

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра Вычислительной техники**

**ОТЧЕТ**  
**по лабораторной работе № 2.2**  
**по дисциплине «Операционные системы»**  
**ТЕМА: «Управление памятью»**

Студент гр. 3311

Баймухамедов Р. Р.

Преподаватель

Тимофеев А. В.

Санкт-Петербург

2025

## Цель работы

Использование проецируемых файлов для обмена данными между процессами.

## Задание

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.

Таким образом,

Задание на проецируемые файлы. Два приложения – клиент и сервер.

Сервер создает файл на диске и проецирует его в память, далее записывает информацию и ждет, пока клиент не прочтет, затем отменяет проецирование и удаляет файл.

Клиент открывает файл, проецирует и ждет доступности чтения, затем читает и выводит результат, в завершении работы отменяет проецирование.

Сделать меню в каждой программе:

пункты меню Сервера – «выполнить проецирование», «записать данные», «завершить работу»;

пункты меню Клиента – «выполнить проецирование», «прочитать данные», «завершить работу».

## Изображение работоспособности программы

<pre>bricklng5654@bricklng5654:~/lab_02\$ ./server Server Menu: 1. Map file to memory 2. Write data 3. Exit Enter your choice:</pre>	<pre>bricklng5654@bricklng5654:~/lab_02\$ ./client File not found. Waiting for the server to create it...  Client Menu: 1. Map file to memory 2. Read data 3. Exit Enter your choice:  </pre>
--	---

Изображение 1 – Запуск сервера и клиента

<pre>Server Menu: 1. Map file to memory 2. Write data 3. Exit Enter your choice: 2 Error: You must map the file first (Option 1).  Server Menu: 1. Map file to memory 2. Write data 3. Exit Enter your choice:</pre>	<pre>Client Menu: 1. Map file to memory 2. Read data 3. Exit Enter your choice: 2 Error: You must map the file first (Option 1).  Client Menu: 1. Map file to memory 2. Read data 3. Exit Enter your choice:  </pre>
--	--

Изображение 2 – Попытка записи и чтения данных без проецирования файла в память

<pre> Server Menu: 1. Map file to memory 2. Write data 3. Exit Enter your choice: 1 File successfully mapped to memory.  Server Menu: 1. Map file to memory 2. Write data 3. Exit Enter your choice: 2 Enter data to write: 123 Data written successfully!  Server Menu: 1. Map file to memory 2. Write data 3. Exit Enter your choice: </pre>	<pre> Client Menu: 1. Map file to memory 2. Read data 3. Exit Enter your choice: 1 File successfully mapped to memory.  Client Menu: 1. Map file to memory 2. Read data 3. Exit Enter your choice: 2 Reading data... Received data: 123  Client Menu: 1. Map file to memory 2. Read data 3. Exit Enter your choice:   </pre>
--	--

*Изображение 3 – Проецирование файла в память, запись и чтение данных файла*

## Заключение

Работа с проецируемыми файлами в операционных системах позволяет эффективно управлять вводом-выводом данных, обеспечивая быстрый доступ к файлам через оперативную память. Проецирование файлов позволяет отображать содержимое файла непосредственно в память, что ускоряет операции чтения и записи. В данном задании реализованы два консольных приложения, которые демонстрируют применение этой техники.

Таким образом, использование проецируемых файлов обеспечивает высокую производительность при работе с файлами, позволяя приложениям быстро обмениваться данными через общую память. Этот подход особенно полезен в сценариях, где требуется минимизировать задержки при вводе-выводе данных.

## Как два процесса работают с общей областью памяти?

Каждый процесс в ОС работает в своём виртуальном адресном пространстве, которое изолировано от других процессов. Это означает, что физическая память, выделенная одному процессу, недоступна другому по умолчанию. Адреса, используемые процессом, соответствуют разным физическим ячейкам памяти для разных процессов.

Такая изоляция обеспечивает стабильность, поскольку ошибка в одном процессе не повлияет на другие.

Для обмена данными между процессами ОС предоставляет возможность создать общую область физической памяти, которая отображается в виртуальное адресное пространство нескольких процессов. Это реализуется через файлы, отображенные в память.

- Область памяти связывается с файлом на диске.
- Процессы проецируют этот файл в своё виртуальное пространство.
- Изменения в памяти автоматически синхронизируются с файлом

Несмотря на общий доступ, безопасность обеспечивается за счёт прав доступа при создании общей памяти и синхронизации, а также изоляцию через виртуальные адреса

## Код программы

### Код сервера

```
#include <iostream>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include <cstring>
#include <sys/stat.h>

#define FILENAME "/tmp/shared_memory_file"
#define FILESIZE 1024

using namespace std;

int main() {
    int fd = open(FILENAME, O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
    if (fd == -1) {
        cout << "Error creating file.\n";
        return 1;
    }

    if (ftruncate(fd, FILESIZE) == -1) {
        cout << "Error setting file size.\n";
        close(fd);
        return 1;
    }

    char* ptr = nullptr;
    bool mapped = false;

    int choice;
    bool running = true;

    while (running) {
        cout << "\nServer Menu:\n";
        cout << "1. Map file to memory\n";
        cout << "2. Write data\n";
        cout << "3. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        cin.ignore(); // Clear input buffer

        switch (choice) {
            case 1:
                ptr = static_cast<char*>(mmap(NULL, FILESIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0));
                if (ptr == MAP_FAILED) {
```

```

        cout << "Error mapping file to memory.\n";
    } else {
        cout << "File successfully mapped to memory.\n";
        mapped = true;
    }
    break;

case 2:
    if (!mapped) {
        cout << "Error: You must map the file first (Option 1).\n";
    } else {
        cout << "Enter data to write: ";
        cin.getline(ptr, FILESIZE);
        cout << "Data written successfully!\n";
    }
    break;

case 3:
    running = false;
    break;

default:
    cout << "Invalid choice, please try again.\n";
}
}

if (mapped) {
    munmap(ptr, FILESIZE);
}
close(fd);
unlink(FILENAME);
return 0;
}

```

## Код клиента

```

#include <iostream>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/select.h>
#include <unistd.h>
#include <cstring>
#include <sys/stat.h>

#define FILENAME "/tmp/shared_memory_file"
#define FILESIZE 1024

using namespace std;

int main() {
    int fd = open(FILENAME, O_RDWR);
    char* ptr = nullptr;
    bool mapped = false;

    if (fd == -1) {
        cout << "File not found. Waiting for the server to create it...\n";
    }

    int choice;
    bool running = true;

    while (running) {
        cout << "\nClient Menu:\n";
        cout << "1. Map file to memory\n";
        cout << "2. Read data\n";
        cout << "3. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
    }
}

```

```

switch (choice) {
    case 1:
        if (fd == -1) {
            cout << "File is not available. Waiting for the server...\n";
        } else {
            ptr = static_cast<char*>(mmap(NULL, FILESIZE, PROT_READ, MAP_SHARED, fd, 0));
            if (ptr == MAP_FAILED) {
                cout << "Error mapping file to memory.\n";
            } else {
                cout << "File successfully mapped to memory.\n";
                mapped = true;
            }
        }
        break;

    case 2:
        if (!mapped) {
            cout << "Error: You must map the file first (Option 1).\n";
        } else {
            cout << "Reading data...\n";

            struct timeval timeout = {5, 0};
            fd_set read_fds;
            FD_ZERO(&read_fds);
            FD_SET(fd, &read_fds);

            int result = select(fd + 1, &read_fds, NULL, NULL, &timeout);
            if (result > 0) {
                cout << "Received data: " << ptr << endl;
            } else {
                cout << "Error or timeout while waiting for data.\n";
            }
        }
        break;

    case 3:
        running = false;
        break;

    default:
        cout << "Invalid choice, please try again.\n";
}
}

if (mapped) {
    munmap(ptr, FILESIZE);
}
if (fd != -1) {
    close(fd);
}

return 0;
}

```