**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра Вычислительной техники**

**ОТЧЕТ**

**по лабораторной работе №2**

**по дисциплине «Операционные системы»**

**Тема: «Управление памятью»**

Студентки гр. 3312 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шахов К.С.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тимофеев А.В.

Санкт-Петербург

2025

**Цель работы.**

Использование проецируемых файлов для обмена данными между процессами.

**Задание 2.1.**

Указания к выполнению.

1. Создайте два консольных приложения с меню (каждая выполняемая функция и/или операция должна быть доступна по отдельному пункту меню), которые выполняют:

− приложение-писатель создает проецируемый файл, проецирует фрагмент файла в память, осуществляет ввод данных с клавиатуры и их запись в спроецированный файл;

− приложение-читатель открывает проецируемый файл, проецирует фрагмент файла в память, считывает содержимое из спроецированного файла и отображает на экран.

2. Запустите приложения и проверьте обмен данных между процессами, удостоверьтесь в надлежащем выполнении задания. Запротоколируйте результаты в отчет. Дайте свои комментарии в отчете относительно выполнения функций.

3. Подготовьте итоговый отчет с развернутыми выводами по заданию.

**Описание серверной части**

Серверная программа выполняет следующие операции:

1. **Проецирование файла в память** (выбор пункта меню 1):

Открывает файл sharedfile.txt с правами записи и чтения. Если файл не существует, он создается.

Устанавливает размер файла в 1024 байта с помощью функции ftruncate.

Отображает файл в память с помощью функции mmap. В случае успешного отображения файл становится доступен для чтения и записи через указатель ptr.

Если проецирование выполнено успешно, выводится сообщение о завершении операции.

1. **Запись данных в память** (выбор пункта меню 2):

Если файл был проецирован (то есть указатель ptr не равен MAP\_FAILED), в память записывается строка "Hello, shared memory!".

После записи данных сервер ожидает реакции от клиента. Для этого используется механизм select, который отслеживает состояние файла. Если клиент не прочитал данные в течение 10 секунд, сервер выводит соответствующее сообщение.

1. **Завершение работы** (выбор пункта меню 3):

Если проецирование было выполнено, оно отменяется с помощью функции munmap.

Файл закрывается и удаляется с помощью unlink.

**Описание клиентской части**

Клиентская программа выполняет следующие операции:

1. **Проецирование файла в память** (выбор пункта меню 1):

Открывает файл sharedfile.txt с правами чтения и записи.

Отображает файл в память с помощью функции mmap. Если проецирование прошло успешно, файл становится доступен для чтения и записи.

1. **Чтение данных из памяти** (выбор пункта меню 2):

Если файл был проецирован, клиент считывает данные из памяти и выводит их на экран. В этом случае выводится строка, записанная сервером, например: "Hello, shared memory!".

1. **Завершение работы** (выбор пункта меню 3):

Если проецирование было выполнено, оно отменяется с помощью munmap.

Файл закрывается и программа завершает работу.

**Использование системных вызовов**

**open**: Открытие файла с указанием прав доступа.

**ftruncate**: Установка размера файла.

**mmap**: Отображение файла в память для совместного доступа между процессами.

**munmap**: Отмена проецирования файла в память.

**close**: Закрытие файла.

**unlink**: Удаление файла.

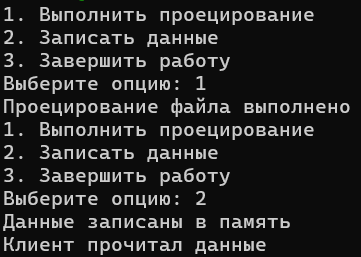
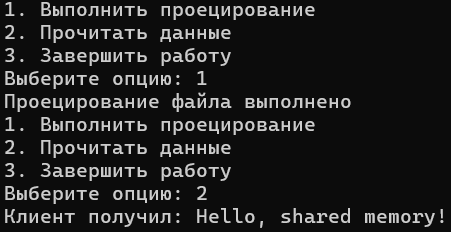
**select**: Ожидание событий на файловом дескрипторе, используемое для синхронизации между сервером и клиентом.

**Пояснение к использованию select**

В серверной части используется механизм select, который проверяет, был ли файл прочитан клиентом. Это позволяет серверу ожидать реакции клиента в течение 10 секунд. Если клиент не читает данные, сервер выводит предупреждение, что клиент не успел прочитать данные за указанное время.

**Потенциальные улучшения**

1. Обработка ошибок более детально, например, проверка статуса select и других системных вызовов.
2. Обработка ситуации, когда клиент и сервер взаимодействуют более активно, например, с использованием многозадачности или потоков для повышения производительности.
3. Улучшение синхронизации между сервером и клиентом (например, использование семафоров или других механизмов синхронизации, если необходимо избежать гонок).

Пример выполнения:  
  
server  
  
  
client

**Механизм проецируемых файлов**

Когда процессы отображают один и тот же файл с флагом MAP\_SHARED, они получают разные указатели в своём адресном пространстве, но эти указатели указывают на одну и ту же физическую память, ассоциированную с файлом. Это и позволяет им "видеть" изменения друг друга.

Когда вызывается:

*ptr = mmap(NULL, FILESIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);*

ОС:

1. Проверяет файл (по fd).
2. Убеждается, что пользователь имеет права на проецирование (PROT\_READ, PROT\_WRITE).
3. Создаёт отображение файла в виртуальное адресное пространство вызывающего процесса:

Выделяет виртуальные страницы в памяти процесса.

Эти страницы ссылаются на общие физические страницы, которые соответствуют содержимому файла.

Если другой процесс вызывает mmap() с тем же файлом и флагом MAP\_SHARED, ОС использует те же физические страницы.

У каждого процесса своё виртуальное адресное пространство (ВАП).

Но mmap() может заставить разные ВАП ссыла́ться на одни и те же физические страницы — именно это и позволяет процессам видеть общие данные.

Как это видно коде

Сервер:

1. Открывает или создаёт файл sharedfile.txt.
2. Увеличивает его до нужного размера через ftruncate.
3. Отображает его в память с флагом MAP\_SHARED.
4. Записывает строку в ptr.
5. Ожидает (условно) реакцию клиента.

Клиент:

1. Открывает тот же файл.
2. Делает mmap() — ОС видит, что файл уже отображён в физической памяти, и просто подключает те же страницы в память клиента.
3. Читает содержимое ptr — видит строку "Hello, shared memory!".

Клиент "видит" данные сервера следующим образом

На уровне ОС:

После mmap() у клиента ptr указывает на виртуальную область, отображающую те же физические страницы, что и сервер.

Изменения сервера — это просто запись в физическую память.

Клиент читает эти изменения из своей виртуальной памяти, которая физически та же.

Как разные процессы с разными ВАП работают с общей памятью:

Они отображают один и тот же файл с MAP\_SHARED.

ОС обеспечивает, чтобы их виртуальные адреса ссылались на одну и ту же физическую память.

Процессы читают/пишут в разные указатели, но под капотом это одни и те же байты.

**Заключение**

Представленная программа демонстрирует базовый механизм взаимодействия между процессами через разделяемую память в Linux. Она использует проецирование файла в память для обмена данными и реализует простое меню для взаимодействия между сервером и клиентом.

**Текст программы**

*server.cpp*

#include <iostream>

#include <fstream>

#include <sys/mman.h>

#include <sys/select.h>

#include <fcntl.h>

#include <unistd.h>

#include <cstring>

#define FILENAME "sharedfile.txt"

#define FILESIZE 1024

void server\_menu() {

    int fd;

    void\* ptr;

    while (true) {

        std::cout << "1. Выполнить проецирование\n";

        std::cout << "2. Записать данные\n";

        std::cout << "3. Завершить работу\n";

        std::cout << "Выберите опцию: ";

        int choice;

        std::cin >> choice;

        switch (choice) {

            case 1: {

                fd = open(FILENAME, O\_RDWR | O\_CREAT, S\_IRUSR | S\_IWUSR);

                if (fd == -1) {

                    perror("Ошибка при открытии файла");

                    return;

                }

                if (ftruncate(fd, FILESIZE) == -1) {

                    perror("Ошибка при изменении размера файла");

                    return;

                }

                ptr = mmap(NULL, FILESIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

                if (ptr == MAP\_FAILED) {

                    perror("Ошибка при отображении файла в память");

                    return;

                }

                std::cout << "Проецирование файла выполнено\n";

                break;

            }

            case 2: {

                if (fd == -1) {

                    std::cerr << "Проецирование не выполнено, сначала выберите пункт 1\n";

                    break;

                }

                sprintf((char\*)ptr, "Hello, shared memory!");

                std::cout << "Данные записаны в память\n";

                // Ожидание клиента

                fd\_set read\_fds;

                FD\_ZERO(&read\_fds);

                FD\_SET(fd, &read\_fds);

                struct timeval timeout = {10, 0}; // Таймаут на 10 секунд

                int result = select(fd + 1, &read\_fds, NULL, NULL, &timeout);

                if (result == -1) {

                    perror("Ошибка select");

                } else if (result == 0) {

                    std::cout << "Клиент не прочитал данные в течение 10 секунд\n";

                } else {

                    std::cout << "Клиент прочитал данные\n";

                }

                break;

            }

            case 3: {

                if (ptr != MAP\_FAILED) {

                    munmap(ptr, FILESIZE);

                    std::cout << "Проецирование отменено\n";

                }

                close(fd);

                unlink(FILENAME);

                std::cout << "Файл удален. Завершаю работу сервера.\n";

                return;

            }

            default:

                std::cout << "Неверный выбор, попробуйте снова.\n";

                break;

        }

    }

}

int main() {

    server\_menu();

    return 0;

}

*Client.cpp*

#include <iostream>

#include <sys/mman.h>

#include <fcntl.h>

#include <unistd.h>

#include <cstring>

#define FILENAME "sharedfile.txt"

#define FILESIZE 1024

void client\_menu() {

    int fd;

    void\* ptr;

    while (true) {

        std::cout << "1. Выполнить проецирование\n";

        std::cout << "2. Прочитать данные\n";

        std::cout << "3. Завершить работу\n";

        std::cout << "Выберите опцию: ";

        int choice;

        std::cin >> choice;

        switch (choice) {

            case 1: {

                fd = open(FILENAME, O\_RDWR);

                if (fd == -1) {

                    perror("Ошибка при открытии файла");

                    return;

                }

                ptr = mmap(NULL, FILESIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

                if (ptr == MAP\_FAILED) {

                    perror("Ошибка при отображении файла в память");

                    return;

                }

                std::cout << "Проецирование файла выполнено\n";

                break;

            }

            case 2: {

                if (fd == -1) {

                    std::cerr << "Проецирование не выполнено, сначала выберите пункт 1\n";

                    break;

                }

                std::cout << "Клиент получил: " << (char\*)ptr << "\n";

                break;

            }

            case 3: {

                if (ptr != MAP\_FAILED) {

                    munmap(ptr, FILESIZE);

                    std::cout << "Проецирование отменено\n";

                }

                close(fd);

                std::cout << "Завершаю работу клиента.\n";

                return;

            }

            default:

                std::cout << "Неверный выбор, попробуйте снова.\n";

                break;

        }

    }

}

int main() {

    client\_menu();

    return 0;

}