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 № 14**

по дисциплине:” Системное программирование”

на тему: “ РАЗДЕЛЯЕМАЯ ПАМЯТЬ (Shared memory), СИНХРОНИЗАЦИЯ ”

Выполнил**:** студент группы 10701321

Грудинский К. А.

Принял**:** Кондратёнок Е.В.

Минск 2023

# Лабораторная работа № 14. РАЗДЕЛЯЕМАЯ ПАМЯТЬ (Shared memory), СИНХРОНИЗАЦИЯ

# Цель работы: Освоить механизм взаимодействия между процессами на основе использова-ния общей памяти.

**Задание 1**

Используя листинги упражнения №1 создайте еще несколько программ shm1-ownerХХ.c (количество программ «ownerХХ» должно соответствовать количеству членов команды). Программы должны выводить в сообщениях имена членов команды. Перестройте коды программ так, чтобы процессы обращались к одному сегменту памяти.

**Решение**

*Owner*

#include <stdio.h>

#include <string.h>

#include <sys/shm.h>

int main() {

int shm\_id;

char \*shm\_buf;

struct shmid\_ds ds;

// Создаем сегмент общей памяти

shm\_id = shmget(IPC\_PRIVATE, 4096, IPC\_CREAT | IPC\_EXCL | 0600);

if (shm\_id == -1) {

fprintf(stderr, "shmget() error\n");

return 1;

}

// Присоединяем сегмент к адресному пространству процесса

shm\_buf = (char \*)shmat(shm\_id, NULL, 0);

if (shm\_buf == (char \*)-1) {

fprintf(stderr, "shmat() error\n");

return 1;

}

// Узнаем информацию о сегменте

shmctl(shm\_id, IPC\_STAT, &ds);

printf("ID: %d\n", shm\_id);

// Добавляем свое сообщение в общий сегмент

strcat(shm\_buf, "Hello from owner01!\n");

printf("Press <Enter> to exit...");

fgetc(stdin);

// Отсоединяем сегмент от адресного пространства

shmdt(shm\_buf);

// Удаляем сегмент общей памяти

shmctl(shm\_id, IPC\_RMID, NULL);

return 0;

}

*User*

#include <sys/shm.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*\*argv) {

int shm\_id;

char \*shm\_buf;

if (argc < 2) {

fprintf(stderr, "Usage: %s <shm\_id>\n", argv[0]);

return 1;

}

// Получаем идентификатор сегмента из аргумента командной строки

shm\_id = atoi(argv[1]);

shm\_buf = (char \*)shmat(shm\_id, 0, 0);

if (shm\_buf == (char \*)-1) {

fprintf(stderr, "shmat() error\n");

return 1;

}

// Выводим сообщение из общего сегмента

printf("Message: %s\n", shm\_buf);

// Отсоединяем сегмент от адресного пространства

shmdt(shm\_buf);

return 0;

}

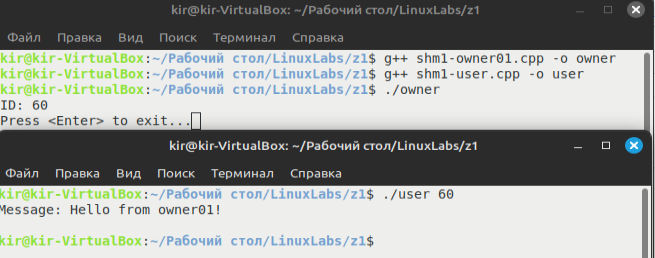


Рисунок 1 – выполнение задания 1

Задание 2.

Использования условия задания №1, создайте несколько программ **shm1-ownerХХ.c**, с использованием одного семафора.

1. Подключить к *общей памяти* двух клиентов (один из них выполняется как поток).
2. Создать два *объекта общие памяти*. Первый процесс пишет в первый *объект* и читает из *второго объекта памяти*, второй процесс, наоборот, читает из первого объекта и пишет во второй (программа-чат).
3. Сделать два сервера сообщений и одного клиента. Серверы используют одну и ту же общую память. Клиент читает и определяет, от какого сервера пришло сообщение.

**Решение**

**2.1**

*Сервер*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <unistd.h>

#define SHM\_SIZE 128

int main() {

key\_t key = ftok("shmfile", 65);

// Создаем общую память

int shmid = shmget(key, SHM\_SIZE, IPC\_CREAT | 0666);

if (shmid == -1) {

perror("shmget");

exit(EXIT\_FAILURE);

}

// Получаем указатель на разделяемую память

char\* data = (char\*)shmat(shmid, (void\*)0, 0);

if (data == (char\*)(-1)) {

perror("shmat");

exit(EXIT\_FAILURE);

}

// Создаем семафор

int semid = semget(key, 1, IPC\_CREAT | 0666);

if (semid == -1) {

perror("semget");

exit(EXIT\_FAILURE);

}

// Устанавливаем начальное значение семафора

semctl(semid, 0, SETVAL, 1);

// Сервер выводит слово "Сервер" и ждет клиента

while (1) {

printf("Сервер: %s\n", data);

sleep(1);

}

// Освобождаем ресурсы

shmdt(data);

shmctl(shmid, IPC\_RMID, NULL);

semctl(semid, 0, IPC\_RMID);

return 0;

}

*Сервер*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <unistd.h>

#define SHM\_SIZE 128

int main() {

key\_t key = ftok("shmfile", 65);

// Получаем идентификатор общей памяти

int shmid = shmget(key, SHM\_SIZE, 0666);

if (shmid == -1) {

perror("shmget");

exit(EXIT\_FAILURE);

}

// Получаем указатель на разделяемую память

char\* data = (char\*)shmat(shmid, (void\*)0, 0);

if (data == (char\*)(-1)) {

perror("shmat");

exit(EXIT\_FAILURE);

}

// Получаем идентификатор семафора

int semid = semget(key, 1, 0666);

if (semid == -1) {

perror("semget");

exit(EXIT\_FAILURE);

}

while (1) {

// Захватываем семафор

struct sembuf sem\_op;

sem\_op.sem\_num = 0;

sem\_op.sem\_op = -1;

sem\_op.sem\_flg = 0;

semop(semid, &sem\_op, 1);

// Клиент записывает свое слово

strcpy(data, "Клиент");

// Освобождаем семафор

sem\_op.sem\_op = 1;

semop(semid, &sem\_op, 1);

sleep(2);

}

// Освобождаем ресурсы

shmdt(data);

return 0;

}

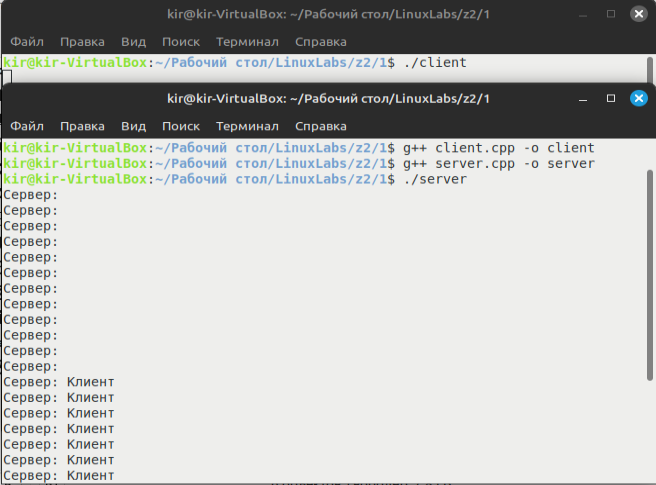


Рисунок 2 – выполнение задания 2.1

**2.2**

*Сервер*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <unistd.h>

#define SHM\_SIZE 128

int main() {

key\_t key = ftok("shmfile", 65);

// Создаем первый сегмент общей памяти

int shmid1 = shmget(key, SHM\_SIZE, IPC\_CREAT | 0666);

if (shmid1 == -1) {

perror("shmget");

exit(EXIT\_FAILURE);

}

// Получаем указатель на первый сегмент общей памяти

char\* data1 = (char\*)shmat(shmid1, (void\*)0, 0);

if (data1 == (char\*)(-1)) {

perror("shmat");

exit(EXIT\_FAILURE);

}

// Создаем второй сегмент общей памяти

int shmid2 = shmget(key + 1, SHM\_SIZE, IPC\_CREAT | 0666);

if (shmid2 == -1) {

perror("shmget");

exit(EXIT\_FAILURE);

}

// Получаем указатель на второй сегмент общей памяти

char\* data2 = (char\*)shmat(shmid2, (void\*)0, 0);

if (data2 == (char\*)(-1)) {

perror("shmat");

exit(EXIT\_FAILURE);

}

// Создаем семафор

int semid = semget(key, 1, IPC\_CREAT | 0666);

if (semid == -1) {

perror("semget");

exit(EXIT\_FAILURE);

}

semctl(semid, 0, SETVAL, 1);

while (1) {

// Ждем, пока первый процесс освободит семафор

struct sembuf sem\_op;

sem\_op.sem\_num = 0;

sem\_op.sem\_op = -1;

sem\_op.sem\_flg = 0;

semop(semid, &sem\_op, 1);

// Читаем из первого сегмента и пишем во второй

printf("Процесс 1: %s\n", data1);

printf("Введите сообщение для отправки процессу 1: ");

fgets(data2, SHM\_SIZE, stdin);

// Освобождаем семафор для первого процесса

sem\_op.sem\_op = 1;

semop(semid, &sem\_op, 1);

}

// Освобождаем ресурсы

shmdt(data1);

shmdt(data2);

shmctl(shmid1, IPC\_RMID, NULL);

shmctl(shmid2, IPC\_RMID, NULL);

semctl(semid, 0, IPC\_RMID);

return 0;

}

*Клиент*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <unistd.h>

#define SHM\_SIZE 128

int main() {

key\_t key = ftok("shmfile", 65);

// Создаем первый сегмент общей памяти

int shmid1 = shmget(key, SHM\_SIZE, IPC\_CREAT | 0666);

if (shmid1 == -1) {

perror("shmget");

exit(EXIT\_FAILURE);

}

// Получаем указатель на первый сегмент общей памяти

char\* data1 = (char\*)shmat(shmid1, (void\*)0, 0);

if (data1 == (char\*)(-1)) {

perror("shmat");

exit(EXIT\_FAILURE);

}

// Создаем второй сегмент общей памяти

int shmid2 = shmget(key + 1, SHM\_SIZE, IPC\_CREAT | 0666);

if (shmid2 == -1) {

perror("shmget");

exit(EXIT\_FAILURE);

}

// Получаем указатель на второй сегмент общей памяти

char\* data2 = (char\*)shmat(shmid2, (void\*)0, 0);

if (data2 == (char\*)(-1)) {

perror("shmat");

exit(EXIT\_FAILURE);

}

// Создаем семафор

int semid = semget(key, 1, IPC\_CREAT | 0666);

if (semid == -1) {

perror("semget");

exit(EXIT\_FAILURE);

}

while (1) {

// Ждем, пока второй процесс освободит семафор

struct sembuf sem\_op;

sem\_op.sem\_num = 0;

sem\_op.sem\_op = -1;

sem\_op.sem\_flg = 0;

semop(semid, &sem\_op, 1);

// Читаем из второго сегмента и пишем в первый

printf("Процесс 2: %s\n", data2);

printf("Введите сообщение для отправки процессу 2: ");

fgets(data1, SHM\_SIZE, stdin);

// Освобождаем семафор для второго процесса

sem\_op.sem\_op = 1;

semop(semid, &sem\_op, 1);

}

// Освобождаем ресурсы

shmdt(data1);

shmdt(data2);

shmctl(shmid1, IPC\_RMID, NULL);

shmctl(shmid2, IPC\_RMID, NULL);

semctl(semid, 0, IPC\_RMID);

return 0;

}

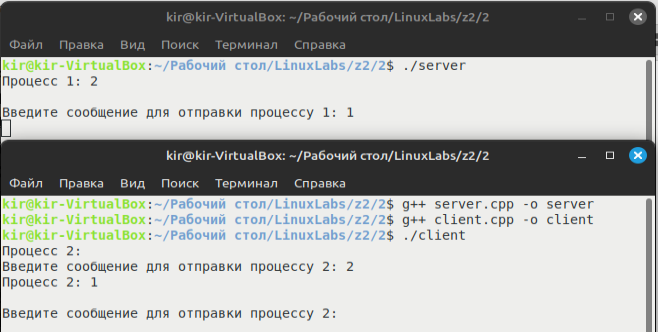


Рисунок 3 – выполнение задания 2.2

2.3

*Сервер1*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <unistd.h>

#define SHM\_SIZE 128

int main() {

key\_t key = ftok("shmfile", 65);

// Создаем общую память

int shmid = shmget(key, SHM\_SIZE, IPC\_CREAT | 0666);

if (shmid == -1) {

perror("shmget");

exit(EXIT\_FAILURE);

}

// Получаем указатель на разделяемую память

char\* data = (char\*)shmat(shmid, (void\*)0, 0);

if (data == (char\*)(-1)) {

perror("shmat");

exit(EXIT\_FAILURE);

}

// Создаем семафор

int semid = semget(key, 1, IPC\_CREAT | 0666);

if (semid == -1) {

perror("semget");

exit(EXIT\_FAILURE);

}

// Устанавливаем начальное значение семафора

semctl(semid, 0, SETVAL, 1);

while(1)

{

// Захватываем семафор

struct sembuf sem\_op;

sem\_op.sem\_num = 0;

sem\_op.sem\_op = -1;

sem\_op.sem\_flg = 0;

semop(semid, &sem\_op, 1);

// Сервер 1 пишет сообщение

strcpy(data, "1: Привет!");

// Освобождаем семафор

sem\_op.sem\_op = 1;

semop(semid, &sem\_op, 1);

printf("записано");

sleep(2);

}

// Освобождаем ресурсы

shmdt(data);

shmctl(shmid, IPC\_RMID, NULL);

semctl(semid, 0, IPC\_RMID);

return 0;

}

*Сервер2*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <unistd.h>

#define SHM\_SIZE 128

int main() {

key\_t key = ftok("shmfile", 65);

// Получаем идентификатор общей памяти

int shmid = shmget(key, SHM\_SIZE, 0666);

if (shmid == -1) {

perror("shmget");

exit(EXIT\_FAILURE);

}

// Получаем указатель на разделяемую память

char\* data = (char\*)shmat(shmid, (void\*)0, 0);

if (data == (char\*)(-1)) {

perror("shmat");

exit(EXIT\_FAILURE);

}

// Получаем идентификатор семафора

int semid = semget(key, 1, 0666);

if (semid == -1) {

perror("semget");

exit(EXIT\_FAILURE);

}

while(1)

{

struct sembuf sem\_op;

sem\_op.sem\_num = 0;

sem\_op.sem\_op = -1;

sem\_op.sem\_flg = 0;

semop(semid, &sem\_op, 1);

// Сервер 2 пишет сообщение

strcpy(data, "2: Привет");

// Освобождаем семафор

sem\_op.sem\_op = 1;

semop(semid, &sem\_op, 1);

printf("записано");

sleep(2);

}

// Освобождаем ресурсы

shmdt(data);

return 0;

}

*Клиент*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <unistd.h>

#define SHM\_SIZE 128

int main() {

key\_t key = ftok("shmfile", 65);

// Получаем идентификатор общей памяти

int shmid = shmget(key, SHM\_SIZE, 0666);

if (shmid == -1) {

perror("shmget");

exit(EXIT\_FAILURE);

}

// Получаем указатель на разделяемую память

char\* data = (char\*)shmat(shmid, (void\*)0, 0);

if (data == (char\*)(-1)) {

perror("shmat");

exit(EXIT\_FAILURE);

}

// Получаем идентификатор семафора

int semid = semget(key, 1, 0666);

if (semid == -1) {

perror("semget");

exit(EXIT\_FAILURE);

}

while (1) {

// Захватываем семафор

struct sembuf sem\_op;

sem\_op.sem\_num = 0;

sem\_op.sem\_op = -1;

sem\_op.sem\_flg = 0;

semop(semid, &sem\_op, 1);

// Читаем сообщение и определяем от кого оно пришло

if (strncmp(data, "1: ", 3) == 0) {

printf("Получено сообщение от Сервера 1: %s\n", data + 3);

} else if (strncmp(data, "2: ", 3) == 0) {

printf("Получено сообщение от Сервера 2: %s\n", data + 3);

} else {

printf("Некорректное сообщение: %s\n", data);

}

// Освобождаем семафор

sem\_op.sem\_op = 1;

semop(semid, &sem\_op, 1);

sleep(1);

}

// Освобождаем ресурсы

shmdt(data);

return 0;

}

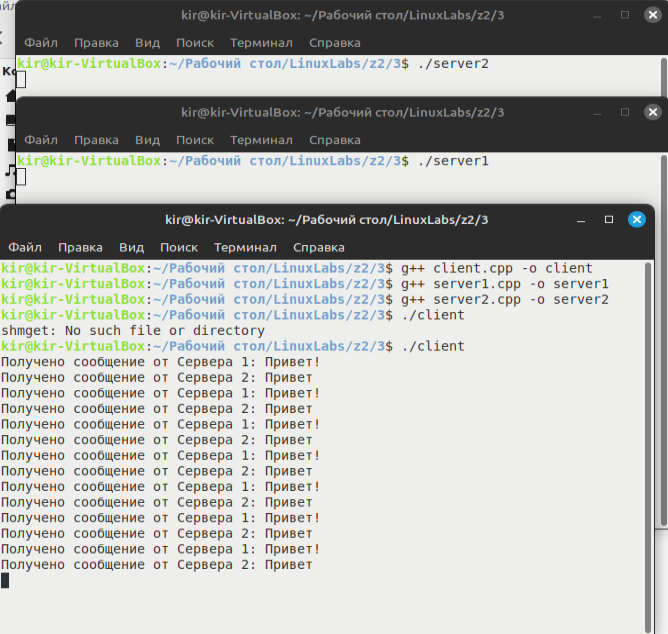


Рисунок 4 – выполнение задания 2.3

Вывод*:* В ходе выполнения лабораторной работы ознакомились с разделяемой памятью и синхронизацией в Linux.

**Контрольные вопросы**

*Что такое* ***сегмент*** *памяти, и его атрибуты?*

Сегмент памяти в компьютерных системах — это логически или физически непрерывная область адресного пространства, предназначенная для хранения данных или инструкций программы. Сегменты используются операционной системой и аппаратным обеспечением для эффективного управления памятью.

Атрибуты сегмента памяти могут варьироваться в зависимости от архитектуры компьютера и операционной системы, но обычно они включают в себя следующие характеристики:

Базовый адрес (Base Address): Это адрес, с которого начинается сегмент в физической или виртуальной памяти. Все адреса внутри сегмента считаются относительными этому базовому адресу.

Длина (Length): Размер сегмента, который указывает, сколько памяти занимает данный сегмент. Он может быть выражен в байтах, килобайтах, мегабайтах и т.д., в зависимости от используемой единицы измерения.

Права доступа (Access Rights): Этот атрибут определяет, как можно обращаться к данным в сегменте. Например, права доступа могут включать чтение, запись и выполнение.

Тип сегмента (Segment Type): Этот атрибут указывает, является ли сегмент кодовым сегментом (хранит исполняемый код), сегментом данных (хранит переменные данных) или другим типом.

Дескриптор сегмента (Segment Descriptor): В архитектуре x86, например, сегменты описываются сегментными дескрипторами, которые содержат информацию о базовом адресе, длине, правах доступа и других атрибутах.

Специфические атрибуты: В зависимости от архитектуры и ОС могут также присутствовать дополнительные атрибуты, такие как флаги сегмента, указывающие на присутствие данных в памяти, атрибуты защиты от записи, кэширование и др.

Общие принципы управления сегментами памяти могут различаться, например, между архитектурами x86 и ARM, а также между разными операционными системами.