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»

ТЕМА: «ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ НА ОСНОВЕ СООБЩЕНИЙ»

Студент гр. 8308 _____ Тайсумов И.И.

Преподаватель _____ Разумовский Г.В.

Санкт-Петербург

2021

Цель работы.

Целью лабораторной работы является 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.

Обработка результатов эксперимента.

Программы были разработаны и откомпилированы. После чего программы были запушена через три терминала. Результаты работы программ приведены на рисунках (содержание исходного файла – «Taisumov Islam Group 8308»):

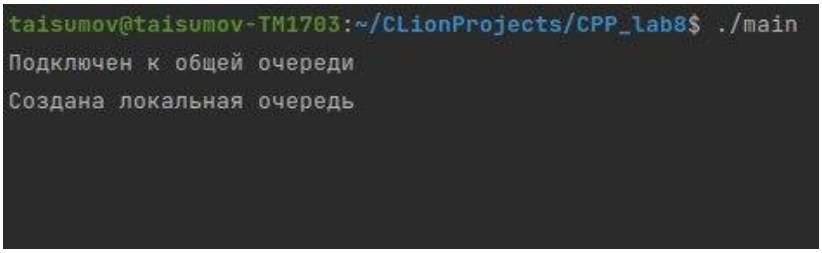

Рисунок 1. Запуск первой программы

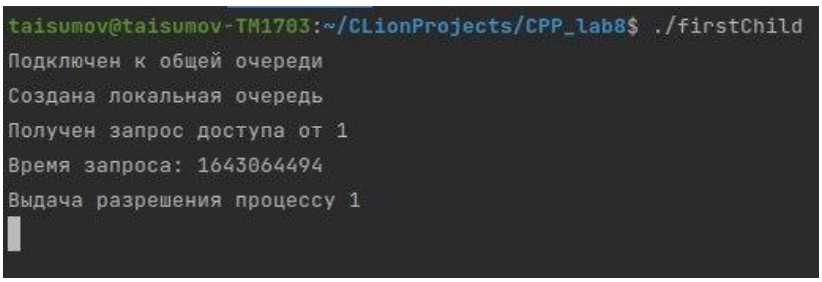

Рисунок 2. Запуск второй программы

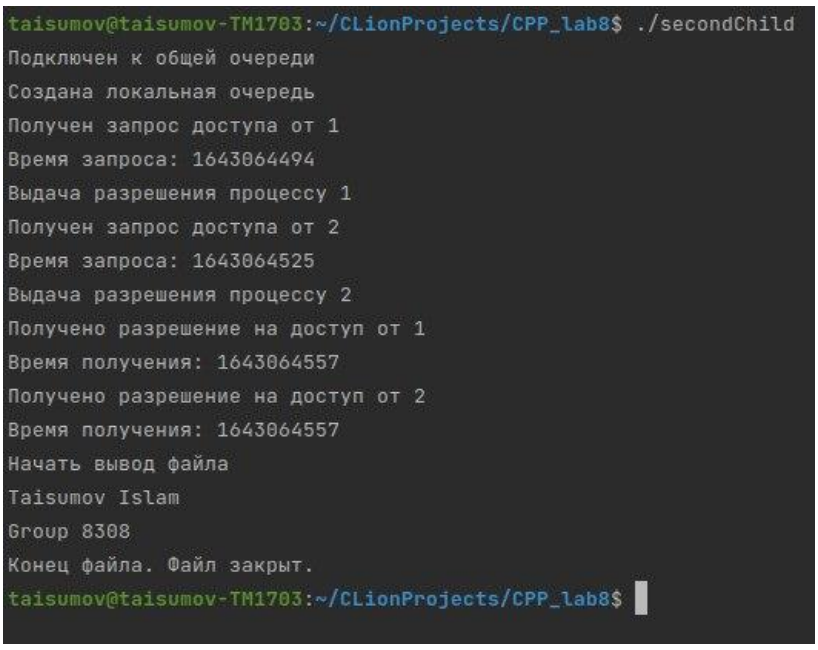

Рисунок 3. Запуск третьей программы и конец её работы

```
taisumov@taisumov-TM1703:~/CLionProjects/CPP_lab8$ ./main
Подключен к общей очереди
Создана локальная очередь
Получен запрос доступа от 2
Время запроса: 1643064525
Получено разрешение на доступ от 2
Время получения: 1643064525
Получен запрос доступа от 3
Время запроса: 1643064557
Получено разрешение на доступ от 3
Время получения: 1643064557
Начать вывод файла
Taisumov Islam
Group 8308
Конец файла. Файл закрыт.
Выдача разрешения процессу 3
Выдача разрешения процессу 2
taisumov@taisumov-TM1703:~/CLionProjects/CPP_lab8$
```

Рисунок 4. Окончание работы первой программы

```
taisumov@taisumov-TM1703:~/CLionProjects/CPP_lab8$ ./firstChild
Подключен к общей очереди
Создана локальная очередь
Получен запрос доступа от 1
Время запроса: 1643064494
Выдача разрешения процессу 1
Получен запрос доступа от 3
Время запроса: 1643064557
Получено разрешение на доступ от 3
Время получения: 1643064557
Получено разрешение на доступ от 1
Время получения: 1643064557
Начать вывод файла
Taisumov Islam
Group 8308
Конец файла. Файл закрыт.
Выдача разрешения процессу 3
taisumov@taisumov-TM1703:~/CLionProjects/CPP_lab8$
```

Рисунок 5. Окончание работы второй программы

Текст программы и распечатки вывода приведены в приложении.

Вывод.

При выполнении лабораторной работы изучены и использованы механизмы обмена сообщениями и системные вызовы приема и передачи сообщений. Программа, разработанная в соответствии с заданием, работает корректно.

ПРИЛОЖЕНИЕ

main.cpp:

```
/**
 * Ислам Тайсумов, группа 8308
 *
 * Компиляция программы:
 * 1. g++ -o main main.cpp
 * 2. g++ -o firstChild firstChild.cpp
 * 3. g++ -o secondChild secondChild.cpp
 * 4. ./main
 * 5. ./firstChild
 * 6. ./secondChild
 *
**/

#include <iostream>
#include <fstream>
#include <sys/msg.h>
#include <ctime>

#define PID 1

using namespace std;

                //структура сообщений общей очереди
struct askMessage
{
   long mtype;          //тип сообщения
   int sender;          //отправитель
   int answer_queue;     //локальная очередь заказчика
   int ask_time;         //время запроса
};

                //структура сообщений локальной очереди
struct answerMessage
{
   long mtype;          //тип сообщения
   int sender;          //отправитель
};

int main()
{
   bool isMainProcess = false;
   int doneProcesses=0;
   int localQueue;
   int commonQueue;
   askMessage ask;              //отправка запроса
   askMessage askBuffer[2];      //полученные запросы
   int indexBuffer=0;           //количество запросов в ожидании
```

```
answerMessage answer;        //отправка ответов
answerMessage access;        //получение разрешений
int numOfGetAccess = 0;      //количество полученных
int numOfSendAccess = 0;     //количество отправленных

//создаем общую очередь
//
// msgget возвращает идентификатор очереди сообщений в
//случае успеха; -1 в случае ошибки.

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

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

//инициализация запросов и занесение их в очередь
ask.ask_time = time(NULL);
ask.mtype = (PID) % 3 + 1;
ask.sender = PID;
ask.answer_queue = localQueue;

msgsnd(commonQueue,
     &ask,
     sizeof(askMessage),
     0);

ask.mtype = (PID + 1) % 3 + 1;

msgsnd(commonQueue,
     &ask,
     sizeof(askMessage),
     0);

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

```
//ожидание двух разрешений на файл (проверка общей очереди)
while(numOfGetAccess < 2)
{
  if(msgrcv(commonQueue,
        &askBuffer[indexBuffer],
        sizeof(askMessage),
        PID,
        IPC_NOWAIT) != -1)
  {//проверка запросов в общей очереди для этой программы
    cout << "Получен запрос доступа от "  << askBuffer[indexBuffer].sender    << endl;
    cout << "Время запроса: "          << askBuffer[indexBuffer].ask_time   << endl;

    if(askBuffer[indexBuffer].ask_time  < ask.ask_time ||
     (askBuffer[indexBuffer].ask_time == ask.ask_time &&
      askBuffer[indexBuffer].sender < PID))
    {//приоритет младшего и меньший индификатор
      msgsnd(askBuffer[indexBuffer].answer_queue,
          &answer,
          sizeof(answerMessage),
          0);
      ++numOfSendAccess;
      cout << "Выдача разрешения процессу " << askBuffer[indexBuffer].sender << endl;
    }
    else //запомнить в буфере если старше
      ++indexBuffer;
  }

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


//вывод файла
cout << "Начать вывод файла" << endl;
ifstream fileStream("TEXT.txt");
string str;

while(getline(fileStream,str))
  cout << str << endl;

fileStream.close();
cout << "Конец файла. Файл закрыт." << endl;
```

```
//выдача разрешений всем ожидающим
while(indexBuffer > 0)
{
  --indexBuffer;
  msgsnd(askBuffer[indexBuffer].answer_queue,
      &answer,
      sizeof(answerMessage),
      0);
  ++numOfSendAccess;
  cout << "Выдача разрешения процессу " << askBuffer[indexBuffer].sender << endl;
}


//если еще не все запросили доступ
while(numOfSendAccess < 2)
{//проверка запросов из общей очереди для этой программы
  if(msgrcv(commonQueue,
        &askBuffer[indexBuffer],
        sizeof(askMessage),
        PID,
        IPC_NOWAIT) != -1)
  {
    msgsnd(askBuffer[indexBuffer].answer_queue,
        &answer,
        sizeof(answerMessage),
        0);
    ++numOfSendAccess;
    cout<<"Выдача разрешения процессу " << askBuffer[indexBuffer].sender << endl;
  }
}


//отпрвака готовности завершения общей очереди
ask.mtype = 0;
msgsnd(commonQueue,
    &ask,
    sizeof(askMessage),
    0);


//ожиание готовности остальных процессов
if(isMainProcess)
{
  while(doneProcesses < 3)
  {
    if(msgrcv(commonQueue,
          &ask,
          sizeof(askMessage),
          0,
          0) != -1)
      ++doneProcesses;
```

```
    }
    msgctl(commonQueue,
        IPC_RMID,
        0);
  }

  //удалене локальной очереди
  msgctl(localQueue,
      IPC_RMID,
      0);

  return 0;
}
```

firstChild.cpp:

```
#include <iostream>
#include <fstream>
#include <sys/msg.h>

#define PID 2

using namespace std;

struct askMessage
{
  long mtype;
  int sender;
  int answer_queue;
  int ask_time;
};

struct answerMessage
{
  long mtype;
  int sender;
};

int main()
{
  bool isMainProcess = false;
  int doneProcesses=0;
  int localQueue;
  int commonQueue;
  askMessage ask;            //отправка запроса
  askMessage askBuffer[2];     //полученные запросы
  int indexBuffer=0;          //количество запросов в ожидании
  answerMessage answer;       //отправка ответов
  answerMessage access;       //получение разрешений
  int numOfGetAccess = 0;     //количество полученных
```

```
int numOfSendAccess = 0;      //количество отправленных


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

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


//инициализация запросов и занесение их в очередь
ask.ask_time      = time(NULL);
ask.mtype         = (PID) % 3 + 1;
ask.sender        = PID;
ask.answer_queue    = localQueue;

msgsnd(commonQueue,
     &ask,
     sizeof(askMessage),
     0);

ask.mtype = (PID + 1) % 3 + 1;

msgsnd(commonQueue,
     &ask,
     sizeof(askMessage),
     0);

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



//ожидание двух разрешений на файл (проверка общей очереди)
while(numOfGetAccess < 2)
{
```

```
  if(msgrcv(commonQueue,
        &askBuffer[indexBuffer],
        sizeof(askMessage),
        PID,
        IPC_NOWAIT) != -1)
  {//проверка запросов в общей очереди для этой программы
    cout << "Получен запрос доступа от "    << askBuffer[indexBuffer].sender    << endl;
    cout << "Время запроса: "             << askBuffer[indexBuffer].ask_time  << endl;
    if(askBuffer[indexBuffer].ask_time  < ask.ask_time ||
     (askBuffer[indexBuffer].ask_time == ask.ask_time &&
      askBuffer[indexBuffer].sender < PID))
    {//приоритет младшего и меньший индификатор
      msgsnd(askBuffer[indexBuffer].answer_queue,
          &answer,
          sizeof(answerMessage),
          0);
      ++numOfSendAccess;
      cout << "Выдача разрешения процессу " << askBuffer[indexBuffer].sender << endl;
    }
    else//запомнить в буфере если старше
      ++indexBuffer;
  }

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


//вывод файла
cout << "Начать вывод файла" << endl;
ifstream fileStream("TEXT.txt");
string str;

while(getline(fileStream,str))
  cout << str << endl;
fileStream.close();
cout << "Конец файла. Файл закрыт." << endl;


//выдача разрешений всем ожидающим
while(indexBuffer > 0)
{
  --indexBuffer;
```

```
    msgsnd(askBuffer[indexBuffer].answer_queue,
           &answer,
           sizeof(answerMessage),
           0);
    ++numOfSendAccess;
    cout << "Выдача разрешения процессу " << askBuffer[indexBuffer].sender << endl;
}
//если еще не все запросили доступ
while(numOfSendAccess < 2)
{//проверка запросов из общей очереди для этой программы
    if(msgrcv(commonQueue,
              &askBuffer[indexBuffer],
              sizeof(askMessage),
              PID,
              IPC_NOWAIT) != -1)
    {
        msgsnd(askBuffer[indexBuffer].answer_queue,
               &answer,
               sizeof(answerMessage),
               0);
        ++numOfSendAccess;
        cout << "Выдача разрешения процессу " << askBuffer[indexBuffer].sender << endl;
    }
}


//отпрвака готовности завершения общей очереди
ask.mtype = 0;
msgsnd(commonQueue,
       &ask,
       sizeof(askMessage),
       0);
//ожиание готовности остальных процессов
if(isMainProcess)
{
    while(doneProcesses < 3)
    {
        if(msgrcv(commonQueue,
                  &ask,
                  sizeof(askMessage),
                  0,
                  0) != -1)
            ++doneProcesses;
    }
    msgctl(commonQueue,
           IPC_RMID,
           0);
}
//удалене локальной очереди
msgctl(localQueue,
       IPC_RMID,
```

```
        0);

    return 0;
}
```

secondChild.cpp:

```
#include <iostream>
#include <fstream>
#include <sys/msg.h>

#define PID 3

using namespace std;

struct askMessage
{
    long mtype;
    int sender;
    int answer_queue;
    int ask_time;
};

struct answerMessage
{
    long mtype;
    int sender;
};

int main()
{
    bool isMainProcess = false;
    int doneProcesses=0;
    int localQueue;
    int commonQueue;
    askMessage ask;                 //отправка запроса
    askMessage askBuffer[2];        //полученные запросы
    int indexBuffer=0;              //количество запросов в ожидании
    answerMessage answer;           //отправка ответов
    answerMessage access;           //получение разрешений
    int numOfGetAccess = 0;         //количество полученных
    int numOfSendAccess = 0;        //количество отправленных


    //создаем общую очередь
    commonQueue = msgget(777,
                0606 | IPC_CREAT | IPC_EXCL);
    if(commonQueue != -1)
    {
        isMainProcess=true;
```

```
    cout << "Создана общая очередь" << endl;
}
else
{//подключаемся если уже создана
    commonQueue = msgget(777,
                  0606 | IPC_CREAT);
    cout << "Подключен к общей очереди" << endl;
}

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


//инициализация запросов и занесение их в очередь
ask.ask_time      = time(NULL);
ask.mtype         = (PID) % 3 + 1;
ask.sender        = PID;
ask.answer_queue  = localQueue;

msgsnd(commonQueue,
    &ask,
    sizeof(askMessage),
    0);

ask.mtype = (PID + 1) % 3 + 1;

msgsnd(commonQueue,
    &ask,
    sizeof(askMessage),
    0);

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


//ожидание двух разрешений на файл (проверка общей очереди)
while(numOfGetAccess < 2)
{
  if(msgrcv(commonQueue,
        &askBuffer[indexBuffer],
        sizeof(askMessage),
        PID,
        IPC_NOWAIT) != -1)
  {//проверка запросов в общей очереди для этой программы
    cout << "Получен запрос доступа от "   << askBuffer[indexBuffer].sender   << endl;
    cout << "Время запроса: "              << askBuffer[indexBuffer].ask_time  << endl;
    if(askBuffer[indexBuffer].ask_time  < ask.ask_time ||
      (askBuffer[indexBuffer].ask_time == ask.ask_time &&
```

```
        askBuffer[indexBuffer].sender < PID))
      {//приоритет младшего и меньший индификатор
        msgsnd(askBuffer[indexBuffer].answer_queue,
            &answer,
            sizeof(answerMessage),
            0);
        ++numOfSendAccess;
        cout << "Выдача разрешения процессу " << askBuffer[indexBuffer].sender << endl;
      }
      else//запомнить в буфере если старше
        ++indexBuffer;
    }

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


//вывод файла
cout << "Начать вывод файла" << endl;
ifstream fileStream("TEXT.txt");
string str;

while(getline(fileStream,str))
    cout << str << endl;
fileStream.close();
cout << "Конец файла. Файл закрыт." << endl;


//выдача разрешений всем ожидающим
while(indexBuffer > 0)
{
    --indexBuffer;
    msgsnd(askBuffer[indexBuffer].answer_queue,
        &answer,
        sizeof(answerMessage),
        0);
    ++numOfSendAccess;
    cout << "Выдача разрешения процессу " << askBuffer[indexBuffer].sender << endl;
}
//если еще не все запросили доступ
while(numOfSendAccess < 2)
{//проверка запросов из общей очереди для этой программы
```

```
    if(msgrcv(commonQueue,
            &askBuffer[indexBuffer],
            sizeof(askMessage),
            PID,
            IPC_NOWAIT) != -1)
    {
       msgsnd(askBuffer[indexBuffer].answer_queue,
            &answer,
            sizeof(answerMessage),
            0);
       ++numOfSendAccess;
       cout << "Выдача разрешения процессу " << askBuffer[indexBuffer].sender << endl;
    }
  }


  //отпрвака готовности завершения общей очереди
  ask.mtype = 0;
  msgsnd(commonQueue,
      &ask,
      sizeof(askMessage),
      0);
  //ожиание готовности остальных процессов
  if(isMainProcess)
  {
    while(doneProcesses < 3)
    {
       if(msgrcv(commonQueue,
            &ask,
            sizeof(askMessage),
            0,
            0) != -1)
         ++doneProcesses;
    }
    msgctl(commonQueue,
         IPC_RMID,
         0);
  }
  //удалене локальной очереди
  msgctl(localQueue,
      IPC_RMID,
      0);

  return 0;
}
```