Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Кафедра «Информатика»
кафедра

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 6

Использование сигналов и средств хронометража в ОС GNU/Linux
Тема

Преподаватель _____ А. С. Кузнецов
Подпись, дата Инициалы, Фамилия

Студент КИ19-17/1Б, №031939174 _____ А. К. Никитин
Номер группы, зачетной книжки Подпись, дата Инициалы, Фамилия

Красноярск 2021

1 Цель

Изучение особенностей работы с системными часами и сигналами в ОС GNU/Linux.

2 Задачи

1. Ознакомиться с краткими теоретическими сведениями по использованию сигналов и средств работы с системными часами в ОС GNU/Linux.

2. Модифицировать результаты выполнения ЛР 5 использованием программных средств для работы с сигналами и средствами хронометража. Обеспечить сборку программы с использованием скрипта *configure* и утилиты GNU *make*.

3. Используя изученные механизмы, разработать и отладить:

a. серверную часть, в которой должен быть обеспечен перехват сигналов, приводящих к аварийному завершению процесса и настроен таймер неактивности клиентской стороны. Аргументы: а) имя *log*-файла, фиксирующего происходящие события; б) количество единиц времени, по прошествии которых сервер автоматически разрывает соединение с клиентом (для TCP) или не ждет сообщений от клиента (для UDP); в) другие - по необходимости.

b. клиентскую часть, в которой должен быть обеспечен перехват сигналов, приводящих к аварийному завершению процесса и настроен таймер неактивности пользователя. Аргументы: а) имя *log*-файла, фиксирующего происходящие события; б) количество единиц времени, по прошествии которых клиент автоматически завершает функционирование; в) другие - по необходимости.

4. Написать отчет и представить его к защите с исходными текстами программ, предварительно загрузив код и отчет в электронный курс в виде

единственного архива формата *.*tar.gz*. Исходные тексты программ должны содержать комментарии в стиле системы *doxygen*.

Описание варианта:

Программа принимает от пользователя три строки, (первая и третья строки — это правильные рациональные или десятичные дроби вида «1/3» или «0,5», вторая строка — это знак арифметической операции вида «+», «-», «*», «/» либо операции сравнения «<», «>», «=», «!=», «>=», «<=»), выполняет требуемую операцию над полученными операндами, и выводит результат на экран. Обеспечить также сокращение дроби при необходимости. Если оба операнда арифметической операции являются рациональными дробями, результатом тоже должна быть рациональная дробь. Для операций сравнения достаточно результата «Истина» или «Ложь».

3 Исходные тексты программ

На листинге 1 представлен код программы algorithm.c.

Листинг 1 – Код программы с основным алгоритмом

```
/*! \file   algorithm.c
 *  \brief  Fraction calculation and comparison
 *  \author Nikitin Alexander, KI19-17/1Б
 */
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include <stdio.h>
#include "algorithm.h"

/*! \brief Checks if the string contains only digits
 *
 *  \param[in] string Numeric or non-numeric string
 *
 *  \return Is the string contains only digits or not (true or false)
 */
bool checkInt(char* string)
{
    char* intChars = "0123456789";

    for (int i = 0; i < strlen(string); i++)
    {
        // Проверка минуса перед числом
        if (i == 0 && string[i] == '-')
            continue;

        if (strchr(intChars, string[i]) == NULL)
            return false;
    }
    return true;
}

/*! \brief Divides fraction in two parts and returns both of the values. First
part is numeric part, second part is
 *  fractional part. Both are integers.
 *
```

```
 *  \param[in] string Source fraction
 *  \param[in] pos Position of separator ('.' or '/')
 *
 *  \return First and second numbers of fraction
 */
struct fractionParts splitFraction(char* string, int pos)
{
    struct fractionParts output;

    char* firstPart = NULL;
    char* secondPart = NULL;

    int firstNumber;
    int secondNumber;

    char* pEnd = NULL;

    if (pos <= 0 || pos > strlen(string))
    {
        output.isWrong = true;
        return output;
    }

    // Срез целой части
    for (int i = 0; i < pos; i++)
    {
        firstPart = realloc(firstPart, i + 2);
        firstPart[i] = string[i];
    }
    firstPart[pos] = '\0';

    // Срез дробной части
    for (int i = pos + 1, j = 0; i < strlen(string); i++, j++)
    {
        secondPart = realloc(secondPart, j + 2);
        secondPart[j] = string[i];
    }
    secondPart[strlen(string) - pos - 1] = '\0';

    if (!checkInt(firstPart) || !checkInt(secondPart))
    {
        output.isWrong = true;
```

```
        return output;
    }

    if (firstPart[0] == '-')
        output.isNegative = true;
    else
        output.isNegative = false;

    firstNumber = abs(strtol(firstPart, &pEnd, 10));
    secondNumber = abs(strtol(secondPart, &pEnd, 10));

    output.firstPart = firstNumber;
    output.secondPart = secondNumber;
    output.isWrong = false;

    return output;
}

/*! \brief Tries to transform string into fractionString.
 *
 *  \param[in] fractionString Potential fractionString (X.X or X/X)
 *
 *   \return All essential information about fraction (if fraction.isWrong ==
false)
 */
fractionInfo_t makeIntoFraction(char* fractionString)
{
    char fractionSeparator;
    char fractionSeparatorString[2];

    struct fractionParts fraction;
    fractionInfo_t convertedFraction;

    printf("%s\n", fractionString);

    fractionSeparator = strpbrk(fractionString, "/,.")[0];
    fractionSeparatorString[0] = fractionSeparator;
    int posSeparator = (int) strcspn(fractionString, fractionSeparatorString) +
1;

    fraction = splitFraction(fractionString, posSeparator - 1);
```

```
    if (fraction.isWrong)
    {
        convertedFraction.isWrong = true;
        return convertedFraction;
    }

    // Заполнение информации о дроби
    convertedFraction.firstPart = fraction.firstPart;
    convertedFraction.secondPart = fraction.secondPart;
    convertedFraction.isNegative = fraction.isNegative;

    switch (fractionSeparator)
    {
    case '/':
    {
        convertedFraction.type = COMMON;
        if (convertedFraction.secondPart == 0)
            convertedFraction.isWrong = true;
        else
            convertedFraction.isWrong = false;
        return convertedFraction;
    }

    case ',':
    case '.':
    {
        convertedFraction.type = DECIMAL;
        convertedFraction.isWrong = false;
        return convertedFraction;
    }
    default:
    {
        convertedFraction.isWrong = true;
        return convertedFraction;
    }
    }
}

/*! \brief Calculates the highest power of the number. For example, 1274 = 1 *
10^3 + 2 * 10^2 + 7 * 10^1 + 4 * 10^0.
 * So, the highest power (10^3) is 4.
 *
```

```
 *  \param[in] number Source number
 *
 *  \return The highest power
 */
int calculateHighestPower(int number)
{
    int power = 0;

    while (abs(number) > 0)
    {
        number /= 10;
        power++;
    }
    return power;
}

/*! \brief Calculates the highest negative power of the decimal number. For
example, 12.74 = 1 * 10^1 + 2 * 10^0
 * + 7 * 10^-1 + 4 * 10^-2. So, the highest negative power (10^-2) is -2.
 *
 *  \param[in] number Source number
 *
 *  \return The highest negative power. Maximum value is MAX_DECIMAL_PART_LENGTH.
 */
int calculateHighestNegativePower(double number)
{
    int power = 0;
    while ((number - (int) number) != 0 && power <= MAX_DECIMAL_PART_LENGTH)
    {
        number *= 10;
        power++;
    }

    return power;
}

/*! \brief Finds and returns simple dividers of the number. For example, simple
dividers of 60 are 2, 2, 3, 5.
 *
 *  \param[in] number Positive integer number
 *
 *  \return Simple dividers of a number
```

```
 */
arrayInfo_t findSimpleDividers(int number)
{
    arrayInfo_t dividers;
    int length = 0;
    int i = 2;
    int j = 0;

    int numberCopy = number;

    while (numberCopy > 1)
    {
        while (numberCopy % i == 0)
        {
            numberCopy /= i;
            length++;
        }
        i++;
    }

    // !
    int* dividersArray = (int*) malloc(length * sizeof(int));
    i = 2;
    numberCopy = number;
    while (numberCopy > 1)
    {
        while (numberCopy % i == 0)
        {
            numberCopy /= i;
            dividersArray[j] = i;
            j++;
        }
        i++;
    }
    dividers.array = dividersArray;
    dividers.length = length;

    return dividers;
}

/*! \brief Subtract one array from another and returns the result. For example,
(3, 4, 2, 6, 2) - (2, 6) = (3, 4, 2)
```

```
 *
 *  \param[in] minuend The minuend array
 *  \param[in] subtrahend The subtrahend array
 *
 *  \return The residual array
 */
arrayInfo_t subtractArrays(arrayInfo_t minuend, arrayInfo_t subtrahend)
{
    arrayInfo_t residual;

    if (subtrahend.length == 0)
        return minuend;

    if (minuend.length == 0)
    {
        residual.array = NULL;
        residual.length = 0;
        return residual;
    }

    // Вспомогательный массив для определения, какие элементы удалять
    int minuendIndexes[minuend.length];
    for (int i = 0; i < minuend.length; i++)
    {
        minuendIndexes[i] = i;
    }

    int count = 0;
    for (int i = 0; i < subtrahend.length; i++)
    {
        for (int j = 0; j < minuend.length; j++)
        {
            if ((minuend.array[j] == subtrahend.array[i]) && (minuendIndexes[j]
!= -1))
            {
                minuendIndexes[j] = -1;
                count++;
                break;
            }
        }
    }
```

```
    int* residualArray = (int*) malloc((minuend.length - count) * sizeof(int));

    count = 0;
    for (int i = 0; i < minuend.length; i++)
        if (minuendIndexes[i] != -1)
        {
            residualArray[count] = minuend.array[i];
            count++;
        }

    residual.array = residualArray;
    residual.length = count;

    return residual;
}

/*! \brief Finds least common multiple of two numbers.
 *
 *  \param[in] number1 First number of LCM
 *  \param[in] number2 Second number of LCM
 *
 *  \return Least common multiple
 */
int findLCM(int number1, int number2)
{
    arrayInfo_t dividers1 = findSimpleDividers(number1);

    arrayInfo_t dividers2 = findSimpleDividers(number2);
    arrayInfo_t remainingDividers = subtractArrays(dividers2, dividers1);
    int LCM = 1;

    for (int i = 0; i < dividers1.length; i++)
        LCM *= dividers1.array[i];

    // Домножаю только неповторяющиеся значения
    for (int i = 0; i < remainingDividers.length; i++)
    {
        LCM *= remainingDividers.array[i];
    }

    return LCM;
}
```

```
/*! \brief Finds greatest common divisor of two numbers.
 *
 *  \param[in] number1 First number of GCD
 *  \param[in] number2 Second number of GCD
 *
 *  \return Greatest common divisor
 */
int findGCD(int number1, int number2)
{
    int GCD = 1;
    if (number1 == 0 || number2 == 0)
        return GCD;

    GCD = number1 * number2 / findLCM(number1, number2);
    return GCD;
}

/*! \brief Reduce fraction to a common denominator.
 *
 *  \param[in] fraction Source fraction
 *
 *  \return Nothing
 */
void reduceFraction(fractionInfo_t* fraction)
{
    if (fraction->type == DECIMAL)
    {
        fraction->isWrong = true;
        return;
    }

    int GCD = findGCD(fraction->firstPart, fraction->secondPart);
    fraction->firstPart /= GCD;
    fraction->secondPart /= GCD;
}

/*! \brief Transforms fraction into decimal number.
 *
 *  \param[in] fraction Source fraction
 *
 *  \return Decimal (double) format of fraction
```

```
 */
double toDouble(fractionInfo_t fraction)
{
    double doubleFraction;
    if (fraction.type == COMMON)
        doubleFraction    =    (double)    fraction.firstPart    /    (double)
fraction.secondPart;
    else
        doubleFraction = (double) fraction.firstPart +
                         (double)       fraction.secondPart       /       pow(10,
calculateHighestPower(fraction.secondPart));

    if (fraction.isNegative)
        doubleFraction *= -1;

    return doubleFraction;
}

/*! \brief Performs arithmetic operations on two fractions. Possible operations
are '+', '-', '*', '/'. Fractions
 * can be decimal or common. If both are common, the result is common, too. In
other cases it is decimal.
 *
 *  \param[in] fraction1 First part of arithmetic operation
 *  \param[in] fraction2 Second part of arithmetic operation
 *  \param[in] operation Operation type ('+', '-', '*', or '/')
 *
 *  \return Result of arithmetic operation
 */
fractionInfo_t  calculate(fractionInfo_t  fraction1,  fractionInfo_t  fraction2,
char* operation)
{
    enum Case
    {
        Plus, Minus, Multiply, Divide
    };

    enum Case operationCode;
    if (strcmp(operation, "+") == 0)
        operationCode = Plus;
    else if (strcmp(operation, "-") == 0)
        operationCode = Minus;
```

```
else if (strcmp(operation, "*") == 0)
    operationCode = Multiply;
else
    operationCode = Divide;

fractionInfo_t result;
if (fraction1.type == COMMON && fraction2.type == COMMON)
{
    int LCM = findLCM(fraction1.secondPart, fraction2.secondPart);
    result.type = COMMON;

    // Учет знака
    if (fraction1.isNegative)
        fraction1.firstPart *= -1;

    if (fraction2.isNegative)
        fraction2.firstPart *= -1;


    switch (operationCode)
    {
    case Plus:
        result.firstPart = fraction1.firstPart * LCM / fraction1.secondPart +
                           fraction2.firstPart * LCM / fraction2.secondPart;
        result.secondPart = LCM;
        break;

    case Minus:
        result.firstPart = fraction1.firstPart * LCM / fraction1.secondPart -
                           fraction2.firstPart * LCM / fraction2.secondPart;
        result.secondPart = LCM;
        break;

    case Multiply:
        result.firstPart = fraction1.firstPart * fraction2.firstPart;
        result.secondPart = fraction1.secondPart * fraction2.secondPart;
        break;

    case Divide:
        result.firstPart = fraction1.firstPart * fraction2.secondPart;
        result.secondPart = fraction1.secondPart * fraction2.firstPart;
        break;
```

```
        }

        if ((result.firstPart < 0 && result.secondPart > 0) || (result.firstPart
> 0 && result.secondPart < 0))
            result.isNegative = true;
        else
            result.isNegative = false;

        result.firstPart = abs(result.firstPart);
        result.secondPart = abs(result.secondPart);

        result.isWrong = false;

        reduceFraction(&result);
        return result;
    }
    else
    {
        result.type = DECIMAL;

        double decimal1 = toDouble(fraction1);
        double decimal2 = toDouble(fraction2);

        double resultDecimal;

        switch (operationCode)
        {
        case Plus:
            resultDecimal = decimal1 + decimal2;
            break;

        case Minus:
            resultDecimal = decimal1 - decimal2;
            break;

        case Multiply:
            resultDecimal = decimal1 * decimal2;
            break;

        case Divide:
            resultDecimal = decimal1 / decimal2;
            break;
```

```
        }

        if (resultDecimal < 0)
        {
            result.isNegative = true;
            resultDecimal *= -1;
        }
        else
            result.isNegative = false;

        int negativePower = calculateHighestNegativePower(resultDecimal);

        result.firstPart = (int) resultDecimal;
        result.secondPart = (int) (resultDecimal * pow(10, negativePower) -
                                   (int) resultDecimal * pow(10, negativePower));

        result.isWrong = false;
        return result;
    }
}

/*! \brief Compares two fractions. Possible comparison operations are '>', '<',
'=', '!=', '>=', '<='.
 * Fractions can be decimal or common. Returns true or false.
 *
 *  \param[in] fraction1 First part of comparison
 *  \param[in] fraction2 Second part of comparison
 *  \param[in] operation Operation type ('>', '<', '=', '!=', '>=', or '<=')
 *
 *  \return Result of comparison (true or false)
 */
int compare(fractionInfo_t fraction1, fractionInfo_t fraction2, char* operation)
{
    enum Case
    {
        Lesser, Greater, Equal, NotEqual, GreaterEqual, LesserEqual
    };

    enum Case operationCode;
    if (strcmp(operation, "<") == 0)
        operationCode = Lesser;
    else if (strcmp(operation, ">") == 0)
```

```
        operationCode = Greater;
    else if (strcmp(operation, "=") == 0)
        operationCode = Equal;
    else if (strcmp(operation, "!=") == 0)
        operationCode = NotEqual;
    else if (strcmp(operation, ">=") == 0)
        operationCode = GreaterEqual;
    else
        operationCode = LesserEqual;

    double decimal1 = toDouble(fraction1);
    double decimal2 = toDouble(fraction2);

    switch (operationCode)
    {
    case Lesser:
        return decimal1 < decimal2;

    case Greater:
        return decimal1 > decimal2;

    case Equal:
        return decimal1 == decimal2;

    case NotEqual:
        return decimal1 != decimal2;

    case GreaterEqual:
        return decimal1 >= decimal2;

    case LesserEqual:
        return decimal1 <= decimal2;
    }
}
```

На листинге 2 представлен код программы socketOperations.c.

Листинг 2 – Код программы с операциями по управлению TCP-сокетами

```
/*! \file    socketOperations.c
 *  \brief   Working with TCP sockets.
 *
 *  \details Has a various amount of TCP socket operating functions like server
and client creation
```

```
 *  and sending data between them.
 *
 *  \author Nikitin Alexander, KI19-17/1Б
 */

#include <string.h>
#include <stdlib.h>
#include <sys/un.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdbool.h>

#include "socketOperations.h"

/*! \brief Writes the buffer content into socket.
 *
 *  \param[in] s File descriptor of socket
 *  \param[in] buf The information to be written
 *  \param[in] len Length of this information
 *
 *  \return Is the operation successful.
 */
bool sendSocketBuf(int s, void* buf, int len)
{
    int writeError = send(s, buf, len, 0);
    if (writeError == -1)
    {
        perror("Write length error");
        return false;
    }
    return true;
}

/*! \brief Writes the text into socket.
 *
 *  \param[in] socketFileDescriptor File descriptor of socket
 *  \param[in] text Text that will be written into socket
 *
 *  \return Nothing.
 */
```

```
void sendSocketText(int socketFileDescriptor, char* text)
{
    int length = strlen(text) + 1;

    // Записываем длину
    sendSocketBuf(socketFileDescriptor, &length, sizeof(int));
    // Записываем строку
    sendSocketBuf(socketFileDescriptor, text, length);
}

/*! \brief Read the information from socket and saves it into the buffer.
 *
 *  \param[in] s File descriptor of socket
 *  \param[out] buf The information to be read
 *  \param[in] len Length of this information
 *
 *  \return Is the operation successful.
 */
bool readSocketBuf(int s, void* buf, int len)
{
    int recvError = recv(s, buf, len, 0);
    if (recvError == -1)
    {
        perror("Socket read error");
        return false;
    }
    return true;
}

/*! \brief Read the text from socket.
 *
 *  \param[in] clientSocketFileDescriptor File descriptor of socket
 *  \param[out] text Text that where the information from socket will be written
 *
 *  \return Is the operation successful.
 */
bool receiveSocketText(int clientSocketFileDescriptor, char** text)
{
    int length;

    if (!readSocketBuf(clientSocketFileDescriptor, &length, sizeof(int)))
        return false;
```

```
    char temp[length];
    *text = malloc(sizeof(char) * length);

    if (!readSocketBuf(clientSocketFileDescriptor, temp, length))
        return false;

    strcpy(*text, temp);

    return true;
}

/*! \brief Creates a TCP socket for a client part.
 *
 *  \return File descriptor of the socket.
 */
int createClientTCPSocket()
{
    int socketFileDescriptor = socket(AF_INET, SOCK_STREAM, 0);
    if (socketFileDescriptor == -1)
    {
        perror("Socket creation error");
        return -1;
    }
    return socketFileDescriptor;
}

/*! \brief Creates a TCP socket for a server part.
 *
 *  \return File descriptor of the socket.
 */
int createServerTCPSocket()
{
    int socketFileDescriptor = socket(AF_INET, SOCK_STREAM, 0);
    int i = 1;
    int setsockError = setsockopt(socketFileDescriptor, SOL_SOCKET, SO_REUSEADDR,
                                  (const char*) &i, sizeof(i)
    );

    if (socketFileDescriptor == -1 || setsockError == -1)
    {
        perror("Socket creation error");
```

```
        return -1;
    }
    return socketFileDescriptor;
}

/*! \brief Creates the name to the TCP socket and binds it to it.
 *
 *  \param[in] socketFileDescriptor File descriptor of socket
 *  \param[in] port TCP port
 *
 *  \return Is the operation successful.
 */
bool bindTCPSocket(int socketFileDescriptor, char* port)
{
    int portNumber = strtol(port, NULL, 10);

    struct sockaddr_in name;
    name.sin_family = AF_INET;
    name.sin_port = htons((u_short) portNumber);
    name.sin_addr.s_addr = INADDR_ANY;

    int bindError = bind(socketFileDescriptor, (const struct sockaddr*) &name,
sizeof(name));
    if (bindError == -1)
    {
        perror("Bind error");
        return false;
    }

    return true;
}

/*! \brief Connects socket to the server.
 *
 *  \param[in] socketFileDescriptor File descriptor of socket
  * \param[in] serverIP An IP to connect the socket
 *  \param[in] port TCP port
 *
 *  \return Is the operation successful.
 */
bool connectTCPSocket(int socketFileDescriptor, char* serverIP, char* port)
{
```

```
    struct sockaddr_in name;
    memset((char*) &name, 0, sizeof(name));

    name.sin_family = AF_INET;
    name.sin_addr.s_addr = inet_addr(serverIP);

    int portNumber = strtol(port, NULL, 10);

    if (name.sin_addr.s_addr == INADDR_NONE)
    {
        puts("Incorrect IP address!");
        return EXIT_FAILURE;
    }
    name.sin_port = htons((u_short) portNumber);

    int connectionError = connect(socketFileDescriptor, (struct sockaddr*) &name,
(socklen_t)
    sizeof(name));
    if (connectionError == -1)
    {
        perror("Connection error");
        return false;
    }

    return true;
}

/*! \brief Switch server into the ready-to-listen to sockets state.
 *
 *  \param[in] socketFileDescriptor File descriptor of socket
 *
 *  \return Is the operation successful.
 */
bool serverListen(int socketFileDescriptor)
{
    int listenError = listen(socketFileDescriptor, BACKLOG_NUMBER);
    if (listenError == -1)
    {
        perror("Listen error");
        return false;
    }
```

```
    return true;
}

/*! \brief Switch server into the waiting-for-sockets state.
 *
 *  \param[in] socketFileDescriptor File descriptor of socket
 *
 *  \return Is the operation successful.
 */
int acceptTCPSocket(int socketFileDescriptor)
{

    struct sockaddr_in clientName;
    socklen_t clientNameLength = sizeof(clientName);

    int clientSocketFileDescriptor = accept(socketFileDescriptor, (struct
sockaddr*) &clientName, &clientNameLength
    );

    if (clientSocketFileDescriptor == -1)
    {
        perror("Accept error");
        return -1;
    }

    return clientSocketFileDescriptor;
}
```

На листинге 3 представлен код программы client.c.

Листинг 3 – Код клиентской части программы

```
/*! \file    client.c
 *  \brief   client part of project.
 *
 *  \details Sends the info to server and takes the processed info.
 *
 *  \author Nikitin Alexander, KI19-17/1Б
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
```

```
#include <stdbool.h>
#include <ctype.h>

#include "client.h"
#include "socketOperations.h"
#include "signals.h"
#include "log.h"

#define RADIX 10
#define NUMBER_OF_COMPARISON_OPERATORS 10
#define MAX_LENGTH_COMPARISON_OPERATORS 2

/*! \brief Strrev function from <string.h> which is not available in Linux.
 *
 *  \param[in] str Source string
 *
 *  \return Reversed string
 */
char* strrev(char* str)
{
    if (!str || !*str)
        return str;

    int i = strlen(str) - 1, j = 0;

    char ch;
    while (i > j)
    {
        ch = str[i];
        str[i] = str[j];
        str[j] = ch;
        i--;
        j++;
    }
    return str;
}

/*! \brief Inputs the string from user console into the variable
 *
 *  \param[out] word String variable of arbitrary length
 *
 *  \return Nothing
```

```
 */
void inputString(char** word)
{
    int count = 0;
    char inputChar = 0;

    fflush(stdin);
    *word = NULL;

    while (1)
    {
        inputChar = getchar();
        if (inputChar == '\n')
            break;
        else
        {
            *word = realloc(*word, count + 2);
            (*word)[count] = inputChar;
            count++;
        }
    }
    (*word)[count] = '\0';
}

/*! \brief Checks whether the string is fraction-type (x/x, x.x) or not.
 *
 *  \param[in] string Verifiable string
 *
 *  \return Is the string fraction-type or not
 */
bool checkFraction(char* string)
{
    char* endPtr = NULL;

    strtol(string, &endPtr, RADIX);
    if (strcmp(string, endPtr) == 0 || (endPtr[0] != '.' && endPtr[0] != '/' &&
endPtr[0] != ','))  // Если строка не
        // начинается с символа и первая часть не заканчивается разделителем
        return false;

    strcpy(string, strrev(string));
    strtol(string, &endPtr, RADIX);
```

```
    if (strcmp(string, endPtr) == 0 || (endPtr[0] != '.' && endPtr[0] != '/' &&
endPtr[0] != ','))  // Если строка не
        // начинается с символа и первая часть не заканчивается разделителем
        return false;

    strrev(string);
    return true;
}

/*! \brief Checks if the operation type is supportable in the program.
* List of operations: ("+", "-", "*", "/", "<", ">", "=", "!=", ">=", "<=")
*
*  \param[in] operation Operation string
*
*  \return If the operator is appropriate (true or false)
*/
bool checkOperation(char* operation)
{
    char
allOperations[NUMBER_OF_COMPARISON_OPERATORS][MAX_LENGTH_COMPARISON_OPERATORS  +
1]
            = {"+", "-", "*", "/", "<", ">", "=", "!=", ">=", "<="};

    for (int i = 0; i < sizeof(allOperations) / sizeof(*allOperations); i++)
        if (strcmp(allOperations[i], operation) == 0)
            return true;
    return false;
}

/*! \brief Checks if the string is a natural number
 *
 * \param[in] string String that may be a number or not
 *
 *  \return true or false
 */
bool checkNat(char* string)
{
    for (int i = 0; string[i] != '\0'; i++)
        if (!isdigit(string[i]))
            return false;

    if (strtol(string, NULL, 10) <= 0)
```

```
        return false;
    return true;
}

/*! \brief Client main function
 *  \param argc  Number of command line arguments
 *  \param argv  An array of command line arguments.
 *               argv[0] - the program name,
 *               argv[1] - a socket IP-address,
 *               argv[2] - a socket port number,
 *               argv[3] - log file name,
 *               argv[4] - server existence time without activity in seconds.
 *  \return Integer 0 upon exit success,
 *          or EXIT_FAILURE otherwise.
 */
int main(int argc, const char* argv[])
{
    registerHandler();
    char* firstFraction = NULL;
    char* secondFraction = NULL;
    char* operationSign = NULL;

    if (argc != 5)
    {
        puts("Socket name, port number, log file and client existence time
expected.\n");
        return EXIT_FAILURE;
    }

    if (!checkNat((char*) argv[2]))
    {
        puts("Incorrect port number expected.\n");
        return EXIT_FAILURE;
    }

    if (strchr(argv[3], '/') != NULL)
    {
        puts("Incorrect log name.\n");
        return EXIT_FAILURE;
    }

    if (!checkNat((char*) argv[4]))
```

```
{
    puts("Incorrect time.\n");
    return EXIT_FAILURE;
}

char* serverIP = (char*) argv[1];
char* port = (char*) argv[2];

char* logDirectoryPath = "./logs";
char logPath[strlen(logDirectoryPath) + strlen(argv[3]) + 1];
strcpy(logPath, logDirectoryPath);
strcat(logPath, "/");
strcat(logPath, argv[3]);

int time = strtol(argv[4], NULL, 10);

int socketFileDescriptor;
while (true)
{
    setTimerClient(time);
    do
    {
        puts("Input first fraction:");
        inputString(&firstFraction);
    } while (!checkFraction(firstFraction));

    do
    {
        puts("Input operation:");
        inputString(&operationSign);
    } while (!checkOperation(operationSign));

    do
    {
        puts("Input second fraction:");
        inputString(&secondFraction);
    } while (!checkFraction(secondFraction));

    socketFileDescriptor = createClientTCPSocket();
    connectTCPSocket(socketFileDescriptor, serverIP, port);

    sendSocketText(socketFileDescriptor, firstFraction);
```

```
        sendSocketText(socketFileDescriptor, operationSign);
        sendSocketText(socketFileDescriptor, secondFraction);

        char    buffer[strlen(firstFraction)    +    strlen(operationSign)    +
strlen(secondFraction) + 3];
        strcpy(buffer, firstFraction);
        strcat(buffer, " ");
        strcat(buffer, operationSign);
        strcat(buffer, " ");
        strcat(buffer, secondFraction);
        strcat(buffer, "\n");

        writeLog(logPath, buffer);

        char* info = NULL;
        receiveSocketText(socketFileDescriptor, &info);
        printf("\n%s\n", info);
        free(info);

        close(socketFileDescriptor);
    }
    return 0;
}
```

На листинге 4 представлен код программы server.c.

Листинг 4 – Код серверной части программы

```
/*! \file    server.c
 *  \brief   Server part of project.
 *
 *  \details Accepts the info from client, process it and returns back.
 *
 *  \author Nikitin Alexander, KI19-17/1Б
 */

#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <stdbool.h>
#include <ctype.h>
#include <stdlib.h>

#include "algorithm.h"
```

```
#include "server.h"
#include "socketOperations.h"
#include "signals.h"
#include "log.h"

/*! \brief Checks if the string is a natural number
 *
 * \param[in] string String that may be a number or not
 *
 *  \return true or false
 */
bool checkNat(char* string)
{
    for (int i = 0; string[i] != '\0'; i++)
        if (!isdigit(string[i]))
            return false;

    if (strtol(string, NULL, 10) <= 0)
        return false;
    return true;
}

/*! \brief Server main function
 *  \param[in] argc  Number of command line arguments
 *  \param[in] argv  An array of command line arguments.
 *               argv[0] - the program name,
 *               argv[1] - the port number,
 *               argv[2] - log file name,
 *               argv[3] - server existence time without activity in seconds.
 *  \return Integer 0 upon exit success,
 *          or EXIT_FAILURE otherwise.
 */
int main(int argc, const char* argv[])
{
    registerHandler();

    char* socketFirstFraction = NULL;
    char* socketOperation = NULL;
    char* socketSecondFraction = NULL;

    fractionInfo_t firstFraction;
    fractionInfo_t secondFraction;
```

```
fractionInfo_t result;

if (argc != 4)
{
    puts("Port number, log file and server existence time expected.\n");
    return EXIT_FAILURE;
}

if (!checkNat((char*) argv[1]))
{
    puts("Incorrect port number.\n");
    return EXIT_FAILURE;
}

if (strchr(argv[2], '/') != NULL)
{
    puts("Incorrect log name.\n");
    return EXIT_FAILURE;
}

if (!checkNat((char*) argv[3]))
{
    puts("Incorrect time.\n");
    return EXIT_FAILURE;
}

puts("Server is running.");

char* port = (char*) argv[1];

char* logDirectoryPath = "./logs";
char logPath[strlen(logDirectoryPath) + strlen(argv[2]) + 1];
strcpy(logPath, logDirectoryPath);
strcat(logPath, "/");
strcat(logPath, argv[2]);

int time = strtol(argv[3], NULL, 10);

int socketFileDescriptor = createServerTCPSocket();

bindTCPSocket(socketFileDescriptor, port);
```

```
    serverListen(socketFileDescriptor);
    puts("Server is listening.");

    int clientSocketFileDescriptor;
    while (true)
    {
        setTimerServer(time);
        clientSocketFileDescriptor = acceptTCPSocket(socketFileDescriptor);
        printf("The socket has been connected to the server.\n");

        receiveSocketText(clientSocketFileDescriptor, &socketFirstFraction);
        printf("The information from socket has been received. Content: %s\n",
socketFirstFraction);
        receiveSocketText(clientSocketFileDescriptor, &socketOperation);
        printf("The information from socket has been received. Content: %s\n",
socketOperation);
        receiveSocketText(clientSocketFileDescriptor, &socketSecondFraction);
        printf("The information from socket has been received. Content: %s\n",
socketSecondFraction);

        firstFraction = makeIntoFraction(socketFirstFraction);
        secondFraction = makeIntoFraction(socketSecondFraction);

        char* sentInfo = malloc( sizeof (char) * (sizeof (socketFirstFraction) +
sizeof (socketOperation)
                                                  +                       sizeof
(socketSecondFraction) + 40));

        if (firstFraction.isWrong || secondFraction.isWrong)
        {
            sprintf(sentInfo, "%s", "Incorrect fraction information!");
            continue;
        }

        if (strcmp(socketOperation, "+") == 0 || strcmp(socketOperation, "-") ==
0 ||
            strcmp(socketOperation, "/") == 0 || strcmp(socketOperation, "*") ==
0)
        {
            result = calculate(firstFraction, secondFraction, socketOperation);

            if (result.type == COMMON)
```

```
                if (result.isNegative)
                    sprintf(sentInfo, "%s %s %s = -%d/%d\n", socketFirstFraction,
socketOperation,
                            socketSecondFraction,              result.firstPart,
result.secondPart);
                else
                    sprintf(sentInfo, "%s %s %s = %d/%d\n", socketFirstFraction,
socketOperation,
                            socketSecondFraction,              result.firstPart,
result.secondPart);
            else
                if (result.isNegative)
                    sprintf(sentInfo, "%s %s %s = -%d.%d\n", socketFirstFraction,
socketOperation,
                            socketSecondFraction,              result.firstPart,
result.secondPart);
                else
                    sprintf(sentInfo, "%s %s %s = %d.%d\n", socketFirstFraction,
socketOperation,
                            socketSecondFraction,              result.firstPart,
result.secondPart);
        }
        else if (compare(firstFraction, secondFraction, socketOperation))
            sprintf(sentInfo,   "%s   %s   %s:   True\n",   socketFirstFraction,
socketOperation,
                     socketSecondFraction);
        else
            sprintf(sentInfo,   "%s   %s   %s:   False\n",   socketFirstFraction,
socketOperation,
                    socketSecondFraction);

        printf("\n%s\n", sentInfo);
        writeLog(logPath, sentInfo);
        sendSocketText(clientSocketFileDescriptor, sentInfo);

        free(socketFirstFraction);
        free(socketOperation);
        free(socketSecondFraction);
        free(sentInfo);

        close(clientSocketFileDescriptor);
    }
```

```
    return 0;
}
```

На листинге 5 представлен код программы signals.c.

Листинг 5 – Код программы по обработке сигналов

```
/*! \file    signals.c
 *  \brief   Working with signals.
 *
 *  \details Have functions of processing different signals and setting out the
timer for program existence.
 *
 *  \author Nikitin Alexander, KI19-17/1Б
 */

#include <sys/time.h>
#include <signal.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#include "signals.h"
#include "log.h"

const char* g_errorLogPath = "./logs/errors.log";

/*! \brief Crashes handler.
 *
 *  \param   signo The number of signal to handle
 *
 *  \return Nothing
 */
void crashHandler(int signo)
{
    if (signo == SIGABRT)
    {
        printf("Abort has happened!\n");
        writeLog((char*) g_errorLogPath, "Abort has happened.");
    }
    else if (signo == SIGFPE)
    {
        printf("Arithmetical operation error has happened!\n");
```

```
        writeLog((char*)  g_errorLogPath,  "Arithmetical  operation  error  has
happened.");
    }
    else if (signo == SIGBUS)
    {
        printf("Hardware error has happened!\n");
        writeLog((char*) g_errorLogPath, "Hardware error has happened.");
    }
    else if (signo == SIGILL)
    {
        printf("Not permitted instruction error has happened!\n");
        writeLog((char*)  g_errorLogPath,  "Not  permitted  instruction  error  has
happened.");
    }
    else if (signo == SIGSEGV)
    {
        printf("Segmentation error has happened!\n");
        writeLog((char*) g_errorLogPath, "Segmentation error has happened.");
    }
    else if (signo == SIGSYS)
    {
        printf("Impermissible system call error has happened!\n");
        writeLog((char*)  g_errorLogPath,  "Impermissible  system  call  error  has
happened.");
    }
    else if (signo == SIGTRAP)
    {
        printf("Breakpoint has been reached!\n");
        writeLog((char*) g_errorLogPath, "Breakpoint has been reached.");
    }
    else if (signo == SIGXCPU)
    {
        printf("Process resources limit has been exceeded!\n");
        writeLog((char*)  g_errorLogPath,  "Process  resources  limit  has  been
exceeded.");
    }
    else if (signo == SIGXFSZ)
    {
        printf("File resources limit has been exceeded!\n");
        writeLog((char*)  g_errorLogPath,  "File  resources  limit  has  been
exceeded.");
    }
```

```
    else
    {
        fprintf(stderr, "Unexpected signal!\n");
        exit(ERROR_CODE);
    }
    exit(SUCCESS_CODE);
}

/*! \brief Handles handle function to all crash signals.
 *
 *  \return Nothing
 */
void registerHandler()
{
    if (signal(SIGABRT, crashHandler) == SIG_ERR)
    {
        fprintf(stderr, "It's impossible to handle SIGABRT!\n");
        writeLog((char*) g_errorLogPath, "It's impossible to handle SIGABRT.");
        exit (ERROR_CODE);
    }
    if (signal(SIGFPE, crashHandler) == SIG_ERR)
    {
        fprintf(stderr, "It's impossible to handle SIGFPE!\n");
        writeLog((char*) g_errorLogPath, "It's impossible to handle SIGFPE.");
        exit (ERROR_CODE);
    }
    if (signal(SIGBUS, crashHandler) == SIG_ERR)
    {
        fprintf(stderr, "It's impossible to handle SIGBUS!\n");
        writeLog((char*) g_errorLogPath, "It's impossible to handle SIGBUS.");
        exit (ERROR_CODE);
    }
    if (signal(SIGILL, crashHandler) == SIG_ERR)
    {
        fprintf(stderr, "It's impossible to handle SIGILL!\n");
        writeLog((char*) g_errorLogPath, "It's impossible to handle SIGILL.");
        exit (ERROR_CODE);
    }
    if (signal(SIGSEGV, crashHandler) == SIG_ERR)
    {
        fprintf(stderr, "It's impossible to handle SIGSEGV!\n");
        writeLog((char*) g_errorLogPath, "It's impossible to handle SIGSEGV.");
```

```
        exit (ERROR_CODE);
    }
    if (signal(SIGSYS, crashHandler) == SIG_ERR)
    {
        fprintf(stderr, "It's impossible to handle SIGSYS!\n");
        writeLog((char*) g_errorLogPath, "It's impossible to handle SIGSYS.");
        exit (ERROR_CODE);
    }
    if (signal(SIGTRAP, crashHandler) == SIG_ERR)
    {
        fprintf(stderr, "It's impossible to handle SIGTRAP!\n");
        writeLog((char*) g_errorLogPath, "It's impossible to handle SIGTRAP.");
        exit (ERROR_CODE);
    }
    if (signal(SIGXCPU, crashHandler) == SIG_ERR)
    {
        fprintf(stderr, "It's impossible to handle SIGXCPU!\n");
        writeLog((char*) g_errorLogPath, "It's impossible to handle SIGXCPU.");
        exit (ERROR_CODE);
    }
    if (signal(SIGXFSZ, crashHandler) == SIG_ERR)
    {
        fprintf(stderr, "It's impossible to handle SIGXFSZ!\n");
        writeLog((char*) g_errorLogPath, "It's impossible to handle SIGXFSZ.");
        exit (ERROR_CODE);
    }
}

/*! \brief Server timer handler.
 *
 *  \param signo The number of signal to handle
 *
 *  \return Nothing
 */
void serverExpirationHandler(int signum)
{
    puts("Server is shutting down.");
    exit(EXPIRATION_EXIT_CODE);
}

/*! \brief Client timer handler.
 *
```

```
 *  \param signo The number of signal to handle
 *
 *  \return Nothing
 */
void clientExpirationHandler(int signum)
{
    puts("Client has stopped working.");
    exit(EXPIRATION_EXIT_CODE);
}

/*! \brief Sets a time to a server and shuts down the server after the expiration
of it.
 *
 *  \param sec The time for server to survive
 *
 *  \return Nothing
 */
void setTimerServer(int sec)
{
    struct sigaction sa;
    struct itimerval timer;

    memset (&sa, 0, sizeof (sa));
    sa.sa_handler = &serverExpirationHandler;
    sigaction (SIGALRM, &sa, NULL);

    timer.it_value.tv_sec = sec;
    timer.it_value.tv_usec = SERVER_INACTIVITY_TIME_MS;

    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, NULL);
}

/*! \brief Sets a time to a client and kills him after the expiration of it.
 *
 *  \param sec The time for client to survive
 *
 *  \return Nothing
 */
void setTimerClient(int sec)
```

```
{
    struct sigaction sa;
    struct itimerval timer;

    memset (&sa, 0, sizeof (sa));
    sa.sa_handler = &clientExpirationHandler;
    sigaction (SIGALRM, &sa, NULL);

    timer.it_value.tv_sec = sec;
    timer.it_value.tv_usec = CLIENT_INACTIVITY_TIME_MS;

    timer.it_interval.tv_sec = 0;
    timer.it_interval.tv_usec = 0;

    setitimer(ITIMER_REAL, &timer, NULL);
}
```

На листинге 6 представлен код программы log.c.

Листинг 6 – Код программы по работе с log-файлами

```
/*! \file    log.c
 *  \brief   Working with log files.
 *
 *  \details Writes the information into log files.
 *
 *  \author Nikitin Alexander, KI19-17/1Б
 */

#include <fcntl.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <time.h>

/*! \brief Writes the information into the log file adding current time.
 *
 *  \param[in] logPath Path to the file to write
 *  \param[in] info Information to be written
 *
 *  \return Is the operation successful.
 */
```

```
void writeLog(char* logPath, char* info)
{
    time_t rawtime;
    struct tm * timeinfo;

    time ( &rawtime );
    timeinfo = localtime ( &rawtime );
    char* timeStr = asctime (timeinfo);
    timeStr[strlen(timeStr) - 1] = ':';

    int fd = open(logPath, O_WRONLY | O_CREAT | O_APPEND, 0660);
    write(fd, timeStr, strlen(timeStr));
    write(fd, "\n", 1);
    write(fd, info, strlen(info));
    write(fd, "\n", 1);
    fsync(fd);
    close(fd);
}
```

4 Содержимое файла configure.ac

На листинге 5 представлено содержимое configure.

Листинг 4 – Файл configure.ac

```
#                                               -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.

AC_PREREQ([2.69])
AC_INIT([FULL-PACKAGE-NAME], [VERSION], [BUG-REPORT-ADDRESS])
AC_CONFIG_SRCDIR([algorithm.c])
AC_CONFIG_HEADERS([config.h])

# Checks for programs.
AC_PROG_CXX
AC_PROG_CC

# Checks for libraries.
# FIXME: Replace `main' with a function in `-lm':
AC_CHECK_LIB([m], [main])

# Checks for header files.
```

```
AC_CHECK_HEADERS([arpa/inet.h   netinet/in.h   stdlib.h   string.h   sys/socket.h
unistd.h])

# Checks for typedefs, structures, and compiler characteristics.
AC_CHECK_HEADER_STDBOOL

# Checks for library functions.
AC_FUNC_MALLOC
AC_FUNC_REALLOC
AC_CHECK_FUNCS([memset pow socket strchr strcspn strpbrk strtol])

AC_CONFIG_FILES([Makefile
                 cmake-build-debug/Makefile])
AC_OUTPUT
```

5 Тестовые примеры работы программы

Рисунок 1 – Запуск двух клиентов и сервера с аргументами командной строки

```
root@brain:/mnt/lab6# ./client1.o 127.0.0.1 1234 client.log 200
Input first fraction:
1/3
Input operation:
+
Input second fraction:
1/2

1/3 + 1/2 = 5/6
root@brain:/mnt/lab6# ./client.o 127.0.0.1 1234 client.log 200
Input first fraction:
1/4
Input operation:
+
Input second fraction:
1/6

1/4 + 1/6 = 5/12
root@brain:/mnt/lab6# ./server.o 1234 server.log 300
Server is running.
Server is listening.
The socket has been connected to the server.
The information from socket has been received. Content: 1/3
The information from socket has been received. Content: +
The information from socket has been received. Content: 1/2

1/3 + 1/2 = 5/6

The socket has been connected to the server.
The information from socket has been received. Content: 1/4
The information from socket has been received. Content: +
The information from socket has been received. Content: 1/6

1/4 + 1/6 = 5/12
```

Рисунок 2 – Одновременная обработка сервером двух клиентов

```
Input first fraction:
1/3
Input operation:
+
Input second fraction:
23/45

1/3 + 23/45 = 38/45

Input first fraction:
```

Рисунок 3 – Сложение обыкновенных дробей

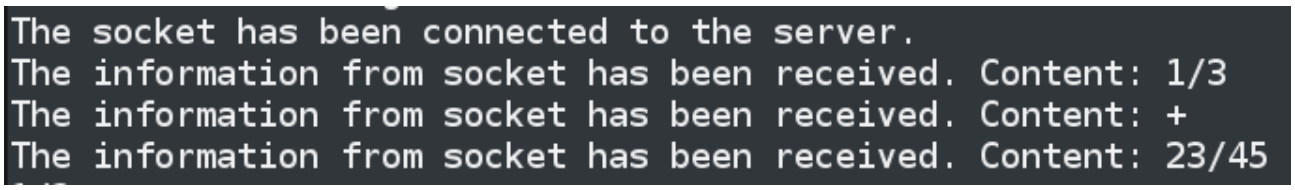

Рисунок 4 – Серверный вывод информации о полученном сокете

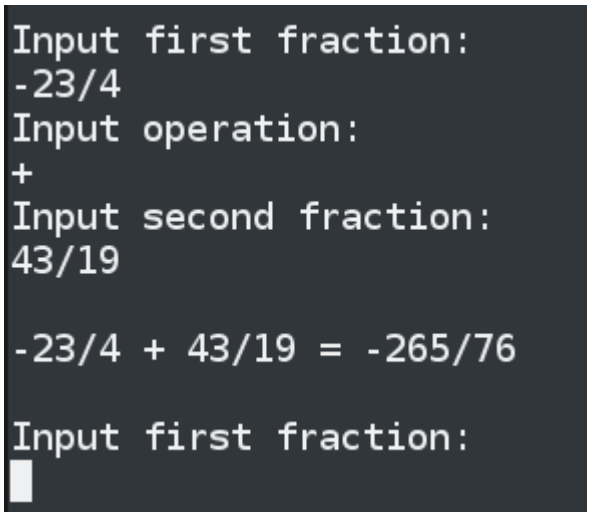

Рисунок 5 – Ввод и обработка отрицательных значений

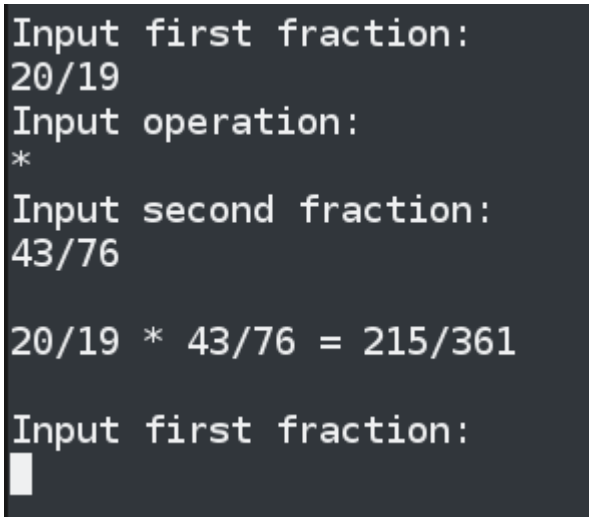

Рисунок 6 – Умножение обыкновенных дробей

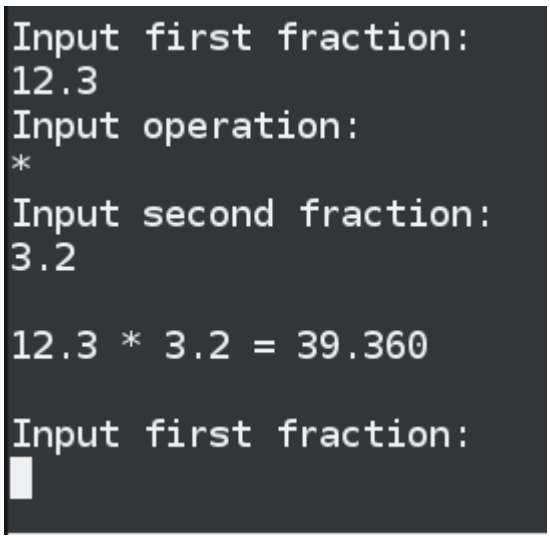

Рисунок 7 – Умножение десятичных дробей

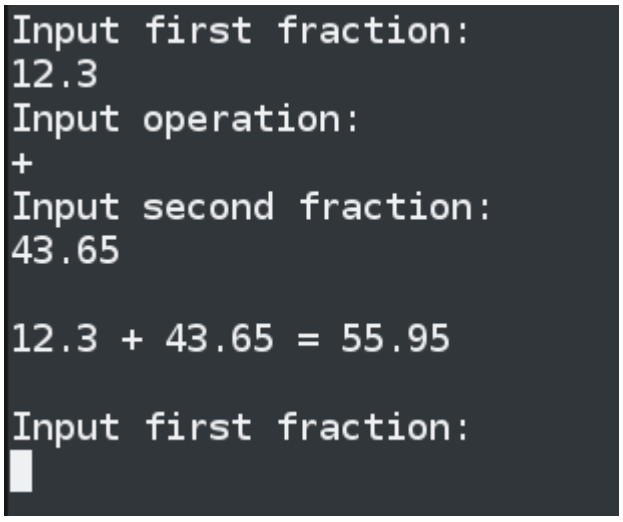

Рисунок 8 – Сложение десятичных дробей

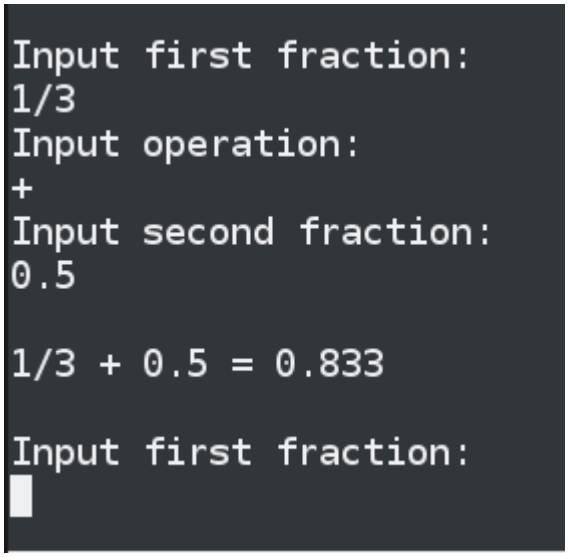

Рисунок 9 – Смешанное сложение дробей

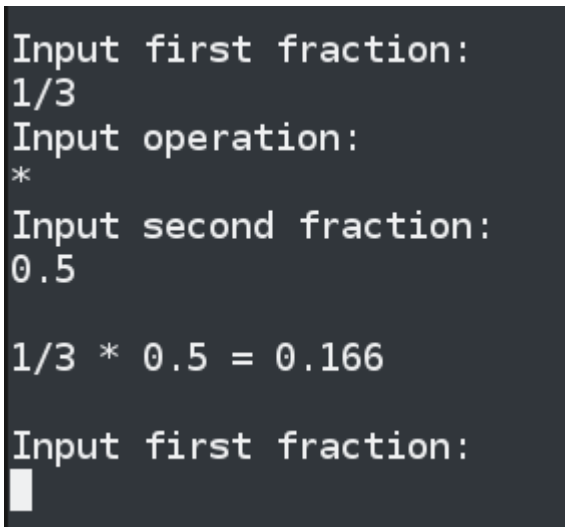

Рисунок 10 – Смешанное произведение дробей