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

ОТЧЁТ

по лабораторной работе №8

по дисциплине «Организация процессов и программирования в среде Linux»

1. Тема: ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ НА ОСНОВЕ СООБЩЕНИЙ

|  |  |  |
| --- | --- | --- |
| Студент гр. 9308 |  | Соболев М.С. |
| Преподаватель |  | Разумовский Г.В. |

Оглавление

[1. Введение 3](#__RefHeading___Toc336_311249038)

[1.1. Введение 3](#__RefHeading___Toc15338_3795591338)

[1.2. Порядок выполнения работы 3](#__RefHeading___Toc15340_3795591338)

[1.3. Содержание отчёта 4](#__RefHeading___Toc15342_3795591338)

[2. Тексты программ 5](#__RefHeading___Toc2036_716552913)

[2.1. executable\_1.cpp 5](#__RefHeading___Toc2038_716552913)

[2.2. executable\_2.cpp 12](#__RefHeading___Toc2040_716552913)

[2.3. executable\_3.cpp 19](#__RefHeading___Toc2042_716552913)

[3. Скриншоты работы каждой программы 26](#__RefHeading___Toc2044_716552913)

[4. Вывод 30](#__RefHeading___Toc358_311249038)

[5. Список использованных источников 31](#__RefHeading___Toc360_311249038)

# 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 прочитать 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; // responce 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 responce 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(200, 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 -- pring 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";

}

## 2.2. 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; // responce 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 responce 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(200, 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 -- pring 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; // responce 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 responce 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(200, 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 -- pring 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

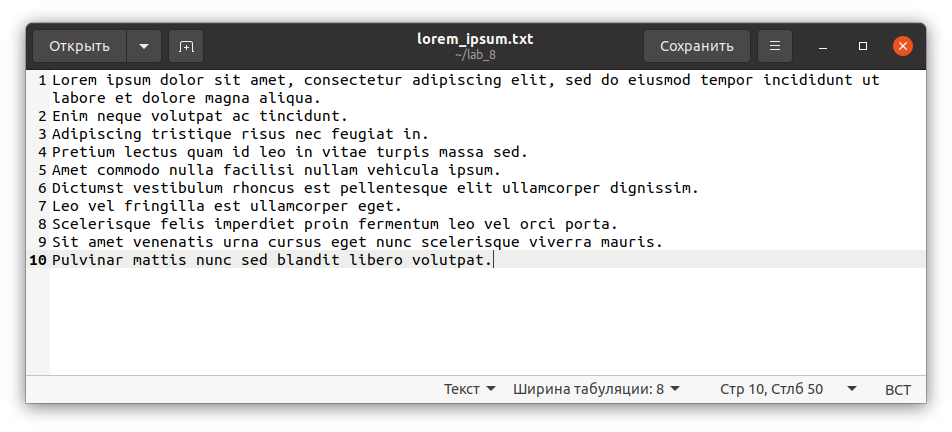


Рисунок 1. Читаемый программами файл «lorem\_ipsum.txt», хранящийся в той же директории, что и исполняемые файлы

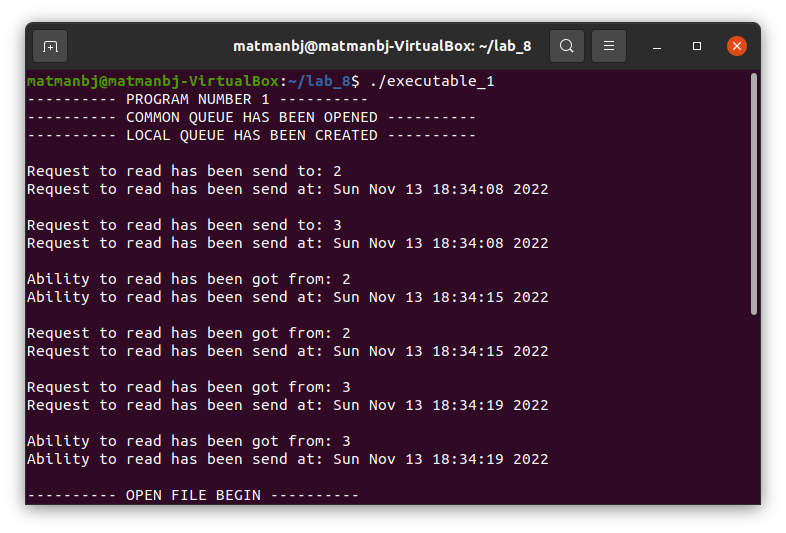


Рисунок 2. Запуск программы «executable\_1»

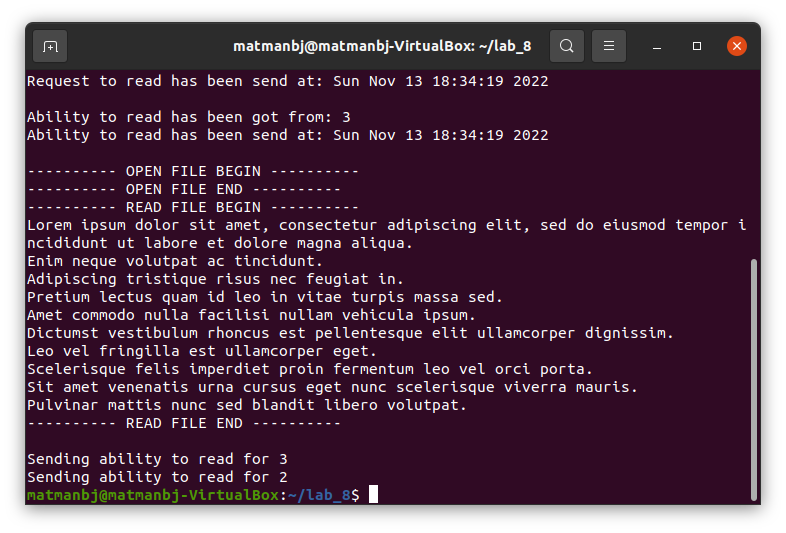


Рисунок 3. Запуск программы «executable\_1»

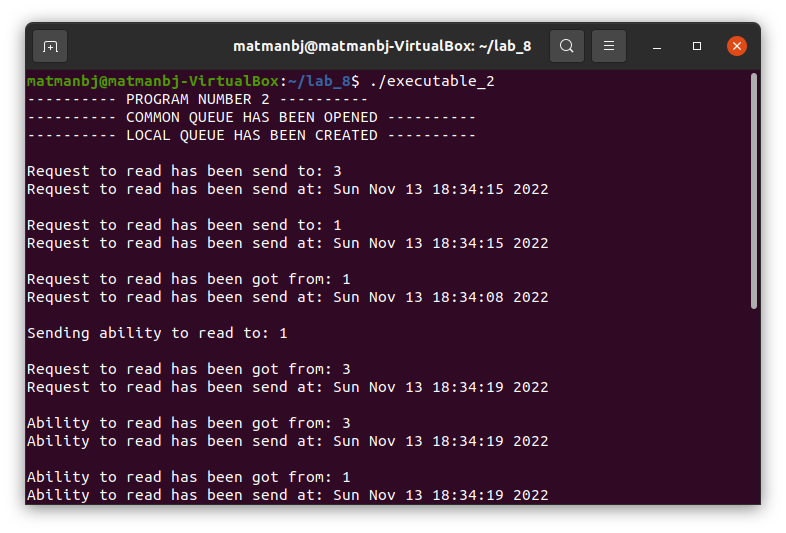


Рисунок 4. Запуск программы «executable\_2»

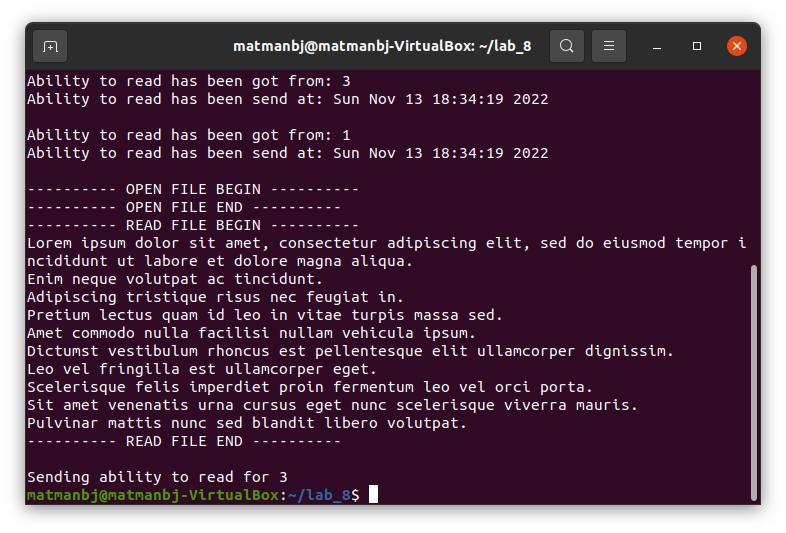


Рисунок 5. Запуск программы «executable\_2»

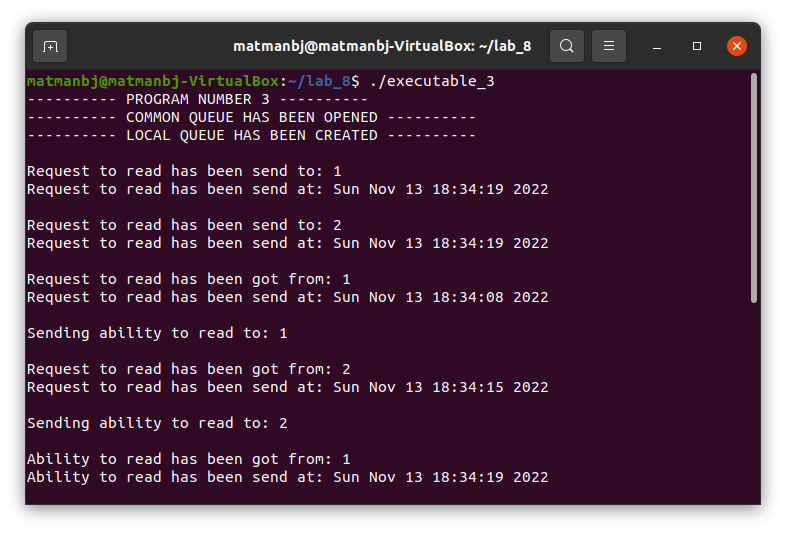


Рисунок 6. Запуск программы «executable\_3»

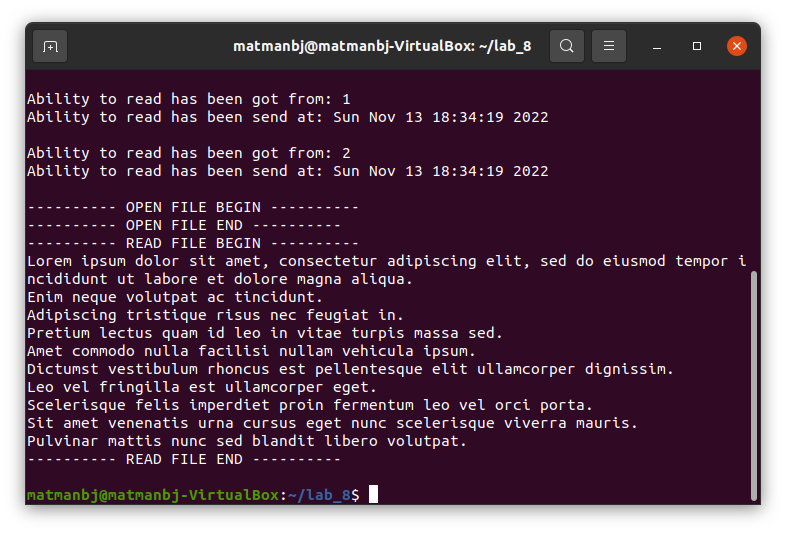


Рисунок 7. Запуск программы «executable\_3»

# 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.