МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 "Компьютерные науки и прикладная математика"

Кафедра 806 "Вычислительная математика и программирование"

Курсовой проект

По курсу «Операционные системы»

Студент: Григорьев Т. А.

Группа: М8О-208Б-23

Преподаватель: Живалев Е. А.

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

**Тема:** Разработка клиент-серверной системы для передачи мгновенных сообщений.

**Цель работы:**  
Целью курсового проекта являлось приобретение практических навыков в:

* разработке клиент-серверных приложений с использованием разделяемой памяти (memory map);
* реализации обмена сообщениями в реальном времени между клиентами;
* поддержке группового общения (групповые чаты).

**Вариант:** 24. Memory Map/Групповые чаты.

**Задачи проекта**

1. Реализовать механизм регистрации клиента на сервере с использованием разделяемой памяти.
2. Обеспечить передачу сообщений между клиентами в реальном времени через memory map.
3. Предусмотреть возможность создания групповых чатов и отправки сообщений в группы.
4. Обеспечить корректную синхронизацию операций с использованием семафоров.

**Описание решения**

Программное решение разработано на языке C++ с использованием системных вызовов POSIX (shm\_open, mmap, sem\_open). Система состоит из двух модулей: **клиент** и **сервер**, взаимодействующих через разделяемую память и семафоры.

**Основные компоненты:**

1. **Регистрация клиента:**
   * Клиент подключается к серверу, передавая уникальный логин.
   * Сервер создаёт для каждого клиента уникальный сегмент разделяемой памяти и набор семафоров.
   * Сервер ведёт список зарегистрированных клиентов.

**Пример реализации регистрации клиента:**

// Подключение к сегменту логинов

int loginFd = shm\_open(LOGIN\_SHM\_NAME, O\_RDWR, 0666);

if (loginFd < 0) {

std::cerr << "Cannot open login shared memory, maybe server is not started." << std::endl;

return 1;

}

void\* loginAddr = mmap(nullptr, sizeof(LoginSegment), PROT\_READ | PROT\_WRITE, MAP\_SHARED, loginFd, 0);

LoginSegment\* loginSeg = reinterpret\_cast<LoginSegment\*>(loginAddr);

std::string nickname;

std::cout << "Enter login: ";

std::cin >> nickname;

// Формирование запроса на логин

std::ostringstream reqStream;

reqStream << "login " << getpid() << " " << nickname;

strncpy(loginSeg->buffer, reqStream.str().c\_str(), MAX\_CMD\_LEN - 1);

loginSeg->requestReady = true;

// Ожидание ответа сервера

sem\_post(loginSemServer);

sem\_wait(loginSemClient);

if (loginSeg->responseReady && loginSeg->buffer[0] == '1') {

std::cout << "Login successful!" << std::endl;

} else {

std::cerr << "Login failed. Reason: " << loginSeg->buffer << std::endl;

}

1. **Передача сообщений между клиентами:**
   * Клиенты могут отправлять сообщения, указав логин получателя.
   * Сообщения отправляются через memory map в сегмент разделяемой памяти получателя.

**Пример отправки сообщения клиентом:**

std::cout << "Enter recipient's login: ";

std::string recipient;

std::cin >> recipient;

std::cout << "Enter your message: ";

std::string message;

std::cin.ignore();

std::getline(std::cin, message);

// Формируем команду для отправки

std::ostringstream cmdStream;

cmdStream << "send " << recipient << " " << message;

strncpy(clientSegment->buffer, cmdStream.str().c\_str(), MAX\_CMD\_LEN - 1);

clientSegment->requestReady = true;

sem\_post(semServer);

// Ожидание подтверждения

sem\_wait(semClient);

if (clientSegment->responseReady) {

std::cout << "Server response: " << clientSegment->buffer << std::endl;

}

**Обработка на сервере:**

void process\_command(pid\_t pid, ClientSegment\* seg, sem\_t\* semClient, const std::string& cmdLine) {

std::istringstream iss(cmdLine);

std::string command;

iss >> command;

if (command == "send") {

std::string recipient, message;

iss >> recipient;

std::getline(iss, message);

if (!recipient.empty() && !message.empty()) {

if (nickname\_to\_pid.find(recipient) != nickname\_to\_pid.end()) {

deliver\_message\_to\_user(recipient, "Message from " + pid\_to\_nickname[pid] + ": " + message);

send\_response(seg, semClient, "Message delivered to " + recipient);

} else {

send\_response(seg, semClient, "Recipient not found.");

}

} else {

send\_response(seg, semClient, "Invalid command format.");

}

}

}

1. **Групповые чаты:**
   * Клиенты могут создавать группы, добавлять участников и отправлять сообщения в группы.
   * Сообщения рассылаются всем участникам группы.

**Пример создания группы и отправки сообщения:**

std::cout << "Enter group name to create: ";

std::string groupName;

std::cin >> groupName;

std::ostringstream groupCmd;

groupCmd << "addgroup " << groupName;

strncpy(clientSegment->buffer, groupCmd.str().c\_str(), MAX\_CMD\_LEN - 1);

clientSegment->requestReady = true;

sem\_post(semServer);

// Ожидание ответа

sem\_wait(semClient);

if (clientSegment->responseReady) {

std::cout << "Server response: " << clientSegment->buffer << std::endl;

}

std::cout << "Enter message for group: ";

std::string groupMessage;

std::cin.ignore();

std::getline(std::cin, groupMessage);

std::ostringstream groupMsgCmd;

groupMsgCmd << "gs " << groupName << " " << groupMessage;

strncpy(clientSegment->buffer, groupMsgCmd.str().c\_str(), MAX\_CMD\_LEN - 1);

clientSegment->requestReady = true;

sem\_post(semServer);

sem\_wait(semClient);

if (clientSegment->responseReady) {

std::cout << "Server response: " << clientSegment->buffer << std::endl;

}

1. **Синхронизация операций:**
   * Семафоры используются для синхронизации взаимодействия между клиентами и сервером.

**Пример использования семафоров:**

sem\_t\* semServer = sem\_open(make\_sem\_server\_name(pid).c\_str(), 0);

sem\_t\* semClient = sem\_open(make\_sem\_client\_name(pid).c\_str(), 0);

if (sem\_wait(semServer) == 0) {

if (clientSegment->requestReady) {

std::string command = clientSegment->buffer;

clientSegment->requestReady = false;

process\_command(pid, command);

sem\_post(semClient);

}

}

**Пример конфигурации и запуска**

1. **Конфигурация:** Клиенты взаимодействуют через сервер с использованием заранее определённых разделяемых объектов (shm, sem).
2. **Запуск:** Запуск сервера и клиентов в отдельных процессах.

**Пример работы программы:**

./server

Server started. Waiting for clients...

./client

Enter login: user1

Login successful!

Send blabla Hello!

Recipient not found.

Enter command: addgroup group1

Server response: group - group1, added to your group list

**Вывод**

В результате работы программы реализована клиент-серверная система с поддержкой:

* мгновенного обмена сообщениями;
* создания и управления групповыми чатами;
* синхронизации взаимодействия через семафоры.

Программа демонстрирует работу с POSIX API для управления memory map и многопоточностью.