Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский авиационный институт
(Национальный исследовательский университет)»

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

КУРСОВОЙ ПРОЕКТ

«Операционные системы»

Студент: Князьков Николай Дмитриевич
Группа: М80–203БВ–24
Вариант: 28
Преподаватель: Соколов А. А.
Оценка:_____
Дата:_____
Подпись:_____

Москва, 2025

Курсовой проект. Клиент-серверная система мгновенных сообщений

1.1 Постановка задачи

Цель проекта: проектирование и реализация клиент-серверной системы мгновенных сообщений с использованием механизмов ОС (именованных FIFO), обеспечивающей:

- Регистрацию клиента на сервере по логину;
- Отправку сообщений от клиента другому клиенту по логину;
- Приём сообщений клиентом в реальном времени;
- Поддержку отложенной отправки сообщений, сохраняемых на сервере;
- Использование pipe'ов (именованных FIFO) для связи между сервером и клиентами.

1.2 Общие сведения о системе и требования

Система состоит из двух модулей:

- `server` — демон/служба, сохраняет состояние клиентов, обрабатывает команды и доставляет сообщения;
- `client` — консольный клиент, подключается к серверу по логину, отправляет команды и получает входящие сообщения в реальном времени.

Все коммуникации реализуются через именованные FIFO (pipe'ы). Сервер управляет очередью сообщений и хранит отложенные сообщения на диске. Клиент одновременно отправляет команды и принимает сообщения через отдельный поток.

1.3.1 Использование pipe'ов

Для связи множества независимых процессов применяются именованные FIFO (`mkfifo`). Сервер создаёт один «well-known» FIFO для регистрации и команд клиентов. Каждый клиент создаёт свой FIFO для приёма сообщений и сообщает серверу путь при регистрации.

1.3.2 Каналы и имена

- `SERVER_CMD_FIFO` — фиксированный FIFO (например, `/tmp/chat_srv_cmd`), куда клиенты отправляют команды;
- Клиентский FIFO: `/tmp/chat_cli_<login>`, создаётся при первом запуске.

1.3.3 Формат команд

- Регистрация: `REGISTER|<login>|<fifo_path>`
- Немедленное сообщение: `SEND|<from>|<to>|<msg_id>|<body>`
- Отложенное сообщение: `SCHEDULE|<from>|<to>|<deliver_ts>|<msg_id>|<b`
- Отключение: `UNREGISTER|<login>`
- Подтверждение доставки: `DELIVERED|<msg_id>`

1.3.4 Поведение сервера

- Чтение команд из `SERVER_CMD_FIFO` (nonblock + select/poll или отдельный поток);
- REGISTER: добавление клиента в таблицу и доставка накопленных сообщений;
- SEND: доставка немедленно или сохранение, если получатель офлайн;

- SCHEDULE: сохранение сообщения с временем доставки, отдельный поток проверяет очередь;
- Обеспечение доставки сообщений при выходе отправителя.

1.3.5 Поведение клиента

- Создание собственного FIFO;
- Отправка REGISTER;
- Два потока: ввод команд и приём сообщений;
- При выходе отправка UNREGISTER (опционально).

1.4 Хранение отложенных сообщений

Сервер хранит сообщения в файле (например, `messages.db`) или SQLite. При старте сервер загружает файл в память и запускает поток, доставляющий сообщения по мере наступления времени. После успешной доставки сообщения удаляются.

1.5 Обработка ошибок

При невозможности записи в FIFO клиента сообщение остаётся в очереди для повторной попытки. Очередь сохраняется на диске при аварийном завершении сервера.

1.6 Примеры последовательностей действий

1. Клиент Alice создаёт `/tmp/chat_cli_Alice` и регистрируется.
2. Клиент Bob регистрируется аналогично.
3. Alice отправляет SEND → Bob, Bob получает сообщение немедленно.
4. Alice отправляет SCHEDULE → Carol, Carol регистрируется позже — сервер доставляет сообщение в назначенное время.
5. Alice выходит, отложенные сообщения доставляются по расписанию.

1.7 Примеры кода (C, POSIX)

1.7.1 Клиент — регистрация и отправка команды

```c
// client_send.c
int fd = open("/tmp/chat_srv_cmd", O_WRONLY);
if (fd == -1) { perror("open server fifo"); exit(1); }
dprintf(fd, "REGISTER|%s|%s\n", login, client_fifo_path);
close(fd);
```

1.7.2 Сервер — чтение команд

```c
// server_main.c
#define CMD_FIFO "/tmp/chat_srv_cmd"
mkfifo(CMD_FIFO, 0666);
int fd = open(CMD_FIFO, O_RDONLY | O_NONBLOCK);
char buf[4096];
while (running) {
    ssize_t r = read(fd, buf, sizeof(buf)-1);
    if (r > 0) {
        buf[r] = '\0';
        //                          REGISTER/SEND/SCHEDULE/UNREGISTER
    } else {
        usleep(100000);
    }
}
```

1.7.3 Сервер — запись в FIFO клиента

```c
int send_to_client(const char *client_fifo, const char *payload) {
    int fd = open(client_fifo, O_WRONLY | O_NONBLOCK);
    if (fd == -1) {
        if (errno == ENXIO || errno == ENOENT) return -1;
        perror("open client fifo");
        return -1;
    }
    write(fd, payload, strlen(payload));
```

```
    close(fd);
    return 0;
}
```

1.7.4 Сервер — таймер доставки сообщений

```
while (running) {
    now = time(NULL);
    //                    scheduled

    sleep(1);
}
```

1.7.5 Клиент — приём сообщений

```
// client_receive.c
int fd = open(client_fifo_path, O_RDONLY);
char buf[2048];
while ((n = read(fd, buf, sizeof(buf)-1)) > 0) {
    buf[n] = '\0';
    printf("%s", buf);
}
close(fd);
```

1.8 Формат передаваемых сообщений

`MSG|<from>|<msg_id>|<timestamp>|<body>`

Клиент парсит строку и печатает: `[HH:MM] from: body (id=msg_id)`.

1.9 Примеры работы

- Два клиента (Alice, Bob). SEND → Bob — немедленное получение.
- Alice отправляет SCHEDULE → Carol, Carol регистрируется позже — доставка по времени.
- Сервер рестартует, очередь сохранена на диске, доставка продолжается.

1.10 Замечания по безопасности

- FIFO в /tmp уязвимы к подмене; для учебного проекта допустимо;
- Для защиты очереди сообщений применять mutex или файл-lock (flock).

1.11 Выводы

Прототип удовлетворяет требованиям: регистрация по логину, отправка/приём сообщений в реальном времени, поддержка отложенных сообщений, сохранение очередей на сервере и доставка при выходе отправителя. Использование именованных FIFO позволило реализовать обмен без сокетов, соблюдая условие задания.