**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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Половникова А.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.

**Отчет по выполнению работы с памятью, проецируемой в файл**

**Описание реализации**

**Сервер:**

1. Открывает или создает файл test с правами чтения и записи.
2. Устанавливает размер файла с помощью ftruncate.
3. Отображает файл в память с правами PROT\_READ | PROT\_WRITE.
4. Обеспечивает три функциональных возможности:

Информирование пользователя о факте проецирования файла в память.

Запись данных в память (а значит, и в файл).

Завершение работы с освобождением ресурсов.

1. После завершения работы снимает отображение, закрывает и удаляет файл.

**Клиент:**

1. Открывает существующий файл test.
2. Отображает его в память с правами PROT\_READ.
3. Обеспечивает три функциональных возможности:

Информирование пользователя о факте проецирования файла в память.

Чтение данных из памяти (и, следовательно, из файла).

Завершение работы с освобождением ресурсов.

1. После завершения работы снимает отображение и закрывает файл.

**3. Ход выполнения работы**

1. Запуск серверной программы.
2. Ввод тестовых данных в серверном приложении.
3. Запуск клиентской программы.
4. Проверка возможности чтения данных, записанных сервером.
5. Завершение работы обеих программ.

**4. Результаты выполнения**

Корректное создание и отображение файла в память.

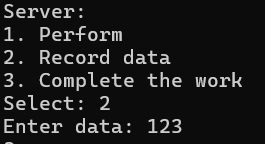
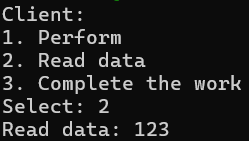
Успешная запись данных в файл через память сервером.

Успешное считывание данных клиентом.

Отсутствие утечек памяти и утечек файловых дескрипторов.

Правильное освобождение ресурсов после завершения работы.

Пример выполнения

  
Сервер  


Клиент

После завершения выполнения сервера файл удаляется

**Механизм ВАП**

Функция mmap() проецирует файл (или другой объект) в виртуальное адресное пространство процесса.

Программа использует:

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

**Механизм отображения**

**1. Процессы имеют свои ВАП**

Каждый процесс в Linux имеет собственное виртуальное адресное пространство (ВАП), то есть свою таблицу соответствий виртуальных адресов к физическим страницам памяти.

**2. mmap создаёт соответствие**

Когда вызывается mmap, ОС:

Резервирует диапазон виртуальной памяти. Находит или выделяет физические страницы, соответствующие этому участку файла. Связывает виртуальные страницы процесса с этими физическими страницами.

**3. С флагом MAP\_SHARED**

Если несколько процессов вызывают mmap с флагом MAP\_SHARED на один и тот же файл, ОС:

Сопоставляет одни и те же физические страницы с виртуальными адресами каждого процесса. Таким образом, несмотря на разные виртуальные адреса, физическая память — общая.

То есть, оба процесса работают с **одной и той же физической памятью**, но по своим виртуальным адресам.

**Как это видно коде**

**server.cpp:**

1. Открывает файл test (или создаёт).
2. Увеличивает его до 1024 байта.
3. mmap с MAP\_SHARED | PROT\_WRITE, т.е. отображает и может писать.
4. Вводит строку — она **сохраняется в отображённой памяти**, и также попадает в файл.

**client.cpp:**

1. Открывает тот же файл test.
2. mmap с MAP\_SHARED | PROT\_READ — отображает для чтения.
3. При выборе пункта 2 — читает данные, которые ввёл сервер.

**В итоге:**

* Один процесс записал строку в память, ОС отразила это изменение в физическую память.
* Второй процесс читает из той же физической памяти — и видит те же данные.

**Что делает ОС?**

1. **Создаёт PTE (page table entries)** для виртуальных адресов, указывающие на одни и те же физические страницы.
2. **Слежение за изменениями:** при MAP\_SHARED, все изменения синхронно записываются в файл, и наоборот — при чтении данных из файла можно получить актуальные данные из памяти.
3. **Кэширование страниц:** ОС может отложить запись на диск, но данные видны в памяти сразу другим процессам.

**Вывод**

В результате выполнения работы был реализован механизм межпроцессного взаимодействия с использованием проецируемой памяти. Операции записи и чтения данных выполнялись корректно, что подтверждает работоспособность предложенного метода. Управление ресурсами (файловыми дескрипторами и выделенной памятью) осуществлялось корректно, что исключает утечки. Данный метод может быть использован в различных системах для организации обмена данными между процессами.

**Текст программы**

*server.cpp*

#include <iostream>

#include <fstream>

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <cstring>

#define FILESIZE 1024

int main()

{

    const char\* fileName = "test";

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

    if (fd == -1)

    {

        perror("open");

        return -1;

    }

    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 -1;

    }

    bool running = true;

    while (running)

    {

        std::cout << "Server:\n";

        std::cout << "1. Perform \n";

        std::cout << "2. Record data \n";

        std::cout << "3. Complete the work \n";

        std::cout << "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 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);

    return 0;

}

*Client.cpp*

#include <iostream>

#include <fstream>

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <cstring>

#define FILESIZE 1024

int main()

{

    const char\* fileName = "test";

    int fd = open(fileName, O\_RDWR);

    if (fd == -1)

    {

        perror("open");

        return -1;

    }

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

    if (ptr == MAP\_FAILED)

    {

        perror("mmap");

        close(fd);

        return -1;

    }

    bool running = true;

    while (running)

    {

        std::cout << "Client:\n";

        std::cout << "1. Perform \n";

        std::cout << "2. Read data \n";

        std::cout << "3. Complete the work \n";

        std::cout << "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);

    return 0;

}