МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»

Институт № 3 «Системы управления, информатика и электроэнергетика»

Кафедра 304 «Вычислительные машины, системы и сети»

Лабораторная работа № 10

по дисциплине «WEB технологии»

на тему «Чат на сокетах»

Выполнил

студент группы М3О-125БВ-24

Егоров А.В.

Приняли

ассистент каф. 304 Борисов А.И.

Москва
2025

Содержание:

Постановка задачи:

1. Разработать программу-сервер, которая хранит список всех подключении клиентов (например, в динамическом массиве).

Необходимо хранить сокет подключения, полученный в результате функции accept.

Для каждого клиента сервер запрашивает кодовое имя (nickname). При получении сообщения от клиента (с соответствующим nickname) происходит передача этого сообщения всем остальным клиентам через сокетное соединение.

Сервер сохраняет сообщение пользователя в файл. При подключении клиента передаются (от сервера к клиенту) последние 10 сообщений из данного файла.

2. Разработать программу-клиент, которая работает в диалоге с пользователем и позволяет ему вводить данные. При подключении к серверу (ip-адрес и порт сервера вводится пользователем) сервер запрашивает nickname, а пользователь вводит его в программе-клиенте.

Программа-клиент позволяет вводить сообщения пользователем от имени nickname и выводит сообщения от сервера на экран. Последние 5 сообщений сохраняются в текстовом файле (эмулирую кэширование). При запуске программы выводится содержимое данного файла.

3. Развернуть данное приложение в сети с одним сервером и 3-мя запущенными программами-клиентами. В отчете привести файлы клиента и сервера, а также скриншот работы программы.

Код

client.py

```
import socket
import threading
import sys
import os

CACHE_FILE = 'client_chat_cache.txt'
CACHE_LINES = 5

def load_cache():
    # При запуске клиента выводим последние сообщения из локального кэша,
чтобы пользователь видел историю прошлых сессий
    try:
        if os.path.exists(CACHE_FILE):
            with open(CACHE_FILE, 'r', encoding='utf-8') as f:
                lines = f.readlines()
            print("--- Последние сообщения из кэша ---")
            for line in lines:
                print(line.strip())
            print("--- Конец кэша ---")
    except Exception as e:
        print(f"Ошибка загрузки кэша: {e}")

def save_to_cache(message):
    # Сохраняем новое сообщение в кэш, чтобы при следующем запуске чата
пользователь видел последние сообщения
    try:
        lines = []
        if os.path.exists(CACHE_FILE):
            with open(CACHE_FILE, 'r', encoding='utf-8') as f:
                lines = f.readlines()
        lines.append(message + '\n')
        # Оставляем только последние CACHE_LINES сообщений
        lines_to_write = lines[-CACHE_LINES:]
        with open(CACHE_FILE, 'w', encoding='utf-8') as f:
            f.writelines(lines_to_write)
    except Exception as e:
        print(f"Ошибка сохранения в кэш: {e}")

def receive_messages(client_socket):
    # Этот поток постоянно слушает сервер и выводит все входящие сообщения на
экран
    while True:
        try:
            message = client_socket.recv(1024).decode('utf-8')
            if not message:
                print("Отключено от сервера (пустое сообщение).")
                break
            if message == "SERVER_SHUTDOWN":
                print("Сервер завершает работу. Отключение.")
                break
            if message == "NICKNAME_REQUEST":
```

```
                continue
            print(message.strip())
        except socket.error as e:
            print(f"Ошибка соединения: {e}")
            break
        except Exception as e:
            print(f"Ошибка получения сообщения: {e}")
            break
    client_socket.close()
    print("Соединение закрыто.")
    os._exit(0) # Если этот поток завершился, выходим из всей программы

def start_client(host, port):
    # Основная функция клиента: подключение, ввод ника, отправка и получение
сообщений
    load_cache()
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        client_socket.connect((host, port))
        print(f"Подключено к серверу {host}:{port}")
    except socket.error as e:
        print(f"Не удалось подключиться к серверу: {e}")
        return
    except Exception as e:
        print(f"Непредвиденная ошибка при подключении: {e}")
        return

    nickname = ""
    try:
        # После подключения ждем от сервера запрос ника
        initial_data = client_socket.recv(1024).decode('utf-8')
        if initial_data.strip() == "NICKNAME_REQUEST":
            while not nickname:
                nickname = input("Введите ваш никнейм: ").strip()
                if not nickname:
                    print("Никнейм не может быть пустым.")
            client_socket.sendall(nickname.encode('utf-8'))
        else:
            print(initial_data.strip())
            if not nickname:
                while not nickname:
                    nickname = input("Введите ваш никнейм: ").strip()
                    if not nickname:
                        print("Никнейм не может быть пустым.")
                client_socket.sendall(nickname.encode('utf-8'))

        # Запускаем отдельный поток для получения сообщений от сервера
        receive_thread = threading.Thread(target=receive_messages,
args=(client_socket,))
        receive_thread.daemon = True
        receive_thread.start()

        if nickname:
            print(f"Вы вошли как {nickname}. Вводите сообщения и нажимайте
Enter. Введите 'выход' для завершения.")
```

```
        else:
            print("Вводите сообщения и нажимайте Enter. Введите 'выход' для
завершения.")

        # Основной цикл: читаем ввод пользователя и отправляем на сервер
        while True:
            message_to_send = input()
            if message_to_send.lower() == 'выход':
                break
            if message_to_send:
                client_socket.sendall(message_to_send.encode('utf-8'))
                save_to_cache(f"Вы: {message_to_send}")

    except KeyboardInterrupt:
        print("\nОтключение...")
    except socket.error as e:
        print(f"Ошибка сокета во время связи: {e}")
    except Exception as e:
        print(f"Произошла ошибка: {e}")
    finally:
        print("Закрытие соединения.")
        client_socket.close()

if __name__ == '__main__':
    # Проверяем аргументы командной строки и запускаем клиент
    if len(sys.argv) != 3:
        print("Использование: python client.py <ip_адрес_хоста> <порт>")
        sys.exit(1)
    host_ip = sys.argv[1]
    try:
        port_num = int(sys.argv[2])
        if not (0 <= port_num <= 65535):
            raise ValueError("Номер порта должен быть от 0 до 65535.")
    except ValueError as e:
        print(f"Неверный номер порта: {e}")
        sys.exit(1)
    start_client(host_ip, port_num)
```

server.py

```
import socket
import threading
import datetime

HOST = '127.0.0.1'
PORT = 65432
MAX_CLIENTS = 10
CHAT_LOG_FILE = 'chat_log.txt'
HISTORY_LINES = 10

clients = []
client_lock = threading.Lock()

def log_message(message):
    # Сохраняем сообщение в файл с датой и временем
    try:
        with open(CHAT_LOG_FILE, 'a', encoding='utf-8') as f:
            timestamp = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:
%S')
            f.write(f"[{timestamp}] {message}\n")
    except Exception as e:
        print(f"Ошибка записи в лог: {e}")

def get_chat_history():
    # Берём последние N сообщений из файла истории
    try:
        with open(CHAT_LOG_FILE, 'r', encoding='utf-8') as f:
            lines = f.readlines()
        return "".join(lines[-HISTORY_LINES:])
    except FileNotFoundError:
        return "Истории чата еще нет.\n"
    except Exception as e:
        print(f"Ошибка чтения истории чата: {e}")
        return "Ошибка получения истории.\n"

def broadcast_message(message, sender_socket):
    # Рассылаем сообщение всем, кроме отправителя
    with client_lock:
        for client_socket, _, nickname in clients:
            if client_socket != sender_socket:
                try:
                    client_socket.sendall(message.encode('utf-8'))
                except socket.error as e:
                    print(f"Ошибка отправки {nickname}: {e}")

def handle_client(conn, addr):
    print(f"Подключен клиент: {addr}")
    nickname = ""
    try:
        # Сначала просим никнейм
        conn.sendall("NICKNAME_REQUEST".encode('utf-8'))
        nickname_data = conn.recv(1024)
        if not nickname_data:
```

```
            print(f"Клиент {addr} отключился до отправки никнейма.")
            return
        nickname = nickname_data.decode('utf-8').strip()

        # Проверяем, не занят ли никнейм
        with client_lock:
            for _, _, existing_nick in clients:
                if existing_nick == nickname:
                    conn.sendall(f"Никнейм '{nickname}' уже занят.
Пожалуйста, переподключитесь с другим никнеймом.\n".encode('utf-8'))
                    print(f"Клиент {addr} попытался использовать занятый
никнейм: {nickname}")
                    return

        # Добавляем клиента в общий список
        with client_lock:
            clients.append((conn, addr, nickname))

        print(f"Клиент {addr} идентифицирован как {nickname}")
        log_message(f"{nickname} присоединился к чату.")
        broadcast_message(f"--- {nickname} присоединился к чату. ---\n",
conn)

        # Отправляем историю чата новому участнику
        history = get_chat_history()
        if history and history.strip() != "Истории чата еще нет.":
            conn.sendall(f"--- Последние {HISTORY_LINES} сообщений ---
\n".encode('utf-8'))
            conn.sendall(history.encode('utf-8'))
            conn.sendall("--- Конец истории ---\n".encode('utf-8'))
        elif history.strip() == "Истории чата еще нет.":
            conn.sendall("Истории чата еще нет.\n".encode('utf-8'))

        # Основной цикл общения с клиентом
        while True:
            data = conn.recv(1024)
            if not data:
                break

            message_text = data.decode('utf-8').strip()
            full_message = f"{nickname}: {message_text}"
            print(f"Получено от {nickname}: {message_text}")

            log_message(full_message)
            broadcast_message(full_message + "\n", conn)

    except socket.error as e:
        print(f"Ошибка сокета с {nickname} ({addr}): {e}")
    except Exception as e:
        print(f"Ошибка обработки клиента {nickname} ({addr}): {e}")
    finally:
        # Удаляем клиента из списка при отключении
        with client_lock:
            clients[:] = [c for c in clients if c[0] != conn]
```

```
        if nickname:
            print(f"{nickname} ({addr}) отключился.")
            log_message(f"{nickname} покинул чат.")
            broadcast_message(f"--- {nickname} покинул чат. ---\n", conn)
        else:
            print(f"Клиент {addr} отключился (без никнейма).")

        conn.close()

def start_server():
    # Запускаем сервер и ждем подключения клиентов
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        try:
            s.bind((HOST, PORT))
        except socket.error as e:
            print(f"Ошибка привязки: {e}")
            return

        s.listen(MAX_CLIENTS)
        print(f"Сервер слушает на {HOST}:{PORT}")

        try:
            while True:
                conn, addr = s.accept()
                thread = threading.Thread(target=handle_client, args=(conn,
addr))
                thread.daemon = True
                thread.start()
        except KeyboardInterrupt:
            print("\nСервер останавливается...")
        finally:
            print("Закрытие всех клиентских соединений...")
            with client_lock:
                for client_socket, _, _ in clients:
                    try:

client_socket.sendall("SERVER_SHUTDOWN".encode('utf-8'))
                        client_socket.close()
                    except socket.error:
                        pass
            s.close()
            print("Сервер остановлен.")

if __name__ == '__main__':
    start_server()
```

Работа программы

Серверная часть

Сервер запускается на выделенном компьютере и ожидает подключения клиентов по заданному IP-адресу и порту. После установления соединения сервер запрашивает у клиента уникальный никнейм. Если никнейм уже занят, сервер отклоняет подключение с соответствующим уведомлением. При успешном подключении сервер отправляет клиенту последние 10 сообщений из истории чата, чтобы новый пользователь мог ознакомиться с предыдущей перепиской. Все сообщения, поступающие от клиентов, сервер сохраняет в файл журнала chat_log.txt. После получения сообщения от одного клиента сервер пересылает его всем остальным подключённым клиентам, обеспечивая групповое взаимодействие в реальном времени. При отключении клиента сервер информирует остальных участников о его выходе из чата.

Сервер реализован на языке Python с использованием стандартного модуля **socket** для сетевого взаимодействия и модуля **threading** для обслуживания нескольких клиентов одновременно. Для каждого подключённого клиента создаётся отдельный поток, что позволяет серверу обрабатывать сообщения параллельно и не блокировать других пользователей. Для хранения истории сообщений используется текстовый файл, в который каждое сообщение записывается с временной меткой. Для синхронизации доступа к общим данным по типу списка клиентов используется объект блокировки **threading.Lock**.

Клиентская часть

Клиентская программа запускается пользователем и подключается к серверу по указанному адресу и порту. После установления соединения клиент вводит свой никнейм, который проверяется на уникальность сервером. После успешной авторизации клиент получает последние 10 сообщений из истории чата. Пользователь может отправлять сообщения, которые будут видны всем участникам чата. Все входящие и исходящие сообщения отображаются на экране клиента. Клиент сохраняет последние 5 сообщений в локальный файл-кэш **client_chat_cache.txt**. При последующем запуске программы эти сообщения отображаются пользователю, даже если сервер временно недоступен.

Клиент также реализован на Python с использованием модуля **socket** для сетевого взаимодействия и **threading** для параллельного приёма сообщений от сервера. Для отображения истории сообщений при запуске

клиента реализована функция чтения локального кэша. Отправка и приём сообщений реализованы в отдельных потоках: основной поток отвечает за ввод пользователя, а дополнительный поток — за приём и отображение сообщений от сервера. Для хранения последних сообщений используется локальный текстовый файл, который обновляется при каждом новом сообщении.

Вывод по работе:

Работа позволила на практике изучить, как реализуется взаимодействие клиент-сервер в сетях, как устанавливается и завершается соединение на уровне сокетов, и как происходит обмен сообщениями между пользователями. В процессе были реализованы функции идентификации пользователей, история сообщений, а также механизмы логирования и кэширования переписки. Использование потоков позволило обеспечить одновременную работу с несколькими клиентами.