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 программированиев среде Linux»

Тема: Взаимодействие процессов на основе сообщений

Студент гр. 9308 _____ Яловега Н.В.

Преподаватель _____ Разумовский Г.В.

Санкт-Петербург

2022

Цель работы

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

Задание

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.

Примеры выполнения программы

В работе запускаются 3 программы, которые обмениваются сообщениями для синхронизации операции с чтением файла. Все программы посылают свои запросы на чтение в общую очередь сообщений. Затем программы проверяют наличие в глобальной очереди запросов на чтение от других программ. Если в очереди имеются запросы с меньшим временем отправки, программа отправляет свое разрешение в личную очередь программы, которая отправила запрос раньше. Когда программа получает разрешения от двух других программ, она приступает к чтению файла. По окончании чтения файла, программа отправляет свое разрешение другим программам, которые ранее его запрашивали.

Результаты запуска программ в последовательности program3 program1 program2 приведен на рисунке 1.

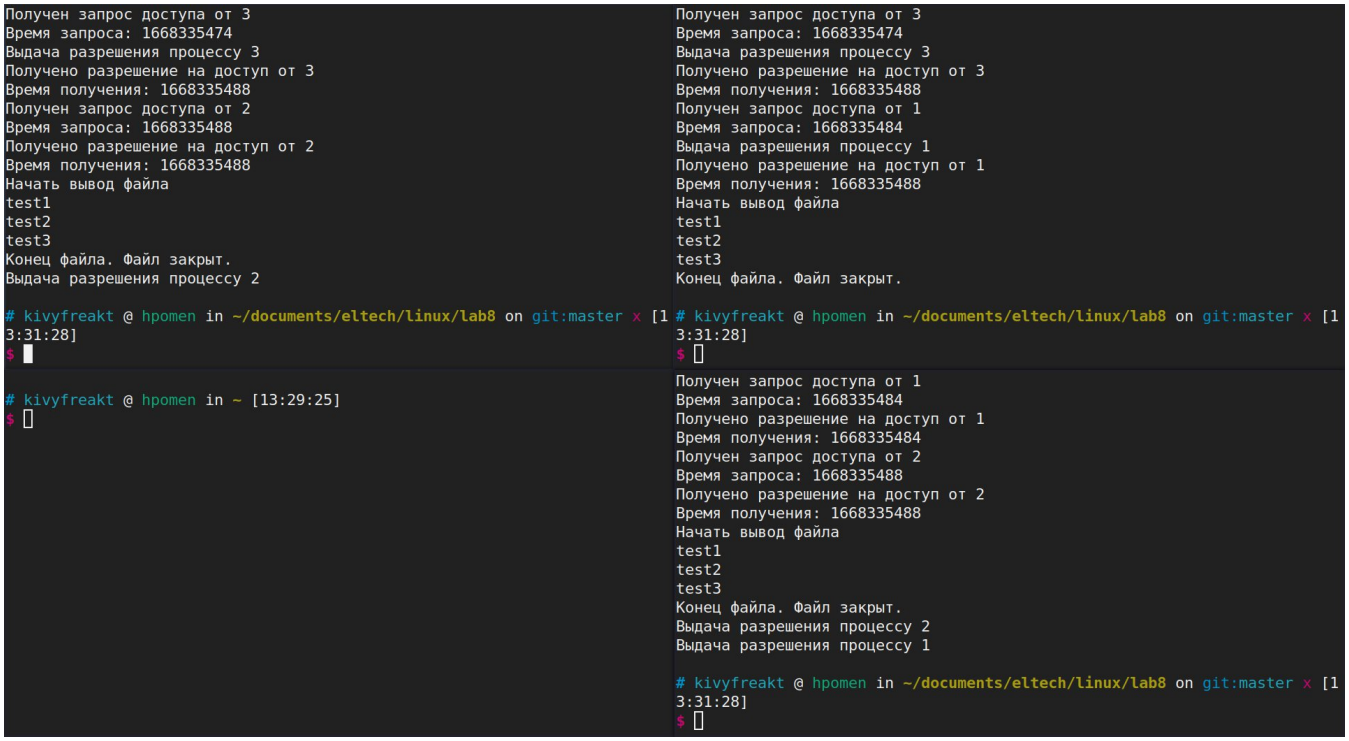

Рисунок 1. Результат выполнения программы.

Результаты запуска программ в последовательности program2 program1 program3 приведен на рисунке 1.

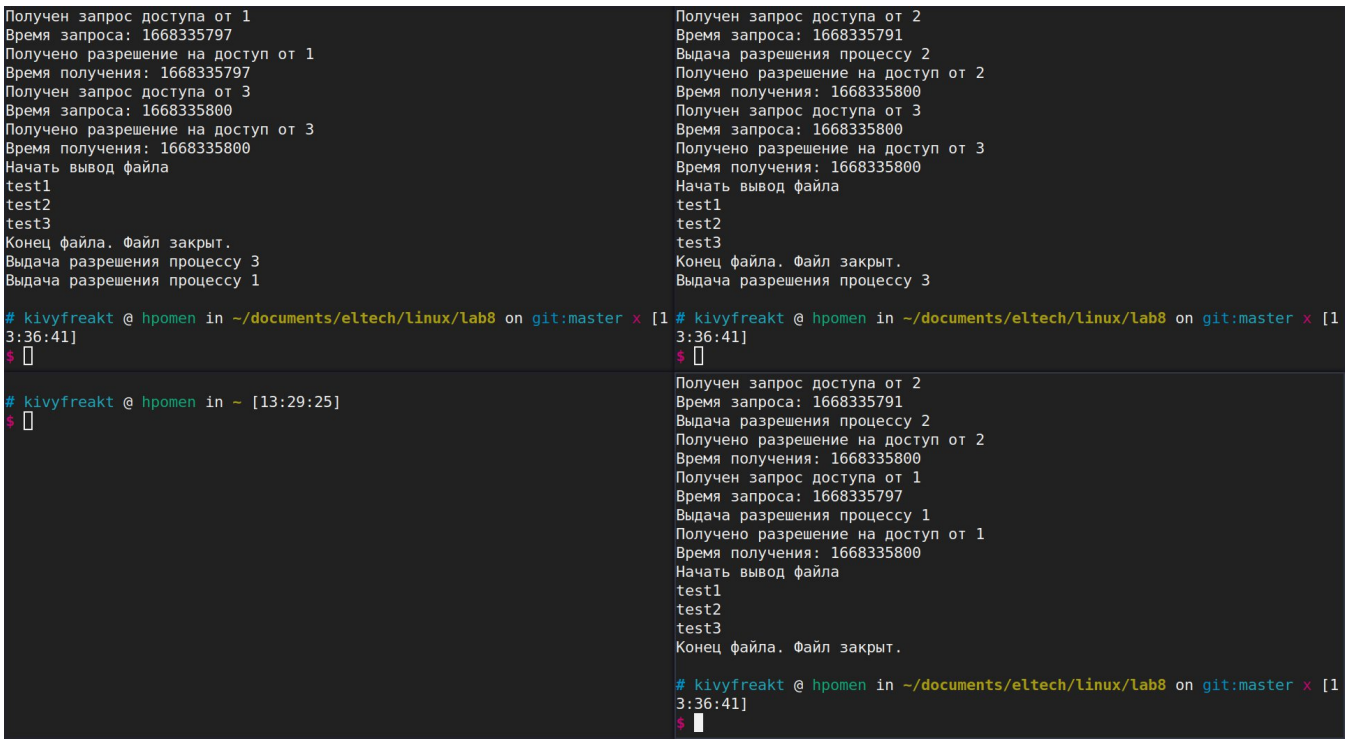

Рисунок 2. Результат выполнения программы.

Исходный код программ

```
#include <iostream>
#include <fstream>
#include <sys/msg.h>
#define PID 1
#define FNAME "file.txt"
//структура сообщений общей очереди
struct ask_msg
{
   long mtype; //тип сообщения
   int sender; //отправитель
   int answer_queue; //локальная очередь заказчика
   int ask_time; //время запроса
};
//структура сообщений локальной очереди
struct answer_msg
{
   long mtype; //тип сообщения
   int sender; //отправитель
};
int main()
{
   bool is_creator=false;
   int proc_end=0;
   int local_queue;
   int joint_queue;
   ask_msg ask; //отправка запроса
   ask_msg ask_buf[2]; //полученные запросы
   int buf_i=0; //количество запросов в ожидании
   answer_msg answer; //отправка ответов
   answer_msg access; //получение разрешений
   int num_get_access=0; //количество полученных
   int num_send_access=0; //количество отправленных

   //создаем общую очередь
   joint_queue = msgget(777,0606|IPC_CREAT|IPC_EXCL);
   if(joint_queue != -1)
   {
      is_creator=true;
      std::cout<<"Создана общая очередь"<<std::endl;
   }
   else
   { //подключаемся если уже создана
      joint_queue = msgget(777,0606|IPC_CREAT);
      std::cout<<"Подключен к общей очереди"<<std::endl;
   }

   //создаем локальную очередь
   local_queue = msgget(IPC_PRIVATE,0606|IPC_CREAT);
   std::cout<<"Создана локальная очередь"<<std::endl;

   //инициализация запросов и занесение их в очередь
```

```
    ask.ask_time=time(NULL);
    ask.mtype=(PID)%3+1;
    ask.sender=PID;
    ask.answer_queue=local_queue;
    msgsnd(joint_queue,&ask,sizeof(ask_msg),0);
    ask.mtype=(PID+1)%3+1;
    msgsnd(joint_queue,&ask,sizeof(ask_msg),0);

    //инициализация ответа
    answer.mtype=1;
    answer.sender=PID;

    //ожидание двух разрешений на файл (проверка общей очереди)
    while(num_get_access < 2)
    {
        if(msgrcv(joint_queue,&ask_buf[buf_i],sizeof(ask_msg),PID,0)!=-1)
        {//проверка запросов в общей очереди для этой программы
            std::cout<<"Получен запрос доступа от "<<ask_buf[buf_i].sender<<std::endl;
            std::cout<<"Время запроса: "<<ask_buf[buf_i].ask_time<<std::endl;
            if(ask_buf[buf_i].ask_time<ask.ask_time || (ask_buf[buf_i].ask_time==ask.ask_time &&
ask_buf[buf_i].sender<PID))
            {//приоритет младшего и меньший индификатор
                msgsnd(ask_buf[buf_i].answer_queue,&answer,sizeof(answer_msg),0);
                ++num_send_access;
                std::cout<<"Выдача разрешения процессу "<<ask_buf[buf_i].sender<<std::endl;
            }
            else//запомнить в буфере если старше
                ++buf_i;
        }

        if(msgrcv(local_queue,&access,sizeof(answer_msg),1,0)!=-1)
        {//проверка разрешений в локальной очереди
            ++num_get_access;
            std::cout<<"Получено разрешение на доступ от "<<access.sender<<std::endl;
            std::cout<<"Время получения: "<<time(NULL)<<std::endl;
        }
    }

    //вывод файла
    std::cout<<"Начать вывод файла"<<std::endl;
    std::ifstream fin(FNAME);
    std::string str;
    while(std::getline(fin,str))
        std::cout<<str<<std::endl;
    fin.close();
    std::cout<<"Конец файла. Файл закрыт."<<std::endl;

    //выдача разрешений всем ожидающим
    while(buf_i>0)
    {
        --buf_i;
        msgsnd(ask_buf[buf_i].answer_queue, &answer, sizeof(answer_msg), 0);
        ++num_send_access;
        std::cout<<"Выдача разрешения процессу "<<ask_buf[buf_i].sender<<std::endl;
```

```
    }
    //если еще не все запросили доступ
    while(num_send_access<2)
    {//проверка запросов из общей очереди для этой программы
        if(msgrcv(joint_queue, &ask_buf[buf_i], sizeof(ask_msg),PID,0) != -1)
        {
            msgsnd(ask_buf[buf_i].answer_queue,&answer,sizeof(answer_msg),0);
            ++num_send_access;
            std::cout<<"Выдача разрешения процессу " << ask_buf[buf_i].sender <<std::endl;
        }
    }

    //отправка готовности завершения общей очереди
    ask.mtype=0;
    msgsnd(joint_queue,&ask,sizeof(ask_msg),0);
    //ожидание готовности остальных процессов
    if(is_creator)
    {
        while(proc_end<3)
        {
            if(msgrcv(joint_queue,&ask,sizeof(ask_msg),0,0)!=-1)
                ++proc_end;
        }
        msgctl(joint_queue,IPC_RMID,0);
    }
    //удалене локальной очереди
    msgctl(local_queue,IPC_RMID,0);

    return 0;
}
```

Вывод

В ходе работы были изучены механизмы обмена сообщениями и системными вызовами приема и передачи сообщений в операционной системе Ubuntu.