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

# ОТЧЕТ

**по лабораторной работе №7**

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

**ТЕМА: «ОБМЕН ДАННЫМИ ЧЕРЕЗ КАНАЛ»**

Студент гр. 8308 _____ Тайсумов И.И.

Преподаватель _____ Разумовский Г.В.

Санкт-Петербург

2021

**Цель работы.**

Целью лабораторной работы является знакомство с механизмом обмена данными через программный канал и системными вызовами, обеспечивающими такой обмен.

**Задание.**

Написать программу, которая обменивается данными через канал с двумя потомками. Программа открывает входной файл, построчно читает из него данные и записывает их в канал. Потомки выполняют свои программы и поочередно читают символы из канала и записывают их в свои выходные файлы: первый потомок – нечетные символы, а второй – четные. Синхронизация работы потомков должна осуществляться напрямую с использованием сигналов SIGUSR1 и SIGUSR2. Об окончании записи файла в канал программа оповещает потомков сигналом SIGQUIT и ожидает завершения работы потомков. Когда они заканчивают работу, программа закрывает канал.

**Обработка результатов эксперимента.**

Ниже приведен вывод программы при файле data.txt, содержащем строку «1234567890», а также содержание выходных файлов:

```
taisumov@taisumov-TM1703:~/CLionProjects/CPP_lab7$ ./main
First: 1
Second: 2
First: 3
Second: 4
First: 5
Second: 6
First: 7
Second: 8
First: 9
Second: 0
First child: parent finished
Second child: parent finished
taisumov@taisumov-TM1703:~/CLionProjects/CPP_lab7$
```

*Рисунок 1. Результат работы программы*

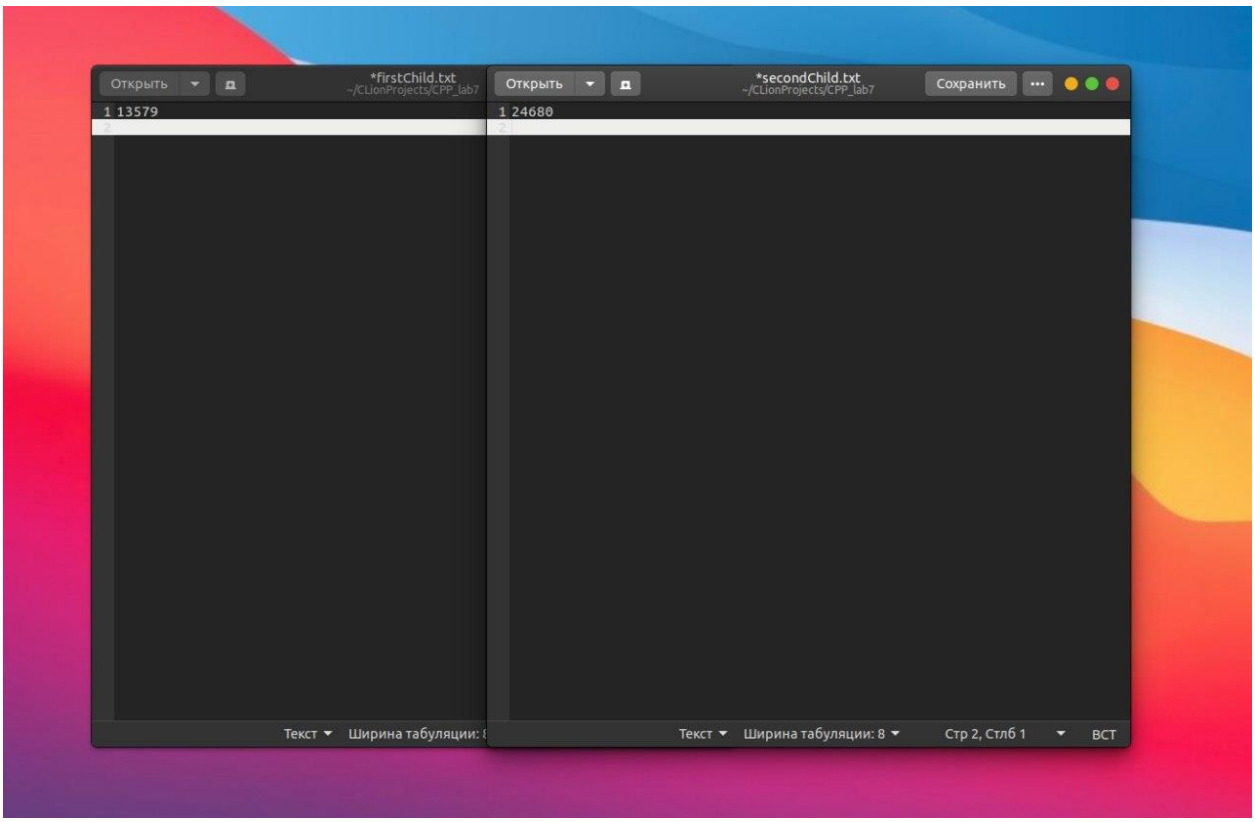


*Рисунок 2. Выходные файлы*

Текст программы приведен в приложении.

**Вывод.**

При выполнении лабораторной работы изучены и использованы механизм обмена данными через программный канал и системные вызовы, обеспечивающие такой обмен. 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
 *
**/

#include <iostream>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <fstream>
#include <fcntl.h>

using namespace std;

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

  // Тип данных sigset_t служит для представления набора сигналов
  sigset_t set;

  // Делает пустым набор сигналов, на который указывает set
  // (никаких сигналов в set представлено не будет)
  // RETURN INT
  sigemptyset(&set);

  // Добавляет сигнал signo в набор set
  // RETURN INT
  sigaddset(&set, SIGUSR1);
  sigaddset(&set, SIGUSR2);
  sigaddset(&set, SIGQUIT);

  // sigprocmask используется для того, чтобы изменить
  // список блокированных в данный момент сигналов
  //
  // SIG_BLOCK:
  // Набор блокируемых сигналов - объединение текущего набора и аргумента set.
  //
  // Если значение поля oldset не равно нулю, то предыдущее значение маски сигналов
  // записывается в oldset.
  sigprocmask(SIG_BLOCK,
          &set,
```

```
        nullptr);

// Массив для открытия каналов
int arrayForPipes[2];

// Открытие каналов
if (pipe(arrayForPipes) == -1) {
   exit(EXIT_FAILURE);
}

// Запрет блокировки чтения или записи
fcntl(arrayForPipes[0], F_SETFL, O_NONBLOCK);

int firstProcess = fork();
int secondProcess;

if(firstProcess != 0)
   secondProcess = fork();

if (firstProcess == 0) {
   // Закрытие канала и запуск первой программы
   close(arrayForPipes[1]);
   execl("./firstChild", "./firstChild", &arrayForPipes[0], nullptr);

} else if (secondProcess == 0) {
   // Закрытие канала и запуск второй программы
   close(arrayForPipes[1]);
   execl("./secondChild", "./secondChild", &arrayForPipes[0], nullptr);

} else {
   ifstream fileStream("data.txt");

   if (fileStream.is_open()) {
      close(arrayForPipes[0]);

      string bufferString;

      while(getline(fileStream, bufferString)){
         write(arrayForPipes[1], &bufferString[0], bufferString.size());
         sleep(2);
      }
      fileStream.close();

      kill(0, SIGQUIT);

      waitpid(firstProcess,
            nullptr,
            0);
      waitpid(secondProcess,
            nullptr,
            0);
```

```
        close(arrayForPipes[1]);

    }
  }

  return 0;
}
```

## firstChild.cpp:

```
#include <iostream>
#include <signal.h>
#include <sys/types.h>
#include <unistd.h>
#include <fstream>

using namespace std;

bool quit = false;

void quitSigAction(int quitSignal) {
  cout << "First child: parent finished" << endl;
  quit = true;
}

int main (int argc,  char* argv[]) {
  int sig;
  sigset_t set;
  sigemptyset(&set);
  sigaddset(&set, SIGUSR1);

      // sigaction используется для изменения действий процесса при получении соответствующего сигнала
  // struct sigaction {
  //    void (*sa_handler)(int);
  //    void (*sa_sigaction)(int, siginfo_t *, void *);
  //    sigset_t sa_mask;
  //    int sa_flags;
  //    void (*sa_restorer)(void);
  // }
  struct sigaction act;
  act.sa_handler = &quitSigAction;
  sigaction(SIGQUIT,
        &act,
        nullptr);

  sigset_t set_quit;
  sigaddset(&set_quit, SIGQUIT);


  // sigprocmask используется для того, чтобы изменить
```

```
    // список блокированных в данный момент сигналов
    //
    // SIG_UNBLOCK
    // Сигналы, устанавливаемое значение битов которых равно set, удаляются из списка блокируемых сигналов.
    // Допускается разблокировать незаблокированные сигналы.
    //
    // Если значение поля oldset не равно нулю, то предыдущее значение маски сигналов
    // записывается в oldset.
    sigprocmask(SIG_UNBLOCK,&set_quit, nullptr);

    ofstream file("firstChild.txt");

    char symb;

    if (file.is_open()) {
        int size = 0;
        while(size != -1 || !quit) {
            size = read(*argv[1], &symb, 1);
            if(size != -1) {
                cout << "First: " << symb << endl;
                file << symb;
                kill(0, SIGUSR2);
                sigwait(&set, &sig);
            }
        }

        //int kill(pid_t pid, int sig);
        // pid > 0, сигнал sig посылается процессу с идентификатором pid.
        // pid = 0, то sig посылается каждому процессу, который входит в группу текущего процесса.
        // pid = -1, то sig посылается каждому процессу, за исключением процесса с номером 1 (init).
        // pid < -1, то sig посылается каждому процессу, который входит в группу процесса -pid.
        // sig = 0, то никакой сигнал не посылается, а только выполняется проверка на существования
        //         процесса или группы.
        kill(0,SIGUSR2);
        file.close();
        close(*argv[1]);
    }
    exit(EXIT_SUCCESS);
}
```

## secondChild.cpp:

```
#include <iostream>
#include <signal.h>
#include <sys/types.h>
#include <unistd.h>
#include <fstream>

using namespace std;
```

```
bool quit = false;

void quitSigAction(int quitSignal) {
    cout << "Second child: parent finished\n";
    quit = true;
}
int main(int argc,char* argv[]) {
    int sig;
    sigset_t set;
    sigemptyset(&set);
    sigaddset(&set, SIGUSR2);

    // sigaction используется для изменения действий процесса при получении соответствующего сигнала
    // struct sigaction {
    //     void (*sa_handler)(int);
    //     void (*sa_sigaction)(int, siginfo_t *, void *);
    //     sigset_t sa_mask;
    //     int sa_flags;
    //     void (*sa_restorer)(void);
    // }
    struct sigaction act;
    act.sa_handler = &quitSigAction;
    sigaction(SIGQUIT,
            &act,
            nullptr);

    sigset_t set_quit;
    sigaddset(&set_quit, SIGQUIT);

    // sigprocmask используется для того, чтобы изменить
    // список блокированных в данный момент сигналов
    //
    // SIG_UNBLOCK
    // Сигналы, устанавливаемое значение битов которых равно set, удаляются из списка блокируемых сигналов.
    // Допускается разблокировать незаблокированные сигналы.
    //
    // Если значение поля oldset не равно нулю, то предыдущее значение маски сигналов
    // записывается в oldset.
    sigprocmask(SIG_UNBLOCK,&set_quit,nullptr);

    ofstream file("secondChild.txt");

    char symb;
    sigwait(&set, &sig);

    if (file.is_open()) {
        int size;
        while(size != -1 || !quit) {
            size = read(*argv[1], &symb, 1);
            if(size != -1) {
                cout << "Second: " << symb << endl;
```

```
            file << symb;
            kill(0, SIGUSR1);
            sigwait(&set, &sig);
        }
    };

    //int kill(pid_t pid, int sig);
    // pid > 0, сигнал sig посылается процессу с идентификатором pid.
    // pid = 0, то sig посылается каждому процессу, который входит в группу текущего процесса.
    // pid = -1, то sig посылается каждому процессу, за исключением процесса с номером 1 (init).
    // pid < -1, то sig посылается каждому процессу, который входит в группу процесса -pid.
    // sig = 0, то никакой сигнал не посылается, а только выполняется проверка на существования
    //         процесса или группы.
    kill(0, SIGUSR1);
    file.close();
    close(*argv[1]);
  }
  exit(EXIT_SUCCESS);
}
```