪装，伪装成浏览器，就可以请求到网站的数据了[10]。

10) GTK+

GTK+是图形界面库，创建窗口、显示控件、信号注册函数等功能[11]。

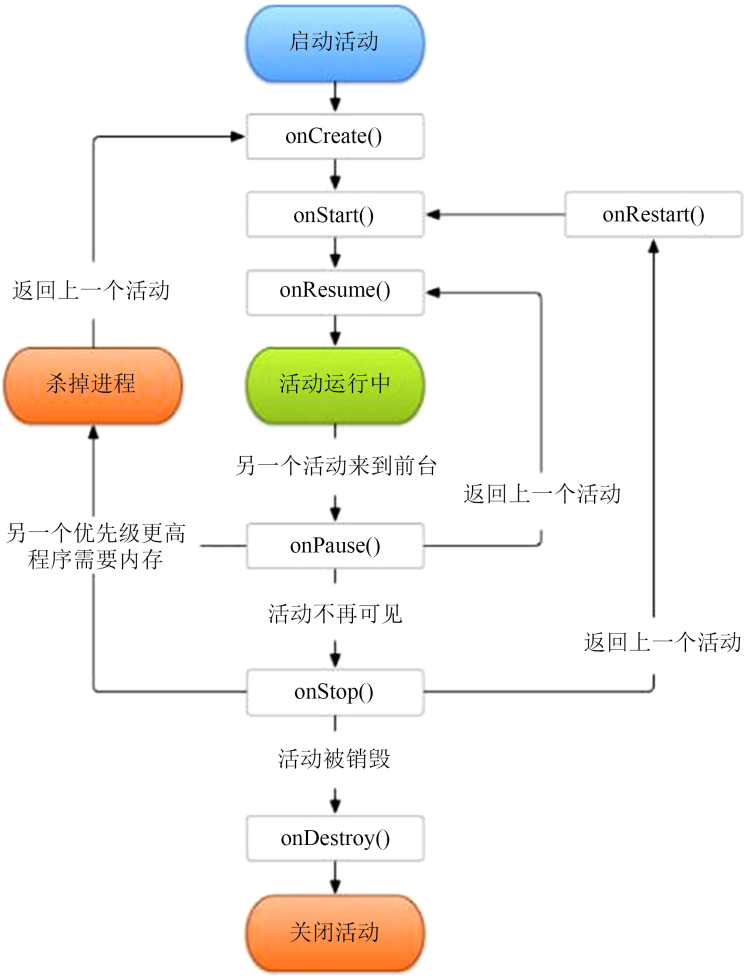

Figure 2. Life cycle of the activity
图 2. 活动的生命周期

4. 程序设计

本设计使用的程序开发环境与技术要点之后，接着分成服务端与客户端两部分来说明本设计的程序设计。

4.1. Linux 服务端程序设计

服务端主要分为 GTK 线程、销毁线程、接收客户端连接、接收客户端数据和处理客户端数据并响应等五个线程，其总体框架如图 3 所示。在服务端初始化时，先读取本地文件，将所需信息进行读取；创建 GTK 线程，以便将信息显示到 GTK 的图形界面上；创建销毁线程，等待销毁信号，将服务端运行时申请的内存释放，以及保存所需数据；创建接收数据线程，接收客户端的数据；创建处理线程，处理来自客户端的数据；然后在运行函数部分使用主进程对客户端的连接请求进行监听，将申请连接的客户端接入。这些线程简单说明如下：

1) GTK 线程

初始化 GTK 的数据后，调用 gtk_main()函数等待有关 GTK 的信号，当其他线程调用有关 GTK 的函数后，会在图形界面上显示信息。

2) 销毁线程

等待 Ctrl + c 的信号，当按下 Ctrl + c 后，或者操作者按下退出后，销毁线程开始执行，对之前运行时申请的内存进行释放，并将相关数据保存到文件中。

3) accept()接收客户端连接的线程

等待连接部分的主体使用 accept()函数对预设好的 ip 地址和端口进行监测，当有客户端进行连接时，就可以得到客户端的套接字标识。以及当每个客户端连接时，给每个客户端设定一个初始的时间，以便后面对客户端发送心跳包进行计时。最后将接收到的客户端的套接字标识写入环形缓冲区 1。

4) select()接收客户端数据的线程

接收客户端数据部分使用 select()函数对已连接的客户端进行监听，监听是否有数据包或者心跳包传输给服务端。客户端超过预定的时间都还没有传输心跳包给服务端，该客户端断开 socket 连接，将其移除监听列表。当收到客户端的数据时，先分析数据包的结构，是否符合自己设定好的通信协议，对数据进行接收，并写入环形缓冲区 2；若是不符合，则回传一条消息给客户端，告诉客户端数据错误。

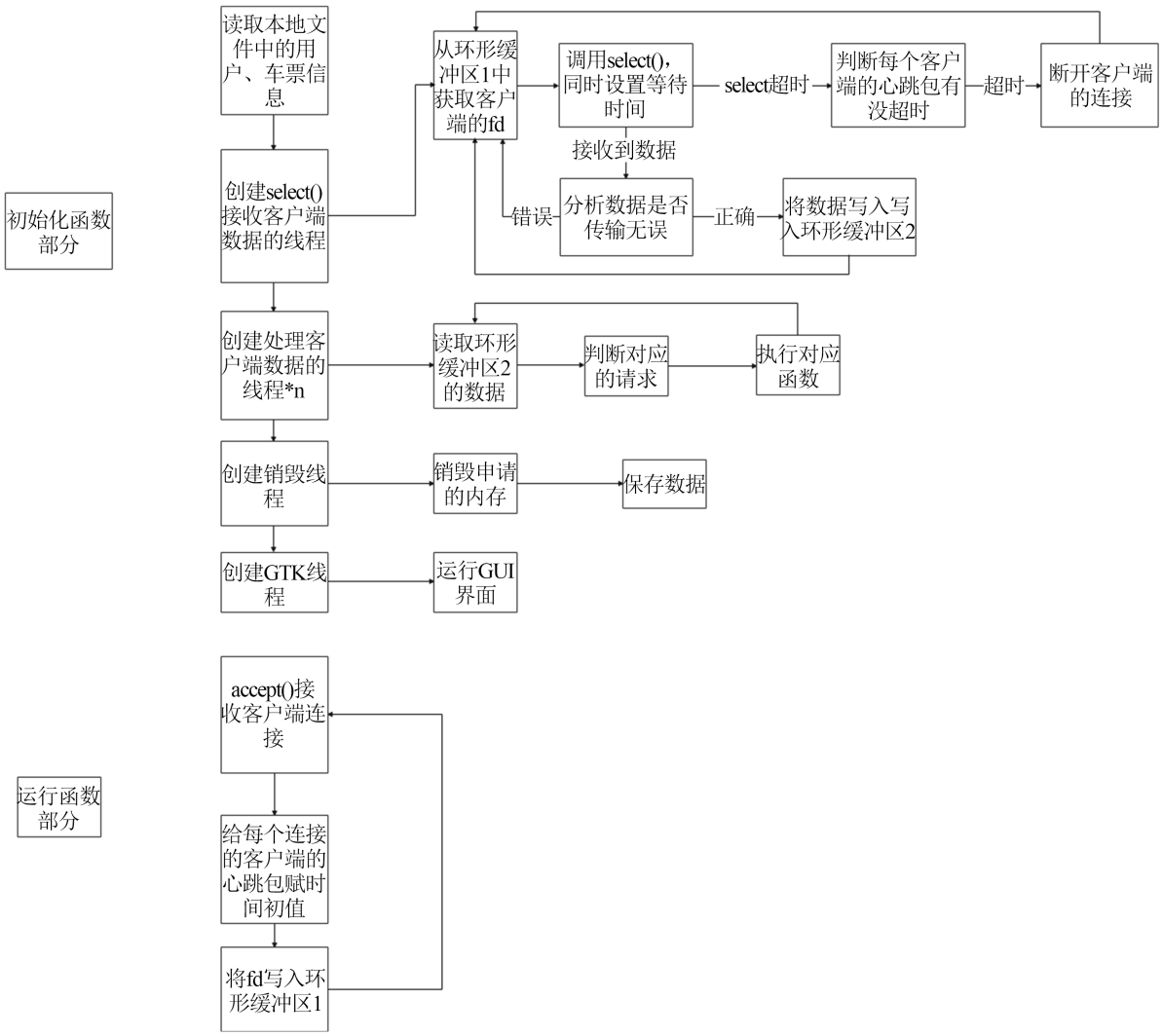

Figure 3. Framework of the server
图 3. 服务端总体框架

5) 处理客户端数据并响应的线程

对来自客户端的不同的数据进行处理，然后将处理结果的数据再发送回客户端。大致分为处理用户数据部分和处理车票数据部分两大类。

在处理用户数据部分，主要是对用户的数据进行增加、删除、修改等操作。用户数据结构体如下：

```
struct user {
    char user_id[USER_ID_SIZE];
    char passwd[USER_PASSWD_SIZE];
    char name[NAME_SIZE];
    char citizen_id[CITIZEN_ID_SIZE];
char phone_num[PHONE_NUM_SIZE];
struct list_head list;
};
```

分别为用户的 ID、密码、姓名、身份证号以及手机号，最后一个 list 负责将每个结构体变量串联进一个链表中。用户数据相关应用函数如下：

用户信息链表初始化函数：

```
int user_init(struct list_head *user_head) ;
```

用户信息链表销毁函数：

```
int user_destroy(struct list_head *user_head) ;
```

用户注册登录：

```
int user_login(struct data_packet data, int clientfd, struct interaction *interaction) ;
```

用户注册函数：

```
int user_register(struct data_packet data, int clientfd, struct interaction *interaction) ;
```

用户信息修改函数：

```
void change_user(struct data_packet data, int clientfd, struct list_head *user_head) ;
```

用户信息删除函数：

```
void delete_user(struct data_packet data, int clientfd, struct list_head *user_head) ;
```

用户信息添加函数：

```
void add_user(struct data_packet data, int clientfd, struct list_head *user_head) ;
```

将用户信息保存到本地文件函数：

```
int write_user_file(struct list_head *user_head) ;
```

将用户信息从本地读取函数：

```
int read_user_file(struct list_head *user_head) ;
```

在处理车票数据部分，主要是对车票的数据进行管理，分为用户已经购买的车票数据和还未购买的车票数据，数据结构如下：

```
struct ticket {
    char origin_place[ORIGIN_PLACE_SIZE];
    char destination[DESTINATION_SIZE];
    char train_number[TRAIN_NUM_SIZE];
    char departure_time[DEPARTURE_TIME_SIZE];
```

```
    char arrival_time[ARRIVAL_TIME_SIZE];
    char type1_price[PRICE_SIZE];
    char type2_price[PRICE_SIZE];
    char type3_price[PRICE_SIZE];
    char type4_price[PRICE_SIZE];
    char date[TO_DATE_SIZE];
    char ticket_type1_num[TICKET_NUM_SIZE];
    char ticket_type2_num[TICKET_NUM_SIZE];
    char ticket_type3_num[TICKET_NUM_SIZE];
    char ticket_type4_num[TICKET_NUM_SIZE];
    struct list_head list;
};
struct user_ticket {
    char id[USER_TICKET_ID_SIZE];
    char name[NAME_SIZE];
    char citizen_id[CITIZEN_ID_SIZE];
    char phone_num[PHONE_NUM_SIZE];
    char origin_place[ORIGIN_PLACE_SIZE];
    char destination[DESTINATION_SIZE];
    char train_number[TRAIN_NUM_SIZE];
char departure_time[DEPARTURE_TIME_SIZE];
    char arrival_time[ARRIVAL_TIME_SIZE];
    char price[PRICE_SIZE];
    char date[TO_DATE_SIZE];
    char ticket_type[TICKET_TYPE_SIZE];
    struct list_head list;
};
```

部分相关应用函数如下：

车票链表初始化：

```
int ticket_init(struct list_head *ticket_head, struct list_head *user_ticket_head) ;
```

更新车票显示：

```
int flash_ticket(int clientfd, struct interaction *interaction) ;
```

获取 12306 的车票：

```
int get_ticket_from_py(struct data_packet data, int clientfd, char *ticket_buffer) ;
```

用户购买车票：

```
int buy_ticket(struct data_packet data, int clientfd, struct list_head *user_ticket_head,
struct list_head *user_head) ;
```

用户查询自己的订单信息：

```
int find_order(struct data_packet data, int clientfd, struct list_head *user_ticket_head,
```

struct list_head *temporary_head) ;

用户退票：

int return_ticket(struct data_packet data, int clientfd, struct list_head *user_ticket_head) ;

查看所有车票信息：

int build_all_ticket_buffer(char *ticket_buffer, struct list_head *ticket_head) ;

查看所有用户信息：

int build_all_user_buffer(char *user_buffer, struct list_head *user_head) ;

管理员添加车票：

void add_ticket(struct data_packet data, int clientfd, struct list_head *ticket_head)

管理员删除车票：

void delete_ticket(struct data_packet data, int clientfd, struct list_head *ticket_head, struct list_head *user_ticket_head)

管理员删除用户：

void delete_user(struct data_packet data, int clientfd, struct list_head *user_head)

管理员修改车票：

void change_ticket(struct data_packet data, int clientfd, struct list_head *ticket_head, struct list_head *user_ticket_head)

管理员修改用户：

void change_user(struct data_packet data, int clientfd, struct list_head *user_head)

用户查找本地已保存的车票：

int user_find_ticket(struct data_packet data, int clientfd, struct list_head *ticket_head, char *ticket_buffer)

需要注意的是，并非一个函数对应一个功能，有些功能是通过几个函数集成在一起实现的，比如说用户改签车票的功能，就是通过搜索本地已保存车票的函数加退票函数加买票函数来实现的。

4.2. 安卓客户端设计

安卓客户端程序逻辑如图 4 所示，安卓客户端主要由多个功能界面的活动以及一个发送心跳包的服务组成。在安卓客户端界面设计上，选择深绿色和浅蓝色为 APP 的主题色，字体大多采用白色、黑色、红色等相对显眼的颜色，控件的布局方式采用相对布局和线性布局相结合，进行说明：

1) 图 5 为用户登录、注册的界面，左边为用户登录界面，采用线性布局的方式，将最上方的图片到最下面的管理员登录 Button 和账户注册 Button 依次排列。其中穿插了部分相对布局的方式，例如：输入 ID 的 EditText 和左边的小人物的图片其实是采用了相对布局。右边为用户注册界面，仅采用相对布局的方式，以第一个输入账号的 EditText 为基准，将其他 EditText 和 Button 放在其下方。

2) 图 6 为查询车票界面和查询结果，左边的查询车票界面采用线性布局和相对布局，以线性布局为主体，将每个部分从上到下依次排列，上方的图片，往下接着的是出发城市和抵达城市等等。每个部分中是采用相对布局的方式。右边的查询结果页面只有两个控件，采用相对布局，一上一下进行分布。下方的 ListView 是用来添加不同车票信息的，一种车票占据一个 item。

3) 提交订单界面采用相对布局，整体分为三个部分：上方的图片及其中间的控件、中间的选取车票类型部分、下方的提交订单 Button。再分别采用相对布局，将每个部分的控件进行布局。

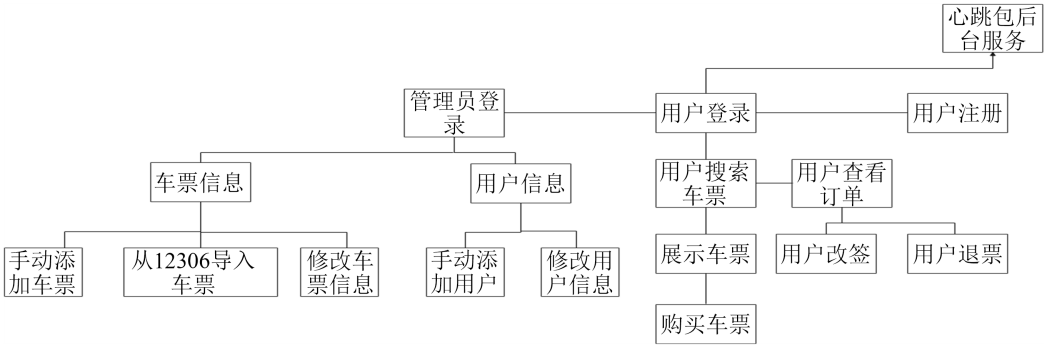

Figure 4. Program logic in user
图 4. 安卓客户端程序逻辑图

Figure 5. Display of login and registration
图 5. 用户登录、注册界面

Figure 6. Display of searching tickets and the results
图 6. 用户查询车票界面、查询结果界面

4) 图 7 为用户查看订单界面和改签退票界面，左边的用户查看页面与上面的查询结果页面的布局和控件是一样的，只是 ListView 中的 item 不一样。右边的改签退票界面是采用相对布局的方式，以上方的图片为基准，将改签和退票的 Button 放下其下方。图片中的控件也是采用相对布局的方式，以正中间的车次为基准进行布局。

5) 改签选取车票界面，该界面是在用户查看自己的订单界面的基础上多加了一行 TextView。每个 item 与查询结果的 item 相同。

6) 管理员登录界面，该界面和用户登录界面是类似的。

7) 图 8 为管理车票信息和用户信息界面各自的 fragment 管理车票信息的 fragment 是采用相对布局的方式，是以整体父布局作为基准进行布局的。上方有刷新、载入、添加三个 ImageButton。中间是输入车次的 EditText 和搜索的 Button。下方是展示车票的 ListView。管理车票信息 fragment 中使用的 item 和查询结果的 item 相同。管理用户信息 fragment 和管理车票信息 fragment 类似，只是去掉了载入的 ImageButton，以及下方的 ListView 中的 item 不一样。

8) 图 9 为管理员搜索 12306 车票以及将车票载入本地界面，管理员搜索 12306 车票的界面和查询车票界面类似，只是删除了右下角的 Button。在运行过程中上下两个界面之间还有个将搜索的车票展示出来的界面，是和查询结果页面一样。车票载入本地界面和改签退票界面类似，更换了一个 Button 中的文字，和将箭头下方的控件更换为由线性布局组成的车票座位类型信息的控件。

9) 图 10 为管理员添加车票和用户界面，管理员添加车票界面采用相对布局，以最上方输入车次的 EditText 为基准向下进行布局，同时最下方加入 TextView 进行提示。管理员添加用户界面采用相对布局，以最上方输入 ID 的 EditText 为基准向下进行布局。

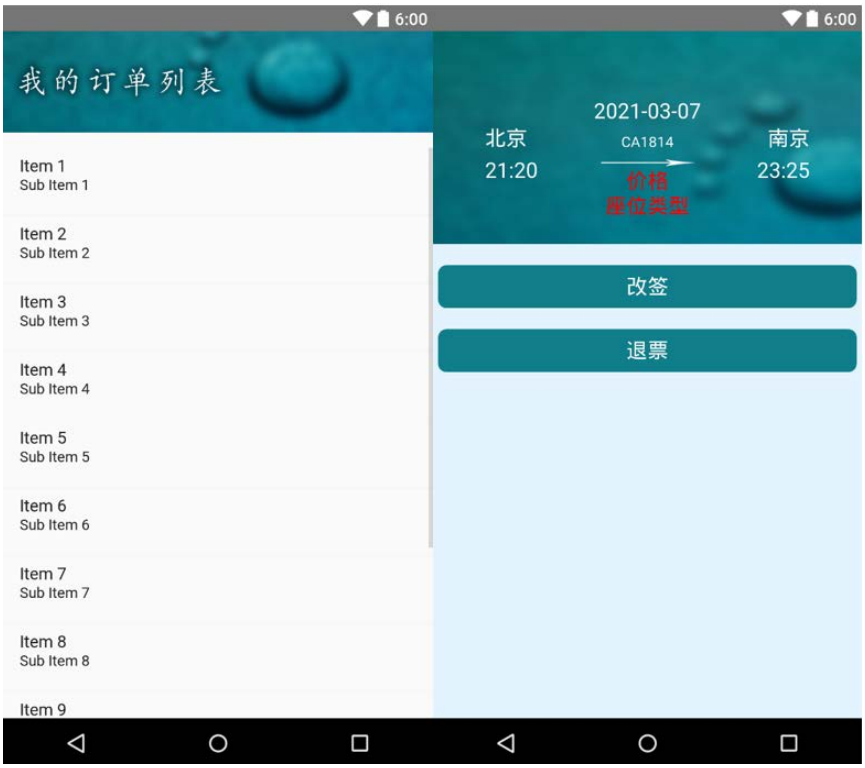

Figure 7. Display of order list and change/refund tickets
图 7. 用户查看自己的订单、改签退票界面

Figure 8. Fragments of administrators and users
图 8. 管理车票信息和用户信息界面各自的 fragment

Figure 9. Display of searching and loading tickets' information from 12306 by administrators
图 9. 管理员搜索 12306 车票以及将车票载入本地界面

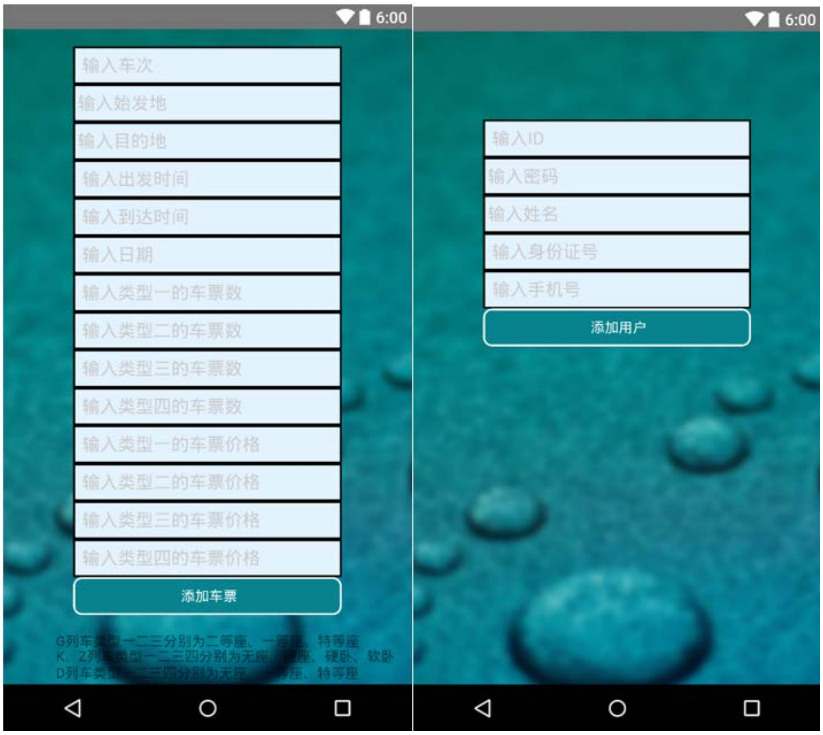

Figure 10. Display of adding tickets and users by administrators
图 10. 管理员添加车票和用户界面

4.3. 安卓客户端与 Linux 服务端之间的通信协议

图 11 为安卓客户端与 Linux 服务端之间的通信协议数据格式，由数据头、数据长度、命令类型、命令吗、返回码、数据域和校验位组成。

数据头让程序知道数据的起始位置。数据长度让程序读取数据在什么位置截止，可以用数据尾来代替数据长度。命令类型是用来区分不同功能所需要的请求码。参数是不同命令需要携带的数据，例如若命令是登录的话，参数携带的就是账号和密码等信息。校验位是判断整段数据有没在传输过程中发生丢失或改变。

数据头(1B)	数据头(1B)	数据长度(2B)	命令类型(1B)	命令码(1B)	返回码(1B)	数据域(N*1B)	校验位(1B)

Figure 11. Data format between server and user in communication protocol
图 11. 安卓客户端与 Linux 服务端之间的通信协议数据格式

5. 系统测试及运行结果

本设计的操作显示画面很多，因此仅截取部分页面展示说明，对于本设计的系统测试及运行结果分别如下所述：

1) 服务端初始化测试：

在终端上输入./train_system 运行 Linux 服务端后，点击运行，Linux 服务端初始化成功，其画面显示如图 12 所示。

2) 客户端连接测试：

将手机联网，启动 Android 客户端，成功接入 Linux 服务端，Android 客户端开始发送心跳包，打开 APP 的画面显示如图 13 所示。

3) 用户登录测试：

在安卓 APP 上输入已注册的账号和密码后，点击登录。服务器收到请求登录的数据。Linux 服务端匹配账号和密码，正确后发送登录成功的数据给安卓客户端，安卓客户端进入搜索车票页面。

4) 用户搜索车票测试：

在搜索车票页面填写出发城市、抵达城市，选择出发时间后，点击搜索，Linux 服务端接收到查看车票信息，输入车辆信息的手机画面显示如图 14 所示。搜索之后，Linux 服务端将与搜索信息相对应的车票数据发送给安卓客户端，手机画面显示如图 15 所示。

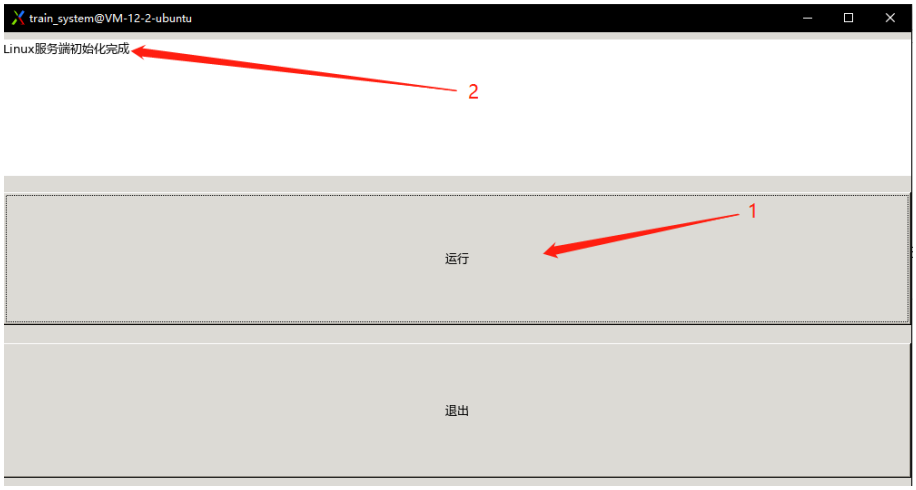

Figure 12. Display of initializing successfully in server
图 12. Linux 服务端初始化成功示意图

Figure 13. Display of executing APP on the mobile phone
图 13. 打开 APP 的手机画面显示图

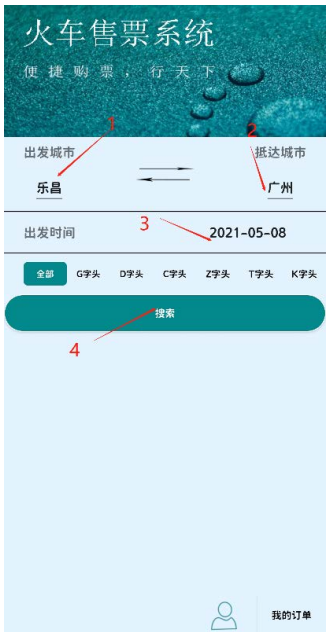

Figure 14. Display of searching trains' information on the mobile phone
图 14. 输入车辆信息的手机画面显示图

Figure 15. Display of tickets' information on the mobile phone
图 15. 用户请求搜索车票，成功后显示车票信息的手机画面显示图

5) 用户购买车票测试：

搜索到的任意一张车票，进入提交订单的界面，选择想要购买的座位类型，点击提交订单后，Linux 服务端接收到请求购买车票的数据。

6) 用户退票测试：

进入改签和退票的界面，点击退票按钮，可以在订单页面看到自己的车票已经退掉了。

7) 用户改签测试：

在自己的订单界面选择想要改签的车票，点击提交订单后，改签成功。

8) 添加用户功能测试：

输入用户的信息，请求添加用户。

9) 添加车票：

输入车票的信息，请求添加用户界面。

10) 从 12306 载入车票信息功能：

填写出发城市、抵达城市、出发时间，点击搜索。选择想要载入的车票，点击载入本地，手机画面显示如图 16 所示。载入成功，手机画面显示如图 17 所示。

11) 删除车票功能：

在管理用户信息界面选择想要删除的车票，点击删除车票。

12) 删除用户：

选择想要删除的用户，点击删除，选择想要删除的用户信息，则手机画面显示如图 18 所示。

13) 修改车票信息：

选择想要修改的车票，填写想要修改的信息，选择想要修改的车票，成功后如图 19 所示。

14) 修改用户信息：

选择想要修改的用户，填写想要修改的信息。

Figure 16. Displays of loading tickets' information from 12306 on the mobile phone
图 16. 从 12306 载入车票信息功能的三张手机画面显示图

Figure 17. Display of loading tickets' information successfully from 12306 on the mobile phone
图 17. 从 12306 载入车票信息成功的手机画面显示图

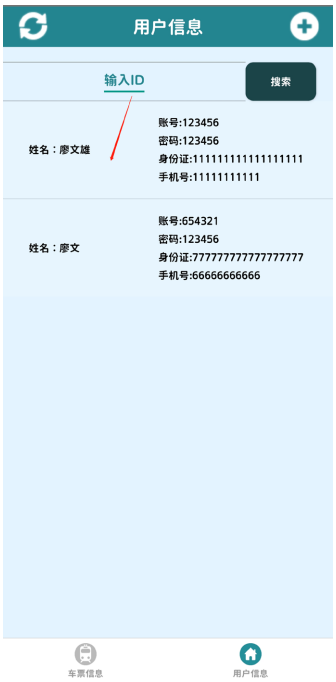

Figure 18. Display of choices in eliminating users' information on the mobile phone
图 18. 选择想要删除的用户信息的手机画面显示图

Figure 19. Display of choices in changing tickets on the mobile phone
图 19. 选择想要修改的车票的手机画面显示图

本设计完成的火车售票系统测试正常、运行正确，具实用性；需要说明的是，由于法规与金流的限制，从购买到付款的阶段，目前尚无法实现，但已具备潜力，满足法规要求便能成为一个真正可用的火车售票系统了。

6. 总结与展望

本设计基于 Linux 和安卓系统实现了一个简单的火车售票系统，于 Linux 开发和安卓开发，python 爬虫的知识，同时结合 Linux 开发和安卓开发的现状，对火车售票系统进行了可用性分析，制定了整体的设计方案。对火车售票系统进行代码的编写，并对整体的功能进行测试，保证其稳定性和可用性。

通过这次对火车售票系统的开发，未来有以下几点可以再做精进与改善：

1) 在上服务端，不能灵活管理处理数据的线程。主要体现在，需要事先开好几个线程，有数据需要处理时，就用上其中的线程，这样就导致有些空闲的线程在占用资源。

2) 不能灵活申请部分内存，主要体现在，对需要传输到 APP 的数据，没法根据数据的大小进行申请内存，只能初始设置一个较大的内存，然后进行使用。

参考文献

[1] W. Richard Stevens, Stephen A. Rago. UNIX 环境高级编程: 影印版[M]. 第 2 版. 尤晋元, 张亚英, 戚正伟, 译. 北京: 人民邮电出版社, 2019.

[2] 杜华. Linux 编程技术详解[M]. 北京: 人民邮电出版社, 2007.

[3] 黄梓昂, 马殿光, 唐厚君. 基于 μCOS 的多轴运动控制器的通信模型[J]. 测控技术, 2019, 38(1): 136-139+148.

[4] 赵宏, 朱忠政, 孔东一. Linux 系统教学中关于套接字文件的解析[J]. 软件, 2020(9): 33-35.

[5] 韩旭, 王娣. C 语言从入门到精通[M]. 第 4 版. 北京: 清华大学出版社, 2019.

[6] 包晓安, 聂凡杰, 徐璐, 张娜, 吴彪. 基于 Reactor 与非阻塞 IO 的服务端框架设计与实现[J]. 浙江理工大学学报(自然科学版), 2020, 43(4): 520-526.

[7] 郭霖. 第一行代码——Android [M]. 第 2 版. 北京: 人民邮电出版社, 2016.

[8] Ed Burnette (2008) Hello, Android. Pragmatic Bookshelf, Dallas.

[9] 袁国忠. Python 编程从入门到实践[M]. 北京: 人民邮电出版社, 2016.

[10] 崔庆才. Python3 网络爬虫开发实战[M]. 北京: 人民邮电出版社, 2018.

[11] 吴向峰, 王景中, 江继军. GNOME/GTK + 编程宝典[M]. 北京: 电子工业出版社, 2000.