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**

*Owner1*

#include <stdio.h>

#include <string.h>

#include <sys/shm.h>

#define SHMEM\_SIZE 4096

int main()

{

int shm\_id;

char \*shm\_buf;

struct shmid\_ds ds;

// Создаем сегмент общей памяти

shm\_id = shmget(IPC\_PRIVATE, SHMEM\_SIZE, 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, "Owner1 say: Grudinskiy Kirill\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);

*Owner2*

#include <stdio.h>

#include <string.h>

#include <sys/shm.h>

#define SHMEM\_SIZE 4096

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

{

int shm\_id;

char \*shm\_buf;

struct shmid\_ds ds;

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]);

// 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;

}

// Добавляем свое сообщение в общий сегмент

strcat(shm\_buf, "Owner2 say: Grudinskiy Kirill!\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;

}

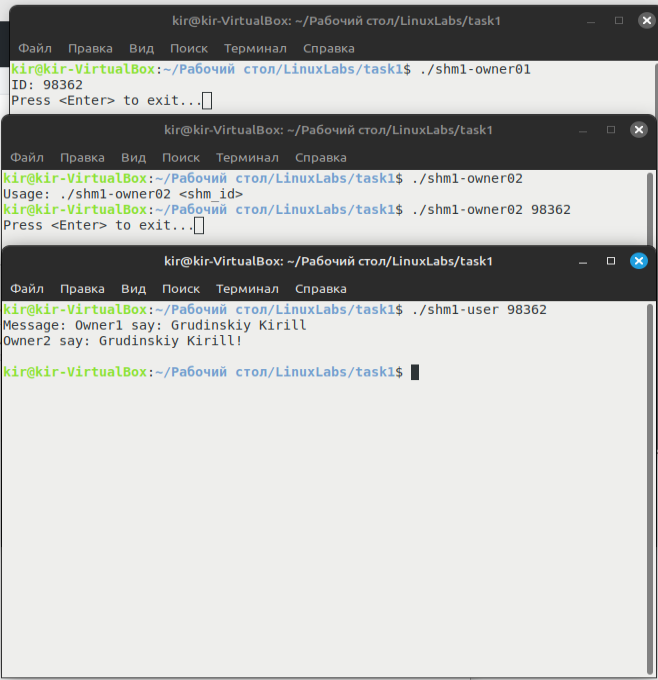


Рисунок 1 – выполнение задания 1

Задание 2.

1. Использования условия задания №1, создайте несколько программ **shm1-ownerХХ.c**, с использованием одного семафора.
2. Подключить к *общей памяти* двух клиентов (один из них выполняется как поток).
3. 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).
4. Сделать два сервера сообщений и одного клиента. Серверы используют одну и ту же общую память. Клиент читает и определяет, от какого сервера пришло сообщение.

*Клиент 1*

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

struct message {

long mtype;

char mtext[100];

};

int main() {

// Создаем очереди сообщений

int msgQueueID1 = msgget(IPC\_PRIVATE, IPC\_CREAT | 0666);

int msgQueueID2 = msgget(IPC\_PRIVATE, IPC\_CREAT | 0666);

printf("Process 1: msgQueueID1 = %d, msgQueueID2 = %d\n", msgQueueID1, msgQueueID2);

struct message msg;

while (1) {

msgrcv(msgQueueID1, &msg, sizeof(msg.mtext), 1, 0);

printf("Received from Process 2: %s\n", msg.mtext);

msg.mtype = 2;

printf("Enter message for Process 2: ");

fgets(msg.mtext, sizeof(msg.mtext), stdin);

msgsnd(msgQueueID2, &msg, sizeof(msg.mtext), 0);

}

return 0;

}

*Клиент 2.*

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

// Структура для сообщения

struct message {

long mtype; // Тип сообщения

char mtext[100]; // Текст сообщения

};

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

if (argc != 3) {

printf("Неверное количество параметров \n");

return 1;

}

int msgQueueID1 = atoi(argv[1]);

int msgQueueID2 = atoi(argv[2]);

struct message msg;

while (1) {

msg.mtype = 1;

printf("Enter message for Process 1: ");

fgets(msg.mtext, sizeof(msg.mtext), stdin);

msgsnd(msgQueueID1, &msg, sizeof(msg.mtext), 0);

msgrcv(msgQueueID2, &msg, sizeof(msg.mtext), 2, 0);

printf("Received from Process 1: %s\n", msg.mtext);

}

return 0;

}

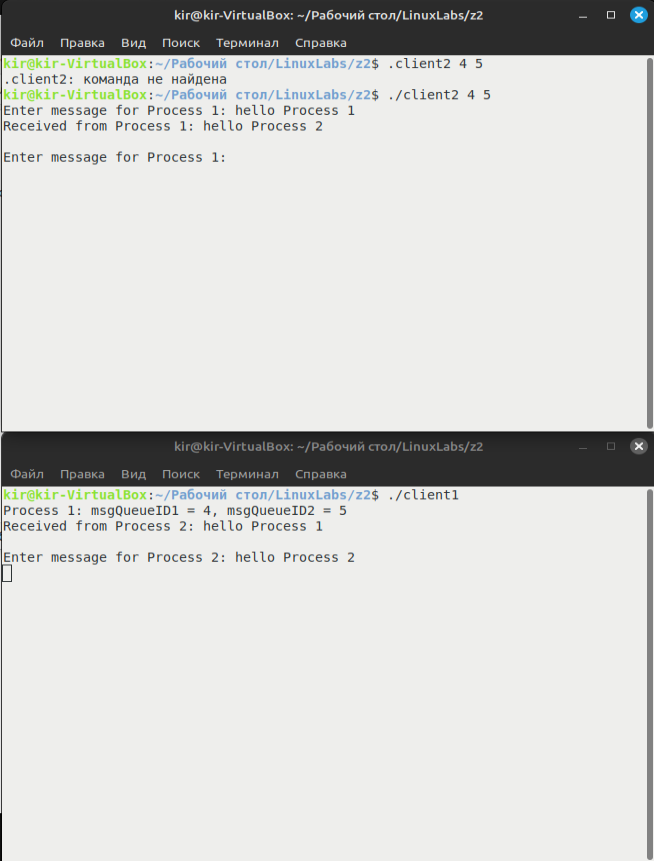


Рисунок 2 – выполнение задания 2

Задание 3.

Сделать два сервера сообщений и одного клиента. Серверы используют одну и ту же очередь. Клиент читает и определяет, от какого сервера пришло сообщение.

*Сервер 1.*

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

struct message {

long mtype;

char mtext[100];

};

int main() {

key\_t key = ftok("server\_client\_communication", 1);

int msgQueueID = msgget(key, IPC\_CREAT | 0666);

struct message msg;

while (1) {

msg.mtype = 1;

snprintf(msg.mtext, sizeof(msg.mtext), "Message from Server 1");

msgsnd(msgQueueID, &msg, sizeof(msg.mtext), 0);

sleep(2);

}

return 0;

}

*Сервер 2.*

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

struct message {

long mtype;

char mtext[100];

};

int main() {

key\_t key = ftok("server\_client\_communication", 1);

int msgQueueID = msgget(key, IPC\_CREAT | 0666);

struct message msg;

while (1) {

msg.mtype = 2;

snprintf(msg.mtext, sizeof(msg.mtext), "Message from Server 2");

msgsnd(msgQueueID, &msg, sizeof(msg.mtext), 0);

sleep(3);

}

return 0;

}

*Клиент.*

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

struct message {

long mtype;

char mtext[100];

};

int main() {

key\_t key = ftok("server\_client\_communication", 1);

int msgQueueID = msgget(key, IPC\_CREAT | 0666);

struct message msg;

while (1) {

msgrcv(msgQueueID, &msg, sizeof(msg.mtext), 0, 0);

if (msg.mtype == 1) {

printf("Received from Server 1: %s\n", msg.mtext);

} else if (msg.mtype == 2) {

printf("Received from Server 2: %s\n", msg.mtext);

}

}

return 0;

}

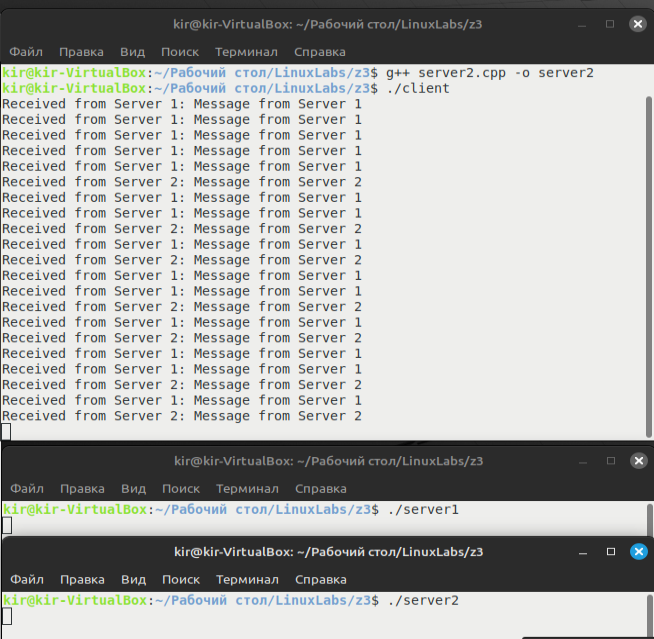


Рисунок 3 – выполнение задания 3

Вывод*:* В ходе выполнения лабораторной работы ознакомились с механизмом работы очередей в Linux. Были созданы программы использующие механизм очередей.

**Контрольные вопросы**

Механизм очередей сообщений в LINUX.

Механизм очередей сообщений в Linux предоставляет средства для обмена данными между процессами через использование сообщений. Очереди сообщений являются одним из механизмов межпроцессного взаимодействия (IPC), предоставляемых операционной системой.

Основные системные вызовы и структуры данных, связанные с механизмом очередей сообщений, включают:

msgget: Создает новую очередь сообщений или получает доступ к существующей. Пример использования:

int msgget(key\_t key, int msgflg);

msgsnd: Помещает сообщение в очередь. Пример использования:

int msgsnd(int msqid, const void \*msgp, size\_t msgsz, int msgflg);

msgrcv: Извлекает сообщение из очереди. Пример использования:

ssize\_t msgrcv(int msqid, void \*msgp, size\_t msgsz, long msgtyp, int msgflg);

msgctl: Управляет очередью сообщений, предоставляет различные операции, такие как удаление очереди. Пример использования:

int msgctl(int msqid, int cmd, struct msqid\_ds \*buf);

struct msgbuf: Структура данных, представляющая сообщение в очереди. Обычно определена следующим образом:

struct msgbuf {

long mtype; /\* Тип сообщения \*/

char mtext[1]; /\* Текст сообщения (может быть массивом переменной длины) \*/

};