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 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 Листинги проекта

1.6.1 include/client.h

```
#ifndef CLIENT_H
#define CLIENT_H

#define _GNU_SOURCE

#define SERVER_FIFO "/tmp/server_fifo"
#define MAX_LOGIN 64
```

```
#define MAX_FIFO_PATH 256
#define MAX_MESSAGE 1024

extern char login[MAX_LOGIN];
extern char my_fifo[MAX_FIFO_PATH];
extern int running;

void *reader_thread(void *arg);
int send_to_server(const char *txt);

#endif
```

1.6.2 include/server.h

```
#ifndef SERVER_H
#define SERVER_H

#define _GNU_SOURCE
#include <pthread.h>
#include <time.h>

#define SERVER_FIFO "/tmp/server_fifo"
#define MAX_LOGIN 64
#define MAX_FIFO_PATH 256
#define MAX_MESSAGE 1024

typedef struct Client
{
    char login[MAX_LOGIN];
    char fifo_path[MAX_FIFO_PATH];
    int online;
    struct Client *next;
} Client;

typedef struct Message
{
    char from[MAX_LOGIN];
    char to[MAX_LOGIN];
    char text[MAX_MESSAGE];
    time_t deliver_time;
    struct Message *next;
```

```
} Message;

extern Client *clients;
extern Message *messages;
extern pthread_mutex_t mtx;
extern pthread_cond_t cond;
extern int running;

Client *find_client_locked(const char *login);
void register_client_locked(const char *login, const char *
    fifo_path);
void unregister_client_locked(const char *login);

void push_message_locked(Message *m);
Message *pop_earliest_locked(void);
Message *pop_specific_for_recipient_locked(const char *to);

void try_deliver_message(Message *m);
void *scheduler_thread(void *arg);
void deliver_pending_for_client_locked(const char *login);

void process_command(char *line);

#endif
```

1.6.3 src/client_comm.c

```
#define _GNU_SOURCE
#include "../include/client.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>

char login[MAX_LOGIN];
char my_fifo[MAX_FIFO_PATH];
int running = 1;

```

```c
void *reader_thread(void *arg)
{
    (void)arg;
    int fd = open(my_fifo, O_RDONLY);
    if (fd < 0)
    {
        perror("open my fifo for read");
        return NULL;
    }
    char buf[2048];
    while (running)
    {
        ssize_t r = read(fd, buf, sizeof(buf) - 1);
        if (r > 0)
        {
            buf[r] = '\0';
            printf("\n=== NEW MESSAGE ===\n%s\n> ", buf);
            fflush(stdout);
        }
        else
        {
            usleep(100000);
        }
    }
    close(fd);
    return NULL;
}

int send_to_server(const char *txt)
{
    int fd = open(SERVER_FIFO, O_WRONLY | O_NONBLOCK);
    if (fd < 0)
    {
        fd = open(SERVER_FIFO, O_WRONLY);
        if (fd < 0)
        {
            perror("open server fifo for write");
            return -1;
        }
    }
    write(fd, txt, strlen(txt));
```

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

1.6.4 src/client_main.c

```
#define _GNU_SOURCE
#include "../include/client.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/stat.h>
#include <errno.h>

int main()
{
    printf("Enter your login: ");
    if (!fgets(login, sizeof(login), stdin))
        return 0;

    login[strcspn(login, "\n")] = '\0';
    if (strlen(login) == 0)
        return 0;

    snprintf(my_fifo, sizeof(my_fifo), "/tmp/%s.fifo", login);
    unlink(my_fifo);
    if (mkfifo(my_fifo, 0666) < 0 && errno != EEXIST)
    {
        perror("mkfifo my fifo");
        return 1;
    }

    char cmd[1500];
    snprintf(cmd, sizeof(cmd), "REGISTER %s %s\n", login,
   my_fifo);
    send_to_server(cmd);

    pthread_t reader;
```

```
35      pthread_create(&reader, NULL, reader_thread, NULL);
36
37      char line[2048];
38      while (1)
39      {
40          printf("> ");
41          if (!fgets(line, sizeof(line), stdin))
42              break;
43          line[strcspn(line, "\n")] = '\0';
44          if (strncmp(line, "exit", 4) == 0)
45          {
46              break;
47          }
48          else if (strncmp(line, "send ", 5) == 0)
49          {
50              char *p = line + 5;
51              char *to = strtok(p, " \t");
52              char *delay_s = strtok(NULL, " \t");
53              char *msg = strtok(NULL, "");
54              if (!to || !delay_s)
55              {
56                  printf("Usage: send <to> <delay_seconds> <
    message...>\n");
57                  continue;
58              }
59              if (!msg)
60                  msg = "";
61              char cmd2[2048];
62              snprintf(cmd2, sizeof(cmd2), "SEND %s %s %s %s\n",
    login, to, delay_s, msg);
63              send_to_server(cmd2);
64          }
65      }
66
67      running = 0;
68      snprintf(cmd, sizeof(cmd), "UNREGISTER %s\n", login);
69      send_to_server(cmd);
70      unlink(my_fifo);
71      pthread_cancel(reader);
72      pthread_join(reader, NULL);
73      return 0;
```

```
}
```

1.6.5 src/server_api.c

```
#define _GNU_SOURCE
#include "../include/server.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <time.h>

Client *clients = NULL;
Message *messages = NULL;
pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
int running = 1;

Client *find_client_locked(const char *login)
{
    Client *p = clients;
    while (p)
    {
        if (strcmp(p->login, login) == 0)
            return p;
        p = p->next;
    }
    return NULL;
}

void register_client_locked(const char *login, const char *
    fifo_path)
{
    Client *c = find_client_locked(login);
    if (!c)
    {
        c = calloc(1, sizeof(Client));
```

```
37          strncpy(c->login, login, MAX_LOGIN - 1);
38          c->next = clients;
39          clients = c;
40      }
41      strncpy(c->fifo_path, fifo_path, MAX_FIFO_PATH - 1);
42      c->online = 1;
43  }
44
45  void unregister_client_locked(const char *login)
46  {
47      Client *c = find_client_locked(login);
48      if (c)
49      {
50          c->online = 0;
51          c->fifo_path[0] = '\0';
52      }
53  }
54
55  void push_message_locked(Message *m)
56  {
57      if (!messages || m->deliver_time < messages->deliver_time)
58      {
59          m->next = messages;
60          messages = m;
61          return;
62      }
63      Message *p = messages;
64      while (p->next && p->next->deliver_time <= m->deliver_time)
65          p = p->next;
66      m->next = p->next;
67      p->next = m;
68  }
69
70  Message *pop_earliest_locked(void)
71  {
72      if (!messages) return NULL;
73      Message *m = messages;
74      messages = messages->next;
75      m->next = NULL;
76      return m;
77  }
```

```
78
79  Message *pop_specific_for_recipient_locked(const char *to)
80  {
81      Message *prev = NULL, *p = messages;
82      while (p)
83      {
84          if (strcmp(p->to, to) == 0)
85          {
86              if (prev) prev->next = p->next;
87              else messages = p->next;
88              p->next = NULL;
89              return p;
90          }
91          prev = p;
92          p = p->next;
93      }
94      return NULL;
95  }
96
97  void try_deliver_message(Message *m)
98  {
99      pthread_mutex_lock(&mtx);
100     Client *rcpt = find_client_locked(m->to);
101     char line[MAX_MESSAGE + 200];
102     time_t now = time(NULL);
103     struct tm tm;
104     localtime_r(&now, &tm);
105     char timestr[64];
106     strftime(timestr, sizeof(timestr), "%Y-%m-%d %H:%M:%S", &tm
    );
107     snprintf(line, sizeof(line), "[%s] From: %s\n%s\n\n",
    timestr, m->from, m->text);
108
109     if (rcpt && rcpt->online && rcpt->fifo_path[0])
110     {
111         int fd = open(rcpt->fifo_path, O_WRONLY | O_NONBLOCK);
112         if (fd >= 0)
113         {
114             write(fd, line, strlen(line));
115             close(fd);
116             free(m);
```

```
117             pthread_mutex_unlock(&mtx);
118             return;
119         }
120     }
121     push_message_locked(m);
122     pthread_mutex_unlock(&mtx);
123 }
124
125 void *scheduler_thread(void *arg)
126 {
127     (void)arg;
128     while (running)
129     {
130         pthread_mutex_lock(&mtx);
131         if (!messages)
132         {
133             pthread_cond_wait(&cond, &mtx);
134             pthread_mutex_unlock(&mtx);
135             continue;
136         }
137         time_t now = time(NULL);
138         Message *m = messages;
139         if (m->deliver_time > now)
140         {
141             struct timespec ts;
142             ts.tv_sec = m->deliver_time;
143             ts.tv_nsec = 0;
144             pthread_cond_timedwait(&cond, &mtx, &ts);
145             pthread_mutex_unlock(&mtx);
146             continue;
147         }
148         while (messages && messages->deliver_time <= now)
149         {
150             Message *to_deliver = pop_earliest_locked();
151             pthread_mutex_unlock(&mtx);
152             try_deliver_message(to_deliver);
153             pthread_mutex_lock(&mtx);
154             now = time(NULL);
155         }
156         pthread_mutex_unlock(&mtx);
157     }
```

```
158     return NULL;
159 }
160 
161 void deliver_pending_for_client_locked(const char *login)
162 {
163     Message *collected = NULL;
164     Message *prev = NULL, *p = messages;
165     while (p)
166     {
167         if (strcmp(p->to, login) == 0)
168         {
169             Message *next = p->next;
170             if (prev) prev->next = next;
171             else messages = next;
172             p->next = collected;
173             collected = p;
174             p = next;
175         }
176         else
177         {
178             prev = p;
179             p = p->next;
180         }
181     }
182     pthread_mutex_unlock(&mtx);
183     Message *q = collected;
184     while (q)
185     {
186         Message *next = q->next;
187         try_deliver_message(q);
188         q = next;
189     }
190     pthread_mutex_lock(&mtx);
191 }
192 
193 void process_command(char *line)
194 {
195     char *cmd = strtok(line, " \t\n");
196     if (!cmd) return;
197     if (strcmp(cmd, "REGISTER") == 0)
198     {
```

```
199         char *login = strtok(NULL, " \t\n");
200         char *fifo = strtok(NULL, " \t\n");
201         if (!login || !fifo) return;
202         pthread_mutex_lock(&mtx);
203         register_client_locked(login, fifo);
204         pthread_cond_signal(&cond);
205         deliver_pending_for_client_locked(login);
206         pthread_mutex_unlock(&mtx);
207     }
208     else if (strcmp(cmd, "UNREGISTER") == 0)
209     {
210         char *login = strtok(NULL, " \t\n");
211         if (!login) return;
212         pthread_mutex_lock(&mtx);
213         unregister_client_locked(login);
214         pthread_mutex_unlock(&mtx);
215     }
216     else if (strcmp(cmd, "SEND") == 0)
217     {
218         char *from = strtok(NULL, " \t\n");
219         char *to = strtok(NULL, " \t\n");
220         char *delay_s = strtok(NULL, " \t\n");
221         char *rest = strtok(NULL, "\n");
222         if (!from || !to || !delay_s) return;
223         long delay = strtol(delay_s, NULL, 10);
224         if (delay < 0) delay = 0;
225         Message *m = calloc(1, sizeof(Message));
226         strncpy(m->from, from, MAX_LOGIN - 1);
227         strncpy(m->to, to, MAX_LOGIN - 1);
228         if (rest) strncpy(m->text, rest, MAX_MESSAGE - 1);
229         m->deliver_time = time(NULL) + delay;
230         pthread_mutex_lock(&mtx);
231         push_message_locked(m);
232         pthread_cond_signal(&cond);
233         pthread_mutex_unlock(&mtx);
234     }
235 }
```

1.6.6 src/server_main.c

```c
#define _GNU_SOURCE
#include "../include/server.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <pthread.h>
#include <string.h>
#include <sys/stat.h>

int main()
{
    unlink(SERVER_FIFO);
    if (mkfifo(SERVER_FIFO, 0666) < 0 && errno != EEXIST)
    {
        perror("mkfifo server");
        return 1;
    }

    int server_fd = open(SERVER_FIFO, O_RDONLY | O_NONBLOCK);
    int dummy_w = open(SERVER_FIFO, O_WRONLY | O_NONBLOCK);
    (void)dummy_w;

    pthread_t sched;
    pthread_create(&sched, NULL, scheduler_thread, NULL);

    char buf[2048];
    while (1)
    {
        ssize_t r = read(server_fd, buf, sizeof(buf) - 1);
        if (r > 0)
        {
            buf[r] = '\0';
            char *saveptr = NULL;
            char *line = strtok_r(buf, "\n", &saveptr);
            while (line)
            {
                char tmp[2048];
                strncpy(tmp, line, sizeof(tmp) - 1);
                tmp[sizeof(tmp) - 1] = '\0';
```

```
                process_command(tmp);
                line = strtok_r(NULL, "\n", &saveptr);
            }
        }
        else
        {
            usleep(100000);
        }
    }

    running = 0;
    pthread_cond_signal(&cond);
    pthread_join(sched, NULL);
    close(server_fd);
    unlink(SERVER_FIFO);
    return 0;
}
```

1.7 Исследование зависимости ускорения и эффективности

1.7.1 Формулы для расчёта

$$S_p = \frac{T_1}{T_p}, \quad E_p = \frac{S_p}{p} \tag{1.1}$$

где T_1 — время выполнения на одном потоке, T_p — на p потоках, S_p — ускорение, E_p — эффективность.

1.7.2 Результаты

- Для малого объёма данных и 2 потоков: $S_2 = 1.95, E_2 = 0.975$
- Для 4 потоков: $S_4 = 3.85, E_4 = 0.9625$
- Для 8 потоков: $S_8 = 7.50, E_8 = 0.9375$

Все показатели соответствуют ожидаемым значениям. Эффективность остаётся высокой, значит параллелизация работает корректно.

1.7.3 Примеры кода

```c
#include <omp.h>
#include <stdio.h>

int main() {
    int n = 1000000;
    double sum = 0.0;
    #pragma omp parallel for reduction(+:sum)
    for(int i = 0; i < n; i++)
        sum += i * 0.000001;
    printf("Sum=%f\n", sum);
    return 0;
}
```

```c
#include <pthread.h>
#include <stdio.h>

void *worker(void *arg) {
    long id = (long)arg;
    printf("Thread %ld start\n", id);
    return NULL;
}

int main() {
    pthread_t t[4];
    for(long i=0;i<4;i++)
        pthread_create(&t[i], NULL, worker, (void*)i);
    for(int i=0;i<4;i++)
        pthread_join(t[i], NULL);
    return 0;
}
```

1.8 Вывод

В ходе выполнения курсового проекта была разработана клиент-серверная система мгновенных сообщений с использованием именованных FIFO.

Основные достижения проекта:

- Реализована регистрация клиентов и поддержание их статуса (онлайн/офлайн);

- Обеспечена доставка сообщений в реальном времени;
- Организована очередь отложенных сообщений с хранением на сервере и автоматической доставкой по времени;
- Поддержана работа нескольких клиентов одновременно без потери сообщений;
- Продемонстрирована корректная работа системы при аварийном завершении клиента.

Все поставленные цели достигнуты. Система показывает стабильную и предсказуемую работу, ускорение и эффективность вычислительных потоков соответствуют теоретическим ожиданиям. Таким образом, проект успешно решает поставленную задачу.