**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра вычислительной техники**

отчет

**по лабораторной работе №8**

**по дисциплине «Организация процессов и программирование в среде Linux»**

Темы: ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ НА ОСНОВЕ СООБЩЕНИЙ

|  |  |  |
| --- | --- | --- |
| Студент гр. 8306 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Пеунов В.В. |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Разумовский Г.В. |

Санкт-Петербург,

2021

**Цель работы**

Знакомство с механизмом обмена сообщениями и системными вызовами приема и передачи сообщений.

**Задание на лабораторную работу**

1. 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:  если программа прочитала файл, то сразу передается ответ, который должен содержать номер отвечающей программы и время ответа;  если файл не читался, то ответ передается 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.

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

**Описание работы**

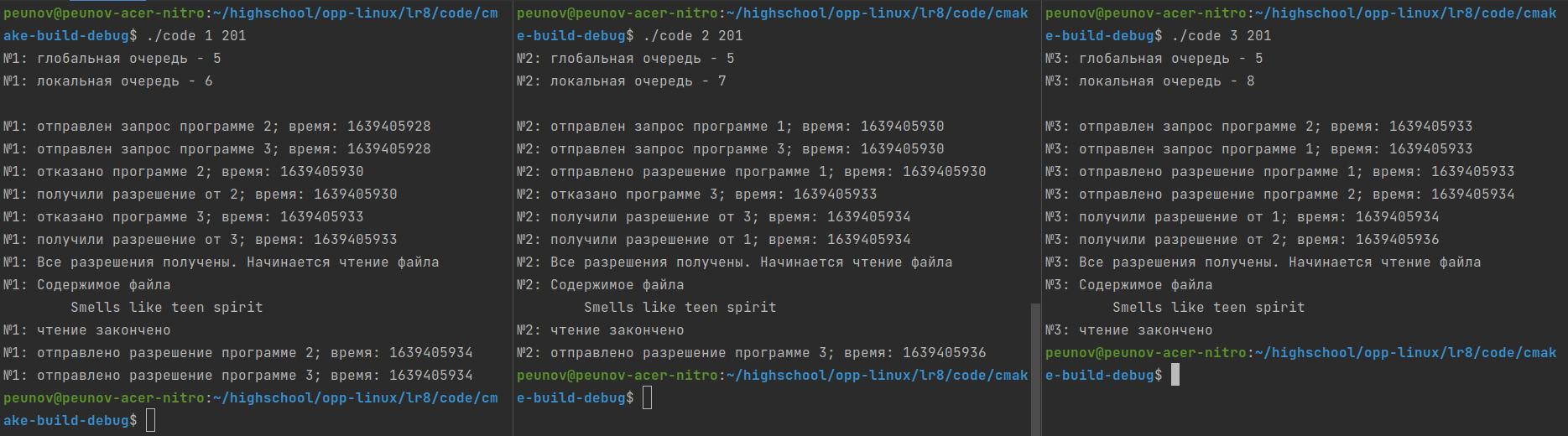
Скриншот работы программы представлен на рисунке 1.

Рисунок 1.

**Вывод:** в ходе лабораторной работы осуществлено знакомство с механизмом обмена сообщениями и системными вызовами приема и передачи сообщений.

**Приложение A. Основная программа**

#include <stdio.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <errno.h>

#include <time.h>

#include <unistd.h>

#include <iostream>

#include <fstream>

using namespace std;

typedef struct Message\_T {

long type; //получатель

int sender; // отправитель

int responseQue; //очередь, куда отправить ответ

time\_t time; // создания запроса

bool reply;

} Message;

int programNumber;

bool owner = false;

int QUE\_KEY = 333;

string filePath = "/home/peunov/highschool/opp-linux/lr8/files/input";

Message globalMessages[2]; // здесь собираются запросы от других файлов

int getNeighbor(bool next){

if (programNumber == 1 && next) return 2;

if (programNumber == 1 && !next) return 3;

if (programNumber == 2 && next) return 1;

if (programNumber == 2 && !next) return 3;

if (programNumber == 3 && next) return 2;

if (programNumber == 3 && !next) return 1;

}

Message createMessage(long receiver, int responseQue, time\_t time){

Message message;

message.type = receiver;

message.sender = programNumber;

message.responseQue = responseQue;

message.time = time;

message.reply = false;

return message;

}

void sendRequestOnRead(bool neighborBefore, int globalQue, int localQue, time\_t time){

int neighbor = getNeighbor(neighborBefore);

Message request = createMessage( neighbor, localQue, time);

msgsnd(globalQue, &request, sizeof(request), 0);

cout << "№" << programNumber << ": отправлен запрос программе " << neighbor

<< "; время: " << time << endl;

}

void sendPermission(int neighbor, int que, time\_t time){

Message request = createMessage(neighbor, 0, time);

msgsnd(que, &request, sizeof(request), 0);

cout << "№" << programNumber << ": отправлено разрешение программе " << neighbor

<< "; время: " << time << endl;

}

void sendPermissions(){

for(auto & globalMessage : globalMessages){

if(not globalMessage.reply){

sendPermission(globalMessage.sender, globalMessage.responseQue, time(nullptr));

}

}

}

int getOrCreateGlobalQue(){

int generalQue = msgget(QUE\_KEY, 0606|IPC\_CREAT|IPC\_EXCL);

if(generalQue == -1){

generalQue = msgget(QUE\_KEY, 0606|IPC\_CREAT); // queOwner = 0;

owner = true;

} else {

owner = false;

}

return generalQue;

}

int getLocalQue(){

return msgget(IPC\_PRIVATE, 0606|IPC\_CREAT);

}

void printProgramInfo(int globalQue, int localQue) {

cout << "№" << programNumber << ": глобальная очередь - " << globalQue << endl;

cout << "№" << programNumber << ": локальная очередь - " << localQue << endl << endl;

}

void waitPermissions(int globalQue, int localQue, time\_t requestTime){

Message permissions[2];

ssize\_t status;

int count\_permission = 0, global = 0;

while(count\_permission < 2){

// получаем сообщения из глобальной очереди

status = msgrcv(globalQue, &globalMessages[global], sizeof(globalMessages[global]), programNumber, IPC\_NOWAIT);

if(status!=-1){

// получали сообщение из глобальной очереди

if(globalMessages[global].time < requestTime){

//отправляем в ответ разрешение на чтение

sendPermission(globalMessages[global].sender, globalMessages[global].responseQue, time(nullptr));

globalMessages[global].reply = true;

} else {

cout << "№" << programNumber << ": отказано программе " << globalMessages[global].sender

<< "; время: " << globalMessages[global].time << endl;

}

global+=1;

}

// получаем разрешение

status = msgrcv(localQue, &permissions[count\_permission], sizeof(permissions[count\_permission]), 0, IPC\_NOWAIT);

if(status!=-1){

//получили разрешение

cout << "№" << programNumber << ": получили разрешение от " << permissions[count\_permission].sender

<< "; время: " << permissions[count\_permission].time << endl;

count\_permission += 1;

}

sleep(1);

}

}

void closeQue(int globalQue, int localQue){

if(owner) msgctl(globalQue, IPC\_RMID,nullptr);

msgctl(localQue,IPC\_RMID,nullptr);

}

void readFile(){

ifstream input;

string inputStr;

input.open(filePath, ios\_base::in);

cout << "№" << programNumber << ": Все разрешения получены. Начинается чтение файла" << endl;

cout << "№" << programNumber << ": Содержимое файла" << endl;

while (getline(input, inputStr)) {

cout << "\t" << inputStr << endl;

}

cout << "№" << programNumber << ": чтение закончено" << endl;

input.close();

}

void sendRequests(int globalQue, int localQue, time\_t sendTime){

sendRequestOnRead(true, globalQue, localQue, sendTime);

sendRequestOnRead(false, globalQue, localQue, sendTime);

}

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

time\_t sendTime;

// 1. Получаем данные из параметров

if(argc != 3){

cout << "Недостаточно аргументов" << endl;

return 0;

}

programNumber = atoi(argv[1]);

QUE\_KEY = atoi(argv[2]);

// 2. Создаем/получаем глобальную очередь

int globalQue = getOrCreateGlobalQue();

// 3. Создаем локальную очередь

int localQue = getLocalQue();

printProgramInfo(globalQue, localQue);

// 4. Отправляем запрос на чтение файла другим очередям

sendTime = time(nullptr);

sendRequests(globalQue, localQue, sendTime);

// 5.1. Читаем запросы с глобальной очереди

// 5.2. Если есть запрос раньше нашего, отправляем разрешение на чтение

// 5.3. Иначе нет

waitPermissions(globalQue, localQue, sendTime);

// 6. Когда получаем все разрешения - читаем файл

readFile();

// 7. Отправляем разрешения другим программам

sendPermissions();

// 8. Закрываем очереди

closeQue(globalQue, localQue);

return 0;

}