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 № 34**

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 № 34 Вариант №1

### Задание 1: Реализация простого чат-приложения

**Цель:** научиться организовывать взаимодействие нескольких процессов с помощью очередей сообщений.

1. Напишите программу на C, которая создает очередь сообщений.
2. Реализуйте простую структуру сообщения, которая будет содержать текстовое сообщение и идентификатор отправителя.
3. Создайте два процесса: один для отправки сообщений, другой для их получения.
4. Процесс отправителя должен отправлять сообщения в очередь, а процесс получателя - получать их и выводить на экран.
5. Подключить к очереди сообщений двух (и более) клиентов (один из них выполняется как поток).
6. Расширьте предыдущее задание, добавив возможность отправки сообщений от нескольких процессов-отправителей к одному процессу-получателю.
7. Реализуйте возможность указания имени отправителя (члена команды) в каждом сообщении.
8. Добавьте в программу обработку сигналов для корректного завершения работы процессов.

Ключ к общей памяти должен задаваться автоматически системой.

Затем **перестройте программу** так, чтобы в ней использовался один ***семафор***.

Код   
  
#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <signal.h>

#include <semaphore.h>

#include <fcntl.h>

#define MAX\_TEXT 100

#define MSG\_TYPE 1

struct message {

long msg\_type;

char text[MAX\_TEXT];

char sender[20];

};

sem\_t \*sem;

void handle\_signal(int sig) {

sem\_close(sem);

sem\_unlink("/msg\_semaphore"); // Удаление семафора

exit(0);

}

int main() {

int msgid;

struct message msg;

key\_t key = ftok("msg\_queue", 65); // Генерация ключа

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

if (msgid == -1) {

perror("msgget");

exit(1);

}

// Создание семафора

sem = sem\_open("/msg\_semaphore", O\_CREAT, 0666, 1);

if (sem == SEM\_FAILED) {

perror("sem\_open");

exit(1);

}

signal(SIGINT, handle\_signal); // Обработка сигнала для завершения

while (1) {

printf("Enter your name: ");

scanf("%s", msg.sender);

printf("Enter message: ");

getchar(); // Очистка буфера

fgets(msg.text, MAX\_TEXT, stdin);

msg.text[strcspn(msg.text, "\n")] = 0; // Удаление новой строки

msg.msg\_type = MSG\_TYPE;

sem\_wait(sem); // Захват семафора

msgsnd(msgid, &msg, sizeof(msg) - sizeof(long), 0);

sem\_post(sem); // Освобождение семафора

printf("Message sent: %s from %s\n", msg.text, msg.sender);

}

sem\_close(sem);

return 0;

}  
  
#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <signal.h>

#include <semaphore.h>

#include <fcntl.h>

#define MAX\_TEXT 100

#define MSG\_TYPE 1

struct message {

long msg\_type;

char text[MAX\_TEXT];

char sender[20];

};

sem\_t \*sem;

void handle\_signal(int sig) {

sem\_close(sem);

sem\_unlink("/msg\_semaphore"); // Удаление семафора

exit(0);

}

int main() {

int msgid;

struct message msg;

key\_t key = ftok("msg\_queue", 65); // Генерация ключа

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

if (msgid == -1) {

perror("msgget");

exit(1);

}

// Открытие семафора

sem = sem\_open("/msg\_semaphore", 0);

if (sem == SEM\_FAILED) {

perror("sem\_open");

exit(1);

}

signal(SIGINT, handle\_signal); // Обработка сигнала для завершения

while (1) {

sem\_wait(sem); // Захват семафора

if (msgrcv(msgid, &msg, sizeof(msg) - sizeof(long), MSG\_TYPE, 0) != -1) {

printf("Received message: %s from %s\n", msg.text, msg.sender);

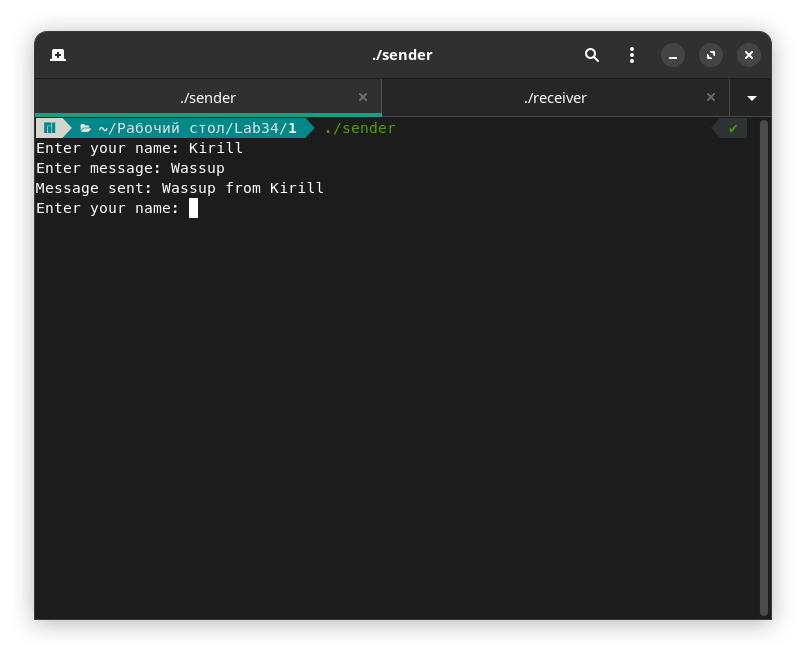
}

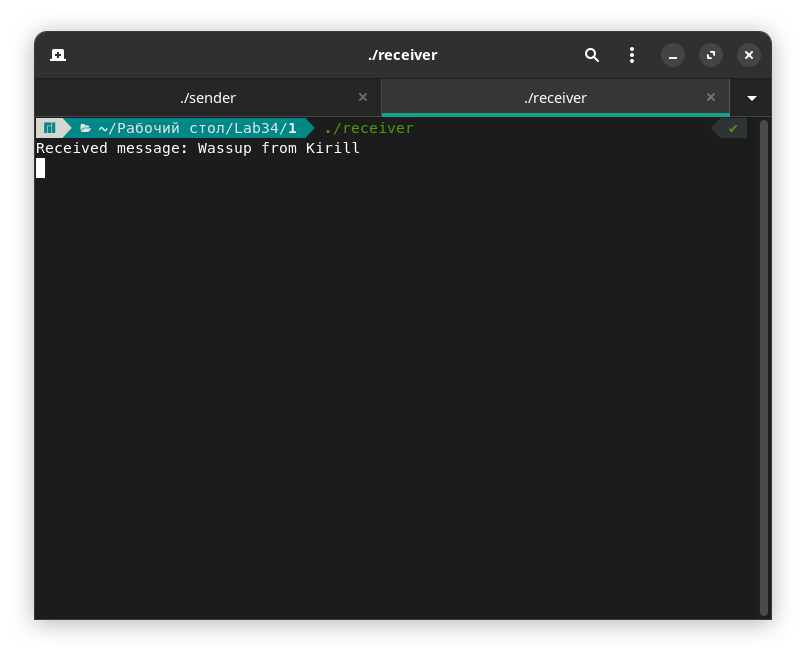
sem\_post(sem); // Освобождение семафора

}

sem\_close(sem);

return 0;

}  
  
  
  
  
Скриншоты  
  
  
  
  
  


 **Отправитель (sender.c):**

* Создает очередь сообщений и семафор.
* Запрашивает у пользователя имя и текст сообщения.
* Использует семафор для синхронизации доступа к очереди сообщений, отправляя сообщения в очередь.

**Получатель (receiver.c):**

* Открывает существующую очередь сообщений и семафор.
* Постоянно слушает сообщения из очереди.
* Использует семафор для синхронизации доступа при получении сообщений.

**Задание 2**

Создать два (и более) 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).

Ключ общей памяти должен задаваться пользователем, например, это могут быть номера телефонов членов команды.  
  
  
Код  
  
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <semaphore.h>

#include <fcntl.h>

#include <unistd.h>

#include <signal.h>

#define SIZE 1024

void handle\_signal(int sig) {

// Очистка ресурсов при завершении

sem\_unlink("/chat\_semaphore");

exit(0);

}

int main() {

char input[SIZE];

key\_t key1, key2;

int shmid1, shmid2;

char \*shm1, \*shm2;

sem\_t \*sem;

// Ввод ключей

printf("Enter key for shared memory 1: ");

scanf("%d", &key1);

printf("Enter key for shared memory 2: ");

scanf("%d", &key2);

// Создание первого объекта общей памяти

shmid1 = shmget(key1, SIZE, 0666 | IPC\_CREAT);

if (shmid1 == -1) {

perror("shmget");

exit(1);

}

shm1 = shmat(shmid1, NULL, 0);

if (shm1 == (char \*)-1) {

perror("shmat");

exit(1);

}

// Создание второго объекта общей памяти

shmid2 = shmget(key2, SIZE, 0666 | IPC\_CREAT);

if (shmid2 == -1) {

perror("shmget");

exit(1);

}

shm2 = shmat(shmid2, NULL, 0);

if (shm2 == (char \*)-1) {

perror("shmat");

exit(1);

}

// Создание семафора

sem = sem\_open("/chat\_semaphore", O\_CREAT, 0666, 1);

if (sem == SEM\_FAILED) {

perror("sem\_open");

exit(1);

}

signal(SIGINT, handle\_signal); // Обработка сигнала для завершения

while (1) {

// Чтение из общего объекта 1

sem\_wait(sem); // Захват семафора

printf("Message from other user: %s\n", shm2);

sem\_post(sem); // Освобождение семафора

// Ввод сообщения

printf("Enter your message: ");

getchar(); // Очистка буфера

fgets(input, SIZE, stdin);

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

// Запись в общий объект 1

sem\_wait(sem); // Захват семафора

strncpy(shm1, input, SIZE);

sem\_post(sem); // Освобождение семафора

}

// Освобождение ресурсов

sem\_close(sem);

shmdt(shm1);

shmdt(shm2);

return 0;

}  
  
  
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <semaphore.h>

#include <fcntl.h>

#include <unistd.h>

#include <signal.h>

#define SIZE 1024

void handle\_signal(int sig) {

// Очистка ресурсов при завершении

sem\_unlink("/chat\_semaphore");

exit(0);

}

int main() {

char input[SIZE];

key\_t key1, key2;

int shmid1, shmid2;

char \*shm1, \*shm2;

sem\_t \*sem;

// Ввод ключей

printf("Enter key for shared memory 1: ");

scanf("%d", &key1);

printf("Enter key for shared memory 2: ");

scanf("%d", &key2);

// Создание первого объекта общей памяти

shmid1 = shmget(key1, SIZE, 0666 | IPC\_CREAT);

if (shmid1 == -1) {

perror("shmget");

exit(1);

}

shm1 = shmat(shmid1, NULL, 0);

if (shm1 == (char \*)-1) {

perror("shmat");

exit(1);

}

// Создание второго объекта общей памяти

shmid2 = shmget(key2, SIZE, 0666 | IPC\_CREAT);

if (shmid2 == -1) {

perror("shmget");

exit(1);

}

shm2 = shmat(shmid2, NULL, 0);

if (shm2 == (char \*)-1) {

perror("shmat");

exit(1);

}

// Открытие семафора

sem = sem\_open("/chat\_semaphore", 0);

if (sem == SEM\_FAILED) {

perror("sem\_open");

exit(1);

}

signal(SIGINT, handle\_signal); // Обработка сигнала для завершения

while (1) {

// Чтение из общего объекта 2

sem\_wait(sem); // Захват семафора

printf("Message from other user: %s\n", shm1);

sem\_post(sem); // Освобождение семафора

// Ввод сообщения

printf("Enter your message: ");

getchar(); // Очистка буфера

fgets(input, SIZE, stdin);

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

// Запись в общий объект 2

sem\_wait(sem); // Захват семафора

strncpy(shm2, input, SIZE);

sem\_post(sem); // Освобождение семафора

}

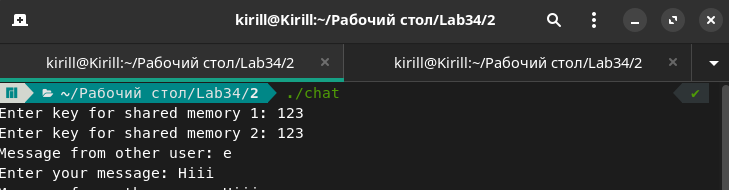
// Освобождение ресурсов

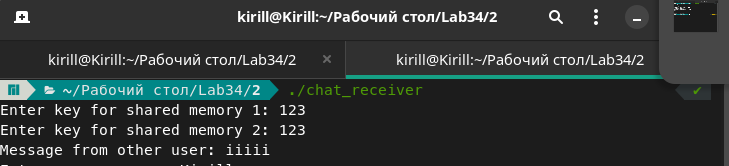
sem\_close(sem);

shmdt(shm1);

shmdt(shm2);

return 0;

}  
  
  


 **Задание 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. Ключ к общей памяти создается системой либо устанавливается пользователем (например, это могут быть номера телефонов членов команды).

Для совершенствования программы можно использовать средства синхронизации, семафоры или мьютексы.

Код  
  
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <semaphore.h>

#include <fcntl.h>

#include <unistd.h>

#include <signal.h>

#define SHM\_SIZE 1024

void handle\_signal(int sig) {

sem\_unlink("/shared\_semaphore");

exit(0);

}

int main() {

key\_t key;

int shmid;

char \*shm;

sem\_t \*sem;

// Ввод ключа

printf("Enter key for shared memory (server 1): ");

scanf("%d", &key);

// Создание общей памяти

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

if (shmid == -1) {

perror("shmget");

exit(1);

}

shm = shmat(shmid, NULL, 0);

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

perror("shmat");

exit(1);

}

// Создание семафора

sem = sem\_open("/shared\_semaphore", O\_CREAT, 0666, 1);

if (sem == SEM\_FAILED) {

perror("sem\_open");

exit(1);

}

signal(SIGINT, handle\_signal); // Обработка сигнала для завершения

while (1) {

char message[SHM\_SIZE];

// Ввод сообщения

printf("Server 1: Enter message: ");

getchar(); // Очистка буфера

fgets(message, SHM\_SIZE, stdin);

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

// Запись в общую память

sem\_wait(sem); // Захват семафора

snprintf(shm, SHM\_SIZE, "Server 1: %s", message);

sem\_post(sem); // Освобождение семафора

}

// Освобождение ресурсов

sem\_close(sem);

shmdt(shm);

return 0;

}  
  
  
 #include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <semaphore.h>

#include <fcntl.h>

#include <unistd.h>

#include <signal.h>

#define SHM\_SIZE 1024

void handle\_signal(int sig) {

sem\_unlink("/shared\_semaphore");

exit(0);

}

int main() {

key\_t key;

int shmid;

char \*shm;

sem\_t \*sem;

// Ввод ключа

printf("Enter key for shared memory (server 2): ");

scanf("%d", &key);

// Создание общей памяти

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

if (shmid == -1) {

perror("shmget");

exit(1);

}

shm = shmat(shmid, NULL, 0);

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

perror("shmat");

exit(1);

}

// Открытие семафора

sem = sem\_open("/shared\_semaphore", 0);

if (sem == SEM\_FAILED) {

perror("sem\_open");

exit(1);

}

signal(SIGINT, handle\_signal); // Обработка сигнала для завершения

while (1) {

char message[SHM\_SIZE];

// Ввод сообщения

printf("Server 2: Enter message: ");

getchar(); // Очистка буфера

fgets(message, SHM\_SIZE, stdin);

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

// Запись в общую память

sem\_wait(sem); // Захват семафора

snprintf(shm, SHM\_SIZE, "Server 2: %s", message);

sem\_post(sem); // Освобождение семафора

}

// Освобождение ресурсов

sem\_close(sem);

shmdt(shm);

return 0;

}  
  
  
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <semaphore.h>

#include <fcntl.h>

#include <unistd.h>

#include <signal.h>

#define SHM\_SIZE 1024

void handle\_signal(int sig) {

exit(0);

}

int main() {

key\_t key;

int shmid;

char \*shm;

sem\_t \*sem;

// Ввод ключа

printf("Enter key for shared memory (client): ");

scanf("%d", &key);

// Получение общей памяти

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

if (shmid == -1) {

perror("shmget");

exit(1);

}

shm = shmat(shmid, NULL, 0);

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

perror("shmat");

exit(1);

}

// Открытие семафора

sem = sem\_open("/shared\_semaphore", 0);

if (sem == SEM\_FAILED) {

perror("sem\_open");

exit(1);

}

signal(SIGINT, handle\_signal); // Обработка сигнала для завершения

while (1) {

sem\_wait(sem); // Захват семафора

printf("Received: %s\n", shm); // Чтение из общей памяти

sem\_post(sem); // Освобождение семафора

sleep(1); // Задержка для уменьшения загрузки ЦП

}

// Освобождение ресурсов

sem\_close(sem);

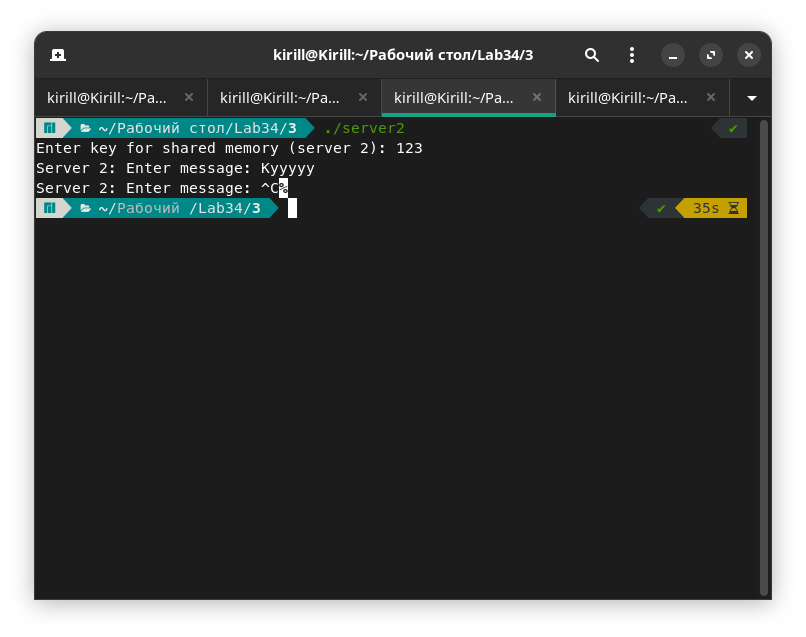
shmdt(shm);

return 0;

}  
  
  
**Создание общей памяти**:

* + Сервера и клиент используют один и тот же ключ для доступа к одному объекту общей памяти.
  + Серверы записывают сообщения в общую память, а клиент читает эти сообщения.

1. **Синхронизация**:
   * Семафор используется для управления доступом к общей памяти, чтобы избежать гонок условий при записи и чтении.
2. **События**:
   * Каждое сообщение записывается в общую память с указанием, от какого сервера оно пришло (например, "Server 1: Hello!").
   * Клиент постоянно читает сообщения из общей памяти и выводит их на экран.

Скриншоты  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  


### **Вопросы Зухта Опишите примерный сценарий работы с разделяемой памятью с использованием семафоров. Сценарий работы**

1. **Инициализация**:
   * **Создание ключа**: Генерируется уникальный ключ для доступа к разделяемой памяти.
   * **Создание объекта памяти**: Процесс создает объект разделяемой памяти с помощью shmget.
   * **Создание семафора**: Создается семафор для синхронизации доступа к памяти.
2. **Запись данных (Сервер)**:
   * **Подключение к памяти**: Процесс подключается к объекту разделяемой памяти с помощью shmat.
   * **Цикл записи**:
     + Пользователь вводит сообщение.
     + Захватывается семафор с помощью sem\_wait.
     + Сообщение записывается в разделяемую память.
     + Семафор освобождается с помощью sem\_post.
3. **Чтение данных (Клиент)**:
   * **Подключение к памяти**: Клиент подключается к тому же объекту с помощью shmat.
   * **Цикл чтения**:
     + Захватывается семафор с помощью sem\_wait.
     + Сообщение читается из памяти.
     + Семафор освобождается с помощью sem\_post.
     + Сообщение выводится на экран.
4. **Завершение работы**:
   * При получении сигнала (например, Ctrl+C), процессы корректно завершаются, освобождают ресурсы и удаляют семафор.