МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

## Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**Отчёт по лабораторной работе № 35**

по дисциплине: ”Системное программирование”

на тему: «***»***

Вариант -

Выполнил**:** студент группы 10701222 Зухта К. М.

Медведский Е. В

Хильчук Н. Батькович

Принял**:** пр. Давыденко Н.В.

# Лабораторная работа № 35

### Задание 1

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

Код  
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#define SHM\_NAME "/my\_shm"

#define SIZE 256

int main() {

int fd;

char \*mapped;

// Создание файла

fd = shm\_open(SHM\_NAME, O\_CREAT | O\_RDWR, 0666);

ftruncate(fd, SIZE);

// Маппинг файла в память

mapped = mmap(0, SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (mapped == MAP\_FAILED) {

perror("mmap");

exit(1);

}

// Ожидание ввода от клиентов

while (1) {

printf("Server: Waiting for input...\n");

sleep(1); // Задержка для предотвращения перегрузки CPU

if (strlen(mapped) > 0) {

printf("Server received: %s\n", mapped);

memset(mapped, 0, SIZE); // Очистка для следующего ввода

}

}

munmap(mapped, SIZE);

close(fd);

shm\_unlink(SHM\_NAME);

return 0;

}  
  
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <unistd.h>

#define SHM\_NAME "/my\_shm"

#define SIZE 256

int main() {

int fd;

char \*mapped;

// Открытие существующего файла

fd = shm\_open(SHM\_NAME, O\_RDWR, 0666);

mapped = mmap(0, SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (mapped == MAP\_FAILED) {

perror("mmap");

exit(1);

}

// Отправка данных в сервер

while (1) {

char input[SIZE];

printf("Client: Enter a message: ");

fgets(input, SIZE, stdin);

input[strcspn(input, "\n")] = 0; // Удаление символа новой строки

strncpy(mapped, input, SIZE);

}

munmap(mapped, SIZE);

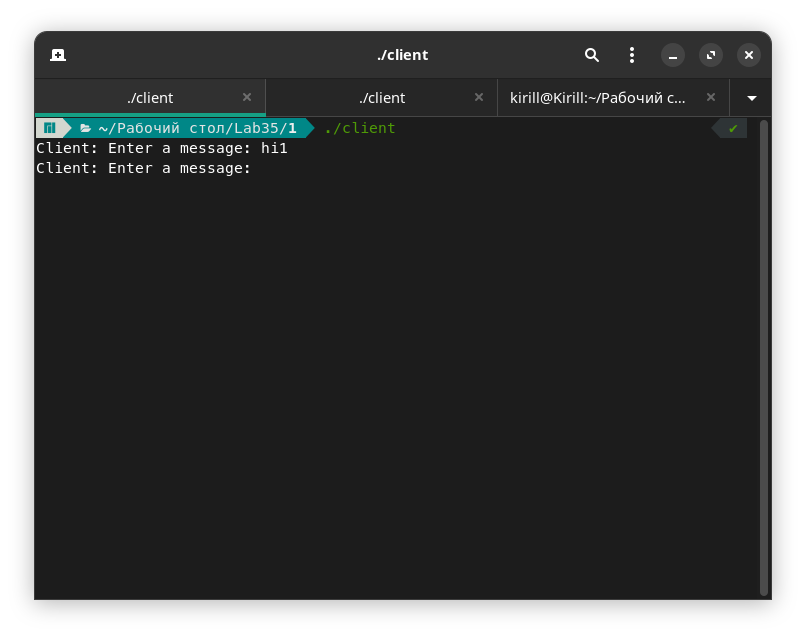
close(fd);

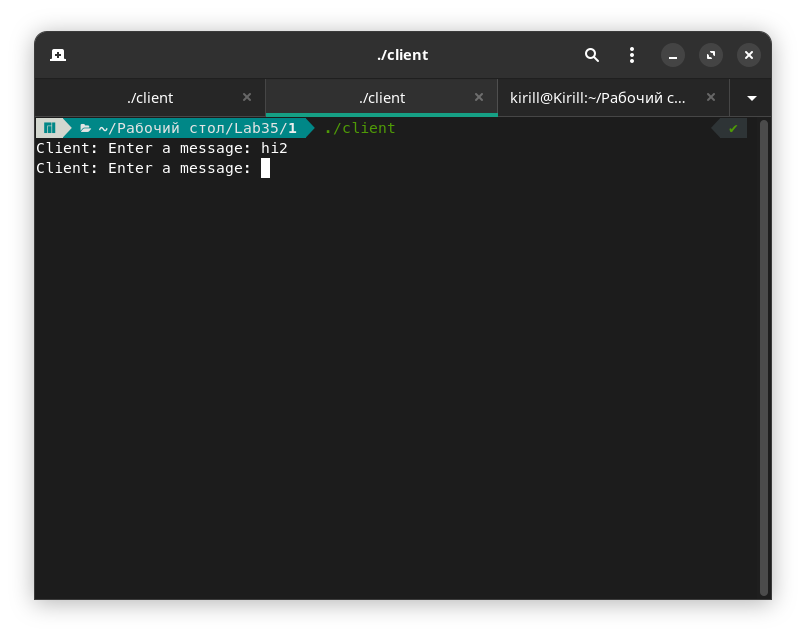
return 0;

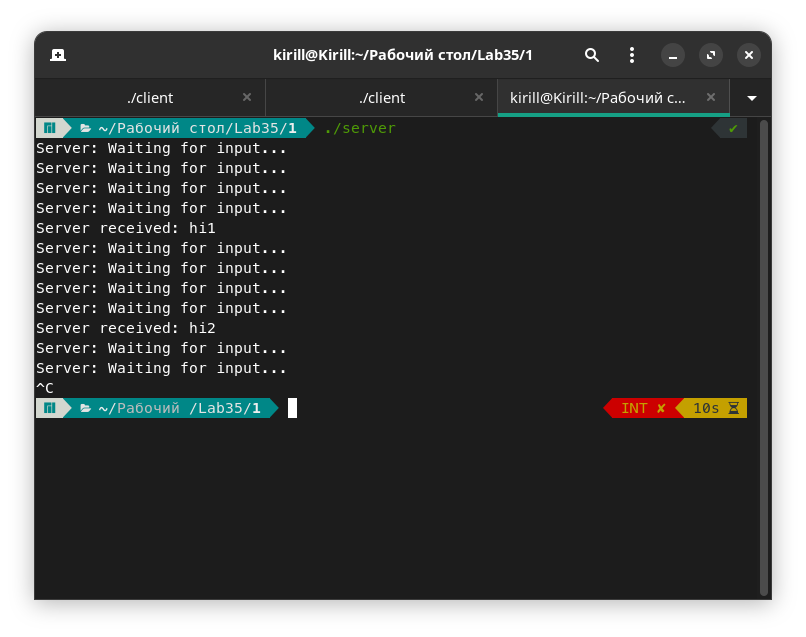
}  
  
  
  
Описание работы

* **Сервер** создает область разделяемой памяти с помощью shm\_open, а затем маппирует ее в адресное пространство процесса. Он ожидает ввода данных от клиентов.
* **Клиенты** открывают ту же область памяти и отправляют сообщения, которые сервер считывает и отображает.

Скриншоты







### Задание 2

Создайте два сервера и одного клиента и передайте данные через маппинг файла. Клиент должен определить от какого сервера приходит сообщение.   
  
Код  
  
 #include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <time.h>

#define SHM\_NAME "/my\_shm"

#define SIZE 256

int main() {

int fd;

char \*mapped;

// Создание файла

fd = shm\_open(SHM\_NAME, O\_CREAT | O\_RDWR, 0666);

ftruncate(fd, SIZE);

// Маппинг файла в память

mapped = mmap(0, SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (mapped == MAP\_FAILED) {

perror("mmap");

exit(1);

}

// Отправка данных клиенту

while (1) {

sleep(2); // Задержка перед отправкой сообщения

snprintf(mapped, SIZE, "Message from Server 1: %ld", time(NULL));

printf("Server 1 sent message.\n");

}

munmap(mapped, SIZE);

close(fd);

shm\_unlink(SHM\_NAME);

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>

#include <time.h>

#define SHM\_NAME "/my\_shm"

#define SIZE 256

int main() {

int fd;

char \*mapped;

// Создание файла

fd = shm\_open(SHM\_NAME, O\_CREAT | O\_RDWR, 0666);

ftruncate(fd, SIZE);

// Маппинг файла в память

mapped = mmap(0, SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (mapped == MAP\_FAILED) {

perror("mmap");

exit(1);

}

// Отправка данных клиенту

while (1) {

sleep(3); // Задержка перед отправкой сообщения

snprintf(mapped, SIZE, "Message from Server 2: %ld", time(NULL));

printf("Server 2 sent message.\n");

}

munmap(mapped, SIZE);

close(fd);

shm\_unlink(SHM\_NAME);

return 0;

}  
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <unistd.h>

#define SHM\_NAME "/my\_shm"

#define SIZE 256

int main() {

int fd;

char \*mapped;

// Открытие существующего файла

fd = shm\_open(SHM\_NAME, O\_RDWR, 0666);

mapped = mmap(0, SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (mapped == MAP\_FAILED) {

perror("mmap");

exit(1);

}

// Получение данных от серверов

while (1) {

sleep(1); // Задержка для получения сообщений

if (strlen(mapped) > 0) {

printf("Client received: %s\n", mapped);

memset(mapped, 0, SIZE); // Очистка для следующего сообщения

}

}

munmap(mapped, SIZE);

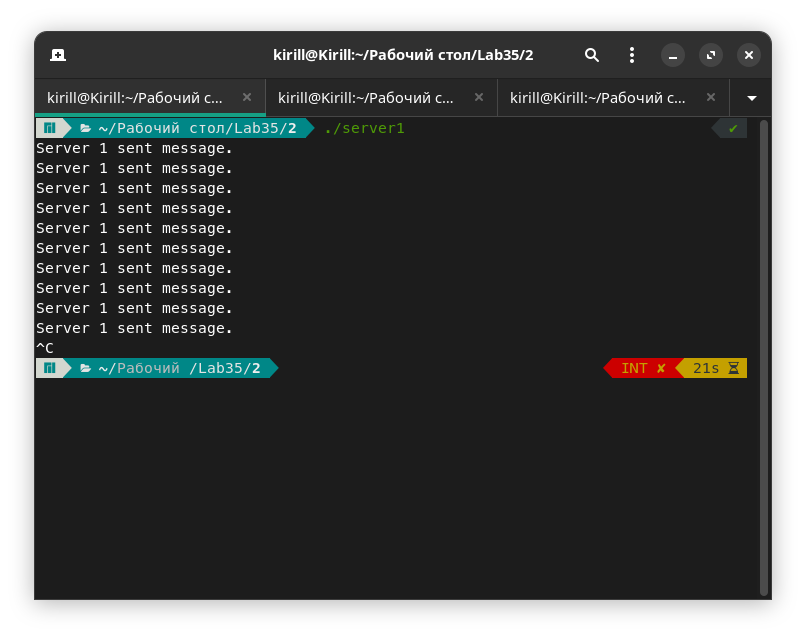
close(fd);

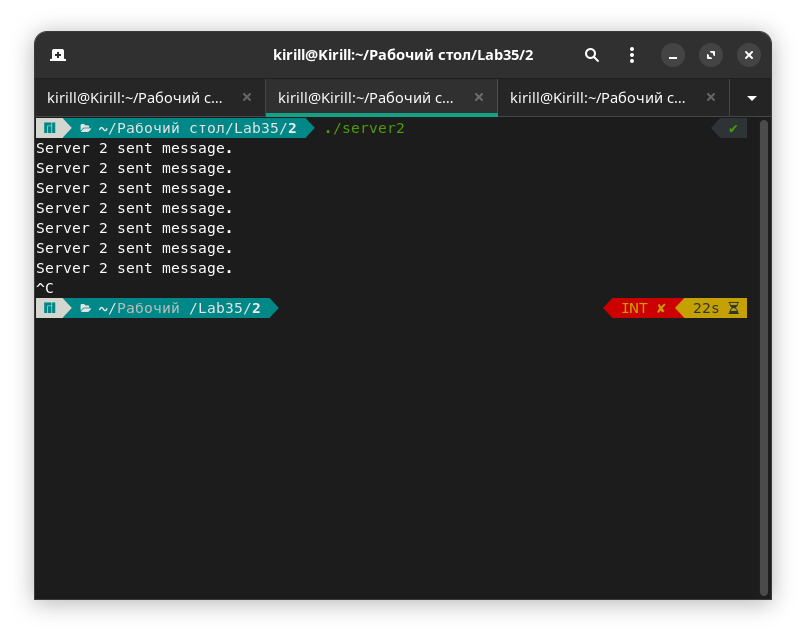
return 0;

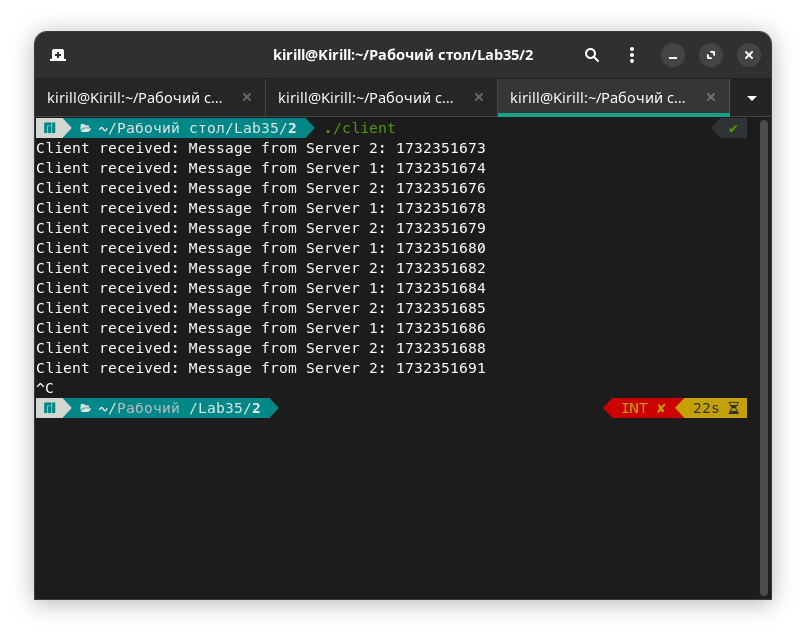
}  
Описание работы

* **Сервер 1** и **Сервер 2** создают область разделяемой памяти и отправляют сообщения с задержкой. Сервер 1 отправляет сообщения каждые 2 секунды, а Сервер 2 — каждые 3 секунды.
* **Клиент** открывает область памяти и постоянно проверяет ее на наличие новых сообщений. Он выводит полученные сообщения на экран.

Скриншоты







### Задание 3

**Измените программы из заданий №1 и №2 с использованием синхронизации общих файлов. Ключи к семафорам должны задаваться системой или пользователем.   
  
  
Код  
  
#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <sys/mman.h>**

**#include <sys/stat.h>**

**#include <fcntl.h>**

**#include <unistd.h>**

**#include <time.h>**

**#include <semaphore.h>**

**#define SHM\_NAME "/my\_shm"**

**#define SEM\_NAME "/my\_semaphore"**

**#define SIZE 256**

**int main() {**

**int fd;**

**char \*mapped;**

**sem\_t \*sem;**

**// Создание файла**

**fd = shm\_open(SHM\_NAME, O\_CREAT | O\_RDWR, 0666);**

**ftruncate(fd, SIZE);**

**// Маппинг файла в память**

**mapped = mmap(0, SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);**

**if (mapped == MAP\_FAILED) {**

**perror("mmap");**

**exit(1);**

**}**

**// Создание семафора**

**sem = sem\_open(SEM\_NAME, O\_CREAT, 0666, 1);**

**// Отправка данных клиенту**

**while (1) {**

**sleep(2); // Задержка перед отправкой сообщения**

**sem\_wait(sem); // Захват семафора**

**snprintf(mapped, SIZE, "Message from Server 1: %ld", time(NULL));**

**printf("Server 1 sent message.\n");**

**sem\_post(sem); // Освобождение семафора**

**}**

**munmap(mapped, SIZE);**

**close(fd);**

**shm\_unlink(SHM\_NAME);**

**sem\_close(sem);**

**sem\_unlink(SEM\_NAME);**

**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>**

**#include <time.h>**

**#include <semaphore.h>**

**#define SHM\_NAME "/my\_shm"**

**#define SEM\_NAME "/my\_semaphore"**

**#define SIZE 256**

**int main() {**

**int fd;**

**char \*mapped;**

**sem\_t \*sem;**

**// Создание файла**

**fd = shm\_open(SHM\_NAME, O\_CREAT | O\_RDWR, 0666);**

**ftruncate(fd, SIZE);**

**// Маппинг файла в память**

**mapped = mmap(0, SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);**

**if (mapped == MAP\_FAILED) {**

**perror("mmap");**

**exit(1);**

**}**

**// Открытие семафора**

**sem = sem\_open(SEM\_NAME, 0);**

**// Отправка данных клиенту**

**while (1) {**

**sleep(3); // Задержка перед отправкой сообщения**

**sem\_wait(sem); // Захват семафора**

**snprintf(mapped, SIZE, "Message from Server 2: %ld", time(NULL));**

**printf("Server 2 sent message.\n");**

**sem\_post(sem); // Освобождение семафора**

**}**

**munmap(mapped, SIZE);**

**close(fd);**

**sem\_close(sem);**

**return 0;**

**}  
  
  
#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <sys/mman.h>**

**#include <fcntl.h>**

**#include <unistd.h>**

**#include <semaphore.h>**

**#define SHM\_NAME "/my\_shm"**

**#define SEM\_NAME "/my\_semaphore"**

**#define SIZE 256**

**int main() {**

**int fd;**

**char \*mapped;**

**sem\_t \*sem;**

**// Открытие существующего файла**

**fd = shm\_open(SHM\_NAME, O\_RDWR, 0666);**

**mapped = mmap(0, SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);**

**if (mapped == MAP\_FAILED) {**

**perror("mmap");**

**exit(1);**

**}**

**// Открытие семафора**

**sem = sem\_open(SEM\_NAME, 0);**

**// Получение данных от серверов**

**while (1) {**

**sleep(1); // Задержка для получения сообщений**

**sem\_wait(sem); // Захват семафора**

**if (strlen(mapped) > 0) {**

**printf("Client received: %s\n", mapped);**

**memset(mapped, 0, SIZE); // Очистка для следующего сообщения**

**}**

**sem\_post(sem); // Освобождение семафора**

**}**

**munmap(mapped, SIZE);**

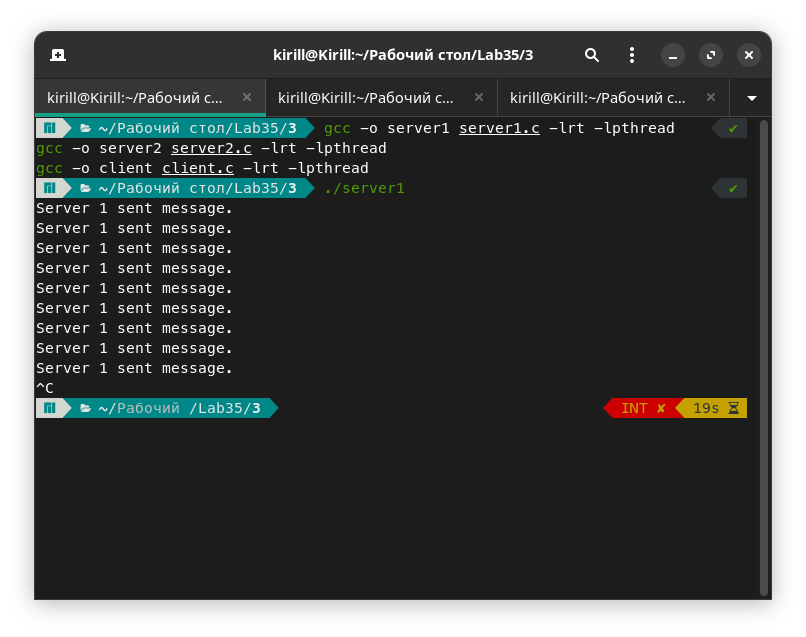
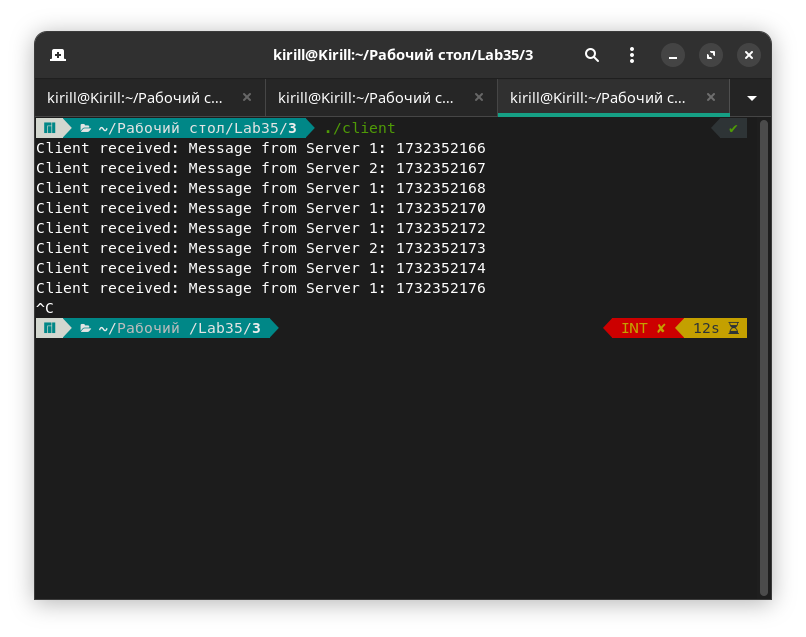
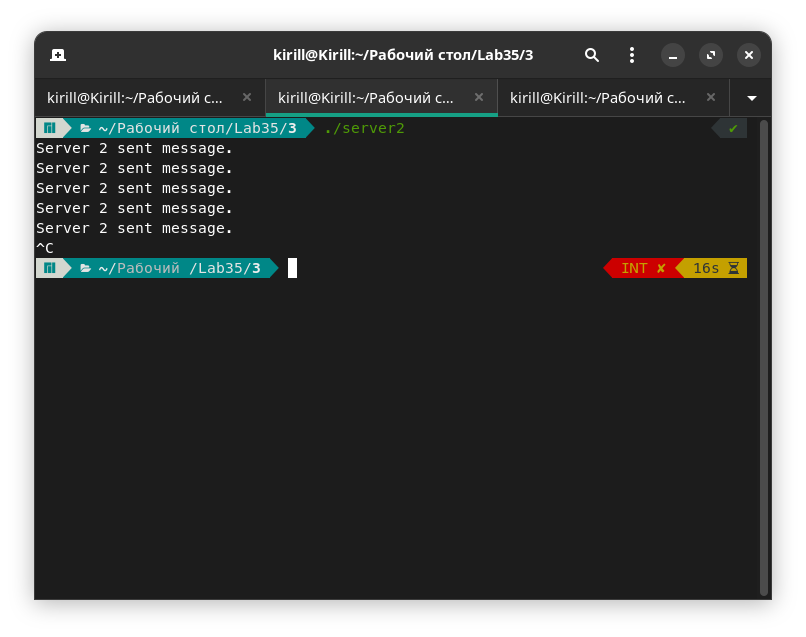
**close(fd);**

**sem\_close(sem);**

**return 0;**

**}  
Описание работы**

* **Сервера** создают область разделяемой памяти и отправляют сообщения с использованием семафоров для синхронизации доступа к этой памяти. Сервер 1 отправляет сообщения каждые 2 секунды, а Сервер 2 — каждые 3 секунды.
* **Клиент** открывает область памяти и постоянно проверяет ее на наличие новых сообщений, также используя семафоры для синхронизации.
* Семафоры гарантируют, что только один процесс в данный момент времени может записывать или читать данные из разделяемой памяти, что предотвращает гонки условий.

**Скригшоты  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
** **Вопросики  
  
Вот краткое резюме о синхронизации и семафорах в C# и Java:**

### C#

1. **Синхронизация**:
   * Использует lock и Monitor для предотвращения одновременного доступа к ресурсам.

**Java**

1. **Синхронизация**:
   * Использует synchronized для методов или блоков кода.

**Семафоры:**

* Класс Semaphore из java.util.concurrent управляет доступом к ресурсам.

**Общие аспекты**

* **Проблемы гонки**: Синхронизация предотвращает одновременные изменения общих ресурсов.
* **Deadlock**: Неправильное использование может привести к взаимной блокировке.
* **Performance**: Синхронизация может снизить производительность, поэтому используйте ее обоснованно.

Хильчук

**Синхронизация процессов** — это задача координации выполнения процессов таким образом, чтобы никакие два процесса не могли иметь доступ к одним и тем же общим данным и ресурсам.