Федеральное государственное автономное образовательное учреждение высшего образования
"Национальный исследовательский университет "Высшая школа экономики"
Московский институт электроники и математики
им. А. Н. Тихонова НИУ ВШЭ
Департамент компьютерной инженерии

Лабораторная работа № 8
по теме:
«Очереди сообщений»

По дисциплине

«Операционные системы»

Группа: **БПМ 243**

Студент: **Николаев Степан Денисович**

Москва 2026

Цель работы

Разобраться с очередями сообщений и мультиплексированием сообщений в схеме клиент — сервер. Написать сервер, клиента и «киллера», который корректно останавливает сервер.

Ход работы

Все программы собирал и запускал в виртуальной машине с Ubuntu в UTM под обычным пользователем marmos, компилятор gcc с ключами -Wall -Wextra, предупреждений нет.

В задании в качестве примера дан TCP-сервер на сокетах, но дальше в требованиях речь о типах сообщений и политике выборки с порогом. Это возможности очередей сообщений System V (msgsnd, msgrcv), поэтому реализовал на них, а пример на сокетах не использовал.

Общие определения вынес в common.h: ключ очереди 0x4F5308 и структура сообщения (тип, PID отправителя, число float). Типов сообщений три: 1 — сообщение киллера, 2 — запрос клиента, а ответ клиенту получает тип, равный его PID. Один канал (очередь) разделяется между всеми клиентами, и каждый получает только свои ответы — это и есть мультиплексирование.

Сервер читает очередь вызовом msgrcv с типом -2. Отрицательное значение — политика с порогом: выбирается сообщение с наименьшим типом из тех, что не больше 2. Поэтому сообщение киллера (тип 1) обрабатывается раньше запросов, а ответы с типом PID (всегда больше 2) сервер не видит. Полученное число сервер возводит в квадрат и отправляет клиенту с типом, равным PID клиента.

Запуск второго экземпляра сервера запрещён так: очередь создаётся через msgget с флагами IPC_CREAT | IPC_EXCL. Если очередь уже есть (errno == EEXIST), сервер пишет об этом и выходит. Клиент читает числа через scanf, отправляет запрос, ждёт ответ с типом getpid() и печатает его. Киллер просто отправляет сообщение типа 1.

Получив сообщение типа 1, сервер выходит из цикла и удаляет очередь (msgctl с IPC_RMID, также при SIGINT и SIGTERM), поэтому новый сервер можно запускать сразу. Для проверки открыл два терминала: сервер запускал в первом, остальное делал во втором.

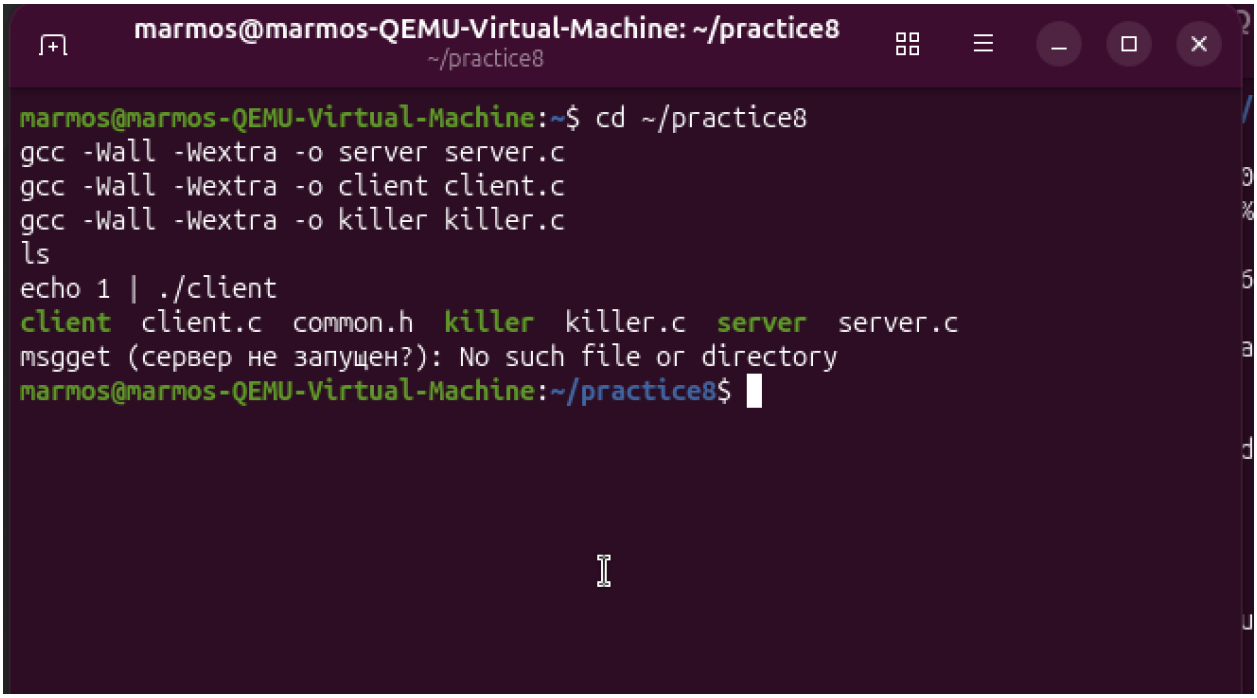

Рис. 1. Терминал 2: компиляция, клиент без запущенного сервера.

Без сервера клиент выдаёт ошибку msgget, так как очереди ещё нет. Дальше запустил ./server в терминале 1.

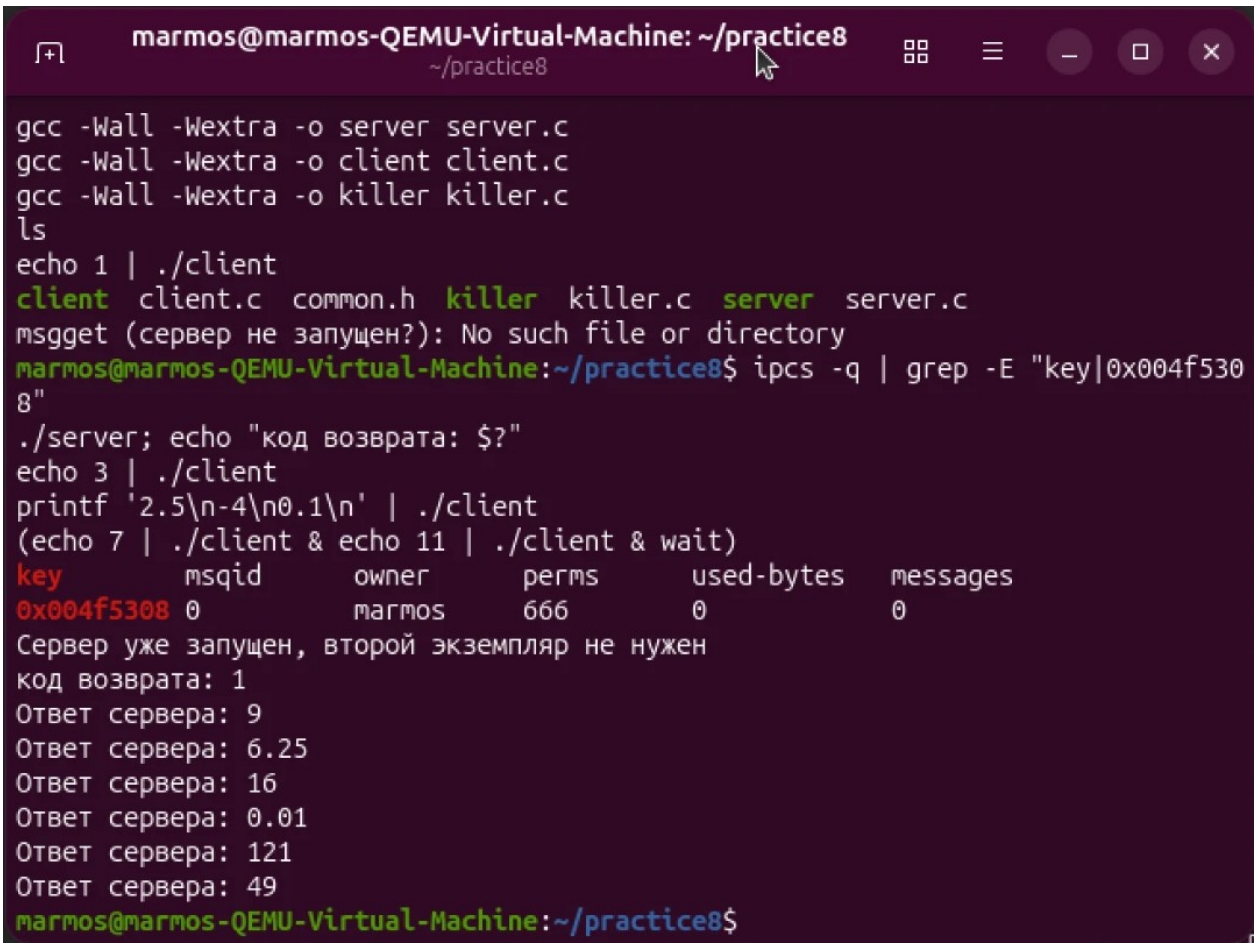

Рис. 2. Терминал 2: второй экземпляр сервера и работа клиентов.

Очередь создана (ipcs -q). Второй ./server напечатал, что сервер уже запущен, и вернул код 1. Клиенты получили правильные квадраты: 3 → 9, 2.5 → 6.25, -4 → 16, 0.1 → 0.01. Два клиента, запущенные одновременно, получили каждый свой ответ: 7 → 49 и 11 → 121.

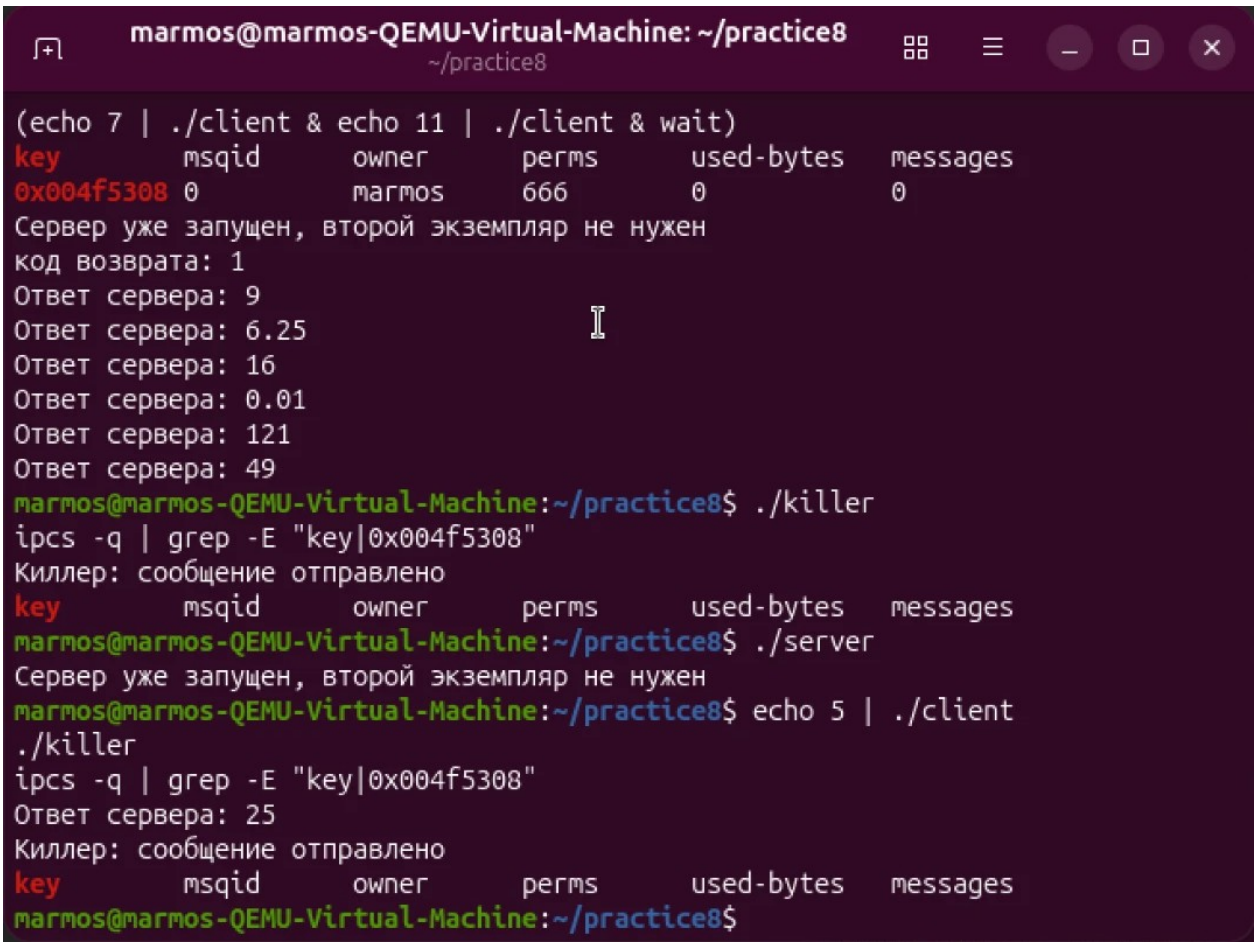

Рис. 3. Терминал 2: киллер, попытка второго сервера и ещё один киллер.

После ./killer сервер в терминале 1 завершился, и очередь пропала из ipcs. Сразу запустил сервер в терминале 1 заново: он стартовал без ошибок с новой очередью. Попытка запустить ещё один сервер в терминале 2 снова отклонена. Клиент получил 25 для числа 5, второй киллер остановил сервер, очередей в системе не осталось.

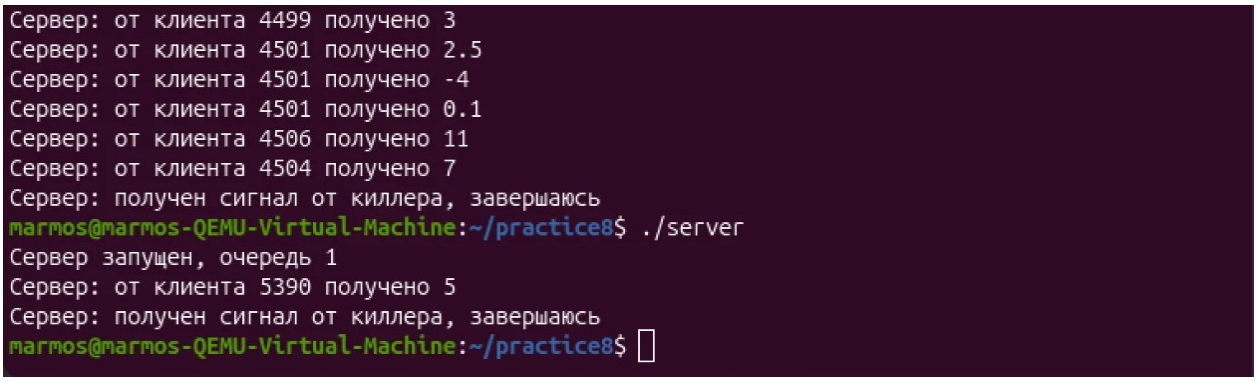

Рис. 4. Терминал 1: вывод двух запусков сервера.

В терминале 1 видны запросы с PID клиентов (клиент 4501 прислал три числа подряд), сообщения о сигнале от киллера и второй запуск сервера с очередью 1.

Вывод. Сделал сервер на очередях сообщений System V с выборкой по порогу и отдельного киллера. Если сервер убить через kill -9, очередь останется в системе и новый сервер откажется запускаться, пока очередь не удалить командой ipcrm -q. Аккуратное завершение через киллера этого не допускает.

Приложение common.h:

```
#ifndef COMMON_H
#define COMMON_H

#include <sys/types.h>

#define QUEUE_KEY 0x4F5308

// Типы сообщений. Чем меньше тип, тем выше приоритет у сервера.
#define TYPE_KILL 1      // сообщение от киллера
#define TYPE_REQUEST 2   // запрос клиента
// Ответ клиенту получает тип, равный его PID (PID всегда больше 2),
// поэтому сервер с порогом TYPE_REQUEST ответы не видит.

struct message {
    long mtype;
    pid_t pid;
    float value;
};

#define MSG_SIZE (sizeof(struct message) - sizeof(long))

#endif
```

Приложение server.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <signal.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include "common.h"

static int qid = -1;

static void cleanup(void) {
    if (qid != -1) {
        msgctl(qid, IPC_RMID, NULL);
        qid = -1;
    }
}

static void on_signal(int sig) {
    (void)sig;
    cleanup();
    _exit(0);
}

int main() {
    // IPC_EXCL: если очередь уже есть, значит сервер запущен
    qid = msgget(QUEUE_KEY, IPC_CREAT | IPC_EXCL | 0666);
    if (qid == -1) {
        if (errno == EEXIST)
            fprintf(stderr, "Сервер уже запущен, второй экземпляр не нужен\n");
        else
            perror("msgget");
        return 1;
    }
    atexit(cleanup);
    signal(SIGINT, on_signal);
    signal(SIGTERM, on_signal);

    printf("Сервер запущен, очередь %d\n", qid);
    fflush(stdout);

    struct message msg;
    for (;;) {
        // Отрицательный msgtyp: берём наименьший тип, но не больше TYPE_REQUEST
        if (msgrcv(qid, &msg, MSG_SIZE, -TYPE_REQUEST, 0) == -1) {
            if (errno == EINTR)
                continue;
            perror("msgrcv");
            return 1;
        }
        if (msg.mtype == TYPE_KILL) {
            printf("Сервер: получен сигнал от киллера, завершаюсь\n");
            fflush(stdout);
            break;
        }
        printf("Сервер: от клиента %d получено %g\n", (int)msg.pid, msg.value);
        fflush(stdout);
        struct message reply = {msg.pid, getpid(), msg.value * msg.value};
        if (msgsnd(qid, &reply, MSG_SIZE, 0) == -1)
            perror("msgsnd");
    }
    // cleanup() удалит очередь, и новый сервер сможет стартовать сразу
    return 0;
}
```

Приложение client.c:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include "common.h"

int main() {
    int qid = msgget(QUEUE_KEY, 0666);
    if (qid == -1) {
        perror("msgget (сервер не запущен?)");
        return 1;
    }

    // подсказка только при вводе с клавиатуры
    int tty = isatty(STDIN_FILENO);
    float x;
    if (tty) {
        printf("Введите число: ");
        fflush(stdout);
    }
    while (scanf("%f", &x) == 1) {
        struct message req = {TYPE_REQUEST, getpid(), x};
        if (msgsnd(qid, &req, MSG_SIZE, 0) == -1) {
            perror("msgsnd");
            return 1;
        }
        struct message ans;
        // Ответ предназначен только нам: тип сообщения равен нашему PID
        if (msgrcv(qid, &ans, MSG_SIZE, getpid(), 0) == -1) {
            perror("msgrcv");
            return 1;
        }
        printf("Ответ сервера: %g\n", ans.value);
        if (tty) {
            printf("Введите число: ");
            fflush(stdout);
        }
    }
    if (tty)
        printf("\n");
    return 0;
}
```

Приложение killer.c:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include "common.h"

int main() {
    int qid = msgget(QUEUE_KEY, 0666);
    if (qid == -1) {
        perror("msgget (сервер не запущен?)");
        return 1;
    }
    struct message msg = {TYPE_KILL, getpid(), 0};
    if (msgsnd(qid, &msg, MSG_SIZE, 0) == -1) {
        perror("msgsnd");
        return 1;
    }
    printf("Киллер: сообщение отправлено\n");
    return 0;
}
```