Тестовое задание для стажера на позицию «Программист на языке C++»

Выполнил: студент СПбГУТ направления «Программная инженерия» Молошников Федор

СОДЕРЖАНИЕ

1. ПОСТАНОВКА ЗАДАЧИ

Задание состоит из двух программ, которые необходимо реализовать. Взаимодействие программ должно быть реализовано через использование сокетов.

Программа №1.
Должна состоять из двух потоков и одного общего буфера.
Поток 1. Принимает строку, которую введет пользователь. Должна быть проверка, что строка состоит только из цифр и не превышает 64 символа. После проверки строка должна быть отсортирована по убыванию и все элементы, значение которых чётно, заменены на латинские буквы «КВ». После данная строка помещается в общий буфер и поток должен ожидать дальнейшего ввода пользователя.
Поток 2. Должен обрабатывать данные которые помещаются в общий буфер. После получения данных общий буфер затирается. Поток должен вывести полученные данные на экран, рассчитать общую сумму всех элементов , которые являются численными значениями. Полученную сумму передать в Программу №2. После этого поток ожидает следующие данные.
Примечание №1 по Программе №1: Взаимодействие потоков должно быть синхронизировано, поток №2 не должен постоянно опрашивать общий буфер. Механизм синхронизации не должен быть глобальной переменной.
Примечание №2 по Программе №1: Работа программы должна быть максимально независима от статуса запуска программы №2. Это значит, что внезапный останов программы №2 не должен приводить к немедленным проблемам ввода у пользователя.
При перезапуске программы №2 необходимо произвести передподключение.

Программа №2.
Ожидает данные от Программы №1. При получении данных происходит анализ из скольки символов состоит переданное значение. Если оно больше 2-ух символов и если оно кратно 32 выводит сообщение о полученных данных , иначе выводится сообщение об ошибке. Далее программа продолжает ожидать данные.
Примечание №1 по Программе №2: Работа программы должна быть максимально независима от статуса запуска программы №1. Внезапный останов программы №1 не должен приводить к немедленным проблемам отображения. Необходимо ожидать подключение программы №1 при потере связи между программами.
Примечание по заданию: Не обязательно все размещать в одном классе. Может быть разработана иерархия классов. Чем более функционален интерфейс класса, тем лучше.

Требования к присылаемым решениям.

- Готовое задание должно быть передано ответным письмом в zip архиве.
- Каждая из программ должна находиться в своей папке.
- Для сборки программа не должна требовать настроек системы или нахождения определенных файлов в специфичном месте.
- Исходный код должен компилироваться средствами **cmake или make с использованием gcc для работы в среде Linux**. В папке с исходным кодом не должно быть мусора: неиспользуемых файлов исходных кодов или ресурсов, промежуточных файлов сборки и т.д.
- Максимальное время на выполнение задания – 2 недели.

2. РАЗРАБОТКА СТРУКТУРЫ ПРОГРАММ №1 И №2

Программа №1 (App1) согласно заданию будет иметь 2 потока и общий буфер. Синхронизировать работу потоков будем с помощью мьютексов и условных переменных. Таким образом, исключается одновременный доступ к общему буферу. Тем же способом обеспечивается очередность ввода и вывода 1-го и 2-го потоков: перед тем, как запросить новые данные у пользователя, 1 поток будет ждать, пока 2-й поток выведет полученные данные. Если это не происходит в течение 1мс, то 1-й поток продолжает работу. Если же 2-й поток успевает вывести данные раньше, чем пройдет 1 мс, то оставшееся время не теряется 1-м потоком.

Программа №1 (App1) и программа №2 (App2) согласно заданию должны взаимодействовать с помощью сокетов. В качестве домена сокетов был выбран домен AF_INET (для сетевого протокола IPv4). В качестве типа сокета был выбран тип SOCK_STREAM (TCP-сокет). App1 выполняет роль клиента, App2 выполняет роль сервера. App1 и App2 максимально независимы друг от друга. При внезапной остановке Программы №2 Программа №1 будет пытаться переподключиться раз в секунду. При этом пользователь имеет возможность продолжить ввод. После того, как соединение будет восстановлено, Программа №1 передаст в Программу №2 данные (сумму), которые не удалось передать во время разрыва, а так же данные(сумму), которые соответствуют последнему вводу пользователя. При внезапной остановке Программы №1 Программа №2 будет ожидать переподключения Программы №1.

При передаче данных учитывается тот факт, что тип int может иметь различное внутреннее представление в различных системах. Поэтому сумма элементов передается из Программы №1 в Программу №2 в виде си-строки.

Код:

2.1. App1

- Makefile – для сборки и очистки результатов сборки

- main.cpp – точка входа в программу.

- app1.h+cpp – содержит класс app1 Программы №1.

- tcpclient.h+cpp – класс tcpclient для взаимодействия с Программой №2.

- InputCalcFn.h+inl+cpp – содержит пространство имен InputCalcFn, в котором объявлены функции (в том числе шаблонные функции) для работы с данными для обоих программ.

2.2. App2

- Makefile – для сборки и очистки результатов сборки

- main.cpp – точка входа в программу.

- tcpserverwork.h+cpp – класс tcpserver и его потомок tcpserverwork для взаимодействия с Программой №1.

- InputCalcFn.h+inl+cpp

3. РАЗРАБОТКА КЛАССОВ И ФУНКЦИЙ

3.1. App1

3.1.1. app1

Класс, реализующий возможности Программы №1.

3.1.1.1. Свойства:

bool BufStatus=false;	0 - пустой; 1 - содержит строку. Нужен для корректной работы cvBuf,т.к. позволяет избежать пропущенных уведомлений.

std::mutex mutBuf;	mutex для Buf
std::condition_variable cvBuf;	для Buf
char Buf[129]={};	общий буфер со строкой
std::mutex mutInput;	Для того, чтобы по возможности ввод 1-го потока следовал за выводом 2-го потока. Причем возможные задержки (на ожидании освобождения мьютексов) потоков будут минимальными.
std::condition_variable cvInput;	
bool IsOutputted=true;	
bool BoolCounter=false;	
bool CurrentCounter=true;	

3.1.1.2. Методы

void Thread1Function();	метод, выполняемый в 1м потоке
void Thread2Function();	метод, выполняемый в 2м потоке

Эти методы передаются в конструкторы класса std::thread ,который обеспечивает многопоточность Программы №1.

3.1.2. tcpclient

3.1.2.1. Свойства:

int ClientSocket;	fd сокета
struct sockaddr_in SockAddr;	Описывает сокет
int retrcv=1;	0 - был разрыв соединения между сокетами. 1 - разрыва не было.
bool IsConnected=false;	
char testBuf[1]={};	с помощью него будем подтверждать соединение после каждой отправки

3.1.2.2. Методы

tcpclient(uint32_t NAdr, uint16_t NPort);	Инициализирует `SockAddr`. NAdr указывает адрес, NPort – порт, на которые мы будем подключаться.
void Reconnect();	Переподключение
void Resend(char *NBuf, int len);	Повторная отправка данных при разрыве соединения
void SendBuf(char *NBuf, int len);	Отправляет содержимое NBuf в количестве len байт
void tcpClose();	Разрывает соединение и закрывает сокет

Алгоритм работы SendBuf() изображен на рис.1

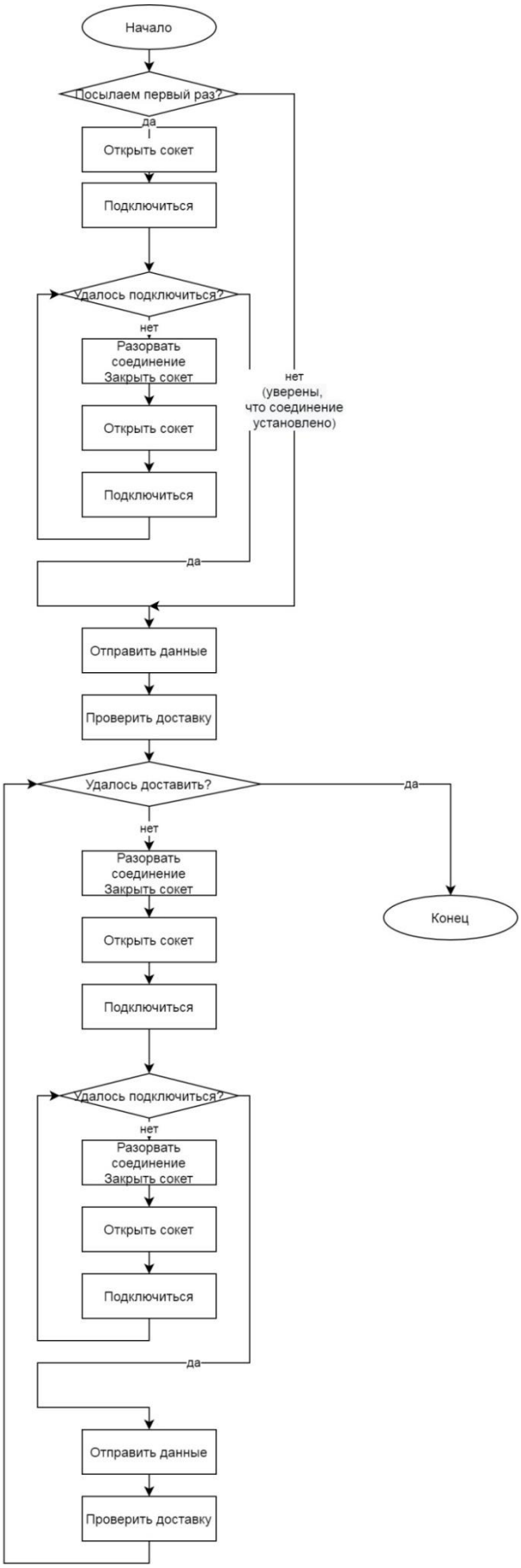

Рисунок 1 – Алгоритм работы SendBuf()

3.1.3. Пространство имен InputCalcFn

3.1.3.1. Функции

void InputDigits(std::string &s1, unsigned int length,std::string InputMsg=std::string(""), std::string str=std::string("1234567890"));	Ввод с проверкой на допустимые символы
void ReplaceKB(std::string &s1);	Замена четных цифр на «КВ»
void CalcSumElements(char *KBstr, char *SumElements);	Находит сумму всех элементов в «КВ» строке
void MultiplicityCheck(char *ch);	(для Программы №2)Проверяет количество цифр суммы и значение этой суммы.

3.1.3.2. Шаблонные функции

template<typename T> void Heapify(T *Arr, int Length, int i, bool SortOrder=false);	Для сортировки двоичной кучей. Позволяют сортировать в обоих направлениях массивы данных, для которых определены операции “<” “>”
template<typename T> void BuildHeap(T *Arr, int Length, bool SortOrder=false);	
template<typename T> void HeapSort(T *Arr, int Length, bool SortOrder=false);	

3.2. App2

3.2.1. tcpserver

3.2.1.1. Свойства:

int MasterSocket;	fd сокета, принимающего соединения
struct sockaddr_in SockAddr;	Описывает сокет, принимающий соединения
int BufLength;	Число байт, которые мы будем принимать в сокет

3.2.1.2. Методы

tcpserver(uint32_t NAdr, uint16_t NPort);	Создает сокет, и связывает его с заданным адресом и номером порта
void tcplisten();	
void AcceptAndWork();	Принимает соединение и вызывает `WorkOnSlaveSocket`
void ShutDownClose();	Закрывает соединение и закрывает сокет, принимающий соединения.
void WorkOnSlaveSocket(int SlaveSocket);	Работает с сокетом
virtual void SetBufLength()=0;	Устанавливают длину принимаемых данных и работу над ними. Определяются в tcpserversoket
virtual void Work(char *)=0;	

3.2.2. tcpserverwork

Наследует tcpserver. Это позволяет отделить работу над данными от работы по получению данных.

3.2.2.1. Свойства:

-	-

3.2.2.2. Методы

virtual void SetBufLength();	Устанавливает длину принимаемых данных в байтах
virtual void Work(char *Buf);	Выполняет работу над данными

4. СБОРКА И ЗАПУСК

Для сборки программ необходимо перейти в папку source и запустить утилиту **make**. После чего программа будет собрана с помощью компилятора gcc.

Для очистки результатов сборки нужно выполнить команду **make clean**.

В архиве присутствуют уже готовые бинарные файлы обеих программ в папке **executable**.

Для связи между собой программы используют порт **12345**. Если он оказался занят, можно попробовать изменить его (App1.cpp 45 строка для Программы №1 и main.cpp 7 строка для ПрограммыN°2)

5. ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ

Обе программы работают в консоли. После запуска Программы №1 она будет предлагать пользователю вводить строки до тех пор, пока пользователь не нажмет Ctrl+C. Программа №2 будет ждать подключения Программы №1 и будет выводить получаемые данные, пока пользователь не нажмет Ctrl+C.

Рис2-4 Нормальная работа программ.

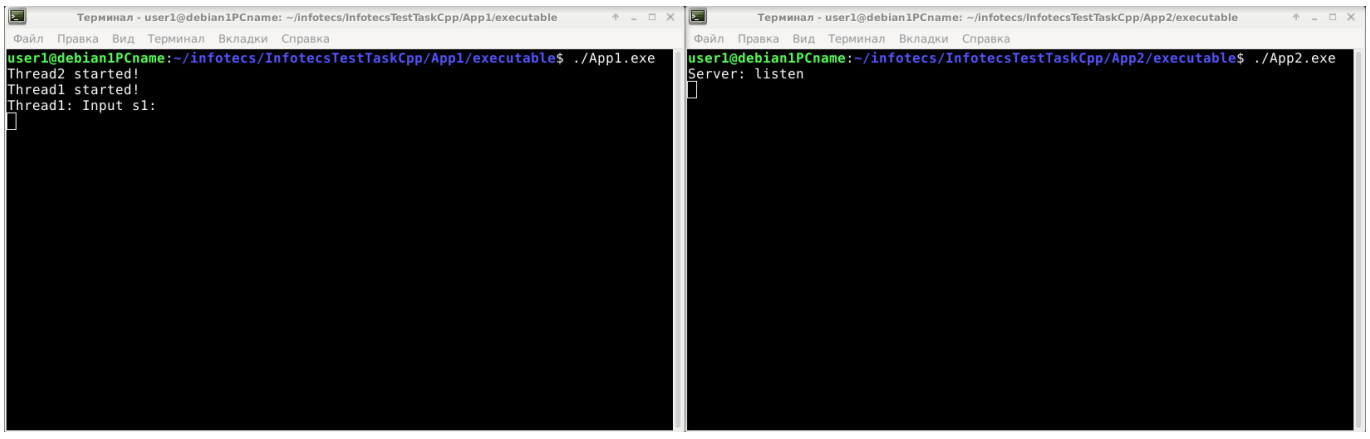

Рисунок 2 – Нормальная работа программ.

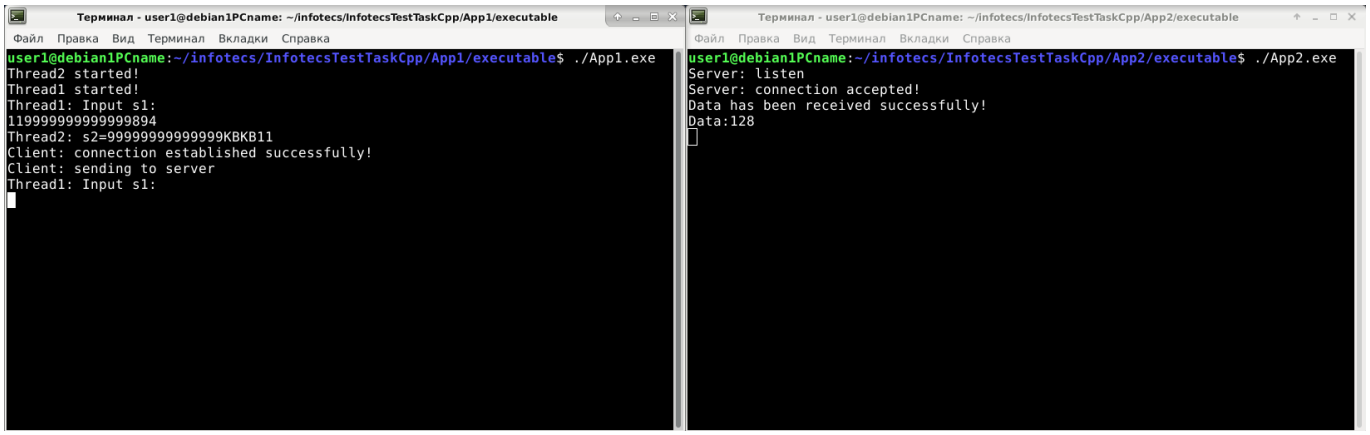

Рисунок 3 – Нормальная работа программ.

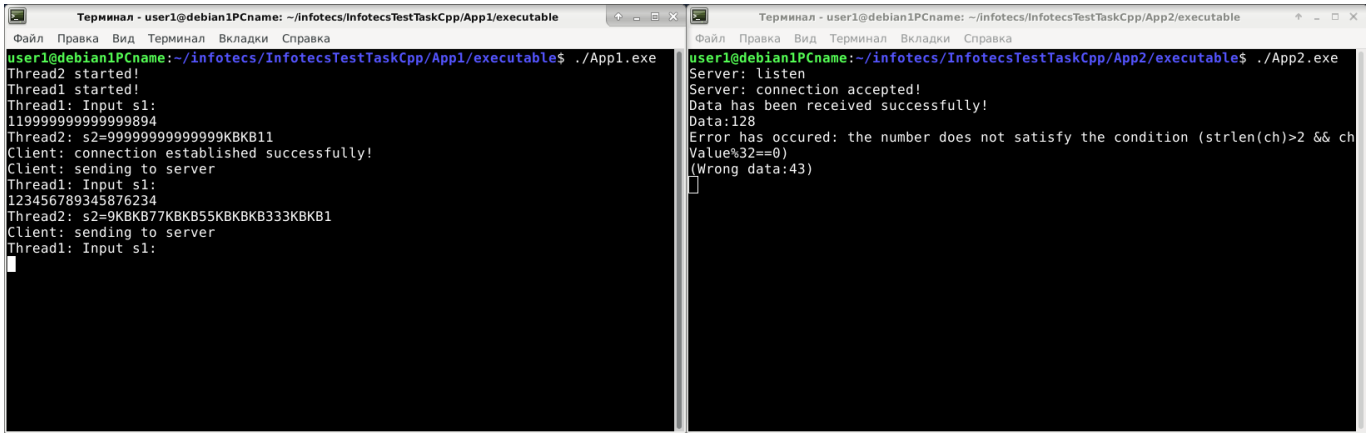

Рисунок 4 – Нормальная работа программ.

Рис 5-6 работа при остановке Программы №2

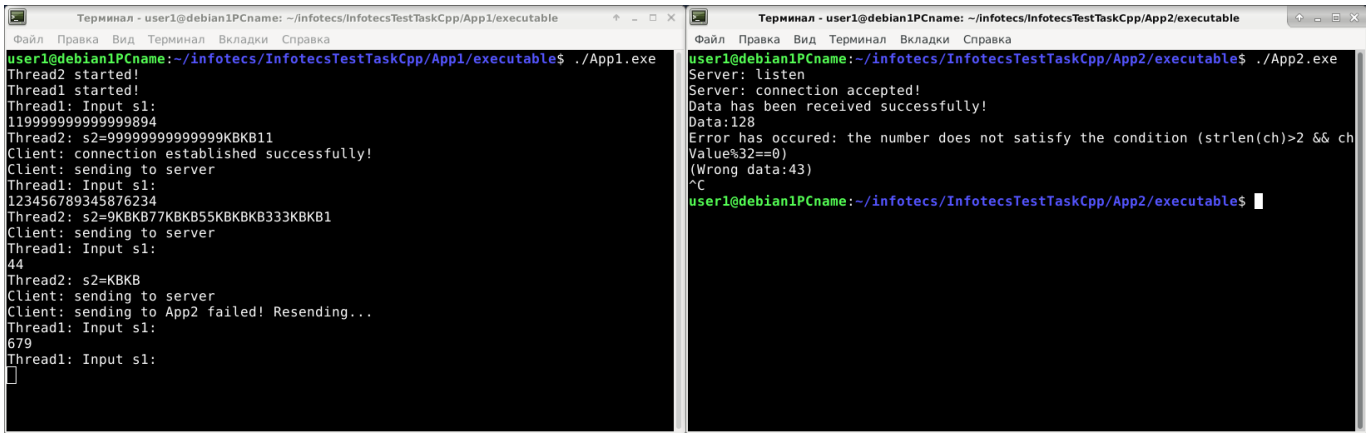

Рисунок 5 – работа при остановке Программы №2

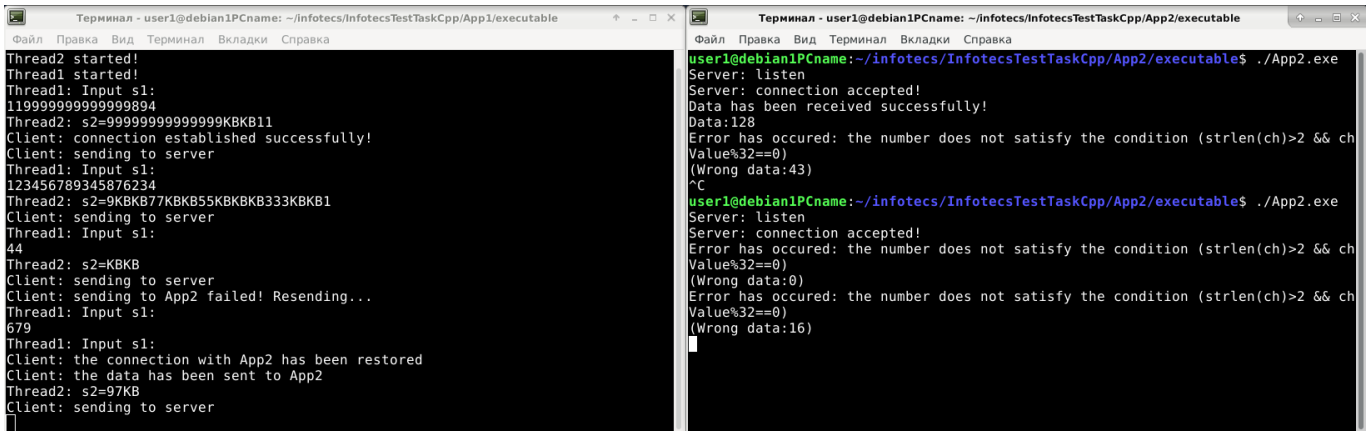

Рисунок 6 – работа при остановке Программы №2 (после перезапуска Программы №2)

Рис 7-8 работа при остановке Программы №1

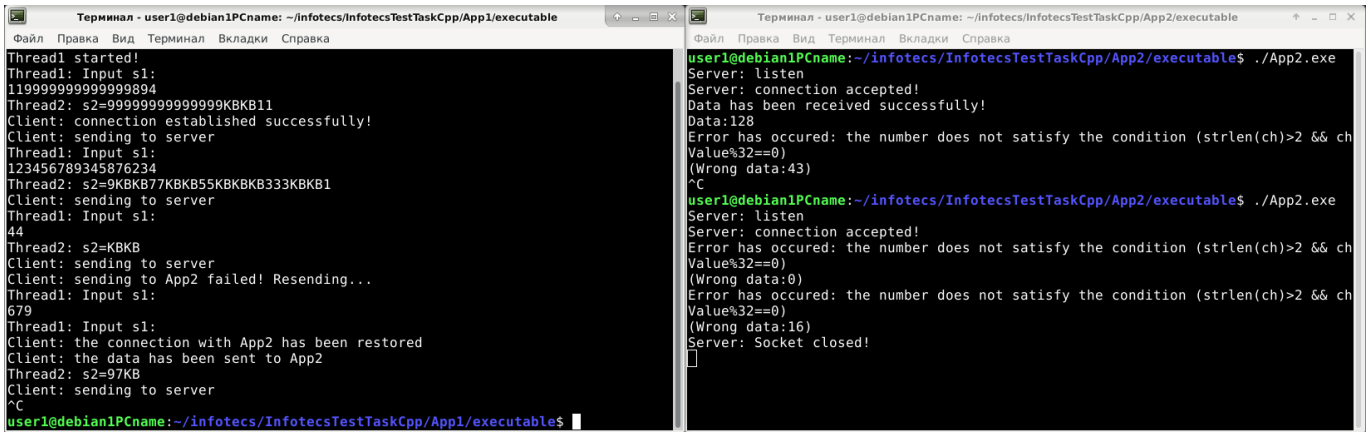

Рисунок 7 – работа при остановке Программы №1

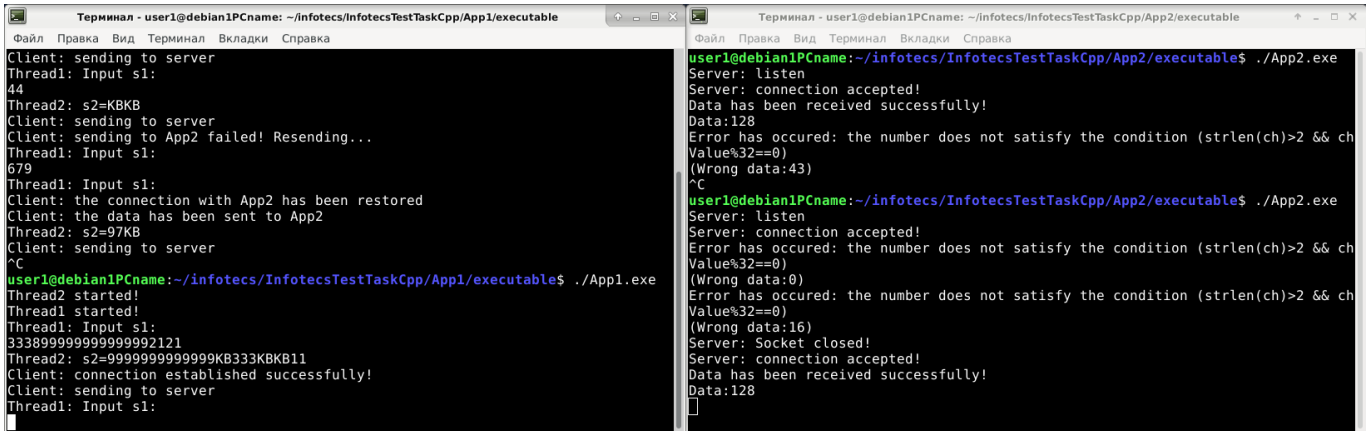

Рисунок 8 – работа при остановке Программы №1

6. ЗАКЛЮЧЕНИЕ

1) Структура 2х поточной Программы №1 построена согласно условиям задания.
2) Взаимодействие Программы №1 и Программы №2 осуществляется с помощью сокетов. Причем программы работают максимально независимо друг от друга.
3) Программы выполняют над данными все необходимые операции.

7. СПИСОК ЛИТЕРАТУРЫ

1) Калугин-Балашов Д.А. Лекции «Многопоточное программирование на C/C++»

2) Коробов С. А. Лекции по дисциплине Объектно–ориентированное программирование.

8. ПРИЛОЖЕНИЕ

8.1. App1

8.1.1. Makefile

```
App1.exe: main.o app1.o tcpclient.o InputCalcFn.o
 g++ main.o app1.o tcpclient.o InputCalcFn.o -o App1.exe -lpthread
main.o:main.cpp
 g++ main.cpp -c
app1.o:app1.cpp
 g++ app1.cpp -c
tcpclient.o: tcpclient.cpp
 g++ tcpclient.cpp -c
InputCalcFn.o: InputCalcFn.cpp
 g++ InputCalcFn.cpp -c
clean:
 rm -rf *.o App1.exe
```

8.1.2. main.cpp

```
/*main.cpp App1*/

#include "app1.h"

int main()
{
    app1 App1;

    std::thread Thread1(&app1::Thread1Function, std::ref(App1));
    std::thread Thread2(&app1::Thread2Function, std::ref(App1));
    Thread1.join();
    Thread2.join();

    return 0;
}
```

8.1.3. app1.h

```
#ifndef APP1_H_INCLUDED
#define APP1_H_INCLUDED
/*app1.h*/
#include <iostream>

#include <string> //для обычных строк

#include <thread> //для потоков
#include <mutex> //для мьютекса
#include <condition_variable> //для условной переменной

#include <chrono> //для mc

#include <cstring> //для работы с буфером

#include "tcpclient.h"
#include "InputCalcFn.h"
```

```
    class app1 //класс, реализующий возможности 1ого приложения
    {
    private:
        bool BufStatus=false; //0 - пустой; 1 - содержит строку. Нужен для корректной работы cvBuf,т.к. позволяет избежать пропущенных уведомлений.
        std::mutex mutBuf; //mutex для Buf
        std::condition_variable cvBuf; //для Buf
        char Buf[129]={}; //общий буфер со строкой

        std::mutex mutInput; //мьютекс, для того, чтобы вводить новое значение по возможности не сразу, а после того, как 2й поток выведет свое (будем ожидать вывода 2го потока в течение небольшого промежутка времени)
        std::condition_variable cvInput;
        bool IsOutputted=true; //true - 2й уже вывел результат на экран. false- 2й поток ещё не вывел результат на экран.
        bool BoolCounter=false; //меняет значение на противоположное каждый раз, когда идет вывод 2м потоком. Поможет обранужить вывод в то время, когда шел ввод с клавиатуры.
        bool CurrentCounter=true; //Хранит текущее значение BoolCounter

    public:
        app1()=default;
        void Thread1Function(); //функция, выполняемая в 1м потоке
        void Thread2Function(); //функция, выполняемая во 2м потоке
    };

    #endif // APP1_H_INCLUDED
```

8.1.4. app1.cpp

```
    #include "app1.h"

    void app1::Thread1Function()
    {
        std::cout << "Thread1 started!" << std::endl;
        std::string s1; //строка для ввода

        while(1) //основной цикл, в котором происходит ввод и обработка строк и где далее стока помещается в общий буфер
        {
            using namespace std::chrono_literals;
            std::unique_lock<std::mutex> ul1(mutInput);
             //    if(!BoolCounter) IsOutputted=false; //если вывели 2 раза между вводами,
                cvInput.wait_for(ul1,        1ms,        [this]{return IsOutputted;});
                CurrentCounter=BoolCounter;
             //   cvInput.wait(ul1);
                IsOutputted=false;
            //    BoolCounter=false;
```

```
            ul1.unlock();

            InputCalcFn::InputDigits(s1, 64, "Thread1: Input s1:");
//Ввод с проверкой символов
            char chSort[65]={}; //Массив символов, который будет
отсортирован
            strcpy(chSort, s1.c_str());
            InputCalcFn::HeapSort<char>(chSort, strlen(chSort), true);
//Сортировка двоичной кучей

            s1=std::string(chSort);
            InputCalcFn::ReplaceKB(s1); //Меняем четный цифры на "KB"

            mutBuf.lock();
                if(CurrentCounter!=BoolCounter) IsOutputted=false;

                strcpy(Buf, s1.c_str());
                BufStatus=true;
                cvBuf.notify_one();
            mutBuf.unlock();

       //     std::this_thread::sleep_for(1ms);
        }
    }

    void app1::Thread2Function()
    {
        std::cout << "Thread2 started!" << std::endl;
        char s2[129]={}; //будем принимать сюда данные из общего
буфера (строку KB)
        char SendBuf[4]={}; //буфер для отправки на сервер
        tcpclient Client1(INADDR_LOOPBACK, 12345); //инициализация
клиента

        while(1)
        {
            std::unique_lock<std::mutex> ul(mutBuf);
                cvBuf.wait(ul, [this]{return BufStatus;}); //если
BufStatus true, то на ожидание не встаем. Эквивалентно while
(!stop_waiting()) {wait(lock);}
                //cvBuf.wait(ul); //Вот так делать нельзя. т.к. если
Thread1Function успеет занять mutex первой, то Thread2Function не
сработает должным образом
                strcpy(s2, Buf);
                strcpy(Buf, ""); //затираем общий буфер
                BufStatus=false;

                mutInput.lock();
                    //"Поток должен вывести полученные данные на
экран"
                    std::cout << "Thread2: s2=" << s2 <<std::endl;
//выводим на экран
                    IsOutputted=true;
```

```
                    BoolCounter=!BoolCounter;
                    cvInput.notify_one();
                mutInput.unlock();

            ul.unlock();
            //"рассчитать общую сумму всех элементов , которые являются численными значениями."
            InputCalcFn::CalcSumElements(s2, SendBuf); //находим сумму элементов си-строке s2 и помещаем её в SendBuf
            //"Полученную сумму передать в Программу номер2. После этого поток ожидает следующие данные."
            Client1.SendBuf(SendBuf, 4);
        }
    }
```

8.1.5. tcpclient.h

```
    #ifndef TCPCLIENT_H_INCLUDED
    #define TCPCLIENT_H_INCLUDED
    /* tcpclient.h*/
    #include <iostream>

    #include <sys/socket.h>
    #include <netdb.h> //для IPP
    #include <cstring> //для chBuf
    #include <unistd.h> //для close дескрипторов сокетов и функции sleep()   (Доступ к POSIX API)

    class tcpclient
    {
    private:
        int ClientSocket;
        struct sockaddr_in SockAddr;
        int retrcv=1; //0 - был разрыв соединения между сокетами. 1 - разрыва не было.
        bool IsConnected=false;

        char testBuf[1]={}; //с помощью него будем подтверждать соединение после каждой отправки

    public:
        tcpclient(uint32_t NAdr, uint16_t NPort);
        void Reconnect();
        void Resend(char *NBuf, int len);
        void SendBuf(char *NBuf, int len);
        void tcpClose();
    };

    #endif // TCPCLIENT_H_INCLUDED
```

8.1.6. tcpclient.cpp

```
    /* tcpclient.cpp*/
    #include "tcpclient.h"
```

```
    tcpclient::tcpclient(uint32_t NAdr, uint16_t NPort)
    {
        SockAddr.sin_family=AF_INET;
        SockAddr.sin_port=htons(NPort);
        SockAddr.sin_addr.s_addr=htonl(NAdr); //INADDR_LOOPBACK
    }
    void tcpclient::Reconnect()
    {
        while(!IsConnected)
        {
            sleep(1);
         //   std::cout << "Client: Reconnect" << std::endl;
            shutdown(ClientSocket, SHUT_RDWR);
            close(ClientSocket);
            ClientSocket=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
            IsConnected=!(bool)connect(ClientSocket,             (const
sockaddr*)&SockAddr, sizeof(SockAddr));
        }
    }
    void tcpclient::Resend(char *NBuf, int len)
    {
        while(!retrcv)
        {
            std::cout   <<   "Client:   sending   to   App2   failed!
Resending..." << std::endl;
            shutdown(ClientSocket, SHUT_RDWR);
            close(ClientSocket);
            ClientSocket=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
            IsConnected=!(bool)connect(ClientSocket,             (const
sockaddr*)&SockAddr, sizeof(SockAddr));
     if(!IsConnected)
     {
               Reconnect();
                    std::cout << "Client: the connection with App2 has
been restored" << std::endl;
     }
            send(ClientSocket, NBuf, len, MSG_NOSIGNAL);
            retrcv=recv(ClientSocket,   testBuf,   1,   MSG_WAITALL);
//проверяем доставку
            if(retrcv) std::cout << "Client: the data has been sent to
App2" << std::endl;
        }
    }
    void tcpclient::SendBuf(char *NBuf, int len)
    {
        if(!IsConnected)
        {
            ClientSocket=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
            IsConnected=!(bool)connect(ClientSocket,             (const
sockaddr*)&SockAddr, sizeof(SockAddr));
            if(IsConnected)   std::cout   <<   "Client:   connection
established successfully!" << std::endl;
            else
```

```
     {
          std::cout   <<   "Client:   connection   not   established!
Reconnecting..." << std::endl;
          Reconnect();
          std::cout  <<  "Client:  the  connection  with  App2  has  been
restored" << std::endl;
     }
        }
        std::cout << "Client: sending to server" << std::endl;
        send(ClientSocket, NBuf, len, MSG_NOSIGNAL);
        retrcv=recv(ClientSocket,     testBuf,     1,     MSG_WAITALL);
//проверяем доставку
        Resend(NBuf, len);
    }
    void tcpclient::tcpClose()
    {
        shutdown(ClientSocket, SHUT_RDWR);
        close(ClientSocket);
    }
```

8.1.7. InputCalcFn.h

```
    #ifndef INPUTCALCFN_H_INCLUDED
    #define INPUTCALCFN_H_INCLUDED

    #include <iostream>
    #include <cmath> //для pow в классе Digits64
    #include <string>
    #include <cstring> //для strlen в MultiplicityCheck

    namespace InputCalcFn
    {
    void InputDigits(std::string &s1, unsigned int length,std::string
InputMsg=std::string(""), std::string str=std::string("1234567890"));
//s1 - строка, куда будет записан ввод, если он корректен. lenght -
максимально допустимая длина строки( для строки "123" это
соответсвтует

//3) str - допустимые символы InputMsg - сообщение при вводе (если
пустое, то просто ввод без пустой строки)
    /*   Шаблонные функции для сортировки кучей (по возрастанию и
убыванию)*/
    template<typename T>
    void Heapify(T *Arr, int Length, int i, bool SortOrder=false);
//Восстанавливает свойства кучи, начиная с корня i. Предполагается,
что левое и правое деревья удовлетворяют куче. SortOrder - направление
сортировки. по умолчанию - по возрастанию(т.е. в корне кучи стоит
наибольший элемент).

    template<typename T>
    void BuildHeap(T *Arr, int Length, bool SortOrder=false); //строит
кучу из неупорядоченного массива SortOrder - направление сортировки.
по умолчанию - по возрастанию.(т.е. в корне кучи стоит наибольший
элемент).
```

```
    template<typename T>
    void HeapSort(T *Arr, int Length, bool SortOrder=false);
//сортировка кучей. По умолчанию - по возрастанию.

    /*  Для замены цифр на символы "КВ"*/
    void ReplaceKB(std::string &s1);

    /*Для потока 2 приложения App1*/
    void CalcSumElements(char *KBstr, char *SumElements); //вычисление
суммы всех элементов

    /*Для App2*/
    void MultiplicityCheck(char *ch);

    }

    #include "InputCalcFn.inl" //Подключаем реализацию наших шаблонных
функций
    #endif // INPUTCALCFN_H_INCLUDED
```

8.1.8. InputCalcFn.inl

```
    #ifndef INPUTCALCFN_INL_INCLUDED
    #define INPUTCALCFN_INL_INCLUDED
    /*InputCalcFn.inl Реализация шаблонных функций*/

    using namespace InputCalcFn;

    template<typename T>
    void InputCalcFn::Heapify(T *Arr, int Length, int i, bool
SortOrder) //Восстанавливает свойства кучи, начиная с корня i.
Предполагается, что левое и правое деревья удовлетворяют куче.
SortOrder - направление сортировки. по умолчанию - по возрастанию(т.е.
в корне кучи стоит наибольший элемент).
    {
        int LargestLowest=i;
        if(!SortOrder) //по умолчанию сортировка по возрастанию
        {
            if(2*i+1<Length      &&      Arr[2*i+1]>Arr[LargestLowest])
LargestLowest=2*i+1;
            if(2*i+2<Length      &&      Arr[2*i+2]>Arr[LargestLowest])
LargestLowest=2*i+2;
        }
        else
        {
            if(2*i+1<Length      &&      Arr[2*i+1]<Arr[LargestLowest])
LargestLowest=2*i+1;
            if(2*i+2<Length      &&      Arr[2*i+2]<Arr[LargestLowest])
LargestLowest=2*i+2;
        }
        if(LargestLowest!=i)
        {
            T temp=Arr[LargestLowest];
```

```
            Arr[LargestLowest]=Arr[i];
            Arr[i]=temp;
            Heapify<T>(Arr, Length, LargestLowest, SortOrder);
        }
    }
    template<typename T>
    void InputCalcFn::BuildHeap(T *Arr, int Length, bool SortOrder)
//строит кучу из неупорядоченного массива SortOrder - направление
сортировки. по умолчанию - по возрастанию.(т.е. в корне кучи стоит
наибольший элемент).
    {
        for(int i=Length/2; i>=0; i--)
        {
            Heapify<T>(Arr, Length, i, SortOrder);
        }
    }

    template<typename T>
    void InputCalcFn::HeapSort(T *Arr, int Length, bool SortOrder)
//сортировка кучей. По умолчанию - по возрастанию.
    {
        BuildHeap<T>(Arr, Length, SortOrder);
        for(int i=0; i<Length-1; i++)
        {
            T temp=Arr[0];
            Arr[0]=Arr[Length-1-i];
            Arr[Length-1-i]=temp;
            Heapify<T>(Arr, Length-1-i, 0, SortOrder);
        }
    }
    #endif // INPUTCALCFN_INL_INCLUDED
```

8.1.9. InputCalcFn.cpp

```
    /*InputCalcFn.cpp*/
    #include "InputCalcFn.h"
    using namespace InputCalcFn;
    void InputCalcFn::InputDigits(std::string &s1, unsigned int
length,std::string InputMsg, std::string str) //s1 - строка, куда
будет записан ввод, если он корректен. lenght - максимально допустимая
длина строки( для строки "123" это соответствует

//3) str - допустимые символы InputMsg - сообщение при вводе (если
пустое, то просто ввод без пустой строки)
    {
        while(1)
        {
            if(InputMsg!="")
            {
                std::cout<<InputMsg<<std::endl;
            }
            getline(std::cin, s1);
            if(s1.length()<=length) //<=64
            {
```

```
                if(s1.find_first_not_of(str)==std::string::npos      &&
s1!="")
                {
                    break;
                }
                else
                {
                    std::cout<< "Extraneous symbols!" << std::endl;
                }
            }
            else //тут уже об ошибке можно выводить
            {
                std::cout<< "string length>" << length<<std::endl;
            }
        }
    }
    void InputCalcFn::ReplaceKB(std::string &s1)
    {
        std::string::size_type pos=s1.find_first_of("02468");
        while(pos!=std::string::npos)
        {
            s1.replace(pos, 1, "KB");
            pos=s1.find_first_of("02468", pos);
        }
    }

    /*Для 2го потока*/
    void  InputCalcFn::CalcSumElements(char  *KBstr,  char  *SumElements)
//в KBstr и SumElements можно подставлять одну и ту же строку
    {
        int Sum=0;
        for(unsigned int i=0; i<strlen(KBstr);i++)
        {
            if(KBstr[i]>='0' && KBstr[i]<='9')
            {
                Sum+=KBstr[i]-'0';
            }
        }
        sprintf(SumElements, "%d", Sum);
    }


    void InputCalcFn::MultiplicityCheck(char *ch)
    {
        int chValue;
        sscanf(ch, "%d", &chValue);
        if(strlen(ch)>2 && chValue%32==0)
        {
            std::cout<<"Data         has         been         received
successfully!"<<std::endl;
            std::cout<<"Data:"<<ch<<std::endl;
        }
        else
```

```
        {
            std::cout<<"Error has occured: the number does not satisfy
the condition (strlen(ch)>2 && chValue%32==0)"<<std::endl;
            std::cout<<"(Wrong data:"<<ch<<")"<<std::endl;
        }
    }
```

8.2. App2

8.2.1. Makefile

```
App2.exe: main.o tcpserverwork.o InputCalcFn.o
 g++ main.o tcpserverwork.o InputCalcFn.o -o App2.exe -lpthread
main.o:main.cpp
 g++ main.cpp -c
tcpserverwork.o:tcpserverwork.cpp
 g++ tcpserverwork.cpp -c
InputCalcFn.o: InputCalcFn.cpp
 g++ InputCalcFn.cpp -c
clean:
 rm -rf *.o App2.exe
```

8.2.2. main.cpp

```
/*main.cpp App2*/
#include <iostream>

#include "tcpserverwork.h"
int main()
{
    tcpserverwork server1(INADDR_ANY, 12345);
    server1.tcplisten();
    server1.AcceptAndWork();
    server1.ShutDownClose();

    return 0;
}
```

8.2.3. tcpserverwork.h

```
#ifndef TCPSERVERWORK_H_INCLUDED
#define TCPSERVERWORK_H_INCLUDED

/*tcpserverwork.h*/

#include <iostream>

#include <sys/types.h>
#include <sys/socket.h>

#include <netdb.h> //для IPPROTO_TCP
#include <unistd.h> //для close дескрипторов сокетов

#include <cstring> //для memset
#include <cstdio> //для sscanf()
```

```
    #include "InputCalcFn.h" //для работы в методе Work()

    class tcpserver //абстрактный класс для запуска tcp сервера. Работает с единственным SlaveSoket-ом. Имеет чисто виртуальные методы SetBufLength и Work
    {
    private:
        int MasterSocket; //Сокет, принимающий соединения
        struct sockaddr_in SockAddr; //содержит адрес и номер порта, к которым биндится MasterSoket
    protected:
        int BufLength; //сколько байт принимаем из сокета
    public:
        tcpserver(uint32_t NAdr, uint16_t NPort); //Передаем адрес и номер порта, к которым будем биндиться
        void tcplisten();
        void AcceptAndWork();
        void ShutDownClose();
        void WorkOnSlaveSocket(int SlaveSocket); //Вызывается внутри AcceptAndWork()
        virtual void SetBufLength()=0; //Эти 2 чисто виртуальные функции будут определяться в классе-потомке. Таким образом, мы отделяем работу с полученными данными от работы сервера по их приему.
        virtual void Work(char *)=0;
    };

    class tcpserverwork: public tcpserver //класс, где реализованы методы по работе с данными.
    {
    public:
        tcpserverwork(uint32_t NAdr, uint16_t NPort);
        virtual void SetBufLength(); //Устанавливаем число байт, которые хотим принять
        virtual void Work(char *Buf);
    };

    #endif // TCPSERVERWORK_H_INCLUDED
```

8.2.4. tcpserverwork.cpp

```
    /*tcpserverwork.cpp*/
    #include "tcpserverwork.h"

    tcpserver::tcpserver(uint32_t NAdr, uint16_t NPort) //Передаем адрес и номер порта, к которым будем биндиться
    {
        MasterSocket=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); //создаем сокет. IPPROTO_TCP - протокол. Можно вместо него 0 поставить
        int optval =1; //для установки опций MasterSoket
        setsockopt(MasterSocket, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval));

        std::memset(&SockAddr, 0, sizeof(SockAddr)); //зануляем, т.к. в конце структуры должны быть нули
```

```
        SockAddr.sin_family=AF_INET;
        SockAddr.sin_port=htons(NPort); //номер порта. htons сетевой порядок байт
        SockAddr.sin_addr.s_addr=htonl(NAdr);    //биндимся на все сетевые интерфесы какие у нас есть (в частности, на все сетевые карты). INADDR_ANY=0.0.0.0 INADDR_LOOPBACK=127.0.0.1

        bind(MasterSocket,      (struct      sockaddr      *)&SockAddr, sizeof(SockAddr)); //связываем сокет с адресом и портом
    }
    void tcpserver::tcplisten()
    {
        listen(MasterSocket,   SOMAXCONN);   //SOMAXCONN   максимальное число одновременно пытающихся подключиться
        std::cout<<"Server: listen"<<std::endl;
    }
    void tcpserver::AcceptAndWork()
    {
        while(1)
        {
            int SlaveSocket=accept(MasterSocket, 0, 0);
            std::cout<<"Server: connection accepted!"<<std::endl;

            WorkOnSlaveSocket(SlaveSocket);

            shutdown(SlaveSocket, SHUT_RDWR);
            close(SlaveSocket);
            std::cout<<"Server: Socket closed!"<<std::endl;
        }
    }
    void tcpserver::ShutDownClose()
    {
        shutdown(MasterSocket, SHUT_RDWR); //закрываем соединение
        close(MasterSocket); //закрываем дескриптор
    }
    void tcpserver::WorkOnSlaveSocket(int SlaveSocket)
    {
        //Работаем с сокетом
        SetBufLength(); //устанавливаем размер буфера, в который мы будем принимать, в классе потомке
        char Buf[BufLength]={};     //В получаемой сумме число цифр не превышает 64. Поэтому 65 байт нам хватит
        char testBuf[1]={};

        int  retrecv=recv(SlaveSocket,  Buf,  BufLength,  MSG_WAITALL); //Получаем данные
        while(retrecv)
        {
            Work(Buf); //Тут работаем с полученными данными

            send(SlaveSocket,  testBuf,  1,  MSG_NOSIGNAL);   //посылаем клиенту 1 байт, чтобы он мог знать, что соединение ещё есть.
```

```
            retrecv = recv(SlaveSocket, Buf, BufLength, MSG_NOSIGNAL);
//MSG_NOSIGNAL
        }
    }
    /*tcpserverwork*/
    tcpserverwork::tcpserverwork(uint32_t        NAdr,        uint16_t
NPort):tcpserver(NAdr, NPort) {}
    void  tcpserverwork::SetBufLength()  //Устанавливаем  число  байт,
которые хотим принять
    {
        BufLength=4; //Сумма не может состоять из более, чем 3 цифр.
Один символ отводим под '\0'
    }
    void tcpserverwork::Work(char *Buf)
    {
        //std::cout<<"Server: Buf="<< Buf <<std::endl;
        //Тут будем читать из буфера и делать всю работу
        InputCalcFn::MultiplicityCheck(Buf);
    }
```