МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

## Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**Отчет по лабораторной работе № 35**

по дисциплине: “Системное программирование”

на тему:**“ИСПОЛЬЗОВАНИЕ ОБЩИХ ФАЙЛОВ, БУФЕРИЗАЦИЯ, СИНХРОНИЗАЦИЯ”**

Вариант 8

Выполнил**:** студент группы 10701222 Хасаншин А.В.

Демянов Л.А

Принял**:** ст. пр. Давыденко Н. В.

Минск 2024

# Лабораторная работа № 3.1.

**Цель: Освоить механизм взаимодействия между процессами на основе использования общей памяти.**

## Задание 1

## Создайте двух клиентов и один сервер, передайте данные с помощью маппинга файла.

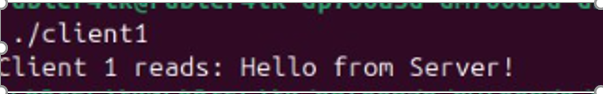
**Сервер**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <sys/mman.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <unistd.h>  
  
#define SHARED\_FILE "/tmp/shared\_file"  
#define FILE\_SIZE 4096  
  
int main() {  
// Создаем или открываем файл  
int fd = open(SHARED\_FILE, O\_RDWR | O\_CREAT, 0666);  
if (fd == -1) {  
perror("open");  
exit(EXIT\_FAILURE);  
}  
  
// Устанавливаем размер файла  
if (ftruncate(fd, FILE\_SIZE) == -1) {  
perror("ftruncate");  
close(fd);  
exit(EXIT\_FAILURE);  
}  
  
// Маппинг файла в память  
char \*mapped\_mem = mmap(NULL, FILE\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);  
if (mapped\_mem == MAP\_FAILED) {  
perror("mmap");  
close(fd);  
exit(EXIT\_FAILURE);  
}  
  
// Запись данных в файл  
const char \*data = "Hello from Server!";  
strncpy(mapped\_mem, data, FILE\_SIZE);  
  
// Ожидаем, чтобы не завершаться сразу  
printf("Server is running, press Enter to exit...\n");  
getchar();  
  
// Размаппинг и закрытие файла  
if (munmap(mapped\_mem, FILE\_SIZE) == -1) {  
perror("munmap");  
}  
close(fd);

**Клиент 1**  
return 0;: #include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <sys/mman.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <unistd.h>  
  
#define SHARED\_FILE "/tmp/shared\_file"  
#define FILE\_SIZE 4096  
  
int main() {  
// Открываем файл  
int fd = open(SHARED\_FILE, O\_RDWR);  
if (fd == -1) {  
perror("open");  
exit(EXIT\_FAILURE);  
}  
  
// Маппинг файла в память  
char \*mapped\_mem = mmap(NULL, FILE\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);  
if (mapped\_mem == MAP\_FAILED) {  
perror("mmap");  
close(fd);  
exit(EXIT\_FAILURE);  
}  
  
// Чтение данных из файла  
printf("Client 1 reads: %s\n", mapped\_mem);  
  
// Размаппинг и закрытие файла  
if (munmap(mapped\_mem, FILE\_SIZE) == -1) {  
perror("munmap");  
}  
close(fd);  
return 0;  
**Клиент 2**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <sys/mman.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <unistd.h>  
  
#define SHARED\_FILE "/tmp/shared\_file"  
#define FILE\_SIZE 4096  
  
int main() {  
// Открываем файл  
int fd = open(SHARED\_FILE, O\_RDWR);  
if (fd == -1) {  
perror("open");  
exit(EXIT\_FAILURE);  
}  
  
// Маппинг файла в память  
char \*mapped\_mem = mmap(NULL, FILE\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);  
if (mapped\_mem == MAP\_FAILED) {  
perror("mmap");  
close(fd);  
exit(EXIT\_FAILURE);  
}  
  
// Чтение данных из файла  
printf("Client 2 reads: %s\n", mapped\_mem);  
  
// Размаппинг и закрытие файла  
if (munmap(mapped\_mem, FILE\_SIZE) == -1) {  
perror("munmap");  
}  
close(fd);  
return 0;  
} > Артем Artemka: gcc server.c -o server  
gcc client1.c -o client1  
gcc client2.c -o client2

## Скриншоты результатов







## Задание 2.

Создайте два сервера и одного клиента и передайте данные через маппинг файла. Клиент должен определить от какого сервера приходит сообщение.

**Сервер 1**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <sys/mman.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <unistd.h>  
  
#define SHARED\_FILE "/tmp/shared\_file1"  
#define FILE\_SIZE 4096  
  
int main() {  
// Создаем или открываем файл  
int fd = open(SHARED\_FILE, O\_RDWR | O\_CREAT, 0666);  
if (fd == -1) {  
perror("open");  
exit(EXIT\_FAILURE);  
}  
  
// Устанавливаем размер файла  
if (ftruncate(fd, FILE\_SIZE) == -1) {  
perror("ftruncate");  
close(fd);  
exit(EXIT\_FAILURE);  
}  
  
// Маппинг файла в память  
char \*mapped\_mem = mmap(NULL, FILE\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);  
if (mapped\_mem == MAP\_FAILED) {  
perror("mmap");  
close(fd);  
exit(EXIT\_FAILURE);  
}  
  
// Запись данных в файл  
const char \*data = "Hello from Server 1!";  
strncpy(mapped\_mem, data, FILE\_SIZE);  
  
// Ожидаем, чтобы не завершаться сразу  
printf("Server 1 is running, press Enter to exit...\n");  
getchar();  
  
// Размаппинг и закрытие файла  
if (munmap(mapped\_mem, FILE\_SIZE) == -1) {  
perror("munmap");  
}  
close(fd);  
return 0

**Сервер 2**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <sys/mman.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <unistd.h>  
  
#define SHARED\_FILE "/tmp/shared\_file2"  
#define FILE\_SIZE 4096  
  
int main() {  
// Создаем или открываем файл  
int fd = open(SHARED\_FILE, O\_RDWR | O\_CREAT, 0666);  
if (fd == -1) {  
perror("open");  
exit(EXIT\_FAILURE);  
}  
  
// Устанавливаем размер файла  
if (ftruncate(fd, FILE\_SIZE) == -1) {  
perror("ftruncate");  
close(fd);  
exit(EXIT\_FAILURE);  
}  
  
// Маппинг файла в память  
char \*mapped\_mem = mmap(NULL, FILE\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);  
if (mapped\_mem == MAP\_FAILED) {  
perror("mmap");  
close(fd);  
exit(EXIT\_FAILURE);  
}  
  
// Запись данных в файл  
const char \*data = "Hello from Server 2!";  
strncpy(mapped\_mem, data, FILE\_SIZE);  
  
// Ожидаем, чтобы не завершаться сразу  
printf("Server 2 is running, press Enter to exit...\n");  
getchar();  
  
// Размаппинг и закрытие файла  
if (munmap(mapped\_mem, FILE\_SIZE) == -1) {  
perror("munmap");  
}  
close(fd);  
return 0;

**Клиент**  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <sys/mman.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <unistd.h>  
  
#define FILE1 "/tmp/shared\_file1"  
#define FILE2 "/tmp/shared\_file2"  
#define FILE\_SIZE 4096  
  
void read\_and\_identify(const char\* file\_path, const char\* server\_name) {  
// Открываем файл  
int fd = open(file\_path, O\_RDWR);  
if (fd == -1) {  
perror("open");  
return;  
}  
  
// Маппинг файла в память  
char \*mapped\_mem = mmap(NULL, FILE\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);  
if (mapped\_mem == MAP\_FAILED) {  
perror("mmap");  
close(fd);  
return;  
}  
  
// Чтение данных из файла  
printf("Client reads from %s: %s\n", server\_name, mapped\_mem);  
  
// Размаппинг и закрытие файла  
if (munmap(mapped\_mem, FILE\_SIZE) == -1) {  
perror("munmap");  
}  
close(fd);  
}  
  
int main() {  
read\_and\_identify(FILE1, "Server 1");  
read\_and\_identify(FILE2, "Server 2");  
return 0;  
}

Скриншоты результата







## Задание 3.

**Сервер 1**

: #include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <sys/mman.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <unistd.h>  
#include <semaphore.h>  
  
#define SHARED\_FILE "/tmp/shared\_file1"  
#define SEM\_NAME "/sem\_server1"  
#define FILE\_SIZE 4096  
  
int main() {  
// Создаем или открываем файл  
int fd = open(SHARED\_FILE, O\_RDWR | O\_CREAT, 0666);  
if (fd == -1) {  
perror("open");  
exit(EXIT\_FAILURE);  
}  
  
// Устанавливаем размер файла  
if (ftruncate(fd, FILE\_SIZE) == -1) {  
perror("ftruncate");  
close(fd);  
exit(EXIT\_FAILURE);  
}  
  
// Маппинг файла в память  
char \*mapped\_mem = mmap(NULL, FILE\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);  
if (mapped\_mem == MAP\_FAILED) {  
perror("mmap");  
close(fd);  
exit(EXIT\_FAILURE);  
}  
  
// Создаем или открываем семафор  
sem\_t \*sem = sem\_open(SEM\_NAME, O\_CREAT, 0644, 1);  
if (sem == SEM\_FAILED) {  
perror("sem\_open");  
close(fd);  
exit(EXIT\_FAILURE);  
}  
  
// Запись данных в файл с использованием семафора  
sem\_wait(sem); // Ждем доступ к ресурсу  
const char \*data = "Hello from Server 1!";  
strncpy(mapped\_mem, data, FILE\_SIZE);  
sem\_post(sem); // Освобождаем ресурс  
  
// Ожидаем, чтобы не завершаться сразу  
printf("Server 1 is running, press Enter to exit...\n");  
getchar();  
  
// Размаппинг и закрытие файла, уничтожение семафора  
if (munmap(mapped\_mem, FILE\_SIZE) == -1) {  
perror("munmap");  
}  
close(fd);  
sem\_close(sem);  
sem\_unlink(SEM\_NAME);  
return 0;

**Сервер 2**

#include <stdlib.h>  
#include <string.h>  
#include <sys/mman.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <unistd.h>  
#include <semaphore.h>  
  
#define SHARED\_FILE "/tmp/shared\_file2"  
#define SEM\_NAME "/sem\_server2"  
#define FILE\_SIZE 4096  
  
int main() {  
// Создаем или открываем файл  
int fd = open(SHARED\_FILE, O\_RDWR | O\_CREAT, 0666);  
if (fd == -1) {  
perror("open");  
exit(EXIT\_FAILURE);  
}  
  
// Устанавливаем размер файла  
if (ftruncate(fd, FILE\_SIZE) == -1) {  
perror("ftruncate");  
close(fd);  
exit(EXIT\_FAILURE);  
}  
  
// Маппинг файла в память  
char \*mapped\_mem = mmap(NULL, FILE\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);  
if (mapped\_mem == MAP\_FAILED) {  
perror("mmap");  
close(fd);  
exit(EXIT\_FAILURE);  
}  
  
// Создаем или открываем семафор  
sem\_t \*sem = sem\_open(SEM\_NAME, O\_CREAT, 0644, 1);  
if (sem == SEM\_FAILED) {  
perror("sem\_open");  
close(fd);  
exit(EXIT\_FAILURE);  
}  
  
// Запись данных в файл с использованием семафора  
sem\_wait(sem); // Ждем доступ к ресурсу  
const char \*data = "Hello from Server 2!";  
strncpy(mapped\_mem, data, FILE\_SIZE);  
sem\_post(sem); // Освобождаем ресурс  
  
// Ожидаем, чтобы не завершаться сразу  
printf("Server 2 is running, press Enter to exit...\n");  
getchar();  
  
// Размаппинг и закрытие файла, уничтожение семафора  
if (munmap(mapped\_mem, FILE\_SIZE) == -1) {  
perror("munmap");  
}  
close(fd);  
sem\_close(sem);  
sem\_unlink(SEM\_NAME);  
return 0;

**Клиент**  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <sys/mman.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <unistd.h>  
#include <semaphore.h>  
  
#define FILE1 "/tmp/shared\_file1"  
#define FILE2 "/tmp/shared\_file2"  
#define SEM\_NAME1 "/sem\_server1"  
#define SEM\_NAME2 "/sem\_server2"  
#define FILE\_SIZE 4096  
  
void read\_and\_identify(const char\* file\_path, const char\* sem\_name, const char\* server\_name) {  
// Открываем файл  
int fd = open(file\_path, O\_RDWR);  
if (fd == -1) {  
perror("open");  
return;  
}  
  
// Маппинг файла в память  
char \*mapped\_mem = mmap(NULL, FILE\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);  
if (mapped\_mem == MAP\_FAILED) {  
perror("mmap");  
close(fd);  
return;  
}  
  
// Открываем семафор  
sem\_t \*sem = sem\_open(sem\_name, 0);  
if (sem == SEM\_FAILED) {  
perror("sem\_open");  
munmap(mapped\_mem, FILE\_SIZE);  
close(fd);  
return;  
}  
  
// Чтение данных из файла с использованием семафора  
sem\_wait(sem); // Ждем доступ к ресурсу  
printf("Client reads from %s: %s\n", server\_name, mapped\_mem);  
sem\_post(sem); // Освобождаем ресурс  
  
// Размаппинг и закрытие файла  
if (munmap(mapped\_mem, FILE\_SIZE) == -1) {  
perror("munmap");  
}  
close(fd);  
sem\_close(sem);  
}  
  
int main() {  
read\_and\_identify(FILE1, SEM\_NAME1, "Server 1");  
read\_and\_identify(FILE2, SEM\_NAME2, "Server 2");  
return 0;  
}

**Скриншоты результатов:**







**Контрольные вопросы**

**1: Разделяемая память**

Разделяемая память (shared memory) — это механизм межпроцессного взаимодействия (IPC), позволяющий нескольким процессам совместно использовать одну и ту же область памяти. Основные преимущества разделяемой памяти:

Быстрота: Чтение и запись данных происходит быстрее, чем использование традиционных методов IPC, таких как сокеты или каналы.

Простота: Процессы могут обмениваться данными, как если бы они работали с обычными массивами в памяти

**2 : Синхронизация процессов**

Синхронизация процессов — это метод управления доступом нескольких процессов к общим ресурсам или данным, чтобы предотвратить гонки данных и гарантировать корректность выполнения программы. Основные механизмы синхронизации включают:

Мьютексы: Позволяют только одному процессу или потоку одновременно иметь доступ к ресурсу.

Семафоры: Используются для управления доступом к ресурсу с помощью счетчика, который показывает, сколько процессов могут использовать ресурс одновременно.

Блокировки (lock): Обеспечивают эксклюзивный доступ к ресурсам.