

重庆大学大数据与软件学院

# 作业报告

实践项目	ATM 模拟编程作业
课程名称	计算机网络

姓名	黄强	学号	2023xxxx
姓名	谢志昊	学号	2023xxxx

教师	胡海波	班级	
----	-----	----	--

日期	2025.05.25	成绩	
----	------------	----	--

## 1. 目的

通过作业二使同学们掌握所学 TCP 套接字编程方法，理解 TCP 套接字在服务器和客户端之间的应用层通信的流程和形式，巩固计算机网络的理论和知识。此外，通过作业的开展使同学们提高计算机网络程序设计和开发能力，熟悉所用编程语言的开发环境及调试过程，熟悉所用语言中的数据类型、数据结构、语法结构、运算方法等。基于作业一的内容进行程序设计开发，同学们也可以体会计算机网络通信协议(应用层)的制定、标准化等工作对实际应用的价值和意义。

## 2. 作业内容

要求两位同学组队(参考石墨文档 1-作业 2 分组)，基于模拟的应用层通信协议标准(参考石墨文档 2-“RFC20232023”)完成 ATM 机客户端和服务端程序的设计和开发。程序语言不限。

客户端具备 GUI 可以模拟 ATM 机的账户登录和存款取款等功能，同时应该包括错误信息的提示。服务器端应该需要通过后台数据库维护账户信息、具备记录日志的功能。

各组同学独立完成，不能抄袭。作业结束时，每组需提交作业报告，其中包含程序设计思路和软件源代码。

里程碑 1：第 2 次实验课，完成组内程序设计开发和测试，尝试连接其他小组的服务器端。

里程碑 2：第 3 次实验课，分析解决里程碑 1 中的问题，完成程序调试，通过测试脚本完成对其他所有小组服务器端的连接，记录结果。

里程碑 3：第 4 次实验课，继续调试程序，最终提交文档、代码。

## 3. 运行环境

本项目在 Windows11 环境下开发和测试，编码环境主要使用 Visual Studio Code。程序语言为 Python 3.12。

核心技术和库包括：

- **TCP Socket 编程：**利用 Python 内置的 `socket` 模块实现客户端与服务端之间的 TCP/IP 通信。TCP (Transmission Control Protocol) 是一种面向连接的、可靠的、基于字节流的传输层通信协议，它通过三次握手

建立连接，四次挥手断开连接，并提供序列号、确认应答、重传机制、流量控制和拥塞控制等功能，以保证数据传输的可靠性和顺序性。

- **PyQt5**: 用于构建客户端的图形用户界面 (GUI)，提供用户友好的交互体验。
- **JSON**: 用于存储和管理用户数据 ([users.json](#))，方便数据的读取和修改。
- **Logging**: 使用 Python 内置的 logging 模块记录服务器端和客户端的操作日志（分别存储于 [server.log](#) 和 [atm\\_client.log](#)）
- **应用层协议 RFC20232023**: 客户端和服务端之间的通信遵循此协议规范。

## 4. 程序设计

### 4.1 设计思路

采用**客户端-服务器 (Client-Server, C/S) 架构**。

- **服务器端**: 作为后端服务，负责监听客户端连接请求，处理业务逻辑，如用户身份验证、账户余额查询、取款操作等；服务器端维护所有用户账户信息并记录详细的操作日志。
- **客户端**: 作为前端用户界面，提供图形化操作界面 (GUI)，用户通过客户端与服务器进行交互，发送操作请求，并接收和显示服务器返回的处理结果。客户端也记录其操作日志。

**交互流程**（基于 RFC20232023 协议）:

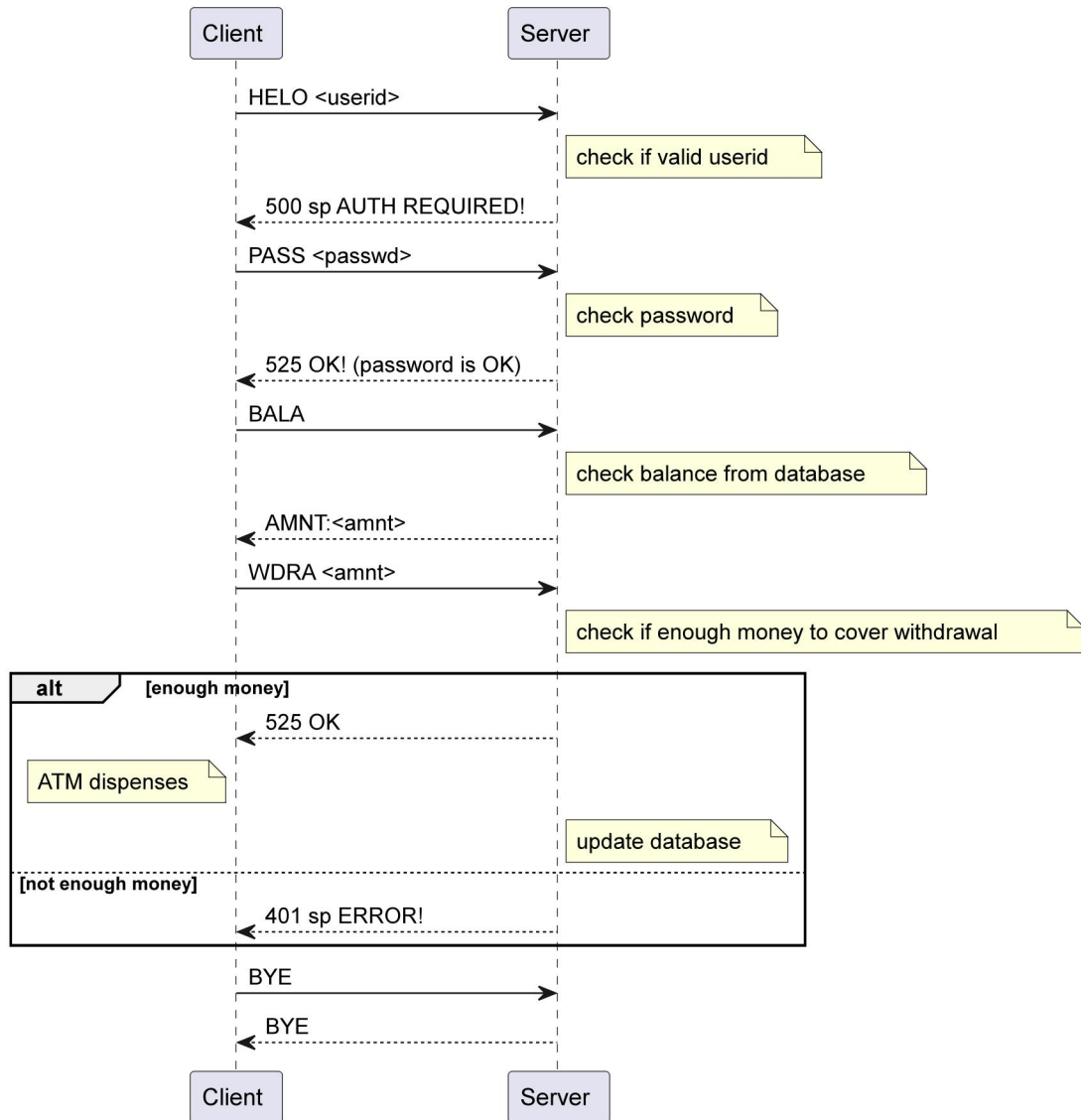
1. 客户端启动，用户在 GUI 界面输入用户 ID（卡号）。
2. 客户端向服务器发送 `HELLO sp <userid>` 消息。
3. 服务器若接收到 `HELLO`，则回复 `500 sp AUTH REQUIRE`，提示客户端需要进行密码验证。
4. 用户在客户端 GUI 输入 PIN 码。
5. 客户端向服务器发送 `PASS sp <passwd>` 消息。
6. 服务器验证 PIN 码：
  - 验证成功：回复 `525 sp OK!`。
  - 验证失败：回复 `401 sp ERROR!`。
7. 登录成功后，用户可进行后续操作：
  - **查询余额**: 客户端发送 `BALA ->` 服务器回复 `AMNT :<amnt>`（成

功) 或 401 sp ERROR! (失败)。

- **取款:** 客户端发送 WDRA sp <amount> -> 服务器回复 525 sp OK! (成功) 或 401 sp ERROR! (如余额不足)。

8. 用户操作结束或退出时, 客户端发送 BYE 消息。

9. 服务器回复 BYE 消息, 随后双方断开连接。



图表 1 交互流程图

## 4.2 服务器端

服务器端的核心逻辑实现在 [server.py](#) 文件中。其主要职责和功能如下：

- **初始化与监听：**启动时，服务器在预设的 IP 地址和端口号上创建 TCP 套接字，并开始监听来自客户端的连接请求。
- **并发处理：**为了能够同时服务多个客户端（或为后续扩展考虑），服务器采用多线程来处理并发连接，每个客户端连接通常会由一个独立的线程或任务来处理。
- **协议解析与请求处理：**
  - 当接收到客户端数据后，服务器按照 RFC2032023 协议规范解析消息。
  - 根据消息类型（HELO, PASS, BALA, WDRA, BYE）执行相应的业务逻辑。
- **用户数据管理：**
  - 用户身份验证时，从 [users.json](#) 文件中读取用户信息进行比对。
  - 处理取款等操作时，会更新 [users.json](#) 文件中对应用户的账户余额。
- **响应生成：**根据业务处理结果，生成符合 RFC2032023 协议的响应消息（如 500, 525, 401, AMNT, BYE）并发送给客户端。
- **日志记录：**所有重要操作、接收到的请求、发送的响应以及发生的错误都会被记录到 [server.log](#) 文件中，包括时间戳、客户端信息、操作类型和结果等。
- **连接管理：**处理客户端的连接建立和断开请求，确保资源的正确释放。

```
import socket
import threading
import json
import logging
import os
# ... (其他导入和日志配置) ...

DATA_FILE = 'data/users.json'
logger = logging.getLogger('ATMServer')

class ATMServer:
    def __init__(self, host='0.0.0.0', port=2525):
        self.host = host
        self.port = port
        self.socket = None
        self.users = self.load_users()

    def load_users(self):
```

```

        """从文件加载用户数据"""
        if not os.path.exists(DATA_FILE):
            # ... (创建默认用户数据) ...
        try:
            with open(DATA_FILE, 'r') as f:
                users = json.load(f)
                logger.info(f"从 {DATA_FILE} 加载了 {len(users)} 个用户")
            return users
        except Exception as e:
            logger.error(f"加载用户数据错误: {str(e)}")
            return {}

    def save_users(self):
        """保存用户数据到文件"""
        try:
            with open(DATA_FILE, 'w') as f:
                json.dump(self.users, f, indent=2)
            logger.info(f"保存了 {len(self.users)} 个用户数据到 {DATA_FILE}")
        except Exception as e:
            logger.error(f"保存用户数据错误: {str(e)}")

    def start(self):
        """启动服务器"""
        try:
            self.socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
            self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,
1)

            self.socket.bind((self.host, self.port))
            self.socket.listen(5)
            logger.info(f"服务器启动于 {self.host}:{self.port}")
            print(f"ATM 服务器已启动, 监听端口 {self.port}")

            while True:
                client_socket, address = self.socket.accept()
                logger.info(f"新连接来自 {address}")
                client_thread = threading.Thread(
                    target=self.handle_client,
                    args=(client_socket, address)
                )
                client_thread.daemon = True
                client_thread.start()

            # ... (异常处理和关闭) ...

```

```

def handle_client(self, client_socket, address):
    """处理客户端连接"""
    user_id = None
    authenticated = False
    try:
        while True:
            data = client_socket.recv(1024).decode('utf-8').strip()
            if not data:
                break
            logger.info(f"收到来自 {address} 的消息: {data}")
            parts = data.split(' ', 1)
            command = parts[0]

            if command == "HELO":
                if len(parts) > 1:
                    user_id = parts[1]
                    response = "500 AUTH REQUIRED!" if user_id in
self.users else "401 ERROR!"
                else:
                    response = "401 ERROR!"
            elif command == "PASS":
                if user_id and len(parts) > 1:
                    password = parts[1]
                    if user_id in self.users and
self.users[user_id]["password"] == password:
                        authenticated = True
                        response = "525 OK!"
                    else:
                        response = "401 ERROR!"
                else:
                    response = "401 ERROR!"
            elif command == "BALA":
                # ... (处理余额查询) ...
            elif command == "WDRA":
                # ... (处理取款) ...
            elif command == "BYE":
                response = "BYE"
            else:
                response = "401 ERROR!"

            client_socket.sendall((response + '\n').encode('utf-8'))
            logger.info(f"发送到 {address}: {response}")
            if command == "BYE":

```

```
                break
            # ... (异常处理和关闭连接) ...

if __name__ == "__main__":
    server = ATMServer()
    server.start()
```

## 4.3 客户端

客户端的实现分散在 [main.py](#) (程序入口)、[atm\\_client.py](#) (核心通信逻辑) 和 [atm\\_gui.py](#) (图形界面实现) 中。其主要职责和功能如下：

- **图形用户界面 (GUI)：**
  - 通过 PyQt5 构建用户交互界面，包括输入卡号、PIN 码、取款金额的输入框，以及执行操作的按钮（登录、查询余额、取款、退出等）。
  - 显示服务器返回的信息、操作结果（成功/失败提示）、账户余额等。
- **用户输入处理：** 获取用户在 GUI 上的输入，并进行基本的格式校验。
- **服务器连接：**
  - 根据用户操作（如点击登录按钮），与服务器指定的 IP 地址和端口建立 TCP 连接。
- **协议封装与消息发送：**
  - 将用户的操作请求（如登录、查询余额、取款）封装成符合 RFC20232023 协议格式的消息。
  - 通过已建立的 TCP 套接字将消息发送给服务器。
- **响应接收与解析：**
  - 接收服务器发送回来的响应数据。
  - 按照 RFC20232023 协议解析响应消息，提取操作结果和相关数据。
- **状态更新与显示：** 根据服务器的响应更新 GUI 界面，例如显示余额、提示操作成功或失败。
- **日志记录：** 客户端执行的各项操作、发送的请求、收到的响应及遇到的错误也会被记录到 [atm\\_client.log](#) 文件中。
- **连接管理：** 在用户退出或操作完成时，向服务器发送 BYE 消息，并关闭与服务器的连接。

1. 程序入口 (src/main.py)：



```

import sys
from PyQt5.QtWidgets import QApplication
from .atm_gui import ATMGUI
from .atm_client import ATMClient # 核心通信逻辑

def main():
    app = QApplication(sys.argv)

    # 创建 ATM 客户端实例
    client = ATMClient(host='localhost', port=2525)

    # 创建 GUI 并传入客户端实例
    gui = ATMGUI(client)
    gui.show()

    sys.exit(app.exec_())

if __name__ == "__main__":
    main()

```

## 2. 客户端核心通信逻辑 (src/atm\_client.py):

```

import socket
import logging

class ATMClient:
    def __init__(self, host='localhost', port=2525):
        self.host = host
        self.port = port
        self.socket = None
        self.user_id = None
        self.logger = self._setup_logger()
        self.callbacks = { # 用于 GUI 更新的回调函数
            "on_error": None, "on_info": None, "on_login_success": None,
            "on_pin_verified": None, "on_balance_result": None,
            "on_withdraw_success": None, "on_exit": None
        }

    def _setup_logger(self):
        # ... (日志配置) ...
        logger = logging.getLogger('ATMClient')
        # ... (添加处理器) ...
        return logger

    def connect(self):

```

```

        try:
            self.socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
            self.socket.connect((self.host, self.port))
            self.logger.info(f"已连接到服务器: {self.host}:{self.port}")
            return True
        except Exception as e:
            self.logger.error(f"无法连接到服务器: {str(e)}")
            return False

def disconnect(self):
    # ... (关闭 socket) ...

def send_receive(self, message):
    if not self.socket: # ... (错误处理) ...
        try:
            self.logger.info(f"发送消息: {message}")
            self.socket.sendall((message + '\n').encode('utf-8'))
            response = self.socket.recv(1024).decode('utf-8').strip() #
strip()移除末尾换行符
            self.logger.info(f"接收响应: {response}")
            return response
        except Exception as e: # ... (错误处理) ...
            return None

def insert_card(self, user_id):
    self.user_id = user_id
    return self.send_receive(f"HELO {user_id}")

def verify_pin(self, pin):
    return self.send_receive(f"PASS {pin}")

def check_balance(self):
    return self.send_receive("BALA")

def withdraw(self, amount):
    return self.send_receive(f"WDRA {amount}")

def exit(self):
    response = self.send_receive("BYE")
    self.disconnect()
    return response

def set_callbacks(self, callbacks):

```

```

        self.callbacks.update(callbacks)

    def _trigger_callback(self, callback_name, *args):
        callback = self.callbacks.get(callback_name)
        if callback and callable(callback):
            callback(*args)

    # --- 业务逻辑处理方法 (示例) ---
    def process_card_insertion(self, card_number):
        if not card_number: # ... (输入校验) ...
            return False
        if not self.connect(): # ... (连接失败处理) ...
            return False
        response = self.insert_card(card_number)
        if not response: # ... (通信错误处理) ...
            return False
        if response.startswith("500"): # 根据 RFC20232023, 500 表示需要认证
            self._trigger_callback("on_login_success")
            return True
        else:
            self._trigger_callback("on_error", "卡号错误", "无效的卡号或服务
            器响应错误")
            self.disconnect()
            return False

    def process_pin_verification(self, pin):
        # ... (类似 process_card_insertion 的逻辑) ...
        response = self.verify_pin(pin)
        if response and response.startswith("525"): # 525 表示成功
            self._trigger_callback("on_pin_verified")
            return True
        # ... (错误处理) ...
        return False

    def process_balance_check(self):
        # ... (类似逻辑) ...
        response = self.check_balance()
        if response and response.startswith("AMNT:"):
            balance = response.split(":", 1)[1]
            self._trigger_callback("on_balance_result", balance)
            return True
        # ... (错误处理) ...
        return False

    def process_withdrawal(self, amount_text):
        # ... (类似逻辑, 包括金额校验) ...
        response = self.withdraw(float(amount_text))
        if response and response.startswith("525"):

```

```

        self._trigger_callback("on_withdraw_success",
float(amount_text))
        return True
    # ... (错误处理) ...

def process_exit(self):
    self.exit() # 发送 BYE 并断开连接
    self._trigger_callback("on_exit")
    return True

```

### 3. 图形用户界面 (src/atm\_gui.py):

```

from PyQt5.QtWidgets import (QMainWindow, QWidget, QPushButton, QLabel,
QLineEdit,
                                QVBoxLayout, QStackedWidget, QMessageBox,
QGridLayout)
from PyQt5.QtCore import Qt, pyqtSignal, QObject
# ... (其他 Qt 导入) ...

class ATMSignals(QObject): # 用于线程安全地更新 GUI
    error_message = pyqtSignal(str, str)
    info_message = pyqtSignal(str, str)

class ATMGUI(QMainWindow):
    def __init__(self, client):
        super().__init__()
        self.client = client
        self.signals = ATMSignals()

    # 将 ATMClient 的回调连接到 GUI 的槽函数或信号
    self.client.set_callbacks({
        "on_error": self.show_error_from_client,
        "on_info": self.show_info_from_client,
        "on_login_success": self.switch_to_pin_page,
        "on_pin_verified": self.switch_to_main_menu,
        "on_balance_result": self.display_balance,
        "on_withdraw_success": self.handle_withdraw_success,
        "on_exit": self.handle_exit_confirmation
    })

    self.signals.error_message.connect(self.show_error_message_box)
    self.signals.info_message.connect(self.show_info_message_box)

    self.setup_ui()

```

```

self.setup_styles() # 应用样式

def setup_ui(self):
    self.setWindowTitle("ATM 终端")
    self.setMinimumSize(800, 600)
    # ... (设置窗口图标等) ...

    self.stack = QStackedWidget()
    self.setCentralWidget(self.stack)

    self.create_welcome_page()    # 页面索引 0
    self.create_pin_page()        # 页面索引 1
    self.create_main_menu_page()  # 页面索引 2
    self.create_balance_page()    # 页面索引 3
    self.create_withdraw_page()   # 页面索引 4
    # ... (其他页面) ...

    self.stack.setCurrentIndex(0) # 初始显示欢迎页

def create_welcome_page(self):
    page = QWidget()
    layout = QVBoxLayout(page)
    # ... (添加标签、卡号输入框 self.card_input、确认按钮) ...
    confirm_button = QPushButton("插卡 (确认卡号)")
    confirm_button.clicked.connect(self.handle_insert_card_button)
    layout.addWidget(confirm_button)
    self.stack.addWidget(page)

def create_pin_page(self):
    # ... (类似地创建 PIN 输入页面, 包含 self.pin_input 和确认按钮) ...
    #
pin_confirm_button.clicked.connect(self.handle_pin_confirm_button)

def create_main_menu_page(self):
    # ... (创建主菜单页面, 包含查询余额、取款、退卡按钮) ...
    # balance_button.clicked.connect(self.handle_balance_button)
    # withdraw_button.clicked.connect(self.handle_withdraw_button)
    # exit_button.clicked.connect(self.handle_exit_button)

# --- 事件处理器 (GUI 按钮点击等) ---
def handle_insert_card_button(self):
    card_number = self.card_input.text().strip()
    # 可以在这里添加一些基本的客户端校验
    self.client.process_card_insertion(card_number) # 调用客户端逻辑

```

```

def handle_pin_confirm_button(self):
    pin = self.pin_input.text().strip()
    self.client.process_pin_verification(pin)

def handle_balance_button(self):
    self.client.process_balance_check()

def handle_withdraw_button(self):
    # 可能先切换到取款金额输入页面
    self.stack.setCurrentIndex(4) # 假设取款页面是索引 4

def handle_exit_button(self):
    self.client.process_exit()

# --- 回调槽函数 (由 ATMClient 调用) ---
def show_error_from_client(self, title, message):
    self.signals.error_message.emit(title, message) # 通过信号在主线程
显示

def show_info_from_client(self, title, message):
    self.signals.info_message.emit(title, message)

def switch_to_pin_page(self):
    self.pin_input.clear() # 清空上次输入
    self.stack.setCurrentIndex(1)

def switch_to_main_menu(self):
    self.stack.setCurrentIndex(2)
    # 可能需要清空一些状态或输入框

def display_balance(self, balance_str):
    self.balance_amount_label.setText(f"您的余额: ¥{balance_str}") #
假设有这个 label
    self.stack.setCurrentIndex(3) # 切换到余额显示页

def handle_withdraw_success(self, amount):
    self.signals.info_message.emit("取款成功", f"成功取出:
¥{amount}")
    self.withdraw_input.clear() # 清空取款输入框
    self.switch_to_main_menu() # 返回主菜单

def handle_exit_confirmation(self):
    self.signals.info_message.emit("再见", "感谢使用, 请取走您的卡片。")

```

```
# 可以选择关闭窗口或返回欢迎页
self.stack.setCurrentIndex(0) # 返回欢迎页
self.card_input.clear()

# --- 消息框 ---
def show_error_message_box(self, title, message):
    QMessageBox.critical(self, title, message)

def show_info_message_box(self, title, message):
    QMessageBox.information(self, title, message)

def closeEvent(self, event):
    """窗口关闭事件，确保断开客户端连接"""
    self.client.disconnect()
    super().closeEvent(event)

# ... (样式设置 setup_styles 方法等) ...
```

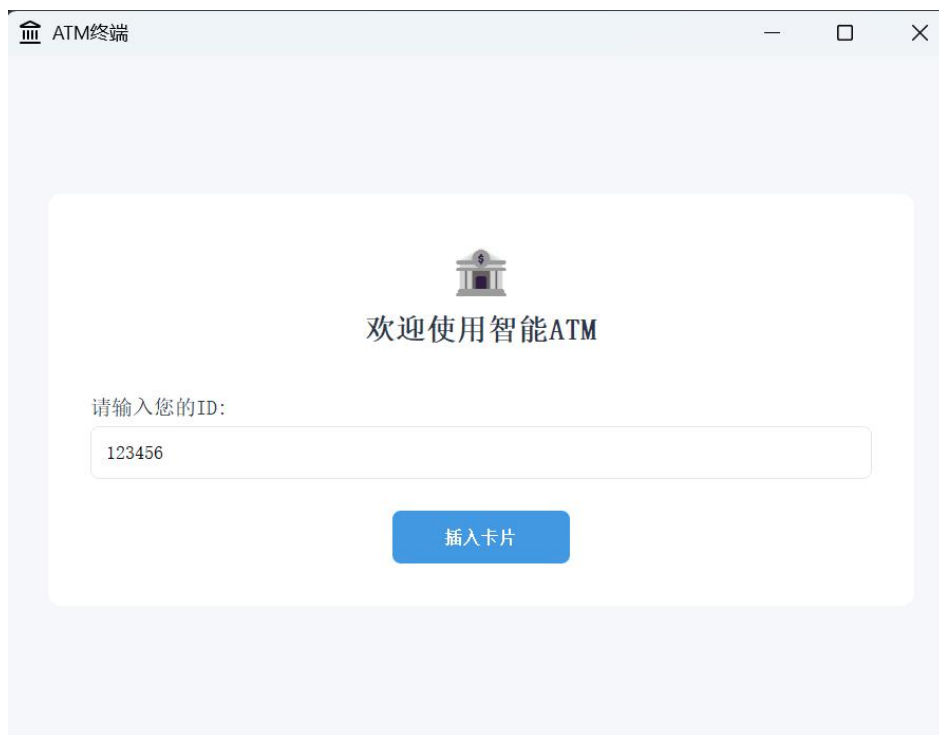
## 5. 测试运行结果

### 5.1 组内测试

经查询，组内成员的 IPv4 地址为 10.244.203.114

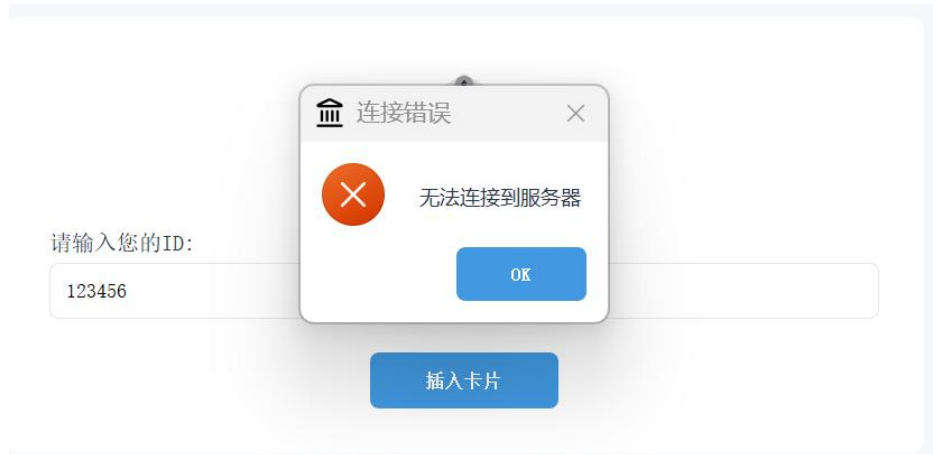
#### 5.1.1 登录界面

输入 [users.json](#) 中的账户 ID，本次为测试账户 123456。



#### 5.1.1.1 连接服务器不存在

若服务器不存在，则弹出窗口，显示无法连接服务器，控制台输出错误信息。



```
2025-05-25 15:37:40,063 - ATMClient - ERROR - 无法连接到服务器: [WinError 10060] 由于连接方在一段时间后没有正确答复或连接的主机没有反应，连接尝试失败。
```

#### 5.1.1.2 输入卡号错误

若输入卡号不在 json 文件内，则弹出窗口显示无效的卡号。



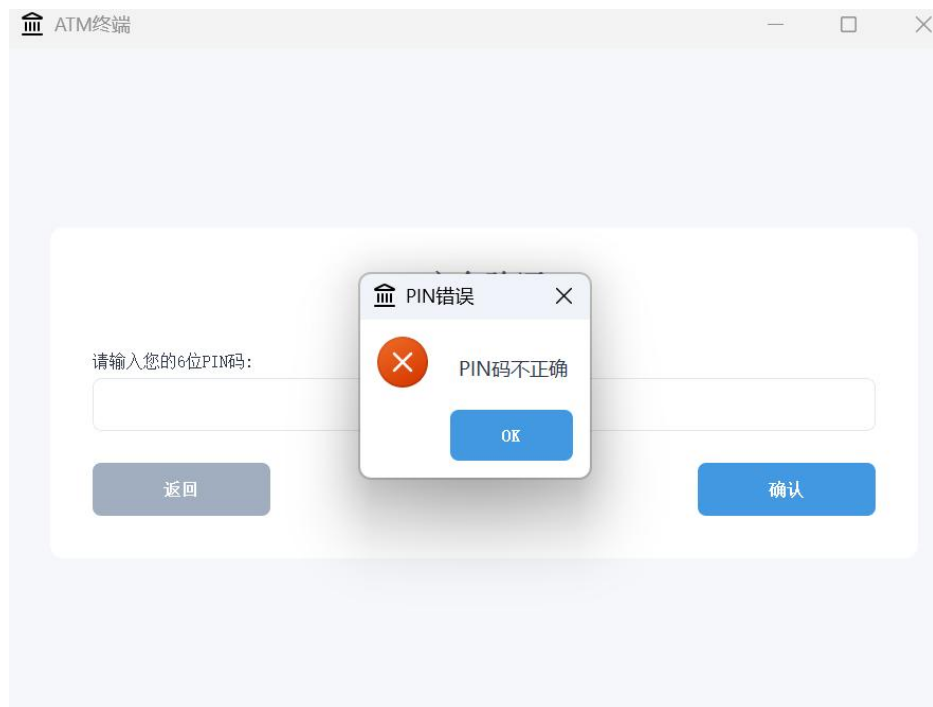


### 5.1.2 输入密码

成功连接到服务器以及卡号输入成功后，转到输入密码界面



若输入密码无效，则提示密码不正确



### 5.1.3 主菜单界面

登录成功进入主菜单界面，选择下一步操作，此处只提供查询余额与取款操作。



### 5.1.4 查询余额

点击查询余额按钮，显示当前账户剩余余额。



### 5.1.5 取款服务

选择取款服务，进入取款界面，可选择预选项，也可自定义取款余额。



取款成功，弹出取款成功窗口，显示取款余额。

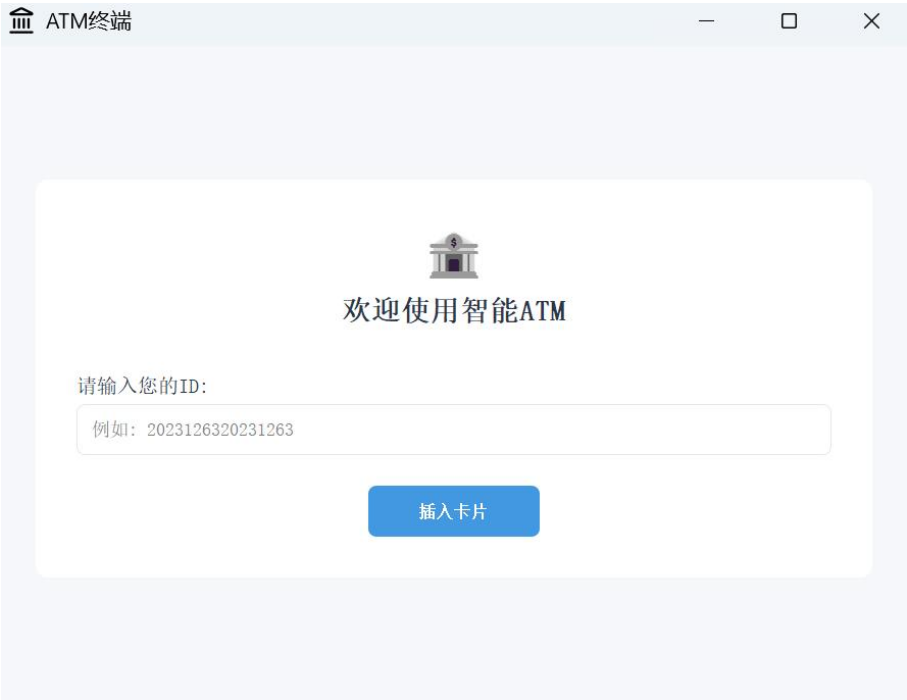


取款不合法（超出余额，取款为负数等），弹出取款失败窗口。



### 5.1.6 退出

点击退出按钮，退出到输入卡号欢迎界面。



日志文件如下：

```
-----组内测试-----
2025-05-23 15:48:49,008 - ATMServer - INFO - 从 data/users.json 加载了 2 个用户
2025-05-23 15:48:49,009 - ATMServer - INFO - 服务器启动于 10.244.203.114:2525
2025-05-23 15:48:50,097 - ATMServer - INFO - 新连接来自 ('10.244.36.124', 9702)
2025-05-23 15:48:50,097 - ATMServer - INFO - 收到来自 ('10.244.36.124', 9702) 的消息: HELO 123456
2025-05-23 15:48:50,097 - ATMServer - INFO - 发送到 ('10.244.36.124', 9702): 500 AUTH REQUIRED!
2025-05-23 15:48:56,842 - ATMServer - INFO - 收到来自 ('10.244.36.124', 9702) 的消息: PASS 1234
2025-05-23 15:48:56,843 - ATMServer - INFO - 发送到 ('10.244.36.124', 9702): 525 OK!
2025-05-23 15:49:05,452 - ATMServer - INFO - 收到来自 ('10.244.36.124', 9702) 的消息: BALA
2025-05-23 15:49:05,452 - ATMServer - INFO - 发送到 ('10.244.36.124', 9702): AMNT:10000.0
2025-05-23 15:49:26,985 - ATMServer - INFO - 收到来自 ('10.244.36.124', 9702) 的消息: WDRA 500.0
2025-05-23 15:49:26,986 - ATMServer - INFO - 保存了 2 个用户数据到 data/users.json
2025-05-23 15:49:26,986 - ATMServer - INFO - 发送到 ('10.244.36.124', 9702): 525 OK
2025-05-23 15:49:38,529 - ATMServer - INFO - 收到来自 ('10.244.36.124', 9702) 的消息: WDRA 50000.0
2025-05-23 15:49:38,529 - ATMServer - INFO - 发送到 ('10.244.36.124', 9702): 401 ERROR!
2025-05-23 15:49:46,208 - ATMServer - INFO - 收到来自 ('10.244.36.124', 9702) 的消息: BYE
2025-05-23 15:49:46,209 - ATMServer - INFO - 发送到 ('10.244.36.124', 9702): BYE
2025-05-23 15:49:46,209 - ATMServer - INFO - 连接关闭: ('10.244.36.124', 9702)
2025-05-23 15:49:48,669 - ATMServer - INFO - 新连接来自 ('10.244.36.124', 9779)
2025-05-23 15:49:48,669 - ATMServer - INFO - 收到来自 ('10.244.36.124', 9779) 的消息: HELO 123456
2025-05-23 15:49:48,670 - ATMServer - INFO - 发送到 ('10.244.36.124', 9779): 500 AUTH REQUIRED!
2025-05-23 15:49:58,087 - ATMServer - INFO - 收到来自 ('10.244.36.124', 9779) 的消息: PASS 111
2025-05-23 15:49:58,087 - ATMServer - INFO - 发送到 ('10.244.36.124', 9779): 401 ERROR!
2025-05-23 15:50:16,818 - ATMServer - INFO - 连接关闭: ('10.244.36.124', 9779)
```

## 5.2 组间测试

### 5.2.1 测试对象：第 30 组 郑宇飞 20231127


30	郑宇飞 20231127	<a href="https://github.com/Lunapicd/CQU-ATM-Homework">https://github.com/Lunapicd/CQU-ATM-Homework</a>	12345	1234	¥1,000.00
----	--------------	---	-------	------	-----------

### 5.2.2 我方作为 Client，对方作为 Server

对方服务器端 IP 地址为：10.244.166.28，在 main.py 输入后启动客户端

程序。

ATM终端



欢迎使用智能ATM

请输入您的ID:

12345

插入卡片

ATM终端

安全验证

请输入您的6位PIN码:

返回

确认

成功登录:



查询余额:



取款 500 元:



#### 日志记录:

```
2025-05-23 14:50:24 - 服务器在端口 2525 上启动
2025-05-23 14:51:11 - 来自 /10.244.36.124 的新连接
2025-05-23 14:51:11 - 收到命令: HELO 12345
2025-05-23 14:51:11 - 用户ID有效: 12345
2025-05-23 14:51:22 - 来自 /10.244.36.124 的新连接
2025-05-23 14:51:22 - 收到命令: HELO 12345
2025-05-23 14:51:22 - 与客户端的连接已关闭
2025-05-23 14:51:22 - 用户ID有效: 12345
2025-05-23 14:51:36 - 收到命令: PASS 1234
2025-05-23 14:51:36 - 用户 12345 认证成功
2025-05-23 14:51:41 - 收到命令: BALA
2025-05-23 14:51:41 - 用户 12345 查询余额: 700.0
2025-05-23 14:52:06 - 收到命令: WDRA 500.0
2025-05-23 14:52:06 - 账户数据已保存到文件
2025-05-23 14:52:06 - 用户 12345 取款成功: 500.0
2025-05-23 14:52:12 - 收到命令: BYE
2025-05-23 14:52:12 - 用户 12345 会话结束
2025-05-23 14:52:12 - 与客户端的连接已关闭
```



5.2.3 我方作为 Server，对方作为 Client  
成功登录：



查询余额：



取款 100 元：



### 日志记录:

```
2025-05-23 14:47:31,576 - ATMServer - INFO - 从 data/users.json 加载了 2 个用户
2025-05-23 14:47:31,578 - ATMServer - INFO - 服务器启动于 0.0.0.0:2525
2025-05-23 14:48:33,046 - ATMServer - INFO - 新连接来自 ('10.244.166.28', 62210)
2025-05-23 14:48:33,048 - ATMServer - INFO - 收到来自 ('10.244.166.28', 62210) 的消息: HELO 2023126320230109
2025-05-23 14:48:33,048 - ATMServer - INFO - 发送到 ('10.244.166.28', 62210): 500 AUTH REQUIRED!
2025-05-23 14:48:38,154 - ATMServer - INFO - 收到来自 ('10.244.166.28', 62210) 的消息: PASS 123456
2025-05-23 14:48:38,154 - ATMServer - INFO - 发送到 ('10.244.166.28', 62210): 525 OK!
2025-05-23 14:48:40,349 - ATMServer - INFO - 收到来自 ('10.244.166.28', 62210) 的消息: BALA
2025-05-23 14:48:40,349 - ATMServer - INFO - 发送到 ('10.244.166.28', 62210): AMNT:50000
2025-05-23 14:48:55,872 - ATMServer - INFO - 收到来自 ('10.244.166.28', 62210) 的消息: WDRA 100.0
2025-05-23 14:48:55,872 - ATMServer - INFO - 保存了 2 个用户数据到 data/users.json
2025-05-23 14:48:55,872 - ATMServer - INFO - 发送到 ('10.244.166.28', 62210): 525 OK
```

## 6 总结

在完成作业一和作业二的过程中，我们学习了计算机网络应用层协议的设计原理与 TCP 套接字编程的实践方法，同时也遇到了一些挑战。

**遇到的主要问题及解决思路：**

### 1. 编程语言选择的问题：

- **问题：**在项目初期，对于选择何种编程语言来实现客户端和服务端曾有过考量。不同的语言在网络编程、GUI 库支持以及开发效率上各有优劣。
- **解决思路：**最终我们选择了 Python。主要原因是 Python 在网络编程方面拥有简洁易用的 socket 库，同时 PyQt5 库能很好地支持图形用户界面的开发。

### 2. 客户端与服务器端协议处理方式不匹配：

- **问题：**在进行组间服务器对接时，遇到了通信问题。具体表现为，本组客户端最初采用连续发送数据包的方式组织协议消息，而对方服务器期望通过类似 Java `BufferedReader` 的 `readLine()` 方式，即逐行读取以换行符 `\n` 分隔的协议指令。这导致对方服务器无法正确解析我方客户端发送的连续数据流，从而连接失败或行为异常。
- **解决思路：**为了解决这个问题，我们对客户端的 `send_receive` 方法进行了调整：在发送每条协议消息时，明确在消息末尾添加换行符 `\n`，确保每条指令都作为单独的一行发送。这样，期望逐行读取的服务器端就能够正确地接收和解析每一条指令，从而保证了通信的兼容性和可靠性，具体为在 `send_receive` 方法中，发送数据的代码从 `self.socket.sendall(message.encode('utf-8'))` 修改为 `self.socket.sendall((message + '\n').encode('utf-8'))`。

**收获与体会：**

通过本次系列作业，我们获得了多方面的提升：

- **理论联系实际：**将《计算机网络》课程中学习到的 TCP/IP 协议、套接字编程、应用层协议设计等抽象概念，成功应用于一个具体的模拟系统中，加深了对核心原理的理解。
- **编程与调试技能增强：**通过实际编写和调试客户端与服务器端的代码，我

们对 Python 网络编程、GUI 设计以及多线程/异步处理有了更熟练的掌握。解决通信故障的过程也极大地锻炼了我们的问题定位和调试能力。

- **标准化与协作的重要性：**在组间测试环节，我们认识到遵循统一标准（即协议规范）对于不同团队开发的组件能够协同工作的基础性作用。清晰的文档和有效的沟通同样不可或缺。