**中国矿业大学计算机科学与技术学院**

**2024-2025(2)《Linux操作系统》课程报告(本科)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 专业班级: | | 计算机科学与技术2022-02班 | | | 姓名: | 杨晓琦 | | | 学号: | 08222213 | |  |
|  | | | | | | | | | | | |  |
| **序号** | **报告题目** | **基础理论掌握程度** | **综合知**  **识应用**  **能力** | **报告**  **内容** | **报告**  **格式** | **完成**  **状况** | **工作量** | **学习、**  **工作**  **态度** | **抄袭**  **现象** | **其它** | **综合**  **成绩** | **批阅日期** |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| 任课教师： | |  | |  |  |  | |  | |  |  |  |

目录

[1 项目概述 2](#_Toc195793727)

[1.1 项目背景 2](#_Toc195793728)

[1.2 项目目标 2](#_Toc195793729)

[1.3 开发环境 2](#_Toc195793730)

[2 需求分析 2](#_Toc195793731)

[2.1 功能性需求分析 2](#_Toc195793732)

[2.1.1 核心需求分析 2](#_Toc195793733)

[2.1.2 命令集需求分析 3](#_Toc195793734)

[2.2 性能需求分析 4](#_Toc195793735)

[2.3 角色用户分析 4](#_Toc195793736)

[3 系统架构 4](#_Toc195793737)

[3.1 整体架构 4](#_Toc195793738)

[3.2 技术栈 4](#_Toc195793739)

[4 系统设计 4](#_Toc195793740)

[4.1 架构设计 5](#_Toc195793741)

[4.2 模块设计 5](#_Toc195793742)

[4.2.1 服务器模块 5](#_Toc195793743)

[4.2.2 客户端模块 5](#_Toc195793744)

[4.2.3 共享模块 5](#_Toc195793745)

[5 详细设计 5](#_Toc195793746)

[5.1 通信协议设计 5](#_Toc195793747)

[5.2 进程与线程设计 6](#_Toc195793748)

[5.2.1 服务器进程设计 6](#_Toc195793749)

[5.2.2 客户端进程设计 6](#_Toc195793750)

[5.2.3 日志进程设计 7](#_Toc195793751)

[5.3 进程与线程通信 7](#_Toc195793752)

[5.3.1 客户端与服务器通信 7](#_Toc195793753)

[5.3.2 线程间通信 8](#_Toc195793754)

[5.3.3 进程间通信 8](#_Toc195793755)

[6 系统实现 8](#_Toc195793756)

[6.1 服务器实现 8](#_Toc195793757)

[6.1.1 多客户端处理 8](#_Toc195793758)

[6.1.2 日志记录实现 9](#_Toc195793759)

[6.2 客户端实现 9](#_Toc195793760)

[6.2.1 用户界面设计 9](#_Toc195793761)

[6.2.2 消息处理 10](#_Toc195793762)

[7 功能展示 10](#_Toc195793763)

[7.1 启动服务器 10](#_Toc195793764)

[7.2启动客户端并登录 11](#_Toc195793765)

[7.3 系统聊天主界面 11](#_Toc195793766)

[7.4 实时展示在线用户列表功能 11](#_Toc195793767)

[7.5 系统消息显示功能 12](#_Toc195793768)

[7.6 私聊功能 12](#_Toc195793769)

[7.7 /quit命令功能 12](#_Toc195793770)

[7.8 不同消息类型及其颜色区分功能 13](#_Toc195793771)

[7.9 文件系统存储日志功能 13](#_Toc195793772)

[8 系统存在的问题 14](#_Toc195793773)

[8.1 架构设计层面 14](#_Toc195793774)

[8.2 功能实现层面 14](#_Toc195793775)

[9 体会 14](#_Toc195793776)

[10 参考文献 14](#_Toc195793777)

**《Linux操作系统》课程综合报告**

题目要求：

综合运用已学习的 Linux 进程控制、进程间通信、进程同步、线程控制、线程间通信、线程同步、文件系统操作、网络通信等知识，选择有创意的题目，用 C/C++ 实现一个综合的应用系统。

报告要求：

⑴ 报告内容符合基本的软件工程思路与规范，逻辑与描述清晰、正确，图表完备；

⑵ 运行与测试必须要相应的说明、数据与运行截图；

⑶ 报告格式符合《计算机学院考查类课程报告撰写规范》要求；

⑷ 综合报告的封面除了正确填写您的专业班级、姓名与学号外，不得更改其它内容

源代码要求：

⑴ 代码必须完整，能够正确运行，有必要的注释；

评阅标准：

⑴ 根据所实现系统的完备性、新颖性、程序正确性酌情扣分；

⑵ 根据上述的报告要求及评阅表内容要求以及源代码要求酌情扣分；

综合报告及源代码提交：

⑴ 将报告文件名中的学号与姓名替换为您的真实学号与姓名，各部分之间用一个英文下划线分隔；

⑵ 以实验成果提交电子档资料：将综合报告电子版与系统源代码及两次作业电子版放在一个压缩包里，并以“学号\_姓名\_Linux操作系统报告与作业”为文件名(将学号与姓名替换为您的真实学号与姓名)；

⑶ 若综合报告、作业及源代码有改动需要重新提交，则应重新压缩并以“学号\_姓名\_Linux操作系统报告与作业(日期)”为文件名，其中日期以“2025.7.1”的格式替换为您的最新资料日期；

⑷ 提交网站：219.219.61.252（课程名称：Linux操作系统2024-2025(2)）

Talk\_CUMT聊天系统

# 1 项目概述

## 1.1 项目背景

本项目是一个在Linux操作系统下开发的基于C语言的终端聊天室系统（TermiTalk），设计并实现了一个多用户聊天室，支持文本消息、私聊、系统消息、用户列表、存储历史聊天记录并将上述以图形化方式展示等功能。该系统综合应用了Linux操作系统的进程控制、进程间通信、进程同步、线程控制、线程间通信、文件系统操作和网络通信等知识。通过设计并实现该系统，能够进一步理解和实践Linux系统编程的核心技术。

## 1.2 项目目标

 开发一个功能较完备的终端聊天室系统，能够支持多个客户端连接到服务器，并进行即时的文本通信。设定该系统需要具备以下基本功能：

* + 多用户在线聊天
  + 私聊功能
  + 显示用户状态（在线、离开、忙碌等）
  + 聊天记录的日志功能
  + 基于ncurses的终端界面显示

## 1.3 开发环境

表1 准备开发环境

|  |  |
| --- | --- |
| 项目 | 配置 |
| 虚拟机 | VMware Workstation Pro |
| 操作系统 | CentOS 7 64位 |
| 处理器 | 1vCPU |
| 内存 | 1GB DDR4 |
| 硬盘(SCSI) | 20 GB |

# 2 需求分析

## 2.1 功能性需求分析

2.1.1 核心需求分析

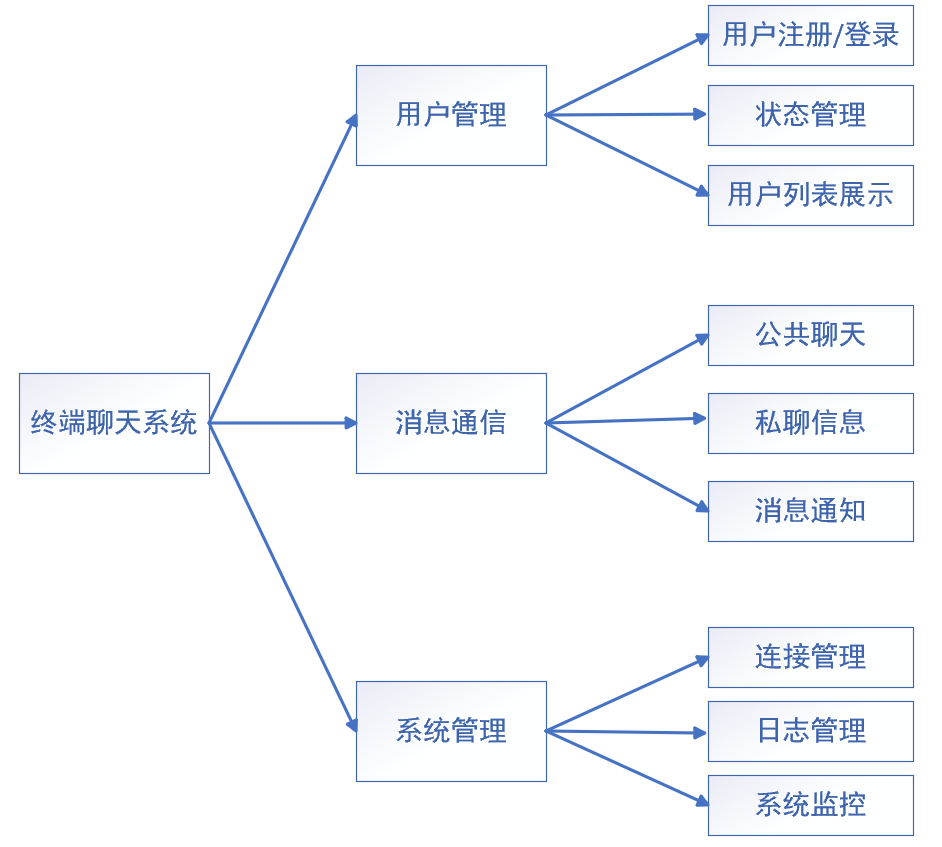


图1 系统核心功能

1. **用户管理功能:**

* 唯一用户名注册（3-16位字母数字组合）
* 三种在线状态切换（在线/离开/忙碌）

实时用户列表更新（心跳检测间隔15秒）

1. **消息通信功能**

表2 支持的消息类型及格式

| **消息类型** | **格式示例** |
| --- | --- |
| 文本消息 | [时间] 用户名: 消息内容 |
| 系统消息 | [系统] 通知内容 |
| 私聊消息 | [私聊] 发送者→接收者: 内容 |

1. **系统管理功能**

* 聊天日志按天分割存储
* 服务器资源监控（CPU/内存/连接数）
* 异常连接自动清理（超时30秒）

2.1.2 命令集需求分析

表3 支持的命令集及作用

|  |  |
| --- | --- |
| 命令 | 作用 |
| /help | 显示帮助菜单 |
| /w<用户> | 发起私聊 |
| /status | 更改状态 |
| /quit | 退出系统 |
| /list | 刷新用户列表 |

## 2.2 性能需求分析

表4 各项性能要求与测试

|  |  |  |  |
| --- | --- | --- | --- |
| 指标项 | 最低要求 | 预期目标 | 测试方法 |
| 并发用户数 | 50 | 100 | Jmeter压力测试 |
| 消息延迟 | 200ms | 50ms | 抓包分析时间戳 |
| 消息吞吐量 | 100条/s | 500条/s | 批量发送测试 |
| 断线重连时间 | 60s | 15s | 模拟网络中断 |

## 2.3 角色用户分析

表5 不同角色与对应需求分析

| **角色类型** | **使用场景** | **核心需求** |
| --- | --- | --- |
| 普通用户 | 日常聊天 | 简洁界面/快速消息发送/状态设置 |
| 管理员 | 系统维护 | 连接监控/异常诊断/日志查询 |
| 审计员 | 安全检查 | 操作追溯/敏感词管理 |

# 3 系统架构

## 3.1 整体架构

图2 系统树状目录

## 3.2 技术栈

表6 系统用到的技术与实现方法

|  |  |
| --- | --- |
| 技术 | 实现方法 |
| 网络通信 | TCP socket 通信 |
| 多线程 | pthread 库实现多客户端并发处理 |
| 进程间通信 | pipe 实现服务器与日志进程通信 |
| 线程同步 | pthread\_mutex\_t 保护共享资源 |
| 线程同步 | ncurses 库实现图形界面 |

# 4 系统设计

## 4.1 架构设计

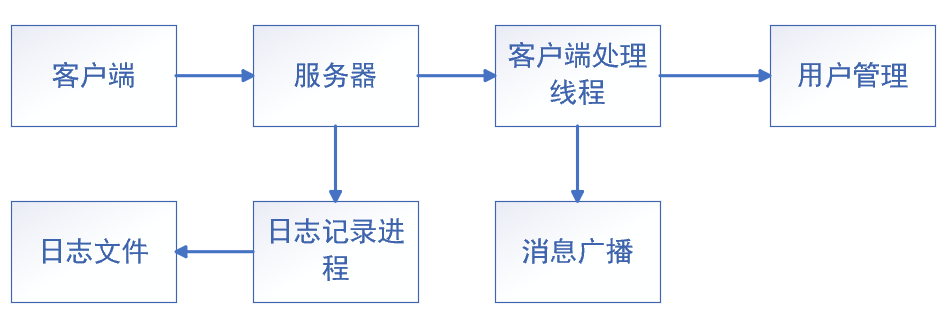


图3 架构模块的交互

## 4.2 模块设计

4.2.1 服务器模块

表7 服务器模块相关功能与技术

| 模块名称 | 功能描述 | 关键技术 |
| --- | --- | --- |
| 主控模块 | 接受客户端连接 | socket监听 |
| 客户端处理模块 | 处理客户端消息 | pthread多线程 |
| 用户管理模块 | 维护在线用户列表 | 互斥锁保护共享数据 |
| 日志记录模块 | 记录聊天记录 | 管道通信+文件锁 |

4.2.2 客户端模块

表8 客户端模块相关功能与技术

| 模块名称 | 功能描述 | 关键技术 |
| --- | --- | --- |
| 网络通信模块 | 与服务器通信 | TCP Socket |
| 用户界面模块 | 显示聊天界面 | ncurses图形库 |
| 消息处理模块 | 解析和处理消息 | 消息队列 |

4.2.3 共享模块

表9 共享模块相关功能与描述

| 模块名称 | 功能描述 | 关键技术 |
| --- | --- | --- |
| 协议定义模块 | 定义通信数据结构 | 结构体封装 |
| 工具函数模块 | 提供公共工具函数 | 时间处理/字符串处理 |
| 常量定义模块 | 系统全局常量定义 | 宏定义/枚举 |

# 5 详细设计

## 5.1 通信协议设计

// 消息数据结构

typedef struct {

    int type;                   // 消息类型

    char username[USERNAME\_LEN]; // 用户名

    char target[USERNAME\_LEN];   // 私聊目标

    char message[MAX\_MSG\_LEN];   // 消息内容

    char timestamp[64];          // 时间戳

    UserInfo userlist[MAX\_CLIENTS]; // 用户列表

    int user\_count;              // 用户数

} ChatMessage;

表10 消息类型定义

| 类型常量 | 值 | 说明 |
| --- | --- | --- |
| MSG\_TYPE\_TEXT | 0 | 普通文本消息 |
| MSG\_TYPE\_SYSTEM | 1 | 系统消息 |
| MSG\_TYPE\_PRIVATE | 3 | 私聊消息 |
| MSG\_TYPE\_USERLIST | 4 | 用户列表更新消息 |

## 5.2 进程与线程设计

5.2.1 服务器进程设计

服务器进程通过server/main.c中的代码实现，主要功能是监听客户端的连接请求，并为每一个连接分配一个新的线程来处理。每个客户端连接都由独立的线程处理，确保服务器能够并发地处理多个客户端的消息。服务器进程的流程如下：

1. **创建套接字**：使用socket()函数创建一个TCP套接字，并通过bind()绑定到指定的端口（8888）。
2. **监听连接**：使用listen()函数开始监听来自客户端的连接请求。
3. **接受连接**：当有客户端请求连接时，使用accept()函数接受连接，返回一个新的套接字用于与客户端进行通信。
4. **处理每个客户端的连接**：服务器会为每个客户端创建一个新的线程，使用pthread\_create()创建线程，并通过handle\_client()函数处理每个客户端的消息传递、用户状态更新以及广播。

int server\_fd = socket(AF\_INET, SOCK\_STREAM, 0);

bind(server\_fd, (struct sockaddr \*)&addr, sizeof(addr));

listen(server\_fd, 10);

// 接受客户端连接并创建新线程

int client\_fd = accept(server\_fd, NULL, NULL);

pthread\_create(&tid, NULL, handle\_client, &client\_fd);

pthread\_detach(tid);

5.2.2 客户端进程设计

客户端进程通过client/main.c中的代码实现，负责与服务器建立连接，并维持一个发送与接收消息的循环。客户端还使用ncurses库绘制聊天界面，显示用户列表、聊天记录等。其主要流程如下：

1. **连接服务器**：客户端首先使用socket()创建TCP套接字，并通过connect()与服务器建立连接。
2. **输入用户名**：客户端接入后，提示用户输入用户名。
3. **创建接收线程**：客户端使用pthread\_create()创建一个新的线程来接收服务器发送过来的消息。该线程会调用receive\_thread()函数，循环读取并处理消息。
4. **绘制界面**：主线程使用ncurses库绘制聊天界面，包括聊天窗口、输入框、在线用户列表等。客户端通过draw\_ui()函数绘制界面并实时更新。
5. **消息发送与接收**：客户端通过标准输入接受用户输入，构造ChatMessage结构体，并通过send()函数将消息发送给服务器。接收到服务器的消息后，客户端在界面上更新聊天内容。

// 创建接收线程

pthread\_t recv\_tid;

pthread\_create(&recv\_tid, NULL, receive\_thread, NULL);

// 绘制聊天界面

draw\_ui();

5.2.3 日志进程设计

日志进程通过server/logger.c中的代码实现，负责记录聊天日志。日志子进程与主服务器进程通过管道进行通信。当有新的聊天消息时，服务器会将消息通过管道传递给日志进程，日志进程会将消息写入日志文件。具体的实现流程如下：

1. **创建管道**：在服务器的server/main.c中，使用pipe()创建一个管道。
2. **创建日志子进程**：服务器通过fork()创建一个子进程来专门处理日志记录。
3. **接收日志信息**：日志子进程从管道中读取聊天消息，并将消息写入日志文件。日志文件名为当前日期，例如：2025-04-16.log。
4. **写入日志**：使用flock()进行文件加锁，确保日志写入的安全性。

pipe(logger\_pipe);

if (fork() == 0)

{

    close(logger\_pipe[1]);

    start\_logger(logger\_pipe); // 启动日志记录

    exit(0);

}

close(logger\_pipe[0]);

// 日志子进程写入日志

start\_logger(logger\_pipe);

## 5.3 进程与线程通信

5.3.1 客户端与服务器通信

客户端与服务器之间通过TCP套接字进行双向通信。客户端发送消息到服务器，服务器通过套接字将消息广播给所有在线用户。消息的格式使用ChatMessage结构体，包括消息类型、发送者、接收者、消息内容等字段。

**客户端发送消息**：客户端通过send()函数将构造好的ChatMessage结构体发送给服务器。

**服务器接收并广播消息**：服务器接收客户端发送的消息后，通过broadcast\_message()函数将消息广播给所有连接的客户端。

// 客户端发送消息

send(sock, &msg, sizeof(msg), 0);

// 服务器广播消息

send(client\_sockets[i], msg, sizeof(\*msg), 0);

5.3.2 线程间通信

在客户端，使用pthread\_create()创建接收消息的线程。该线程调用receive\_thread()函数，循环接收来自服务器的消息。为了确保线程安全，线程之间使用pthread\_mutex\_t进行同步。

// 客户端接收消息线程

void \*receive\_thread(void \*arg)

{

    ChatMessage msg;

    while (1)

    {

        ssize\_t len = recv(sock, &msg, sizeof(msg), 0);

        if (len <= 0)

            break;

        update\_chat(&msg);

    }

    return NULL;

}

5.3.3 进程间通信

在服务器端，日志进程与主服务器进程通过管道进行通信。每当有新的聊天消息时，主服务器进程通过管道将消息传递给日志进程，日志进程接收消息后将其写入日志文件。管道是无名管道，通过pipe()函数创建。主进程和子进程之间通过read()和write()进行数据传输。

// 创建管道

pipe(logger\_pipe);

// 主进程写入管道

write(logger\_pipe[1], &msg, sizeof(msg));

// 日志子进程从管道读取消息

read(pipe\_fd[0], &msg, sizeof(msg));

# 6 系统实现

## 6.1 服务器实现

6.1.1 多客户端处理

服务器采用主从Reactor模式实现高并发，主线程负责接收连接，工作线程处理具体业务。关键实现如下：

// 客户端线程处理函数

void \*handle\_client(void \*arg)

{

    // 接收用户名

    recv(client\_fd, &msg, sizeof(msg), 0);

    // 更新用户列表

    pthread\_mutex\_lock(&clients\_lock);

    strncpy(client\_usernames[i], msg.username, USERNAME\_LEN);

    pthread\_mutex\_unlock(&clients\_lock);

    // 消息处理循环

    while (1)

    {

        recv(client\_fd, &msg, sizeof(msg), 0);

        // 处理不同类型的消息

    }

}

6.1.2 日志记录实现

采用生产者-消费者模型，通过管道实现进程间通信。日志进程独立运行，避免阻塞主线程：

void start\_logger(int pipe\_fd[2])

{

    while (1)

    {

        read(pipe\_fd[0], &msg, sizeof(msg));

        // 按日期生成日志文件

        strftime(filename, sizeof(filename), "chat\_log/%Y-%m-%d.log", tm\_info);

        // 使用文件锁保证写入安全

        flock(fileno(fp), LOCK\_EX);

        fprintf(fp, "[%s] %s: %s\n", msg.timestamp, msg.username, msg.message);

        flock(fileno(fp), LOCK\_UN);

    }

}

## 6.2 客户端实现

6.2.1 用户界面设计

采用MVC模式设计，ncurses负责视图渲染，独立线程处理网络消息。界面分为三个核心区域：

void draw\_ui()

{

    // 初始化ncurses

    initscr();

    cbreak();

    keypad(stdscr, TRUE);

    // 创建窗口

    user\_win = newwin(height, 20, 0, 0);          // 用户列表窗口

    chat\_win = newwin(height, width - 20, 0, 20); // 聊天窗口

    input\_win = newwin(3, width, height, 0);      // 输入窗口

    // 颜色设置

    init\_pair(PAIR\_SELF, COLOR\_CYAN, COLOR\_BLACK);

    init\_pair(PAIR\_SYSTEM, COLOR\_GREEN, COLOR\_BLACK);

}

6.2.2 消息处理

采用事件驱动机制，对不同消息类型进行差异化渲染：

void update\_chat(const ChatMessage \*msg)

{

    // 根据消息类型设置颜色

    switch (msg->type)

    {

    case MSG\_TYPE\_SYSTEM:

        wattron(chat\_win, COLOR\_PAIR(PAIR\_SYSTEM));

        break;

    case MSG\_TYPE\_PRIVATE:

        wattron(chat\_win, COLOR\_PAIR(PAIR\_PRIVATE));

        break;

        // 其他类型处理...

    }

    // 显示消息

    wprintw(chat\_win, "[%s] %s: %s\n",

            msg->timestamp, msg->username, msg->message);

}

# 7 功能展示

## 7.1 启动服务器

打开终端，切换到当前项目Talk\_CUMT目录下，输入.bin/server.out命令启动服务器，如下图所示，表示服务器 TermiTalk 已经启动，并且正在监听端口 8888。



图4 服务器启动

## 7.2启动客户端并登录

打开另一个终端，同样切换到当前项目Talk\_CUMT目录下，输入.bin/client.out命令启动客户端，并且输入唯一用户名进行登录，进入聊天界面，如下图所示，使用 cumter1 作为用户名进入系统。

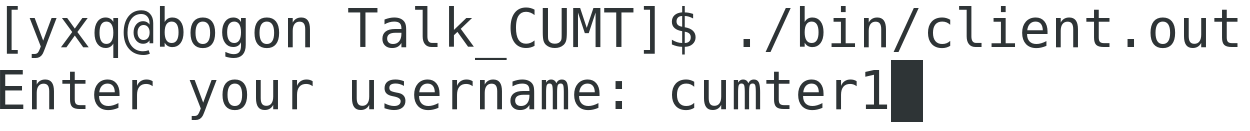


图5 客户端登陆页面

## 7.3 系统聊天主界面

如下图所示，使用ncurses 库中的newwin()创建3个主窗口：①在线用户列表；②聊天消息列表；③输入区域，并通过box() 绘制边框，实现了复杂的终端用户界面。

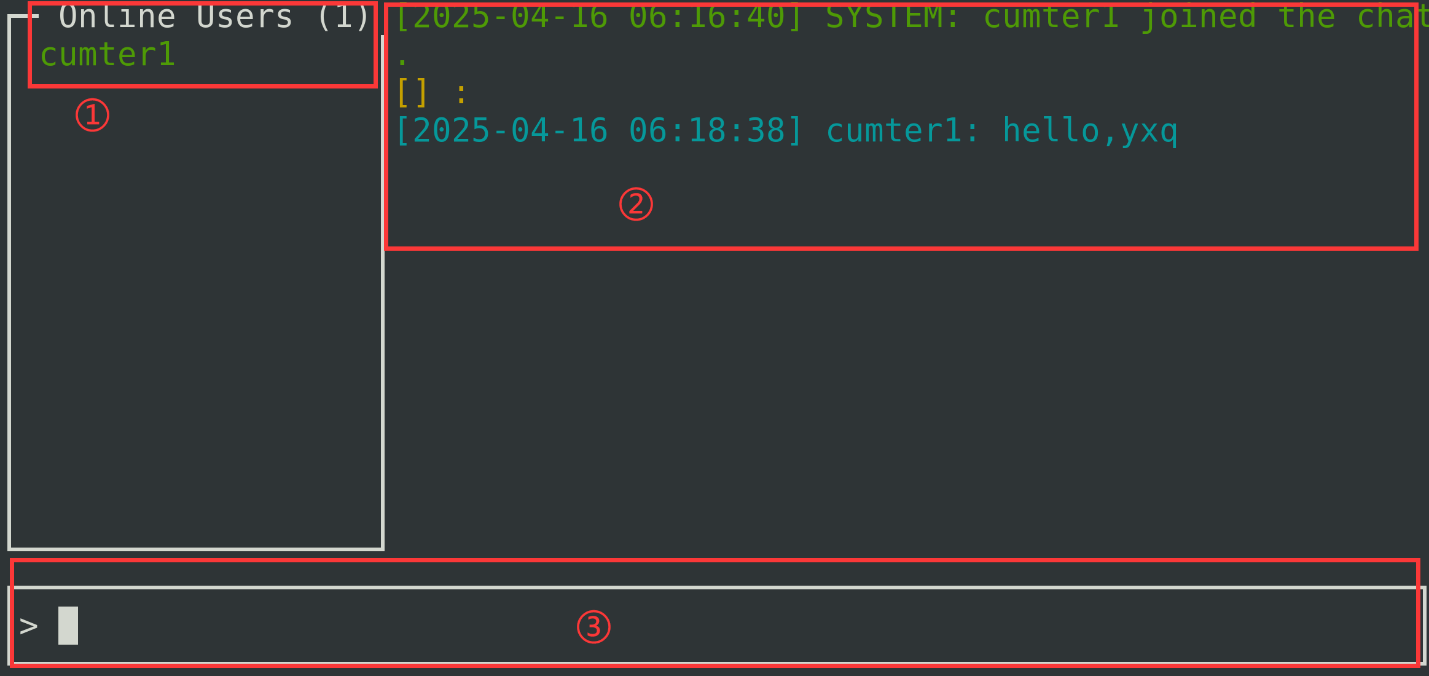


图6 终端用户主界面

## 7.4 实时展示在线用户列表功能

在cumter1登录的前提下，重新再另外两个新终端下启动客户端并进行登录，分别命名为cumter2和cumter3。如下图所示，三个用户的实时在线用户列表均是3个，分别为：cumter1，cumter2，cumter3。

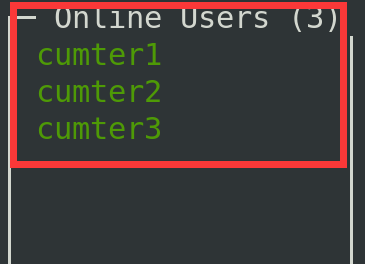


图7 实时显示在线用户

## 7.5 系统消息显示功能

如下图所示，当其他用户登录时，系统会提示：用户x joined the chat；当其他用户退出时，系统会提示：用户x left the chat。

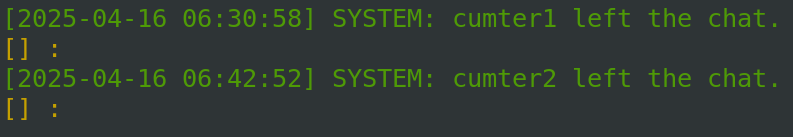
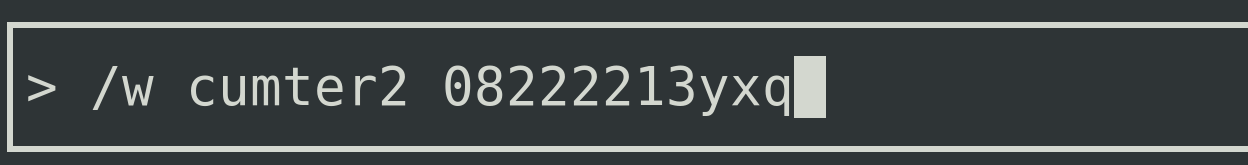


图8 用户登录与退出

## 7.6 私聊功能

如下图所示，cumter1输入 /w cumter2 08222213yxq这个私聊命令后，其他用户无法接收，而用户cumter2会收到这个来自cumter1发送过来的消息。



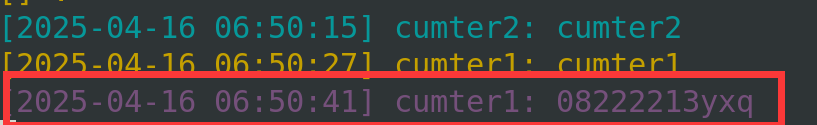


图9 私聊发送与接收

## 7.7 /quit命令功能

如下图所示，当用户cumter2输入 /quit 命令并发送后，这个系统消息会广播发送给其他在线用户，收到cumter3离开聊天的消息。

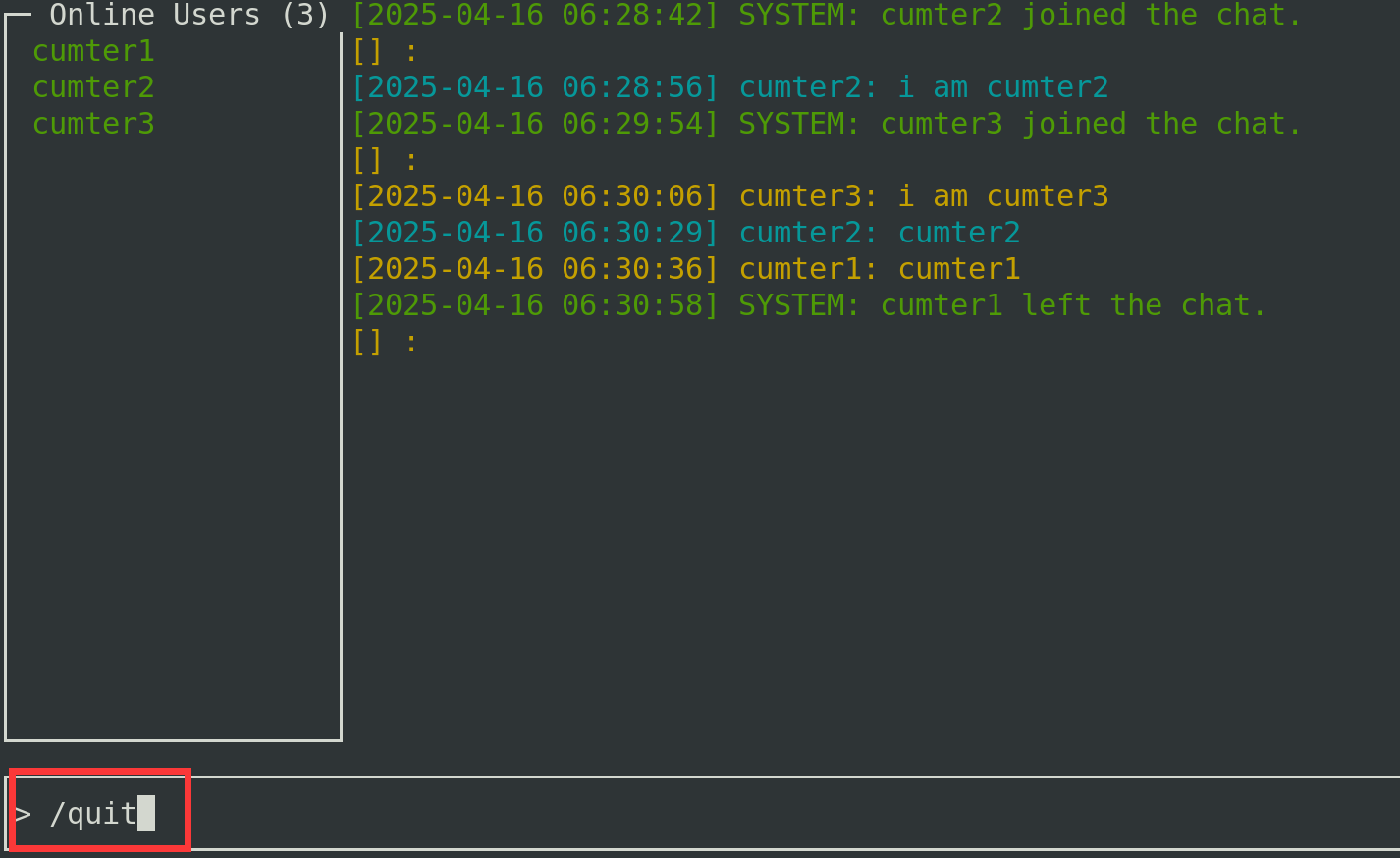




图10 使用/quit命令退出登录

## 7.8 不同消息类型及其颜色区分功能

表11 消息类型与对应显示颜色

|  |  |
| --- | --- |
| 消息类型 | 显示颜色 |
| 自己发送的消息 | 青色 |
| 系统消息 | 绿色 |
| 他人消息 | 黄色 |
| 私聊消息 | 紫色 |

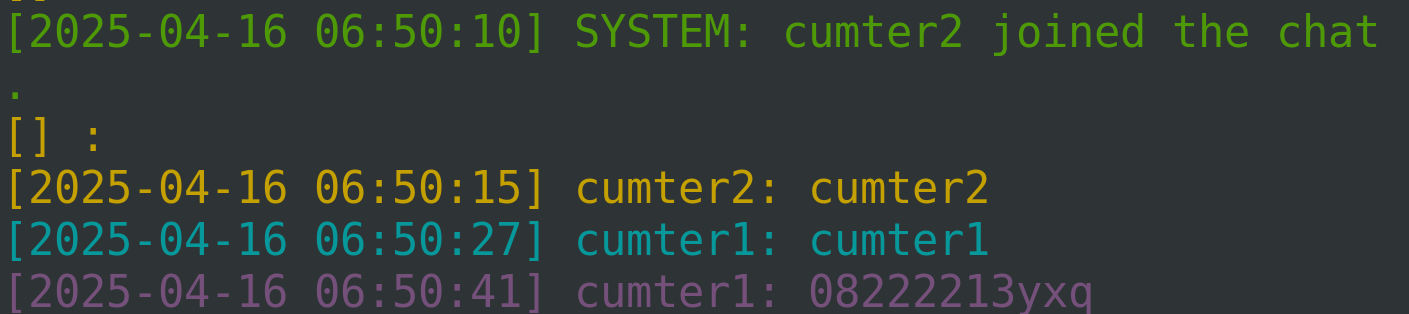
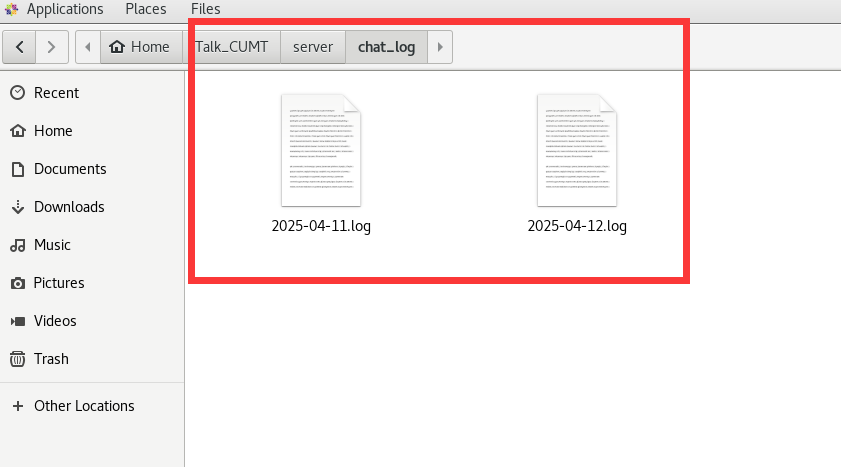


图11 消息颜色显示

## 7.9 文件系统存储日志功能

如下图所示，所有在线用户的历史聊天记录都会以log文件的形式按天来保存。



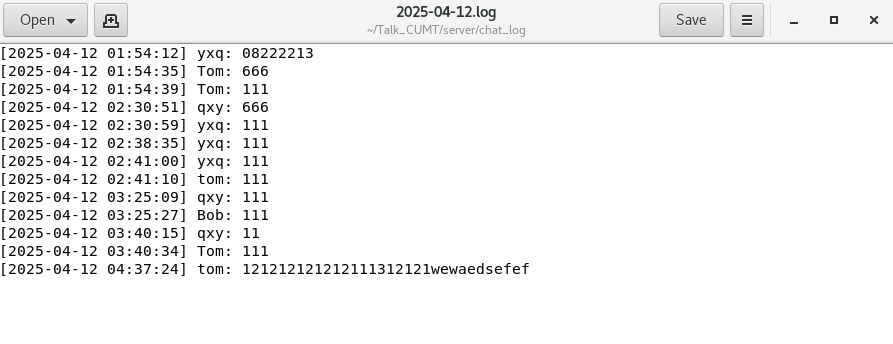


图12 以天为单位查看聊天记录

# 8 系统存在的问题

## 8.1 架构设计层面

**模块耦合度较高**：UI模块与网络处理逻辑存在交叉

**扩展性不足**：协议设计未预留版本兼容字段

**异常处理不完善**：网络中断等场景缺乏恢复机制

## 8.2 功能实现层面

表12 功能层面的缺陷

|  |  |
| --- | --- |
| 问题类型 | 具体表现 |
| 网络通信 | TCP粘包问题未处理 |
| 用户界面 | 输入框缺少历史消息功能 |
| 性能优化 | 日志同步写入导致性能瓶颈 |
| 安全性 | 未实现消息加密传输 |

# 9 体会

1. **对操作系统基本概念的理解加深**

在开发过程中，我需要运用操作系统中的基本概念，如进程、线程、同步、通信等。特别是在多进程和多线程的设计和实现过程中，我深刻认识到操作系统的这些机制如何直接影响系统的稳定性和性能。例如，客户端与服务器之间的通信依赖于TCP/IP协议，进程间的同步和通信（如管道、信号量等）则帮助我实现了服务器日志的记录和处理。通过这些操作系统基本概念的应用，我对操作系统的理解有了更深入的认知。

1. **进程间通信与线程同步的挑战**

在多进程、多线程的设计中，我遇到了不少挑战。尤其是线程间同步与进程间通信的实现。在服务器端，我通过pthread\_create创建了多个线程来处理客户端的连接请求，每个线程负责与一个客户端进行通信。由于线程之间需要共享一些数据（如在线用户列表、聊天记录等），因此在设计时需要特别注意线程同步，避免竞态条件的发生。在此过程中，我使用了互斥锁（mutex）来保证共享资源的安全访问。

1. **使用ncurses构建客户端界面**

在客户端的UI部分，我使用了ncurses库来实现命令行界面的绘制。ncurses使得我能够在终端中实现更加丰富的用户交互界面，包括输入框、聊天信息的展示、在线用户列表等。这不仅让我体会到了文本界面的设计挑战，还让我学会了如何在有限的资源和环境下优化用户体验。虽然ncurses相较于图形界面开发更为简陋，但它的简单、高效的特性适用于我们的终端聊天室系统。

1. **开发遇到的困难与解决方案**

在开发过程中，我遇到了一些困难，例如如何合理地处理并发客户端、如何保证日志进程的稳定性、如何避免死锁等。通过反复调试，我逐步掌握了线程和进程的管理技巧。通过合理设计锁机制和进程间通信，我有效避免了多线程带来的并发问题。在日志记录方面，尽管管道机制非常简洁，但也要求我仔细处理父子进程间的数据传递，避免数据丢失或冲突。

# 10 参考文献

[1] 李超. (2013). *多线程与网络编程基础*. 北京: 电子工业出版社.

[2] 张辉. (2017). *操作系统进程同步与通信*. 北京: 高等教育出版社.

[3] 陆凌牛. (2016). *Linux系统编程（第2版）*. 北京: 人民邮电出版社