

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

ОТЧЁТ

по лабораторной работе №8

по дисциплине «Организация процессов и программирования в среде Linux»

Тема: ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ НА ОСНОВЕ СООБЩЕНИЙ

Студент гр. 9308

Преподаватель

Соболев М.С.

Разумовский Г.В.

Санкт-Петербург,

2022

Оглавление

1. Введение.....	3
1.1. Введение.....	3
1.2. Порядок выполнения работы.....	3
1.3. Содержание отчёта.....	4
2. Тексты программ.....	5
2.1. executable_1.cpp.....	5
2.2. executable_2.cpp.....	12
2.3. executable_3.cpp.....	19
3. Скриншоты работы каждой программы.....	26
4. Вывод.....	30
5. Список использованных источников.....	31

1. Введение

1.1. Введение

Тема работы: Взаимодействие процессов на основе сообщений.

Цель работы: Знакомство с механизмом обмена сообщениями и системными вызовами приёма и передачи сообщений.

1.2. Порядок выполнения работы

1. Написать две программы, обменивающиеся сообщениями. Первая программа создаёт очередь и ожидает сообщение от второй программы определённое время, которое задаётся при запуске первой программы и выводится на экран. Если за это время сообщение от второй программы не поступило, то первая программа завершает свою работу и уничтожает очередь. Вторая программа может запускаться несколько раз и только при условии, что первая программа работает, в противном случае она заканчивает свою работу. При запуске второй программы указывается очередное время ожидания для первой программы.

2. Откомпилировать обе программы. Выполнить 3 варианта их запуска:

2.1. запустить первую программу, не запуская вторую;

2.2. запустить вторую программу, не запуская первую;

2.3. запустить первую программу, и пока она работает, несколько раз запустите вторую с различными значениями времени ожидания.

3. Написать три программы, выполняющиеся параллельно и читающие один и тот же файл. Программа, которая хочет прочесть файл, должна передать другим программам запрос на разрешение операции и ожидать их ответа. Эти запросы программы передают через одну очередь сообщений. Ответы каждая программа должна принимать в свою локальную очередь. В запросе указываются: номер программы, которой посылается запрос, идентификатор

очереди, куда надо передать ответ, и время посылки запроса. Начать выполнять операцию чтения файла программе разрешается только при условии получения ответов от двух других программ. Каждая программа перед отображением файла на экране должна вывести следующую информацию: номер программы и времена ответов, полученных от других программ.

Программа, которая получила запрос от другой программы, должна реагировать следующим образом:

3.1. если программа прочитала файл, то сразу передаётся ответ, который должен содержать номер отвечающей программы и время ответа;

3.2. если файл не читался, то ответ передаётся только при условии, что время посылки запроса в сообщении меньше, чем время запроса на чтение у данной программы.

Запросы, на которые ответы не были переданы, должны быть запомнены и после чтения файла обслужены.

4. Откомпилировать 3 программы и запустить их несколько раз на разных терминалах в различной последовательности.

Выбранные задания: 3, 4.

1.3. Содержание отчёта

Отчёт по лабораторной работе должен содержать:

1. Цель и задание.
2. Тексты программ.
3. Скриншоты работы каждой программы.

2. Тексты программ

2.1. executable_1.cpp

```
// executable 1
// start program
// ./executable 1

#include <iostream>
#include <fstream>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <errno.h>
#include <cstring>
#include <string>

using namespace std;

// https://www.opennet.ru/man.shtml?topic=msgsnd&category=2&russian=0
// https://www.opennet.ru/man.shtml?topic=msgrcv&category=2&russian=0

typedef struct // order is very important
{
    long receiver_id; // program-receiver id
    int sender_id;    // program-sender id
    int local_queue_id; // local queue id
    time_t request_time; // request sending time
} MessageRequest;

typedef struct
{
    int sender_id; // program-sender id
    time_t response_time; // response time of program, who got request
} MessageResponse;

void sendingRequest (MessageRequest *local_buffer, int local_other_program_id, int local_program_id, int
local_local_queue, int local_common_queue);
```

```

int main(int argc, char *argv[])
{
    bool common_queue_owner; // owner of common queue

    int local_queue = 0; // local queue
    int common_queue = 0; // common queue
    int ability_got = 0; // ability to read got counter
    int ability_sended = 0; // ability to read sended counter
    int is_finished = 0; // number of program, who finished reading file
    int other_first_program_id = 0; // other program id
    int other_second_program_id = 0; // other program id
    int program_id = 1; // this program id
    int message_number = 0; // array number of recieved message

    MessageResponse message_response; // message response to send to other programs
    MessageRequest message_request_receive[2]; // message request to receiving from other programs
    MessageRequest message_request; // message request
    MessageRequest message_request_send[4]; // message request to sending to other programs

    cout << "----- PROGRAM NUMBER " << program_id << " -----\\n";

    // ----- CREATING/OPENING COMMON QUEUE -----

    // IPC_CREAT -- if there wasn't queue, it will be created
    // O_EXCL + IPC_CREAT -- if there was queue, msgget will return error
    common_queue = msgget(190, 0606 | IPC_CREAT | IPC_EXCL); // trying to create common queue
    // 190 -- key for identification, 0606 -- r&w for owner and others

    // checking if common queue has been created
    if (common_queue != -1) // if we created common queue, write message
    {
        common_queue_owner = true;
        cout << "----- COMMON QUEUE HAS BEEN CREATED -----\\n";
    }
    else // if we hasn't been created common queue, try to open, write message
    {

```

```

common_queue = msgget(190, IPC_CREAT); // trying to open common queue
if (common_queue == -1) // if we couldn't open, write message & terminate program
{
    cout << "----- COMMON QUEUE HAS NOT BEEN OPENED -----\\n";
    exit(-1);
}
else // if we can open, write message
{
    cout << "----- COMMON QUEUE HAS BEEN OPENED -----\\n";
}
}

// ----- CREATING LOCAL QUEUE -----

local_queue = msgget(IPC_PRIVATE, 0606 | IPC_CREAT); // creating local queue

// checking if local queue has been created or not
if (local_queue == -1) // if not created -- delete remaining object if it has been created & print message
{
    cout << "----- LOCAL QUEUE HAS NOT BEEN CREATED -----\\n\\n";
    if (common_queue_owner == true) // deleting local queue if there is remaining object
    {
        // if we are owner of the common queue, delete it (IPC_RMID means delete queue, alarm all
processes & throw an error)
        msgctl(common_queue, IPC_RMID, NULL);
    }
    exit(-1); // terminate program
}
else // if created -- print message
{
    cout << "----- LOCAL QUEUE HAS BEEN CREATED -----\\n\\n";
}

// ----- SENDING REQUESTS FOR ABILITY TO READ TO OTHER PROGRAMS -----

other_first_program_id = (program_id) % 3 + 1;
other_second_program_id = (program_id + 1) % 3 + 1;

```

```

        sendingRequest (message_request_send, other_first_program_id, program_id, local_queue, common_queue);
        sendingRequest (message_request_send, other_second_program_id, program_id, local_queue,
common_queue);

// ----- GETTING REQUESTS FOR ABILITY AND ABILITIES TO READ FROM OTHER
PROGRAMS -----

while(ability_got < 2) // when we have not got abilities to read from 2 other programs
{
    if(msgrcv(common_queue, &message_request_receive[message_number],
sizeof(message_request_receive[message_number]), program_id, IPC_NOWAIT) != -1) // common queue message
check
    {
        cout << "Request to read has been got from: " <<
message_request_receive[message_number].sender_id << "\n";
        cout << "Request to read has been send at: " <<
ctime(&message_request_receive[message_number].request_time) << "\n";

        // if the request TIME for ability to read from OTHER program <= request TIME for ability
to read from THIS program,
        // THIS program sends the ability to read to OTHER program (< || (= & id_sender < id_this))
        if((message_request_receive[message_number].request_time
<
message_request_send[message_request_receive[message_number].sender_id].request_time)
|| (message_request_receive[message_number].request_time
==
message_request_send[message_request_receive[message_number].sender_id].request_time
&& message_request_receive[message_number].sender_id < program_id))
        {
            message_response.sender_id = program_id;
            message_response.response_time = time(NULL);
            msgsnd(message_request_receive[message_number].local_queue_id,
&message_response, sizeof(message_response), 0);
            ability_sended = ability_sended + 1;

```



```

        cout << "Sending ability to read to: " <<
message_request_receive[message_number].sender_id << "\n\n";
    }
    else // else, untreated request will be placed to "message_request_receive" array
    {
        message_number = message_number + 1;
    }
}
// check messages in local queue for abilities to read from other programs
if(msggrcv(local_queue, &message_response, sizeof(message_response), 0, IPC_NOWAIT) != -1)
{
    ability_got = ability_got + 1;

    cout << "Ability to read has been got from: " << message_response.sender_id << "\n";
    cout << "Ability to read has been send at: " << ctime(&message_response.response_time) <<
"\n";
}
}

// ----- OPENING AND READING THE FILE -----

cout << "----- OPEN FILE BEGIN ----- \n";
fstream local_file("lorem_ipsum.txt");
string local_string;
cout << "----- OPEN FILE END ----- \n";

cout << "----- READ FILE BEGIN ----- \n";
while(!local_file.eof() && getline(local_file, local_string))
{
    cout << local_string << "\n";
}
cout << "----- READ FILE END ----- \n\n";
local_file.close();

// ----- REQUESTS TREATMENT -----

while(message_number > 0) // all requests treatment, if they wasn't treated before

```

```

{
    message_response.sender_id = program_id;
    message_response.response_time = time(NULL);
    msgsnd(message_request_receive[message_number - 1].local_queue_id, &message_response,
sizeof(message_response), 0);
    ability_sended = ability_sended + 1;
    cout << "Sending ability to read for " << message_request_receive[message_number - 1].sender_id
<< "\n";

    message_number = message_number - 1;
}

while(ability_sended < 2) // if other program sended request before checking common queue
{
    if(msgrcv(common_queue, &message_request_receive[0], sizeof(message_request_receive[0]),
program_id, IPC_NOWAIT) != -1) // checking messages from common queue
    {
        message_response.sender_id = program_id;
        message_response.response_time = time(NULL);
        msgsnd(message_request_receive[0].local_queue_id, &message_response,
sizeof(message_response), 0);
        ability_sended = ability_sended + 1;
        cout << "Sending ability to read for " << message_request_receive[0].sender_id << "\n";
    }
}

message_request.receiver_id = 4; // message type -- 4
message_request.request_time = time(NULL);
message_request.local_queue_id = local_queue;
message_request.sender_id = program_id;
msgsnd(common_queue, &message_request, sizeof(message_request), 0); // sending ready signal to delete
common queue

// ----- CLEANING & TERMINATING -----

if(common_queue_owner == true) // waiting till other processes will finish
{
    while(is_finished < 3)
    {

```

```

        if(msggrcv(common_queue, &message_request_send[0], sizeof(message_request_send[0]), 4,
0) != -1)
        {
            is_finished = is_finished + 1;
        }
    }
    msgctl(common_queue, IPC_RMID, 0); // deleting common queue
}

msgctl(local_queue, IPC_RMID, 0); // deleting local queue
return 0;
}

void sendingRequest (MessageRequest *local_buffer, int local_other_program_id, int local_program_id, int
local_local_queue, int local_common_queue)
{
    local_buffer[local_other_program_id].receiver_id = local_other_program_id;
    local_buffer[local_other_program_id].request_time = time(NULL);
    local_buffer[local_other_program_id].local_queue_id = local_local_queue;
    local_buffer[local_other_program_id].sender_id = local_program_id;
    // (common queue, message, message real size, flags)
    msgsnd(local_common_queue,                                &local_buffer[local_other_program_id],
sizeof(local_buffer[local_other_program_id]), 0);
    cout << "Request to read has been send to: " << local_buffer[local_other_program_id].receiver_id << "\n";
    cout << "Request to read has been send at: " << ctime(&local_buffer[local_other_program_id].request_time)
<< "\n";
}

```

2.2. executable_2.cpp

```
// executable 2
// start program
// ./executable 2

#include <iostream>
#include <fstream>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <errno.h>
#include <cstring>
#include <string>

using namespace std;

// https://www.opennet.ru/man.shtml?topic=msgsnd&category=2&russian=0
// https://www.opennet.ru/man.shtml?topic=msgrcv&category=2&russian=0

typedef struct // order is very important
{
    long receiver_id; // program-receiver id
    int sender_id;    // program-sender id
    int local_queue_id; // local queue id
    time_t request_time; // request sending time
} MessageRequest;

typedef struct
{
    int sender_id; // program-sender id
    time_t response_time; // response time of program, who got request
} MessageResponse;

void sendingRequest (MessageRequest *local_buffer, int local_other_program_id, int local_program_id, int
local_local_queue, int local_common_queue);

int main(int argc, char *argv[])
```

```

{

bool common_queue_owner; // owner of common queue


int local_queue = 0; // local queue
int common_queue = 0; // common queue
int ability_got = 0; // ability to read got counter
int ability_sended = 0; // ability to read sended counter
int is_finished = 0; // number of program, who finished reading file
int other_first_program_id = 0; // other program id
int other_second_program_id = 0; // other program id
int program_id = 2; // this program id
int message_number = 0; // array number of recieved message


MessageResponse message_response; // message response to send to other programs
MessageRequest message_request_receive[2]; // message request to receiving from other programs
MessageRequest message_request; // message request
MessageRequest message_request_send[4]; // message request to sending to other programs


cout << "----- PROGRAM NUMBER " << program_id << " -----\\n";


// ----- CREATING/OPENING COMMON QUEUE -----


// IPC_CREAT -- if there wasn't queue, it will be created
// O_EXCL + IPC_CREAT -- if there was queue, msgget will return error
common_queue = msgget(190, 0606 | IPC_CREAT | IPC_EXCL); // trying to create common queue
// 190 -- key for identification, 0606 -- r&w for owner and others


// checking if common queue has been created
if (common_queue != -1) // if we created common queue, write message
{
    common_queue_owner = true;
    cout << "----- COMMON QUEUE HAS BEEN CREATED -----\\n";
}
else // if we hasn't been created common queue, try to open, write message
{
    common_queue = msgget(190, IPC_CREAT); // trying to open common queue
    if (common_queue == -1) // if we couldn't open, write message & terminate program

```

```

    {
        cout << "----- COMMON QUEUE HAS NOT BEEN OPENED -----\\n";
        exit(-1);
    }
    else // if we can open, write message
    {
        cout << "----- COMMON QUEUE HAS BEEN OPENED -----\\n";
    }
}

// ----- CREATING LOCAL QUEUE -----

local_queue = msgget(IPC_PRIVATE, 0606 | IPC_CREAT); // creating local queue

// checking if local queue has been created or not
if (local_queue == -1) // if not created -- delete remaining object if it has been created & print message
{
    cout << "----- LOCAL QUEUE HAS NOT BEEN CREATED -----\\n\\n";
    if (common_queue_owner == true) // deleting local queue if there is remaining object
    {
        // if we are owner of the common queue, delete it (IPC_RMID means delete queue, alarm all
processes & throw an error)
        msgctl(common_queue, IPC_RMID, NULL);
    }
    exit(-1); // terminate program
}
else // if created -- print message
{
    cout << "----- LOCAL QUEUE HAS BEEN CREATED -----\\n\\n";
}

// ----- SENDING REQUESTS FOR ABILITY TO READ TO OTHER PROGRAMS -----

other_first_program_id = (program_id) % 3 + 1;
other_second_program_id = (program_id + 1) % 3 + 1;

sendingRequest (message_request_send, other_first_program_id, program_id, local_queue, common_queue);

```

```

        sendingRequest    (message_request_send,    other_second_program_id,    program_id,    local_queue,
common_queue);

// ----- GETTING REQUESTS FOR ABILITY AND ABILITIES TO READ FROM OTHER
PROGRAMS -----

while(ability_got < 2) // when we have not got abilities to read from 2 other programs
{
    if(msgrcv(common_queue,                                &message_request_receive[message_number],
sizeof(message_request_receive[message_number]), program_id, IPC_NOWAIT) != -1) // common queue message
check
    {
        cout    <<    "Request    to    read    has    been    got    from:    "    <<
message_request_receive[message_number].sender_id << "\n";
        cout    <<    "Request    to    read    has    been    send    at:    "    <<
ctime(&message_request_receive[message_number].request_time) << "\n";

        // if the request TIME for ability to read from OTHER program <= request TIME for ability
to read from THIS program,
        // THIS program sends the ability to read to OTHER program (< || (= & id_sender < id_this))
        if((message_request_receive[message_number].request_time
            <
message_request_send[message_request_receive[message_number].sender_id].request_time)
            || (message_request_receive[message_number].request_time
                ==
message_request_send[message_request_receive[message_number].sender_id].request_time
                && message_request_receive[message_number].sender_id < program_id))
        {
            message_response.sender_id = program_id;
            message_response.response_time = time(NULL);
            msgsnd(message_request_receive[message_number].local_queue_id,
&message_response, sizeof(message_response), 0);
            ability_sended = ability_sended + 1;

            cout    <<    "Sending    ability    to    read    to:    "    <<
message_request_receive[message_number].sender_id << "\n\n";
        }
        else // else, untreated request will be placed to "message_request_receive" array

```

```

        {
            message_number = message_number + 1;
        }
    }
    // check messages in local queue for abilities to read from other programs
    if(msgrcv(local_queue, &message_response, sizeof(message_response), 0, IPC_NOWAIT) != -1)
    {
        ability_got = ability_got + 1;

        cout << "Ability to read has been got from: " << message_response.sender_id << "\n";
        cout << "Ability to read has been send at: " << ctime(&message_response.response_time) <<
"\n";
    }
}

// ----- OPENING AND READING THE FILE -----

cout << "----- OPEN FILE BEGIN ----- \n";
fstream local_file("lorem_ipsum.txt");
string local_string;
cout << "----- OPEN FILE END ----- \n";

cout << "----- READ FILE BEGIN ----- \n";
while(!local_file.eof() && getline(local_file, local_string))
{
    cout << local_string << "\n";
}
cout << "----- READ FILE END ----- \n\n";
local_file.close();

// ----- REQUESTS TREATMENT -----

while(message_number > 0) // all requests treatment, if they wasn't treated before
{
    message_response.sender_id = program_id;
    message_response.response_time = time(NULL);

```



```

        msgsnd(message_request_receive[message_number - 1].local_queue_id, &message_response,
sizeof(message_response), 0);
        ability_sended = ability_sended + 1;
        cout << "Sending ability to read for " << message_request_receive[message_number - 1].sender_id
<< "\n";

        message_number = message_number - 1;
    }

    while(ability_sended < 2) // if other program sended request before checking common queue
    {
        if(msgrcv(common_queue, &message_request_receive[0], sizeof(message_request_receive[0]),
program_id, IPC_NOWAIT) != -1) // checking messages from common queue
        {
            message_response.sender_id = program_id;
            message_response.response_time = time(NULL);
            msgsnd(message_request_receive[0].local_queue_id, &message_response,
sizeof(message_response), 0);
            ability_sended = ability_sended + 1;
            cout << "Sending ability to read for " << message_request_receive[0].sender_id << "\n";
        }
    }

    message_request.receiver_id = 4; // message type -- 4
    message_request.request_time = time(NULL);
    message_request.local_queue_id = local_queue;
    message_request.sender_id = program_id;
    msgsnd(common_queue, &message_request, sizeof(message_request), 0); // sending ready signal to delete
common queue

    // ----- CLEANING & TERMINATING -----

    if(common_queue_owner == true) // waiting till other processes will finish
    {
        while(is_finished < 3)
        {
            if(msgrcv(common_queue, &message_request_send[0], sizeof(message_request_send[0]), 4,
0) != -1)
            {

```

```

        is_finished = is_finished + 1;
    }
}

msgctl(common_queue, IPC_RMID, 0); // deleting common queue
}

msgctl(local_queue, IPC_RMID, 0); // deleting local queue
return 0;
}

void sendingRequest (MessageRequest *local_buffer, int local_other_program_id, int local_program_id, int
local_local_queue, int local_common_queue)
{
    local_buffer[local_other_program_id].receiver_id = local_other_program_id;
    local_buffer[local_other_program_id].request_time = time(NULL);
    local_buffer[local_other_program_id].local_queue_id = local_local_queue;
    local_buffer[local_other_program_id].sender_id = local_program_id;
    // (common queue, message, message real size, flags)
    msgsnd(local_common_queue,
                                                    &local_buffer[local_other_program_id],
sizeof(local_buffer[local_other_program_id]), 0);
    cout << "Request to read has been send to: " << local_buffer[local_other_program_id].receiver_id << "\n";
    cout << "Request to read has been send at: " << ctime(&local_buffer[local_other_program_id].request_time)
<< "\n";
}

```

2.3. executable_3.cpp

```
// executable 3
// start program
// ./executable 3

#include <iostream>
#include <fstream>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <errno.h>
#include <cstring>
#include <string>

using namespace std;

// https://www.opennet.ru/man.shtml?topic=msgsnd&category=2&russian=0
// https://www.opennet.ru/man.shtml?topic=msgrcv&category=2&russian=0

typedef struct // order is very important
{
    long receiver_id; // program-receiver id
    int sender_id;    // program-sender id
    int local_queue_id; // local queue id
    time_t request_time; // request sending time
} MessageRequest;

typedef struct
{
    int sender_id; // program-sender id
    time_t response_time; // response time of program, who got request
} MessageResponse;

void sendingRequest (MessageRequest *local_buffer, int local_other_program_id, int local_program_id, int
local_local_queue, int local_common_queue);

int main(int argc, char *argv[])
```

```

{

bool common_queue_owner; // owner of common queue


int local_queue = 0; // local queue
int common_queue = 0; // common queue
int ability_got = 0; // ability to read got counter
int ability_sended = 0; // ability to read sended counter
int is_finished = 0; // number of program, who finished reading file
int other_first_program_id = 0; // other program id
int other_second_program_id = 0; // other program id
int program_id = 3; // this program id
int message_number = 0; // array number of recieved message


MessageResponse message_response; // message response to send to other programs
MessageRequest message_request_receive[2]; // message request to receiving from other programs
MessageRequest message_request; // message request
MessageRequest message_request_send[4]; // message request to sending to other programs


cout << "----- PROGRAM NUMBER " << program_id << " -----\\n";


// ----- CREATING/OPENING COMMON QUEUE -----


// IPC_CREAT -- if there wasn't queue, it will be created
// O_EXCL + IPC_CREAT -- if there was queue, msgget will return error
common_queue = msgget(190, 0606 | IPC_CREAT | IPC_EXCL); // trying to create common queue
// 190 -- key for identification, 0606 -- r&w for owner and others


// checking if common queue has been created
if (common_queue != -1) // if we created common queue, write message
{
    common_queue_owner = true;
    cout << "----- COMMON QUEUE HAS BEEN CREATED -----\\n";
}
else // if we hasn't been created common queue, try to open, write message
{
    common_queue = msgget(190, IPC_CREAT); // trying to open common queue
    if (common_queue == -1) // if we couldn't open, write message & terminate program

```

```

    {
        cout << "----- COMMON QUEUE HAS NOT BEEN OPENED -----\\n";
        exit(-1);
    }
    else // if we can open, write message
    {
        cout << "----- COMMON QUEUE HAS BEEN OPENED -----\\n";
    }
}

// ----- CREATING LOCAL QUEUE -----

local_queue = msgget(IPC_PRIVATE, 0606 | IPC_CREAT); // creating local queue

// checking if local queue has been created or not
if (local_queue == -1) // if not created -- delete remaining object if it has been created & print message
{
    cout << "----- LOCAL QUEUE HAS NOT BEEN CREATED -----\\n\\n";
    if (common_queue_owner == true) // deleting local queue if there is remaining object
    {
        // if we are owner of the common queue, delete it (IPC_RMID means delete queue, alarm all
        // processes & throw an error)
        msgctl(common_queue, IPC_RMID, NULL);
    }
    exit(-1); // terminate program
}
else // if created -- print message
{
    cout << "----- LOCAL QUEUE HAS BEEN CREATED -----\\n\\n";
}

// ----- SENDING REQUESTS FOR ABILITY TO READ TO OTHER PROGRAMS -----

other_first_program_id = (program_id) % 3 + 1;
other_second_program_id = (program_id + 1) % 3 + 1;

sendingRequest (message_request_send, other_first_program_id, program_id, local_queue, common_queue);

```

```

        sendingRequest    (message_request_send,    other_second_program_id,    program_id,    local_queue,
common_queue);

// ----- GETTING REQUESTS FOR ABILITY AND ABILITIES TO READ FROM OTHER
PROGRAMS -----

while(ability_got < 2) // when we have not got abilities to read from 2 other programs
{
    if(msgrcv(common_queue,                                &message_request_receive[message_number],
sizeof(message_request_receive[message_number]), program_id, IPC_NOWAIT) != -1) // common queue message
check
    {
        cout    <<    "Request    to    read    has    been    got    from:    "    <<
message_request_receive[message_number].sender_id << "\n";
        cout    <<    "Request    to    read    has    been    send    at:    "    <<
ctime(&message_request_receive[message_number].request_time) << "\n";

        // if the request TIME for ability to read from OTHER program <= request TIME for ability
to read from THIS program,
        // THIS program sends the ability to read to OTHER program (< || (= & id_sender < id_this))
        if((message_request_receive[message_number].request_time
            <
message_request_send[message_request_receive[message_number].sender_id].request_time)
            || (message_request_receive[message_number].request_time
                ==
message_request_send[message_request_receive[message_number].sender_id].request_time
                && message_request_receive[message_number].sender_id < program_id))
        {
            message_response.sender_id = program_id;
            message_response.response_time = time(NULL);
            msgsnd(message_request_receive[message_number].local_queue_id,
&message_response, sizeof(message_response), 0);
            ability_sended = ability_sended + 1;

            cout    <<    "Sending    ability    to    read    to:    "    <<
message_request_receive[message_number].sender_id << "\n\n";
        }
        else // else, untreated request will be placed to "message_request_receive" array

```

```

        {
            message_number = message_number + 1;
        }
    }
    // check messages in local queue for abilities to read from other programs
    if(msgrcv(local_queue, &message_response, sizeof(message_response), 0, IPC_NOWAIT) != -1)
    {
        ability_got = ability_got + 1;

        cout << "Ability to read has been got from: " << message_response.sender_id << "\n";
        cout << "Ability to read has been send at: " << ctime(&message_response.response_time) <<
"\n";
    }
}

// ----- OPENING AND READING THE FILE -----

cout << "----- OPEN FILE BEGIN ----- \n";
fstream local_file("lorem_ipsum.txt");
string local_string;
cout << "----- OPEN FILE END ----- \n";

cout << "----- READ FILE BEGIN ----- \n";
while(!local_file.eof() && getline(local_file, local_string))
{
    cout << local_string << "\n";
}
cout << "----- READ FILE END ----- \n\n";
local_file.close();

// ----- REQUESTS TREATMENT -----

while(message_number > 0) // all requests treatment, if they wasn't treated before
{
    message_response.sender_id = program_id;
    message_response.response_time = time(NULL);

```

```

        msgsnd(message_request_receive[message_number - 1].local_queue_id, &message_response,
sizeof(message_response), 0);
        ability_sended = ability_sended + 1;
        cout << "Sending ability to read for " << message_request_receive[message_number - 1].sender_id
<< "\n";

        message_number = message_number - 1;
    }

    while(ability_sended < 2) // if other program sended request before checking common queue
    {
        if(msgrcv(common_queue, &message_request_receive[0], sizeof(message_request_receive[0]),
program_id, IPC_NOWAIT) != -1) // checking messages from common queue
        {
            message_response.sender_id = program_id;
            message_response.response_time = time(NULL);
            msgsnd(message_request_receive[0].local_queue_id, &message_response,
sizeof(message_response), 0);
            ability_sended = ability_sended + 1;
            cout << "Sending ability to read for " << message_request_receive[0].sender_id << "\n";
        }
    }

    message_request.receiver_id = 4; // message type -- 4
    message_request.request_time = time(NULL);
    message_request.local_queue_id = local_queue;
    message_request.sender_id = program_id;
    msgsnd(common_queue, &message_request, sizeof(message_request), 0); // sending ready signal to delete
common queue

    // ----- CLEANING & TERMINATING -----

    if(common_queue_owner == true) // waiting till other processes will finish
    {
        while(is_finished < 3)
        {
            if(msgrcv(common_queue, &message_request_send[0], sizeof(message_request_send[0]), 4,
0) != -1)
            {

```



```

        is_finished = is_finished + 1;
    }
}

msgctl(common_queue, IPC_RMID, 0); // deleting common queue
}

msgctl(local_queue, IPC_RMID, 0); // deleting local queue
return 0;
}

void sendingRequest (MessageRequest *local_buffer, int local_other_program_id, int local_program_id, int
local_local_queue, int local_common_queue)
{
    local_buffer[local_other_program_id].receiver_id = local_other_program_id;
    local_buffer[local_other_program_id].request_time = time(NULL);
    local_buffer[local_other_program_id].local_queue_id = local_local_queue;
    local_buffer[local_other_program_id].sender_id = local_program_id;
    // (common queue, message, message real size, flags)
    msgsnd(local_common_queue,                                &local_buffer[local_other_program_id],
sizeof(local_buffer[local_other_program_id]), 0);
    cout << "Request to read has been send to: " << local_buffer[local_other_program_id].receiver_id << "\n";
    cout << "Request to read has been send at: " << ctime(&local_buffer[local_other_program_id].request_time)
<< "\n";
}

```

3. Скриншоты работы каждой программы

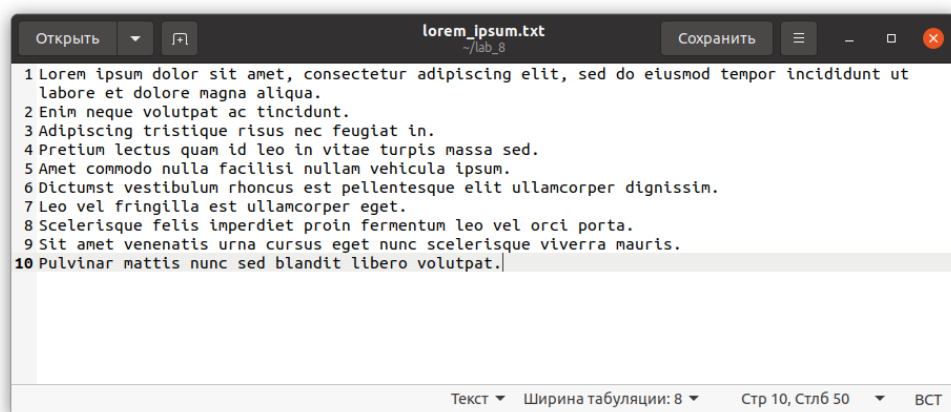


Рисунок 1. Читаемый программами файл «lorem_ipsum.txt», хранящийся в той же директории, что и исполняемые файлы

```
matmanbj@matmanbj-VirtualBox: ~/lab_8
matmanbj@matmanbj-VirtualBox:~/lab_8$ ./executable_1
----- PROGRAM NUMBER 1 -----
----- COMMON QUEUE HAS BEEN CREATED -----
----- LOCAL QUEUE HAS BEEN CREATED -----

Request to read has been send to: 2
Request to read has been send at: Tue Nov 15 23:05:36 2022

Request to read has been send to: 3
Request to read has been send at: Tue Nov 15 23:05:36 2022

Request to read has been got from: 2
Request to read has been send at: Tue Nov 15 23:05:38 2022

Ability to read has been got from: 2
Ability to read has been send at: Tue Nov 15 23:05:38 2022

Request to read has been got from: 3
Request to read has been send at: Tue Nov 15 23:05:41 2022

Ability to read has been got from: 3
Ability to read has been send at: Tue Nov 15 23:05:41 2022

----- OPEN FILE BEGIN -----
```

Рисунок 2. Запуск программы «executable_1»

```
matmanbj@matmanbj-VirtualBox: ~/lab_8
Request to read has been send at: Tue Nov 15 23:05:41 2022

Ability to read has been got from: 3
Ability to read has been send at: Tue Nov 15 23:05:41 2022

----- OPEN FILE BEGIN -----
----- OPEN FILE END -----
----- READ FILE BEGIN -----
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor i
ncididunt ut labore et dolore magna aliqua.
Enim neque volutpat ac tincidunt.
Adipiscing tristique risus nec feugiat in.
Pretium lectus quam id leo in vitae turpis massa sed.
Amet commodo nulla facilisi nullam vehicula ipsum.
Dictumst vestibulum rhoncus est pellentesque elit ullamcorper dignissim.
Leo vel fringilla est ullamcorper eget.
Scelerisque felis imperdiet proin fermentum leo vel orci porta.
Sit amet venenatis urna cursus eget nunc scelerisque viverra mauris.
Pulvinar mattis nunc sed blandit libero volutpat.
----- READ FILE END -----

Sending ability to read for 3
Sending ability to read for 2
matmanbj@matmanbj-VirtualBox:~/lab_8$
```

Рисунок 3. Запуск программы «executable_1»

```
matmanbj@matmanbj-VirtualBox: ~/lab_8
matmanbj@matmanbj-VirtualBox:~/lab_8$ ./executable_2
----- PROGRAM NUMBER 2 -----
----- COMMON QUEUE HAS BEEN OPENED -----
----- LOCAL QUEUE HAS BEEN CREATED -----

Request to read has been send to: 3
Request to read has been send at: Tue Nov 15 23:05:38 2022

Request to read has been send to: 1
Request to read has been send at: Tue Nov 15 23:05:38 2022

Request to read has been got from: 1
Request to read has been send at: Tue Nov 15 23:05:36 2022

Sending ability to read to: 1

Ability to read has been got from: 3
Ability to read has been send at: Tue Nov 15 23:05:41 2022

Request to read has been got from: 3
Request to read has been send at: Tue Nov 15 23:05:41 2022

Ability to read has been got from: 1
Ability to read has been send at: Tue Nov 15 23:05:41 2022
```

Рисунок 4. Запуск программы «executable_2»

```
matmanbj@matmanbj-VirtualBox: ~/lab_8
Request to read has been got from: 3
Request to read has been send at: Tue Nov 15 23:05:41 2022

Ability to read has been got from: 1
Ability to read has been send at: Tue Nov 15 23:05:41 2022

----- OPEN FILE BEGIN -----
----- OPEN FILE END -----
----- READ FILE BEGIN -----
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor i
ncidunt ut labore et dolore magna aliqua.
Enim neque volutpat ac tincidunt.
Adipiscing tristique risus nec feugiat in.
Pretium lectus quam id leo in vitae turpis massa sed.
Amet commodo nulla facilisi nullam vehicula ipsum.
Dictumst vestibulum rhoncus est pellentesque elit ullamcorper dignissim.
Leo vel fringilla est ullamcorper eget.
Scelerisque felis imperdiet proin fermentum leo vel orci porta.
Sit amet venenatis urna cursus eget nunc scelerisque viverra mauris.
Pulvinar mattis nunc sed blandit libero volutpat.
----- READ FILE END -----

Sending ability to read for 3
matmanbj@matmanbj-VirtualBox:~/lab_8$
```

Рисунок 5. Запуск программы «executable_2»

```
matmanbj@matmanbj-VirtualBox: ~/lab_8
matmanbj@matmanbj-VirtualBox:~/lab_8$ ./executable_3
----- PROGRAM NUMBER 3 -----
----- COMMON QUEUE HAS BEEN OPENED -----
----- LOCAL QUEUE HAS BEEN CREATED -----

Request to read has been send to: 1
Request to read has been send at: Tue Nov 15 23:05:41 2022

Request to read has been send to: 2
Request to read has been send at: Tue Nov 15 23:05:41 2022

Request to read has been got from: 1
Request to read has been send at: Tue Nov 15 23:05:36 2022

Sending ability to read to: 1

Request to read has been got from: 2
Request to read has been send at: Tue Nov 15 23:05:38 2022

Sending ability to read to: 2

Ability to read has been got from: 1
Ability to read has been send at: Tue Nov 15 23:05:41 2022
```

Рисунок 6. Запуск программы «executable_3»

```
matmanbj@matmanbj-VirtualBox: ~/lab_8
Ability to read has been got from: 1
Ability to read has been send at: Tue Nov 15 23:05:41 2022

Ability to read has been got from: 2
Ability to read has been send at: Tue Nov 15 23:05:41 2022

----- OPEN FILE BEGIN -----
----- OPEN FILE END -----
----- READ FILE BEGIN -----
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor i
ncididunt ut labore et dolore magna aliqua.
Enim neque volutpat ac tincidunt.
Adipiscing tristique risus nec feugiat in.
Pretium lectus quam id leo in vitae turpis massa sed.
Amet commodo nulla facilisi nullam vehicula ipsum.
Dictumst vestibulum rhoncus est pellentesque elit ullamcorper dignissim.
Leo vel fringilla est ullamcorper eget.
Scelerisque felis imperdiet proin fermentum leo vel orci porta.
Sit amet venenatis urna cursus eget nunc scelerisque viverra mauris.
Pulvinar mattis nunc sed blandit libero volutpat.
----- READ FILE END -----
matmanbj@matmanbj-VirtualBox:~/lab_8$
```

Рисунок 7. Запуск программы «executable_3»

4. Вывод

В ходе выполнения лабораторной работы №8 «Взаимодействие процессов на основе сообщений» были изучены системные функции, отвечающие за создание или открытие очереди («msgget»), в том числе локальной (с ключом «IPC_PRIVATE») и общей (с ненулевым целым ключом), за удаление очереди («msgctl»), за получения сообщений из очереди («msgrcv») и за их отправку («msgsnd»). Также были изучены структуры, отвечающие за запрос сообщения и за ответ на сообщение. Таким образом и было произведено знакомство с механизмом обмена сообщениями и системными вызовами приёма и передачи сообщений.

5. Список использованных источников

1. Онлайн-курс «Организация процессов и программирование в среде Linux» в LMS Moodle [сайт]. URL: <https://vec.etu.ru/moodle/course/view.php?id=9703>.

2. Разумовский Г.В. Организация процессов и программирование в среде Linux: учебно-методическое пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2018. 40с.