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»

Группа: М8О-215Б-23

Студент: Закарейшвили Г. М.

Преподаватель: Миронов Е.С.

Оценка:

_____

Дата: 11.03.25

Москва, 2025

Постановка задачи

Вариант 27
Цель курсового проекта

1. Приобретение практических навыков в использовании знаний, полученных в течении курса
2. Проведение исследования в выбранной предметной области

Задание
Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом. Произвести анализ и сделать вывод на основании данных, полученных при работе программного прототипа.

Необходимо создать собственный сервер сообщений. В качестве механизма передачи сообщений возможно использовать следующие:

1. Pipes
2. Sockets
3. Files/Shared memory

При работе с сервером сообщений необходимо предусмотреть следующие механизмы (в зависимости от варианта):

- Долговременное хранение сообщений
- Транзактивность
- Система имен очередей
- Возможность настройки переадресации сообщений по фильтрам
- Приоритеты сообщений
- И другие возможные, в зависимости от варианта

Основные операции, которые должен поддерживать сервер сообщений и библиотека по работе с ним:

- CreateQueue
- DeleteQueue
- ConnectToQueue
- Push (Send)
- Pop/Top (Receive)

Клиент-серверная система для передачи мгновенных сообщений. Базовый функционал должен быть следующим:

- Клиент может присоединиться к серверу, введя логин
- Клиент может отправить сообщение другому клиенту по его логину
- Клиент в реальном времени принимает сообщения от других клиентов

27. Необходимо предусмотреть возможность хранения истории переписок (на сервере) и поиска по ним. Связь между сервером и клиентом должна быть реализована при помощи memory map

Введение

В данной курсовой работе рассматривается разработка клиент-серверной системы для обмена сообщениями. Программа позволяет пользователям обмениваться личными и групповыми сообщениями в реальном времени. Основная цель работы — изучить механизмы межпроцессного взаимодействия (IPC) и многопоточности в операционных системах на примере реализации системы чата.

Технологии, используемые при решении:

1. Memory-Mapped Files (mmap)

Цель: Организация межпроцессного взаимодействия (IPC) между клиентом и сервером.

Особенности: Файл или область памяти отображается в адресное пространство процессов.
Используется для передачи сообщений через разделяемую память (/chat_server, /chat_client_*).
Преимущества: Высокая скорость обмена данными, минимизация накладных расходов.

2. Потоки (pthread)

Цель: Асинхронная обработка входящих сообщений.

Реализация:

Клиент: Отдельный поток receive_messages для чтения уведомлений из mmap.

Сервер: Основной поток обрабатывает команды из цикла событий.

Преимущества: Неблокирующий ввод/вывод, параллельная обработка задач.

3. Синхронизация (мьютексы)

Цель: Защита разделяемых данных от состояния гонки.

Реализация:

Мьютекс pthread_mutex_t в структуре SharedData.

Используется при доступе к очереди команд.

Преимущества: Гарантия целостности данных в многопоточной среде.

5. Обработка сигналов (SIGINT)

Цель: Корректное завершение работы.

Реализация:

Перехват Ctrl+C для закрытия ресурсов (mmap, shm_unlink).

Преимущества: Предотвращение утечек памяти и "зомби-процессов".

Алгоритм работы программы:

1. Запуск сервера

1. Инициализация разделяемой памяти:
   - Создается объект разделяемой памяти /chat_server через shm_open.
   - Настраивается мьютекс с атрибутом PTHREAD_PROCESS_SHARED для синхронизации между процессами.

2. Основной цикл обработки:
   - Сервер постоянно проверяет очередь команд в разделяемой памяти.

- При обнаружении новой команды (processed == 0):
    - Блокирует мьютекс для безопасного доступа к данным.
    - Обрабатывает команду (подключение, отправка сообщения, поиск, отключение).
    - Помечает команду как обработанную (processed = 1).

2. Подключение клиента

1. Регистрация клиента:
  - Пользователь вводит логин, клиент создает свою область памяти (/chat_client_<логин>).
  - Отправляет команду CONNECT через общую память сервера, передавая имя своей mmap-области.
  - Сервер сохраняет клиента в массив clients и инициализирует его историю сообщений.

2. Запуск потока приема сообщений:
  - Клиент создает отдельный поток receive_messages, который:
    - Постоянно проверяет его mmap-область.
    - Выводит новые сообщения на экран.
    - Сбрасывает буфер после чтения.

3. Отправка сообщения

1. Клиент-отправитель:
  - Пользователь вводит команду send, получателя и текст.
  - Клиент формирует команду SEND и записывает ее в общую память сервера.

2. Обработка на сервере:
  - Сервер находит получателя в массиве clients.
  - Сохраняет сообщение в историю получателя (кольцевой буфер).
  - Дублирует сообщение в историю отправителя.
  - Записывает сообщение в mmap-область получателя для мгновенной доставки.

4. Поиск по истории

1. Запрос поиска:
  - Пользователь вводит команду search и запрос.
  - Клиент отправляет команду SEARCH через общую память.

2. Обработка на сервере:
  - Сервер находит историю запрашивающего клиента.
  - Ищет совпадения в его кольцевом буфере сообщений.
  - Возвращает результаты (макс. 10 последних совпадений) через mmap клиента.

5. Отключение клиента

1. Команда exit:
  - Клиент отправляет DISCONNECT, сервер удаляет его из массива clients.
  - Закрывает mmap-область и удаляет объект разделяемой памяти.

2. Аварийное завершение:
  - Обработчик сигнала SIGINT корректно освобождает ресурсы:
    - munmap — отключает mmap.
    - shm_unlink — удаляет объекты разделяемой памяти.

6. Завершение работы сервера

- При получении SIGINT (Ctrl+C):
    - Закрывает все mmap-области.
    - Удаляет разделяемую память /chat_server.
    - Завершает работу с выводом лога.

Заключение

В ходе выполнения лабораторной работы была разработана клиент-серверная система чата с использованием современных механизмов операционной системы. Основные достижения проекта:

1. Эффективное межпроцессное взаимодействие

Реализация на основе memory-mapped files (mmap) обеспечила высокоскоростной обмен данными между клиентами и сервером без использования сетевых сокетов, что продемонстрировало преимущества разделяемой памяти для IPC.

2. Гибкая архитектура

Применение многопоточности (pthread) позволило организовать асинхронную обработку сообщений, а мьютексы гарантировали корректную работу с разделяемыми ресурсами в конкурентной среде.

3. Контроль данных

Реализация кольцевого буфера для истории сообщений обеспечила эффективное управление памятью, сохраняя только актуальные данные. Интеграция поиска по истории добавила практическую ценность системе.

Работа подтвердила, что сочетание mmap, потоков и кольцевого буфера является эффективным решением для задач, требующих минимальных задержек и контроля памяти.