**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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Поляков А.И.

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

Санкт-Петербург

2025

**Цель работы.**

Использование проецируемых файлов для обмена данными между процессами.

**Задание 2.1.**

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.

**Анализ выполнения программы**

Программа реализует межпроцессное взаимодействие через разделяемую память с использованием механизма mmap(). Она может работать в двух режимах:

* **Сервер** создаёт файл, отображает его в память и предоставляет возможность записи данных.
* **Клиент** открывает тот же файл и считывает данные, записанные сервером.

**Основные компоненты**

1. **Файл для обмена данными**

Используется файл shared\_memory\_file размером 1024 байта (FILESIZE).

Файл создаётся сервером (open() + O\_CREAT), а клиент открывает его для чтения.

1. **Отображение файла в память (mmap)**

Сервер открывает файл с правами PROT\_READ | PROT\_WRITE, что позволяет записывать данные.

Клиент открывает файл только для чтения (PROT\_READ).

1. **Интерактивный режим работы**

Серверу предоставляется меню:

1 — отображение информации о проекции файла.

2 — ввод и запись данных в разделяемую память.

3 — завершение работы.

Клиенту предоставляется аналогичное меню:

1 — информация о проекции файла.

2 — чтение данных, записанных сервером.

3 — завершение работы.

1. **Очистка ресурсов**

После завершения работы сервер удаляет файл (unlink(FILENAME)).

Используется munmap() для освобождения отображённой памяти.

Пример работы программы:



На этом рисунке мы видим создание и проецирование файла сервером и считывание этого файла клиентом из памяти.

**Механизм проецируемых файлов**Это механизм, позволяющий отобразить (спроецировать) содержимое файла в память процесса. После отображения, процесс может работать с данными файла как с обычной областью памяти — читать, писать, и эти изменения будут отражаться в самом файле.

Как это работает на уровне ОС

1. Файл создаётся или открывается через open()

* В server() файл создаётся с правами на чтение и запись:

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

* В client() файл открывается только на чтение:

*int fd = open(FILENAME, O\_RDWR);*

Два процесса открывают один и тот же файл, что означает, что они получат дескриптор на один и тот же объект в файловой системе.

2. Файл отображается в память с помощью mmap()

Server:

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

Client:

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

Параметры:

* NULL — система сама выбирает адрес.
* FILESIZE — размер отображаемой области.
* PROT\_READ | PROT\_WRITE / PROT\_READ — права доступа.
* MAP\_SHARED — флаг, говорящий системе, что изменения в памяти должны быть синхронизированы с файлом, и другие процессы, отобразившие тот же файл, тоже увидят изменения.

Это означает: оба процесса имеют разные указатели ptr в своей памяти, но они указывают на физически одну и ту же область памяти, связанную с этим файлом.

3. Работа с общей памятью

* Server записывает данные:

*std::cin.getline((char\*)ptr, FILESIZE);*

* Client читает данные:

*std::cout << "Read data: " << (char\*)ptr << "\n";*

Они работают с одним и тем же участком памяти — хотя в разных ВАПах, но указатель ptr ведёт к одному и тому же физическому участку памяти благодаря mmap(MAP\_SHARED).

4. Завершение работы

* Процессы вызывают munmap() для снятия отображения:

*munmap(ptr, FILESIZE);*

* Server дополнительно удаляет файл:

*unlink(FILENAME);*

На уровне ОС и памяти

1. Создание файла. ОС создаёт inode и выделяет блоки на диске.
2. mmap() с MAP\_SHARED. ОС:
   * создаёт отображение файла в памяти,
   * вносит записи в таблицу страниц процесса, чтобы виртуальные адреса ptr указывали на соответствующие физические адреса.
3. Если другой процесс также вызывает mmap() на тот же файл с MAP\_SHARED, то он получает другое виртуальное отображение на те же физические страницы.
4. Любые изменения, сделанные одним процессом, становятся видимыми другим — поскольку они разделяют физическую память.

**Заключение**

Программа эффективно демонстрирует использование mmap() для межпроцессного взаимодействия. Она организована в виде двух отдельных функций (server() и client()), что делает код структурированным и легко расширяемым.

**Текст программы**

*Main.cpp*

#include <iostream>

#include <fstream>

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <cstring>

#define FILENAME "shared\_memory\_file"

#define FILESIZE 1024

void server()

{

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

    if (fd == -1)

    {

        perror("open");

        return;

    }

    ftruncate(fd, FILESIZE);

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

    if (ptr == MAP\_FAILED)

    {

        perror("mmap");

        close(fd);

        return;

    }

    bool running = true;

    while (running)

    {

        std::cout << "Server Menu:\n1. Perform \n2. Record data \n3. Complete the work \n Select: ";

        int choice;

        std::cin >> choice;

        std::cin.ignore();

        switch (choice)

        {

        case 1:

            std::cout << "The file is projected in memory.\n";

            break;

        case 2:

            std::cout << "Enter the data: ";

            std::cin.getline((char\*)ptr, FILESIZE);

            break;

        case 3:

            running = false;

            break;

        default:

            std::cout << "The wrong choice.\n";

        }

    }

    munmap(ptr, FILESIZE);

    close(fd);

    unlink(FILENAME);

}

void client()

{

    int fd = open(FILENAME, O\_RDWR);

    if (fd == -1)

    {

        perror("open");

        return;

    }

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

    if (ptr == MAP\_FAILED)

    {

        perror("mmap");

        close(fd);

        return;

    }

    bool running = true;

    while (running)

    {

        std::cout << "Client Menu:\n1. Perform \n2. Read data \n3. Complete the work \n Select: ";

        int choice;

        std::cin >> choice;

        std::cin.ignore();

        switch (choice)

        {

        case 1:

            std::cout << "The file is projected in memory.\n";

            break;

        case 2:

            std::cout << "Read data: " << (char\*)ptr << "\n";

            break;

        case 3:

            running = false;

            break;

        default:

            std::cout << "The wrong choice.\n";

        }

    }

    munmap(ptr, FILESIZE);

    close(fd);

}

int main(int argc, char\* argv[])

{

    if (argc != 2)

    {

        std::cerr << "Usage: " << argv[0] << " <server|client>\n";

        return 1;

    }

    std::string mode = argv[1];

    if (mode == "server")

        server();

    else if (mode == "client")

        client();

    else

    {

        std::cerr << "Unknown mode: " << mode << "\n";

        return 1;

    }

    return 0;

}