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 № 15**

по дисциплине:” Системное программирование”

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

Выполнил**:** студент группы 10701321

Грудинский К. А.

Принял**:** Кондратёнок Е.В.

Минск 2023

# Лабораторная работа № 15. ИСПОЛЬЗОВАНИЕ ОБЩИХ ФАЙЛОВ, БУФЕРИЗАЦИЯ, СИНХРОНИЗАЦИЯ

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

**Задание 1**

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

**Решение**

#include <iostream>

#include <fstream>

#include <sys/mman.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/wait.h>

#define FILENAME "shared\_file"

struct SharedData {

int value;

char message[256];

};

void serverProcess() {

int fd = open(FILENAME, O\_CREAT | O\_RDWR, S\_IRUSR | S\_IWUSR);

if (fd == -1) {

std::cerr << "Failed to open the file" << std::endl;

return;

}

ftruncate(fd, sizeof(SharedData));

SharedData\* sharedData = static\_cast<SharedData\*>(mmap(NULL, sizeof(SharedData), PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0));

close(fd);

while (true) {

// Server reads data from the shared memory

std::cout << "Server: Value = " << sharedData->value << ", Message = " << sharedData->message << std::endl;

sleep(1);

}

munmap(sharedData, sizeof(SharedData));

}

void clientProcess(int clientId) {

int fd = open(FILENAME, O\_RDWR);

if (fd == -1) {

std::cerr << "Failed to open the file" << std::endl;

return;

}

SharedData\* sharedData = static\_cast<SharedData\*>(mmap(NULL, sizeof(SharedData), PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0));

close(fd);

int counter = 0;

while (counter < 5) {

// Clients write data to the shared memory

sharedData->value = counter;

snprintf(sharedData->message, sizeof(sharedData->message), "Message from Client %d", clientId);

std::cout << "Client " << clientId << ": Data written" << std::endl;

sleep(2);

counter++;

}

munmap(sharedData, sizeof(SharedData));

}

int main() {

// Create the server process

if (fork() == 0) {

serverProcess();

exit(0);

}

// Create two client processes

if (fork() == 0) {

clientProcess(1);

exit(0);

}

if (fork() == 0) {

clientProcess(2);

exit(0);

}

// Wait for all child processes to complete

for (int i = 0; i < 3; ++i) {

wait(NULL);

}

return 0;

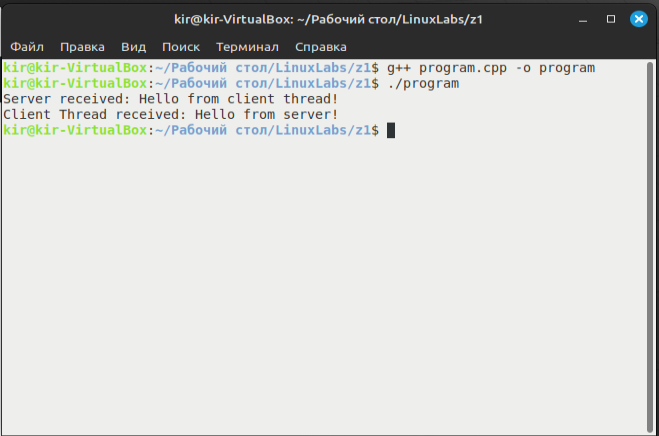
}

Рисунок 1 – выполнение задания 1

Задание 2.

Создать две очереди сообщений. Первый процесс пишет в первую очередь и читает из второй, второй процесс, наоборот, читает из первой очереди и пишет во вторую (программа-чат).

*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;

}

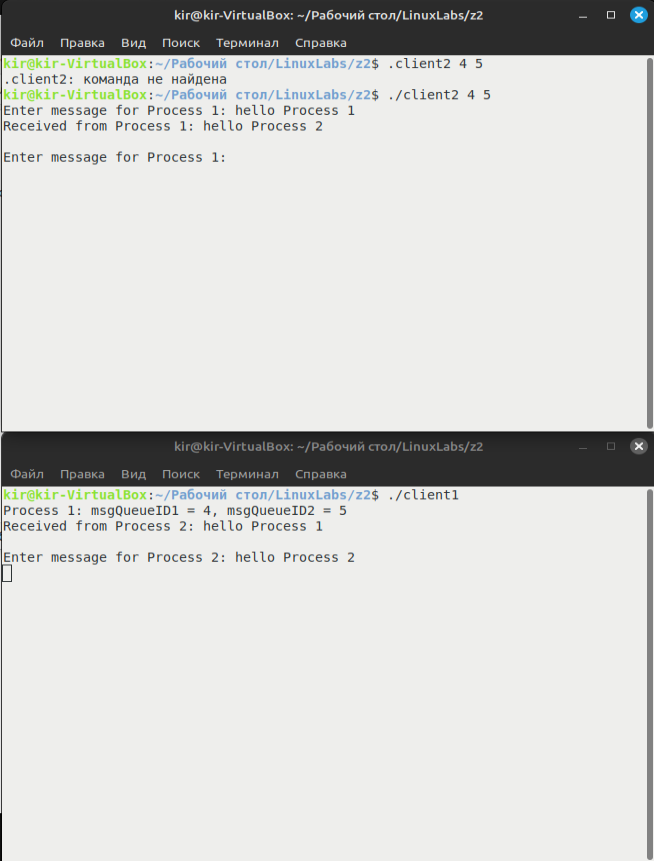


Рисунок 2 – выполнение задания 2

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;

}

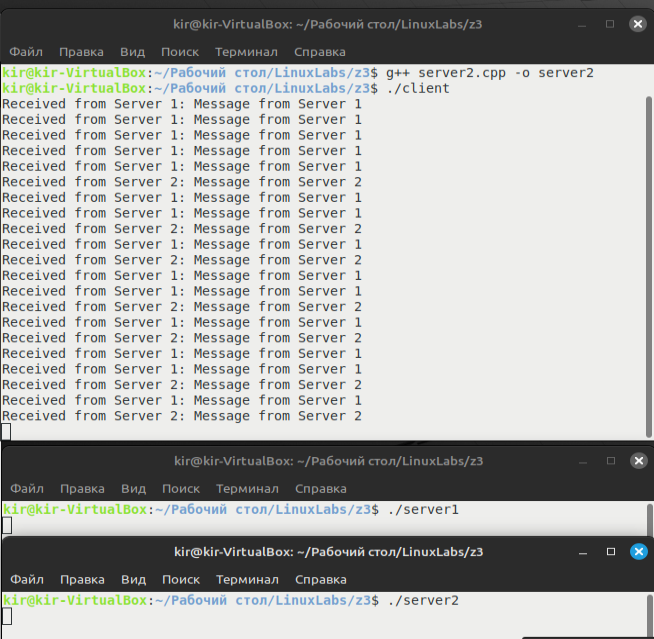


Рисунок 3 – выполнение задания 3

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) \*/

};