Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Санкт-Петербургский государственный политехнический университет»

Институт компьютерных наук и кибербезопасности

**Отчет по Практической работе №2**

по дисциплине

«Технология разработки программного обеспечения»

на тему

«Разработка клиент-серверного приложения»

Выполнила студентка группы 5130203/30001:

Чичкан Елизавета Владимировна

Преподаватель:

Силиненко Александр Витальевич

:

Санкт-Петербург

2024

**СОДЕРЖАНИЕ**

[**Введение** 3](#_Toc187671896)

[**1.** **Общее описание задачи** 4](#_Toc187671897)

[**2.** **Описание интерфейса** 4](#_Toc187671898)

[**3.** **Функциональные требования** 5](#_Toc187671899)

[**3.1. Подключение** 5](#_Toc187671900)

[**3.2. Обмен данными** 6](#_Toc187671901)

[**3.3. Обработка ошибок** 6](#_Toc187671902)

[**4.** **Требования к тестированию** 6](#_Toc187671903)

[**5.** **Проектирование** 6](#_Toc187671904)

[**6.** **Создание программы** 9](#_Toc187671905)

[**6.1. Проверка корректности примера** 9](#_Toc187671906)

[**6.2. Разработка калькулятора** 9](#_Toc187671907)

[**6.3. Создание подключения через TCP сокет** 9](#_Toc187671908)

[**6.4. Создание подключения через UDP сокет** 10](#_Toc187671909)

[**7.** **Верификация (тест-план)** 11](#_Toc187671910)

[**7.1. Тестирование программы валидации** 12](#_Toc187671911)

[**7.2. Тестирование программы калькулятора** 13](#_Toc187671912)

[**8. Статистический анализ** 15](#_Toc187671913)

[**Заключение** 18](#_Toc187671914)

[**Список источников** 19](#_Toc187671915)

[**Приложения** 20](#_Toc187671916)

[**Приложение 1. Исходный код программы валидатора** 20](#_Toc187671917)

[**Приложение 2. Исходный код программы калькулятора** 27](#_Toc187671918)

[**Приложение 3. Исходный код программы сервера** 33](#_Toc187671919)

[**Приложение 4. Исходный код программы клиента** 39](#_Toc187671920)

[**Приложение 5. Исходный код тестирующей программы валидатора** 44](#_Toc187671921)

[**Приложение 6. Исходный код тестирующей программы калькулятора** 47](#_Toc187671922)

[**Приложение 7. Результаты статического анализа** 49](#_Toc187671923)

# **Введение**

В настоящее время операционная система Linux занимает значительное место в мире информационных технологий, благодаря своей стабильности, безопасности и широким возможностям для разработки программного обеспечения. Освоение работы в данной среде является важным шагом для специалистов, стремящихся углубить свои знания в области программирования и сетевых технологий. Данная практическая работа направлена на изучение основ разработки клиент-серверных приложений в системе Linux. [1]

**Цель работы** - разработка клиент-серверного приложения, функционирующего в рамках ОС Linux. В процессе реализации проекта акцент будет сделан на создании приложения-калькулятора, что позволит не только изучить основные принципы проектирования клиент-серверной архитектуры, но и получить практический опыт программирования на языке С.

**Задачи работы**:

– получение базовых знаний по разработке клиент-серверных приложений в ОС Linux;

– разработать клиент-серверное приложение-калькулятор;

– написать тестовое приложение (программа верификации) для проверки созданного приложения;

– провести статический анализ исходного кода приложения.

Таким образом, работа не только способствует овладению теоретическими основами, но и формирует практические навыки, необходимые для успешной разработки программного обеспечения в современных условиях.

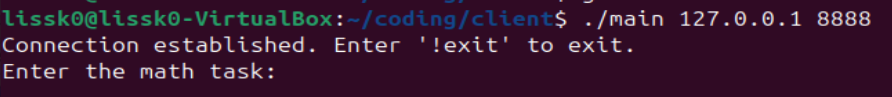
# 

# **Общее описание задачи**

Суть программы заключается настройке связи между сервером и клиентом для отправки и получения сообщений (в обе стороны). После установления подключения, клиент отправляет запросы серверу с математическим примером. При получении примера сервер возвращает клиенту результат подсчетов. Приложение написано на ОС Linux. Для реализации задания был выбран язык программирования С.

# **Описание интерфейса**

Для подключения к серверу в качестве параметра клиентская часть приложения принимает IP-адрес, порт серверной части, а также необязательный параметр **“-u”** для подключения по UDP (рис. 2.1). При запуске приложения на одном компьютере клиентская часть должна использовать в качестве IP-адреса сервера адрес 127.0.0.1 (адрес loopback-интерфейса операционной системы).



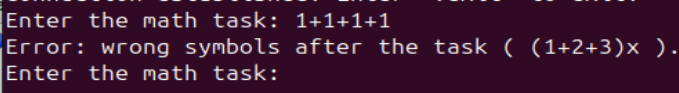
*Рис. 2.1. Пример запуска клиентского приложения*

При успешном подключении клиенту необходимо ввести математический пример, который соответствует заданному формату. Для подсчета доступны типы математических примеров, представленные на рис. 2.2.



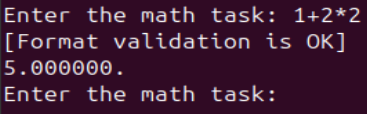
*Рис. 2.2. Типы математических примеров*

В случае, если клиент введет пример неверного типа, в консоль выведется ошибка формата с её возможным местом (рис. 2.3).



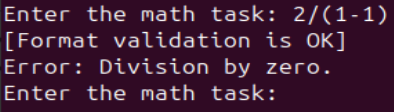
*Рис. 2.3. Демонстрация вывода ошибки формата*

Если пример успешно прошел проверку формата (“*[Format validation is OK]*”), клиент получит от сервера результат подсчета математического выражения (рис. 2.4).



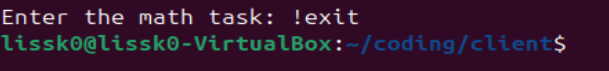
*Рис. 2.4. Демонстрация результата подсчета выражения*

Если же в примере обнаруживается деление на ноль, клиент получит ошибку “*Division by zero*” (рис. 2.5).



*Рис. 2.5. Демонстрация вывода ошибки деления на ноль*

Для выхода из программы клиенту необходимо прописать команду *“!exit” (рис. 2.6).*

**

*Рис. 2.6. Демонстрация выхода из клиентской программы*

# **Функциональные требования**

Приложение должно состоять из клиентской и серверной части, которые взаимодействуют друг с другом по сети. Оно должно быть способно функционировать, как при запуске клиентской и серверной частей приложения на одном компьютере, так и на разных сетевых узлах.

## **3.1. Подключение**

**Клиент** должен иметь возможность подключаться к серверу по заданному IP-адресу и порту, а также выбирать желаемое подключение (TCP или UDP) путем ввода необязательного параметра (*“-u”*). Для выхода из интерфейса приложения клиенту необходимо ввести ключевое слово (*“!exit”*).

**Сервер** должен иметь возможность запускаться в разных режимах. При запуске сервера можно выбрать нужное подключение (TCP или UDP) с помощью ввода необязательного параметра (*“-u”*). После чего сервер должен слушать входящие подключения от клиентов. Для остановки сервера необходимо использовать команду отправки сигналов процессу *“kill <pid >”*, где *pid* – process id, идентификатор процесса сервера.

## **3.2. Обмен данными**

**Клиент** должен иметь возможность отправлять на сервер математические примеры. Отправляемые примеры должны проверяться на корректность формата на стороне клиента. При использовании протокола UDP при отправке примера клиент должен ожидать “ACK” подтверждение от сервера. [2]

**Сервер** должен обрабатывать полученные от клиента примеры и отправлять результат подсчета. При использовании протокола UDP при отправке результата сервер должен ожидать “ACK” подтверждение от клиента.

## **3.3. Обработка ошибок**

**Клиент** должен проверять формат примера. В случае, если введенный математический пример нарушает допустимый формат, клиентская часть выдает ошибку пользователю и не осуществляет отправку на сервер, ожидания повторного ввода примера.

**Сервер** должен корректно обрабатывать ошибки и возвращать сообщения об ошибках клиенту. Если на сервере при подсчете примера результат не определен (было найдено деление на ноль), клиенту отправляется соответствующая ошибка.

# **Требования к тестированию**

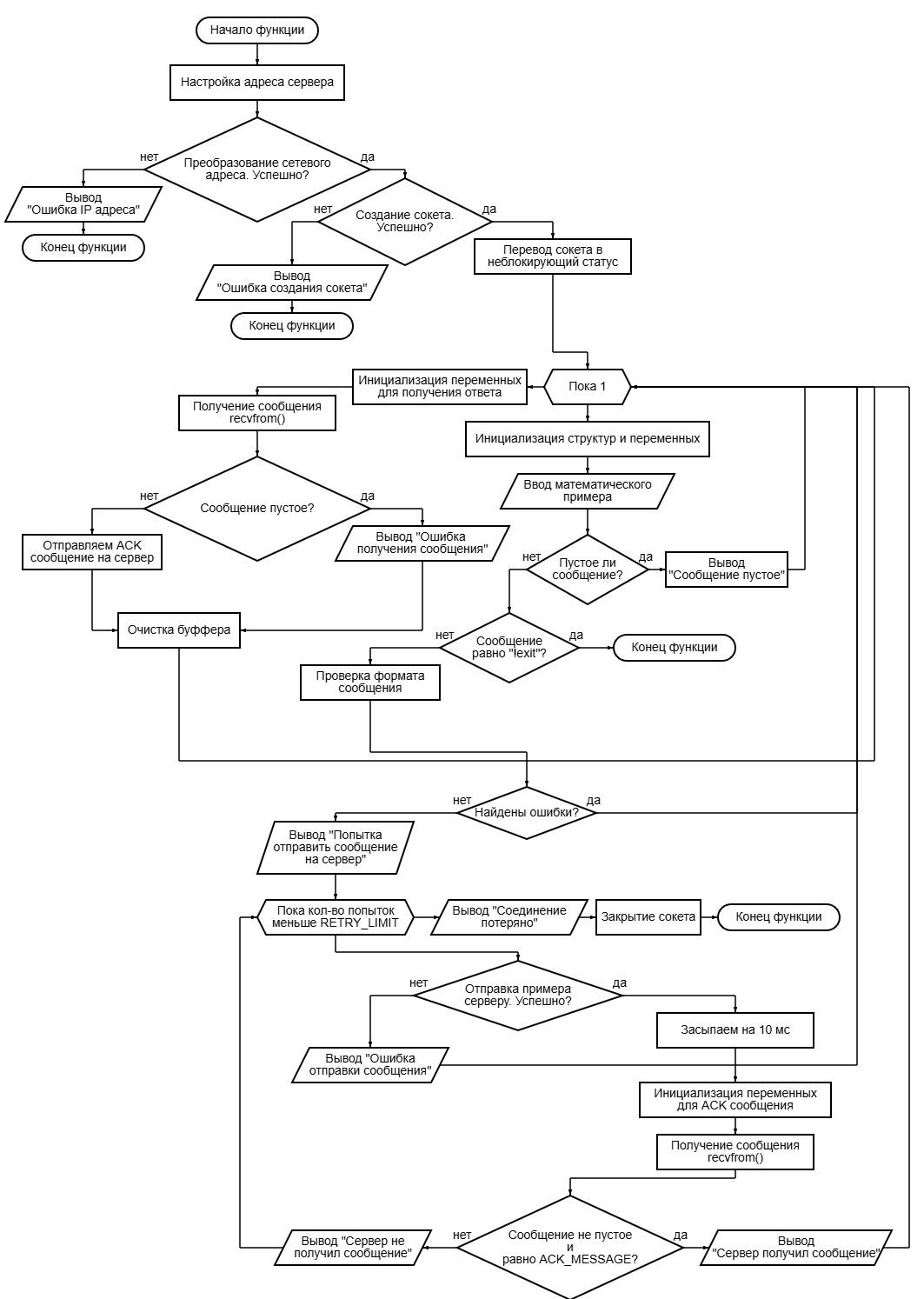
При тестировании необходимо проверить выполнение всех функциональных требований приложения: подключение, корректный обмен данными, обработка ошибок. Проверить сценарии, которые должны завершиться ошибками (например, отключение клиента от сервера, закрытие сервера, ввод несуществующего IP-адреса или неверного количество параметров, и так далее).

Необходимо протестировать программу, проверяющую формат вводимых математических примеров. Для этого необходимо отправлять примеры с верным форматом и ожидать успешное прохождение проверки; отправлять примеры с неверным форматом и ожидать ошибку.

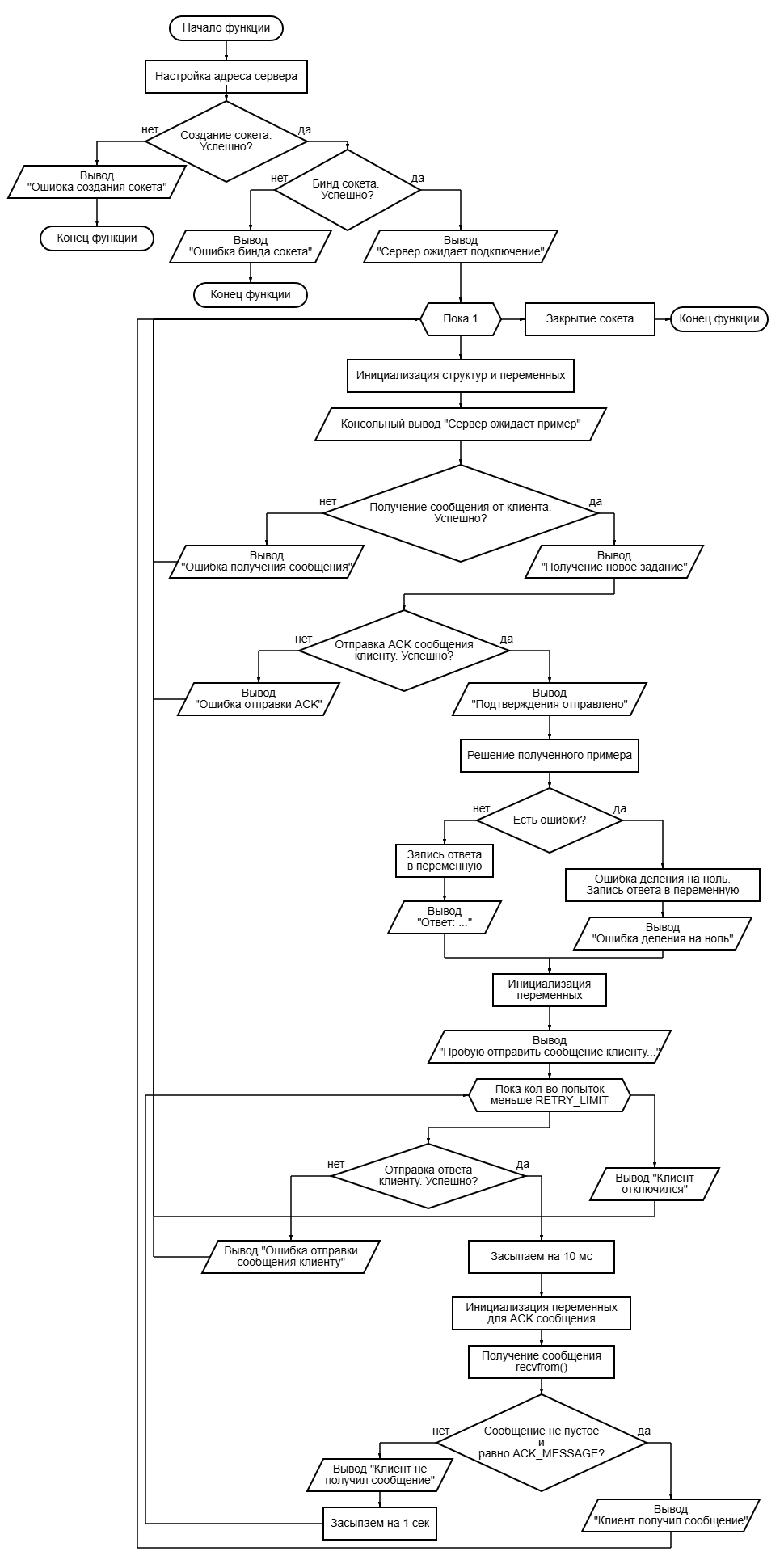
Затем проверить программу подсчета математических примеров. Для этого необходимо проверить правильность подсчета сложных примеров; проверить ошибку деления на ноль.

# **Проектирование**

В этом разделе приведен алгоритм работы при использовании протокола UDP с помощью блок-схем [3]. Блок-схема для клиентской части приложения продемонстрирована на рис. 5.1. Для серверной части - на рис. 5.2.

**

*Рис. 5.1. Блок-схема клиентской части приложения*



*Рис. 5.2. Блок-схема серверной части приложения*

# **Создание программы**

## **6.1. Проверка корректности примера**

В клиентской части приложения происходит консольный ввод математического примера, после чего выполняется синтаксический разбор и проверка корректности входной строки (файл **“validation.h**”, см. Приложение 1). В случае некорректных входных данных (наличие посторонних символов, неверное количество операторов/операндов и др.) на экран выводится соответствующее сообщение о типе ошибке и возможное место проблемы. Данные при этом на серверную часть не отправляются. Если вводимый пример проходит проверку корректности, он отправляется на сервер в виде массива символов.

## **6.2. Разработка калькулятора**

Затем была разработана программа для вычисления математического примера. Поскольку корректность примера уже была проверена в клиентской части, проверять его на стороне сервера не нужно.

В файле **“solver.h”** (см. Приложение 2) происходит подсчет ответа к полученному примеру. Функция *remove\_spaces()* удаляет все пробелы из примера. Затем функция *task\_solver()* поочередно проходится по всему примеру, считывая скобки, дробные числа, целые числа и знаки. Если пример имеет два числа, то вызывается функция *calc\_two\_numbers()*, принимающая эти два числа и знак операции между ними. Возвращает результат подсчета. Если же пример имеет три числа, то вызывается функция *calc\_three\_numbers()*, принимающая три числа, два знака операций и позиции скобок. В этой функции с помощью скобок и приоритетности операций определяется порядок выполнения действий, после чего в ней дважды вызывается уже созданная функция *calc\_two\_numbers()*. В результате работы, функция *task\_solver()* возвращает готовый ответ на полученный математический пример. При возникновении ситуации деления на ноль возвращается ошибка с кодом 2.

## **6.3. Создание подключения через TCP сокет**

В начале было разработано создание подключения между сервером и клиентом через TCP сокет, который гарантирует, что данные будут доставлены без потерь и в правильном порядке.

**Сторона сервера:**

На стороне сервера был создан сокет-коннектор для установки будущего соединения с клиентом (файл **“main.c”**, см. Приложение 3). Посредством функции *bind()* сокет связывается с адресом сервера. Затем через функцию *listen()* осуществляется поиск входящих соединений, после чего командой *accept()* создается TCP-сокет, через который происходит обмен сообщениями между сервером и клиентом. [4]

Для того, чтобы сервер получил сообщение клиента, используется функция *recv()*. В случае, если длина полученного примера не является положительной, сервер выведет ошибку получения сообщения. Если ошибок нет, последним символом полученного сообщения устанавливается символ ‘\0’ для обозначения конца сообщения. Затем через функцию *send()* сервер отправляет найденный ответ клиенту.

Сервер работает до тех пор, пока в консоли не будет введена команда “kill <pid>”, где *pid – process id* (идентификатор процесса сервера).

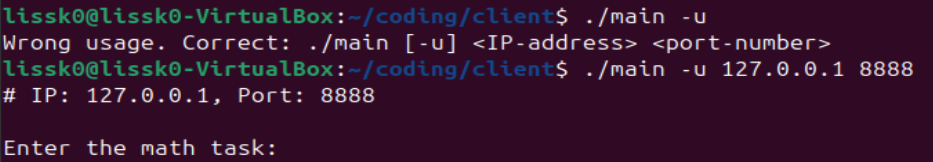
**Сторона клиента:**

Первым делом в коде клиентской части также создается сокет для установки соединения с сервером (файл **“main.c”**, см. Приложение 4). Затем выполняется подключение к серверу с помощью функции *connect().* При успешной установке соединения с сервером программа запрашивает математический пример у пользователя через консоль, проверяет его на корректность формата, а затем через функцию *send()* отправляет пример на сервер. После этого клиент ожидает ответ на заданный пример через функцию *recv().*

Цикл запроса примеров работает до тех пор, пока клиент не введет команду “!exit”.

## **6.4. Создание подключения через UDP сокет**

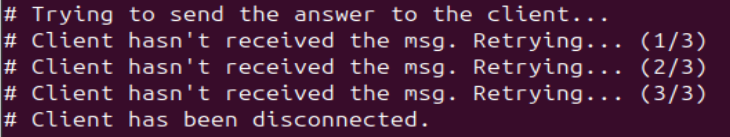
После успешного создания подключения через TCP сокет было разработано подключение с помощью другого сокета - UDP. Он не гарантирует получение и порядок доставки пакетов, однако используется для более быстрой передачи данных. Был добавлен параметр [-u] при запуске сервера и клиента, чтобы можно было выбрать подключение по UDP, а также проверка корректности ввода IP-адреса и порта (рис. 6.4.1).



*Рис. 6.4.1. Демонстрация подключения клиента*

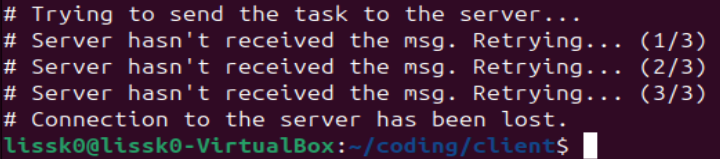
**Сторона сервера:**

Для начала на стороне сервера создается сокет для получения и отправки сообщений. Посредством функции *bind()* сокет связывается с адресом сервера, после чего сервер начинает прослушивать входящие сообщения с помощью функцию *recv()* [5]. При успешном получении сообщения с математическим примером сервер отправляет контрольное “ACK” (“Acknowledgeable”) сообщение клиентской стороне. Затем сервер начинает подсчет полученного примера с помощью описанной ранее функции *task\_solver()*. После подсчета сервер будет пытаться отправить ответ клиенту через функцию *sendto()*, пока не получит в ответ“ACK” сообщение. Если спустя три попытки отправки контрольное сообщение не будет получено - связь с клиентом будет разъединена (рис. 6.4.2). [6]

*Рис. 6.4.2. Отключение клиента от сервера*

**Сторона клиента:**

Аналогично серверной части, на клиенте сначала создается сокет для обмена сообщениями через функцию *socket()*. Затем пользователь вводит математический пример, после чего с помощью функции *sendto()* клиентская часть будет пытаться отправить введенное сообщение на сервер, пока не получит в ответ “ACK” сообщение. Если спустя три попытки отправки контрольное сообщение не будет получено - связь с клиентом будет разъединена (рис. 6.4.3). При успешном получении подтверждения от сервера клиент будет ожидать ответ на заданный пример через функцию *recvfrom()*. В случае, если пользователь захочет выйти из программы (путем ввода команды *“!exit”*), клиентская часть закрывает сокет командой *close()*.

****

*Рис. 6.4.3. Потеря соединения с сервером*

# **Верификация (тест-план)**

Для тестирования программы калькулятора и валидатора (проверяет корректность формата примера на стороне клиента) были созданы bash-сценарии [7, 8]. Описание работы сценария:

1. Для каждого тестового случая сценарий передает выражение как входные данные, затем сравнивает результат выполнения с ожидаемым и выводит результат теста;
2. Сценарий компилирует программу на C (если она еще не скомпилирована). Сценарий использует *gcc* для компиляции программы **“name.c”** в исполняемый файл **“name”** (где *name* - название файла программы калькулятора/валидатора, *solver* ***/*** *validation*).
3. Передает в программу тестовые данные. Для каждого тестового случая сценарий передает выражение как входные данные, затем сравнивает результат выполнения с ожидаемым и выводит результат теста;
4. Сравнивает вывод программы с ожидаемым результатом;
5. Выводит статус тестирования (“OK” или “WA”).

Запуск тестирующей программы **“test.sh”** (где *test.sh* - название файла программы тестирования калькулятора / валидатора - *test\_solver.sh* ***/*** *test\_validation.sh*) нужно:

1. Убедиться, что сценарий исполняемый. Для этого в консоли в директории файла ввести следующую команду: *chmod +x test.sh*
2. Убедиться, что файл **“name.c”** находится в той же директории.
3. Запустить сценарий: *./test.sh*

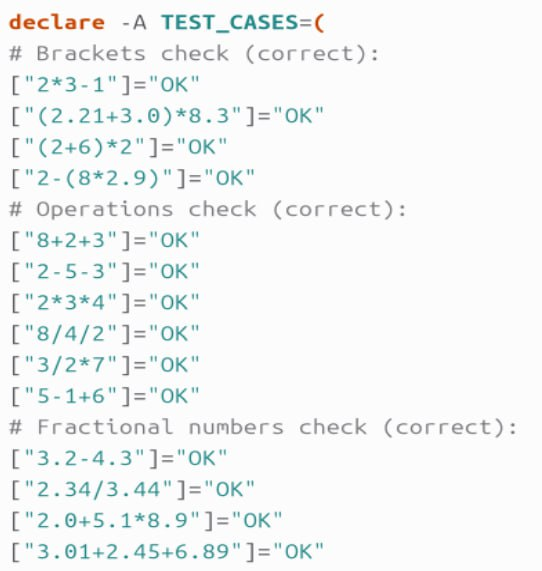
После запуска в консоли будут выведены все исполняемые тесты и их результаты. Каждое тестирование описано подробнее в следующих пунктах.

## **7.1. Тестирование программы валидации**

Для тестирования программы валидации были разработаны следующие группы тестов:

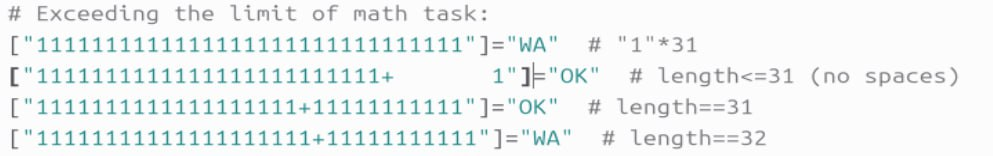
1. Для примеров с правильным форматом:
   1. Все возможные верные варианты расставления скобок;
   2. Различные комбинации знаков операций;
   3. Дробные числа на всех возможных позициях операндов.
2. Для примеров с неверным форматом:
   1. Нарушение порядка скобок;
   2. Нарушение порядка разрешенных символов;
   3. Неверное количество операндов (недостаток, превышение);
   4. Использование недопустимых символов;
   5. Превышение допустимой длины сообщения.

Каждая из подгрупп состоит не менее, чем из четырех тестов (рис. 7.1.1).



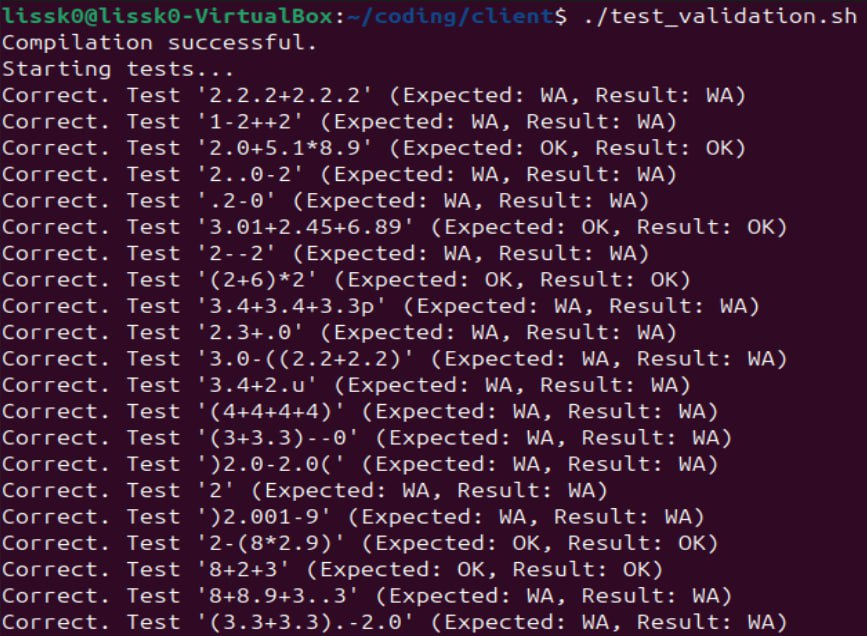
*Рис. 7.1.1. Фрагмент тестов в файле “test\_validation.sh”*

Проверка диапазона значений чисел в примере не является необходимой, поскольку размер числа типа *double* составляет 8 байтов, а размер сообщения всего 31 бит. Разумной будет являться проверка длины сообщения (не более 31 символа, соответственно). Часть написанных тестов изображена на рис. 7.1.2.

**

*Рис. 7.1.2. Тесты на проверку длины сообщения*

Файл тестирования **“test\_validation.sh”** (см. Приложение 5)проверяет программу **“validation.c”** (оба файла находятся в папке **client**) - рис. 7.1.3.



*Рис. 7.1.3. Фрагмент тестирования программы валидации*

Программа валидации успешно прошла все тесты.

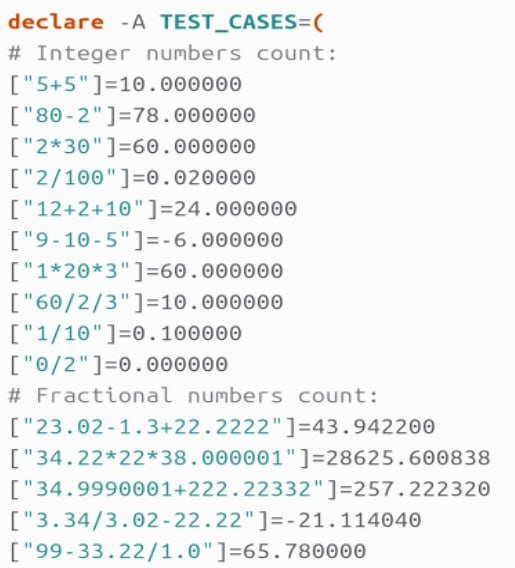
### 

## **7.2. Тестирование программы калькулятора**

Для тестирования программы калькулятора были разработаны следующие группы тестов:

1. Проверка подсчета целочисленных примеров (с двумя и тремя операндами);
2. Проверка подсчета примеров с дробными числами (с двумя и тремя операндами);
3. Проверка подсчета примеров с делением на 0 (с двумя и тремя операндами).

Каждая из групп состоит не менее, чем из четырех тестов (рис. 7.2.1).



*Рис. 7.2.1. Фрагмент тестов в файле “test\_solver.sh”*

Файл тестирования **“test\_solver.sh”** (см. Приложение 6)проверяет программу **“solver.c”** (оба файла находятся в папке **server**) - рис. 7.2.2.



*Рис. 7.2.2. Фрагмент тестирования программы калькулятора*

Во время тестирования была найдена ошибка, вызванная округлением дробных чисел (рис. 7.2.3). Ошибка незначительна, поэтому было принято решение заменить результат теста на тот, который подсчитывает программа калькулятора.

*Рис. 7.2.3. Ошибка при тестировании программы калькулятора*

Остальные тесты программы калькулятора были пройдены успешно.

# **8. Статистический анализ**

Статический анализ кода — анализ исходного кода [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), производимый без реального выполнения исследуемых программ. [9]

Статический анализ исходного кода разработанного приложения проводится с использованием статического анализатора **cppcheck**. Необходимо было установить **cppcheck** и убедиться, что версия анализатора 2.11 или выше (рис. 8.1).



*Рис. 8.1. Проверка версии Cppcheck*

Команда запуска статического анализатора:

*user@ubuntu:~$ cppcheck --cppcheck-build-dir=/tmp/sa-build --std=c99 --library=posix.cfg --check-level=exhaustive --xml ~/coding 2> ~/coding/static\_analysis/report-sa.xml*

где:

**“~”** - домашний каталог пользователя;

**“coding”** - каталог с файлами исходных кодов приложения;

**“static\_analysis”** - каталог хранения результатов статического анализа (создан вручную);

**“2>”** - перенаправление вывода в файл;

**“--std=c99”** - явно задаем используемый стандарт C;

**“--library=posix.cfg”** - задаем поддержку библиотеки POSIX;

**“--check-level=exhaustive”** - задаем уровень проверки Exhaustive;

**“--xml”** - формат отчета XML;

“**/report-sa.xml**” - имя файла отчета.

При первой попытке запустить статический анализатор произошла следующая ошибка:

*cppcheck error Directory '/tmp/sa-build' specified by —cppcheck-build-dir argument has to be existen*

Для ее решения было необходимо создать недостающую директорию командой:

*user@ubuntu:~$ mkdir -p /tmp/sa-build*

На рис. 8.2. продемонстрирован успешный запуск анализатора с исправлением ошибки.

**

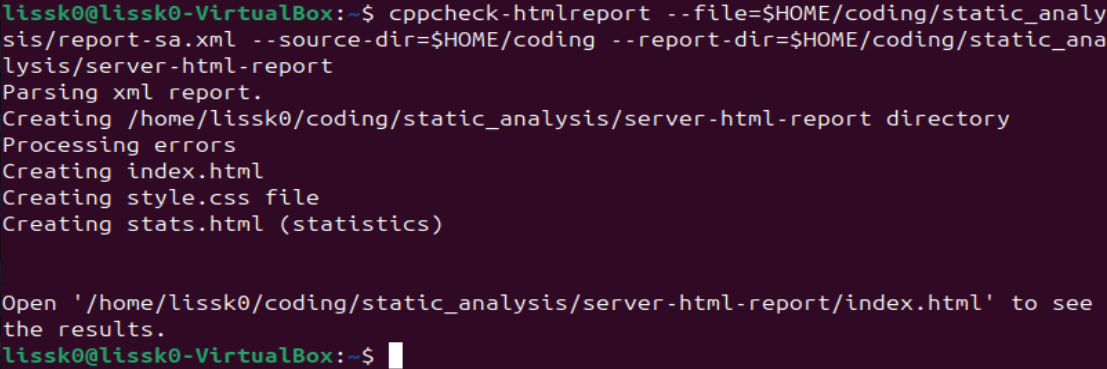
*Рис. 8.2. Успешный запуск анализатора cppcheck*

Затем было необходимо преобразовать отчет анализатора в формат HTML с помощью вспомогательной утилиты **cppcheck-htmlreport**, которая устанавливается вместе с **cppcheck**, автоматически добавляется в переменную окружения PATH и доступна из любой директории (рис. 8.3). [10]

Команда запуска:

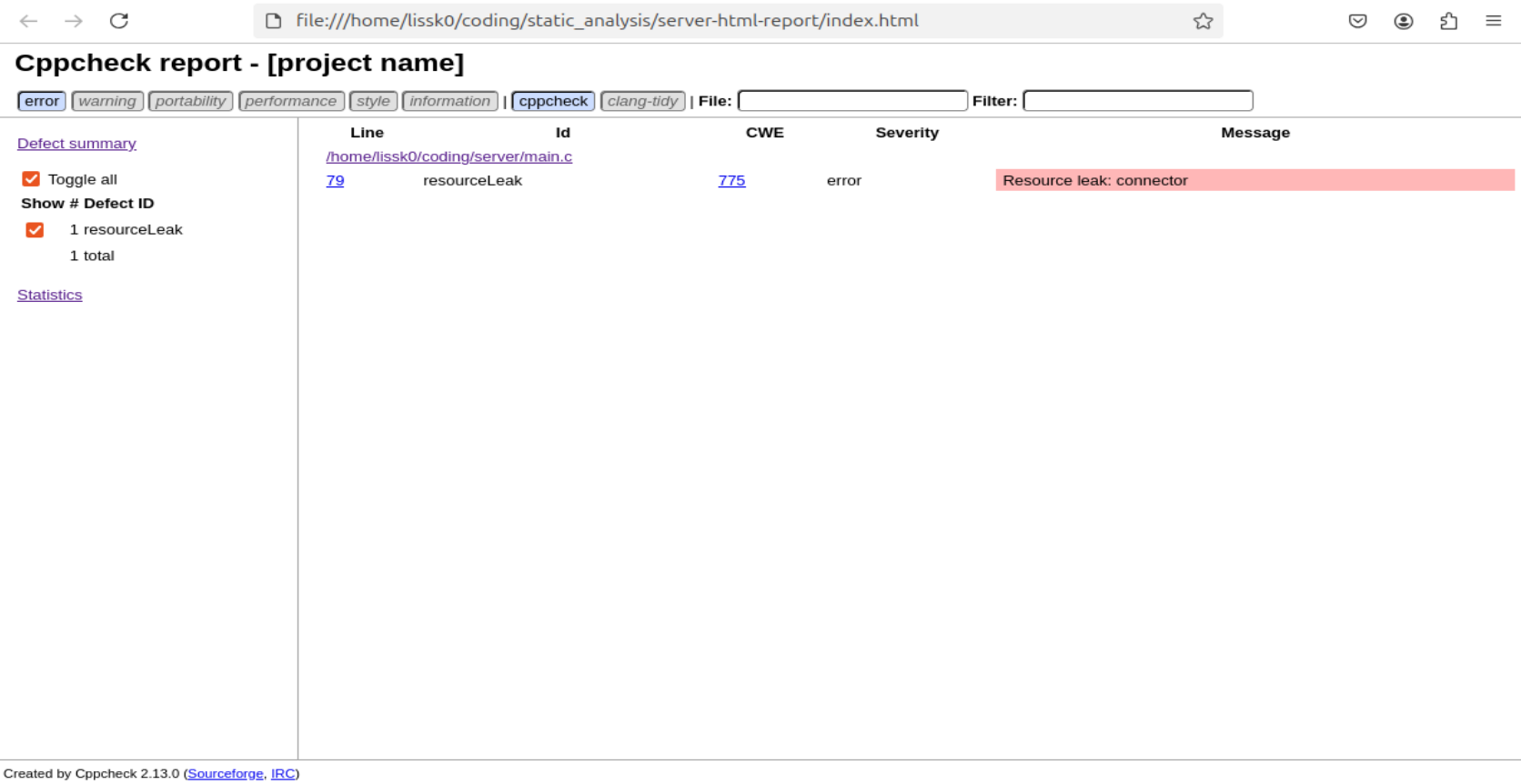
*user@ubuntu:~$ cppcheck-htmlreport --file=$HOME/coding/static\_analysis/report-sa.xml --source-dir=$HOME/coding --report-dir=$HOME/coding/static\_analysis/server-html-report*

В результате выполнения команды в директории *static\_analysis/server-html-report* появился файл *index.html* (см. Приложение 7), содержащий преобразованный в HTML отчет о работе статического анализатора.



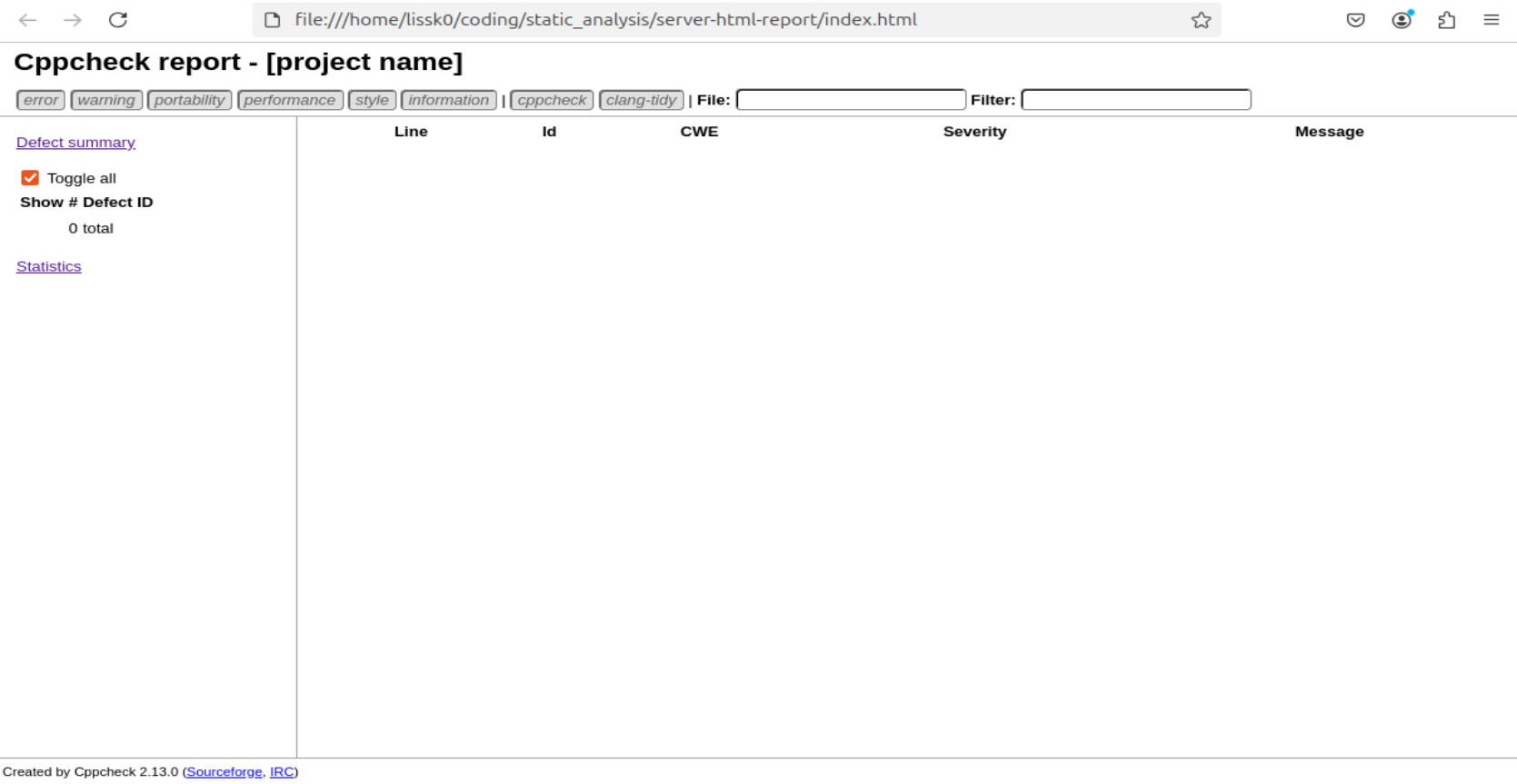
*Рис. 8.3. Преобразование отчета в формат HTML*

В созданном отчете была отображена единственная ошибка, продемонстрированная на рис. 8.4. Проблема вызвана незакрытым сокетом в программе сервера. Для решения данной ошибки было необходимо дописать функцию *close(tcp\_socket)*.



*Рис. 8.4. Первый отчет статического анализатора cppcheck*

После исправления было осуществлено повторное создание отчета. Больше ошибок найдено не было (рис. 8.5).



*Рис. 8.5. Второй отчет статического анализатора cppcheck*

# **Заключение**

В результате работы были получены практические навыки по разработке клиент-серверного приложения, функционирующего в рамках ОС Linux на языке программирования C. Было создано готовое приложение-калькулятор для взаимодействия между клиентской и серверной частями на протоколах TCP и UDP.

По итогам данной работе были изучены основные принципы проектирования клиент-серверной архитектуры, а также базовые знания по разработке приложений в Linux.

Таким образом, выполненная работа не только способствует глубокому изучению и усвоению теоретических основ, связанных с разработкой программного обеспечения, но и играет ключевую роль в развитии практических навыков. Эти навыки крайне важны для успешного выполнения профессиональных задач, стоящих перед специалистами в области информационных технологий, особенно в условиях постоянно меняющихся требований и высокой динамики современного рынка программного обеспечения.

# **Список источников**

1. Операционная система Linux: надежность, гибкость и безопасность [Электронный ресурс] // URL: <https://dzen.ru/a/ZW782azreD2pj0_u> (дата обращения: 28.10.2024)
2. ACK (acknowledgement) definition [Электронный ресурс] // URL: https://www.techtarget.com/searchnetworking/definition/ACK (дата обращения: 23.11.2024)
3. Полное руководство по созданию блок-схемы [Электронный ресурс] // URL: https://slidemodel.com/how-to-make-a-flowchart/ (дата обращения: 26.12.2024)
4. Реализация TCP сервер-клиента на C [Электронный ресурс] // URL: https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/ (дата обращения: 12.11.2024)
5. Реализация UDP сервер-клиента на C [Электронный ресурс] // URL: https://www.geeksforgeeks.org/udp-client-server-using-connect-c-implementation/ (дата обращения: 21.11.2024)
6. Реализация ACK по протоколу UDP [Электронный ресурс] // URL: https://stackoverflow.com/questions/1514839/implementing-ack-over-udp (дата обращения: 23.11.2024)
7. Bash скрипты [Электронный ресурс] // URL: https://habr.com/ru/articles/726316/ (дата обращения: 07.12.2024)
8. Тестирование программы с Bash [Электронный ресурс] // URL: https://stackoverflow.com/questions/33036282/testing-a-program-in-bash (дата обращения: 07.12.2024)
9. Что такое статический анализ кода? [Электронный ресурс] // URL: https://www.jetbrains.com/ru-ru/teamcity/ci-cd-guide/concepts/static-code-analysis/ (дата обращения: 12.12.2024)
10. Тонкости анализа исходного кода C/C++ с помощью cppcheck [Электронный ресурс] // URL: https://habr.com/ru/articles/210256/ (дата обращения: 12.12.2024)

# **Приложения**

## **Приложение 1. Исходный код программы валидатора**

#include <stdio.h>

#include <ctype.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

// deletes spaces in the task

void remove\_spaces(char\* source) {

char\* i = source;

char\* j = source;

while (\*j != '\0') {

\*i = \*j++;

if (\*i != ' ')

i++;

}

\*i = '\0';

}

unsigned short task\_solver(char msg[], int msg\_len)

{

// possible tasks:

// a x b

// a x b x c

// (a x b) x c

// a x (b x c)

unsigned short i = 0; // current pos

bool bracket\_open = false, bracket\_closed = false;

bool bracket\_pos1 = false, bracket\_pos2 = false, bracket\_pos3 = false, bracket\_pos4 = false;

// a x b x c

// 1 2 3 4

if (msg\_len == 0)

{

printf("Error: msg length is 0.\n");

return 1;

}

if (msg\_len > 31)

{

printf("Error: msg length is more than 31.\n");

return 1;

}

if (msg[i] == '(') {

bracket\_open = true;

bracket\_pos1 = true;

i++;

}

// declaring num1

bool num1\_started = false;

bool num1\_point = false;

// if msg[i] is a number

if (i < msg\_len && '0' <= msg[i] && msg[i] <= '9')

{

num1\_started = true;

while (i < msg\_len && i < msg\_len && '0' <= msg[i] && msg[i] <= '9') {

i++;

}

}

else

{

printf("Error: starts not with a wrong symbol or too short.\n");

return 1;

}

if (i >= msg\_len) {

printf("Error: first number ends too early.\n");

return 1;

}

if (msg[i] != '.' && msg[i] != '+' && msg[i] != '-' && msg[i] != '\*' && msg[i] != '/')

{

printf("Error: wrong symbol after first number (123x).\n");

return 1;

}

if (msg[i] == '.')

{

num1\_point = true;

i++;

// continue collecting num1 after point

while (i < msg\_len)

{

if ('0' <= msg[i] && msg[i] <= '9')

{

i++;

}

else if (msg[i] == '+' || msg[i] == '-' || msg[i] == '\*' || msg[i] == '/')

break;

else // not a number and not a mark

{

printf("Error: wrong symbol after num1 (12.34x).\n");

return 1;

}

}

// num1 has ended

}

if (i >= msg\_len) {

printf("Error: too short task (12.34).\n");

return 1;

}

if (msg[i] != '+' && msg[i] != '-' && msg[i] != '\*' && msg[i] != '/')

{

printf("Error: symbol is not a operator after num1 (12.34x).\n");

return 1;

}

// writing an operator

i++;

if (i >= msg\_len) {

printf("Error: too short task (12.34\*).\n");

return 1;

}

if (msg[i] == '(')

{

if (bracket\_open == true) {

printf("Error: two opened brackets ( (( ).\n");

return 1;

}

bracket\_open = true;

bracket\_pos2 = true;

i++;

if (i >= msg\_len) {

printf("Error: too short task ( 12.34\*( ).\n");

return 1;

}

}

if ('0' > msg[i] || msg[i] > '9')

{

printf("Error: not a number after operator (12.34+x).\n");

return 1;

}

// declaring num2

bool num2\_started = false;

bool num2\_point = false;

num2\_started = true; // msg[i] is 100% a number and i < msg\_len (has been checked earlier)

// writing while numbers

while (i < msg\_len && i < msg\_len && '0' <= msg[i] && msg[i] <= '9') {

i++;

}

if (i >= msg\_len) {

if (bracket\_open == false) { // if there are no brackets then it has format "1.0+2" correct

printf("[Format validation is OK]\n");

return 0;

}

else

{

printf("Error: too short ( (1.0+2) ).\n");

return 1;

}

}

if (msg[i] != ')' && msg[i] != '.' && msg[i] != '+' && msg[i] != '-' && msg[i] != '\*' && msg[i] != '/')

{

printf("Error: wrong symbol after second number (12.0+12x).\n");

return 1;

}

if (msg[i] == ')')

{

if (bracket\_open == true)

{

bracket\_pos3 = true;

bracket\_closed = true;

i++;

if (i >= msg\_len) {

printf("Error: can't use such brackets format ( (1.2+2) ).\n");

return 1;

}

}

else

{

printf("Error: closing brackets without opening ( 1.2+0.2) ).\n");

return 1;

}

}

else if (msg[i] == '.')

{

num2\_point = true;

i++;

if (i >= msg\_len) {

printf("Error: ends after point (12.0+12.).\n");

return 1;

}

// continue collecting num2 after point

while (i < msg\_len)

{

if ('0' <= msg[i] && msg[i] <= '9')

{

i++;

}

else if (msg[i] == '+' || msg[i] == '-' || msg[i] == '\*' || msg[i] == '/')

break;

else if (msg[i] == ')')

{

if (bracket\_open == true)

{

bracket\_pos3 = true;

bracket\_closed = true;

i++;

if (i >= msg\_len) {

printf("Error: can't use such brackets format ( (1.2+0.2) ).\n");

return 1;

}

break;

}

else

{

printf("Error: closing brackets without opening ( 1.2+0.2) ).\n");

return 1;

}

}

else // not a number, not a mark, not a ')'

{

printf("Error: wrong symbol after num2 (1.2+3.4x).\n");

return 1;

}

}

// num2 has ended

}

if (i >= msg\_len)

{

if (bracket\_open == true)

{

printf("Error: bracket is not closed ( (1.2+3.4 ).\n");

return 1;

}

printf("[Format validation is OK]\n");

return 0;

}

if (msg[i] != '+' && msg[i] != '-' && msg[i] != '\*' && msg[i] != '/')

{

printf("Error: wrong symbol after num2 or num2).\n");

return 1;

}

// writing an operator

i++;

if (i >= msg\_len) {

printf("Error: too short task (1.2+3.4+).\n");

return 1;

}

if ('0' < msg[i] && msg[i] > '9')

{

printf("Error: not a number after operator (1.2+3.4+x).\n");

return 1;

}

// declaring num3

bool num3\_started = false;

bool num3\_point = false;

num3\_started = true; // msg[i] is 100% a number and i < msg\_len (has been checked earlier)

// writing while numbers

while (i < msg\_len && i < msg\_len && '0' <= msg[i] && msg[i] <= '9') {

i++;

}

if (i >= msg\_len)

{

if (bracket\_closed == true || bracket\_open == false)

{

printf("[Format validation is OK]\n");

return 0;

}

else

{

printf("Error: wrong brackets format (e.g. not closed).\n");

return 1;

}

}

if (msg[i] == ')')

{

if (bracket\_open == true && bracket\_closed == false && bracket\_pos1 == false)

{

bracket\_pos4 = true;

i++;

if (msg\_len != i) {

printf("Error: symbols after last close bracket ( (1+2+3)x ).\n");

return 1;

}

printf("[Format validation is OK]\n");

return 0;

}

else {

printf("Error: wrong close bracket (not opened or already closed).\n");

return 1;

}

i++;

}

if (msg[i] == '.')

{

num3\_point = true;

i++;

if (i >= msg\_len) {

printf("Error: ends after point (1.2+3.4+5.).\n");

return 1;

}

// continue collecting num3 after point

while (i < msg\_len)

{

if ('0' <= msg[i] && msg[i] <= '9')

{

i++;

}

else if (msg[i] == ')')

{

if (bracket\_open == true && bracket\_closed == false && bracket\_pos1 == false)

{

bracket\_pos4 = true;

i++;

if (msg\_len != i) {

printf("Error: symbols after last close bracket ( (1+2+3)x ).\n");

return 1;

}

printf("[Format validation is OK]\n");

return 0;

}

else {

printf("Error: wrong close bracket (not opened or already closed).\n");

return 1;

}

}

else // not a number, not a ')'

{

printf("Error: wrong symbol after num3 (1.2+3.4+5.6x).\n");

return 1;

}

}

// num3 has ended

}

if (i >= msg\_len)

{

if ((bracket\_open == false && bracket\_closed == false) || (bracket\_open == true && bracket\_closed == true))

{

printf("[Format validation is OK]\n");

return 0;

}

else

{

printf("Error: wrong brackets format.\n");

return 1;

}

}

else

{

printf("Error: wrong symbols after the task ( (1+2+3)x ).\n");

return 1;

}

}

## **Приложение 2. Исходный код программы калькулятора**

#include <stdio.h>

#include <ctype.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

struct SolverData {

double result;

unsigned short error;

};

// delete spaces in the task

void remove\_spaces(char\* source) {

char\* i = source;

char\* j = source;

while (\*j != '\0') {

\*i = \*j++;

if (\*i != ' ')

i++;

}

\*i = '\0';

}

// calculate task with two numbers

struct SolverData calc\_two\_nums(double num1, double num2, char oper) {

struct SolverData sd;

sd.result = 0;

sd.error = 0;

if (oper == '+')

sd.result = num1 + num2;

if (oper == '-')

sd.result = num1 - num2;

if (oper == '\*')

sd.result = num1 \* num2;

if (oper == '/') {

if (num2 == 0) {

sd.error = 2; // Division by zero

}

else

sd.result = num1 / num2;

}

return sd;

}

struct SolverData calc\_three\_nums(double n1, double n2, double n3, char o1, char o2, bool bp1, bool bp2, bool bp3, bool bp4)

{

// a x b x c

// 1 2 3 4

// possible options: 13, 24

struct SolverData sd;

sd.error = 0;

if (bp1 == false && bp2 == false && bp3 == false && bp4 == false)

{

if ((o1 != '\*' && o1 != '/') && (o2 == '\*' || o2 == '/')) // o2 then o1

{

sd = calc\_two\_nums(n2, n3, o2);

if (sd.error != 0)

return sd;

double temp\_res = sd.result;

sd = calc\_two\_nums(n1, temp\_res, o1);

return sd; // whatever error

}

else // other cases where o1 then o2

{

sd = calc\_two\_nums(n1, n2, o1);

if (sd.error != 0)

return sd;

double temp\_res = sd.result;

sd = calc\_two\_nums(temp\_res, n3, o2);

return sd; // whatever error

}

}

if (bp1 == true && bp2 == false && bp3 == true && bp4 == false)

{

sd = calc\_two\_nums(n1, n2, o1);

if (sd.error != 0)

return sd;

double temp\_res = sd.result;

sd = calc\_two\_nums(temp\_res, n3, o2);

return sd; // whatever error

}

if (bp1 == false && bp2 == true && bp3 == false && bp4 == true)

{

sd = calc\_two\_nums(n2, n3, o2);

if (sd.error != 0)

return sd;

double temp\_res = sd.result;

sd = calc\_two\_nums(n1, temp\_res, o1);

return sd; // whatever error

}

sd.error = 1; // Wrong format

printf("Error: wrong brackets positions.\n");

return sd;

}

struct SolverData task\_solver(char msg[], int msg\_len)

{

// possible tasks:

// a x b

// a x b x c

// (a x b) x c

// a x (b x c)

struct SolverData solver\_data;

solver\_data.result = 0;

solver\_data.error = 0;

unsigned short i = 0; // current pos

bool bracket\_pos1 = false, bracket\_pos2 = false, bracket\_pos3 = false, bracket\_pos4 = false;

// a x b x c

// 1 2 3 4

if (msg\_len == 0)

{

solver\_data.error = 1; // Wrong format

printf("Error: msg\_len==0.\n");

return solver\_data;

}

if (msg[i] == '(') {

bracket\_pos1 = true;

i++;

}

// declaring num1

int decimal\_amount = 1;

double num1 = 0;

// writing while numbers

while (i < msg\_len && i < msg\_len && '0' <= msg[i] && msg[i] <= '9') {

num1 = num1 \* 10 + (msg[i] - '0');

i++;

}

if (msg[i] == '.')

{

i++;

// continue collecting num1 after point

while (i < msg\_len)

{

if ('0' <= msg[i] && msg[i] <= '9')

{

num1 = num1 \* 10 + (msg[i] - '0');

decimal\_amount \*= 10;

i++;

}

else if (msg[i] == '+' || msg[i] == '-' || msg[i] == '\*' || msg[i] == '/')

break;

}

num1 = num1 / decimal\_amount; // num1 has ended

}

// writing an operator

char oper1;

oper1 = msg[i];

i++;

if (msg[i] == '(')

{

bracket\_pos2 = true;

i++;

}

// declaring num2

decimal\_amount = 1;

double num2 = 0;

// msg[i] is 100% a number and i < msg\_len (has been checked earlier)

// writing while numbers

while (i < msg\_len && i < msg\_len && '0' <= msg[i] && msg[i] <= '9') {

num2 = num2 \* 10 + (msg[i] - '0');

i++;

}

if (i >= msg\_len) {

solver\_data = calc\_two\_nums(num1, num2, oper1);

if (solver\_data.error == 2) {

printf("Error: Division by zero (num1 / num2).\n");

}

return solver\_data;

}

if (msg[i] == ')')

{

bracket\_pos3 = true;

i++;

}

else if (msg[i] == '.')

{

i++;

// continue collecting num2 after point

while (i < msg\_len)

{

if ('0' <= msg[i] && msg[i] <= '9')

{

num2 = num2 \* 10 + (msg[i] - '0');

decimal\_amount \*= 10;

i++;

}

else if (msg[i] == '+' || msg[i] == '-' || msg[i] == '\*' || msg[i] == '/')

break;

else if (msg[i] == ')')

{

bracket\_pos3 = true;

i++;

break;

}

}

num2 = num2 / decimal\_amount; // num2 has ended

}

if (i >= msg\_len)

{

solver\_data = calc\_two\_nums(num1, num2, oper1); // 1.2+3.4

if (solver\_data.error == 2) {

printf("Error: Division by zero (num1 / num2).\n");

}

return solver\_data;

}

// writing an operator

char oper2;

oper2 = msg[i];

i++;

// declaring num3

decimal\_amount = 1;

double num3 = 0;

// writing while numbers

while (i < msg\_len && i < msg\_len && '0' <= msg[i] && msg[i] <= '9') {

num3 = num3 \* 10 + (msg[i] - '0');

i++;

}

if (i >= msg\_len)

{

solver\_data = calc\_three\_nums(num1, num2, num3, oper1, oper2, bracket\_pos1, bracket\_pos2, bracket\_pos3, bracket\_pos4);

if (solver\_data.error == 2) {

printf("Error: Division by zero.\n");

}

return solver\_data;

}

if (msg[i] == ')')

{

bracket\_pos4 = true;

i++;

solver\_data = calc\_three\_nums(num1, num2, num3, oper1, oper2, bracket\_pos1, bracket\_pos2, bracket\_pos3, bracket\_pos4);

if (solver\_data.error == 2) {

printf("Error: Division by zero.\n");

}

return solver\_data;

i++;

}

if (msg[i] == '.')

{

i++;

// continue collecting num3 after point

while (i < msg\_len)

{

if ('0' <= msg[i] && msg[i] <= '9')

{

num3 = num3 \* 10 + (msg[i] - '0');

decimal\_amount \*= 10;

i++;

}

else if (msg[i] == ')')

{

bracket\_pos4 = true;

i++;

num3 = num3 / decimal\_amount;

solver\_data = calc\_three\_nums(num1, num2, num3, oper1, oper2, bracket\_pos1, bracket\_pos2, bracket\_pos3, bracket\_pos4);

if (solver\_data.error == 2) {

printf("Error: Division by zero.\n");

}

return solver\_data;

}

}

num3 = num3 / decimal\_amount; // num3 has ended

}

solver\_data = calc\_three\_nums(num1, num2, num3, oper1, oper2, bracket\_pos1, bracket\_pos2, bracket\_pos3, bracket\_pos4);

if (solver\_data.error == 2) {

printf("Error: Division by zero.\n");

}

return solver\_data;

}

## **Приложение 3. Исходный код программы сервера**

#include <stdio.h>

#include <stdlib.h>

#include <arpa/inet.h> // for inet\_pton, htons

#include <string.h>

#include <sys/socket.h> // for socket (it's already in unistd.h)

#include <unistd.h> // for close()

#include <errno.h> // for errno

#include "solver.h"

#include <fcntl.h> // for F\_SETFL

#define MAX\_CONN 5 // max socket connections

#define LOCAL\_PORT 8888 // server's port

struct SolverData call\_solver(char msg[])

{

remove\_spaces(msg);

int msg\_len = strlen(msg);

struct SolverData solver\_data;

solver\_data = task\_solver(msg, msg\_len);

return solver\_data;

}

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

{

bool connection\_type = 0; // 0 - TCP, 1 - UDP

if (argc > 1) {

if (strcmp(argv[1], "-u") == 0)

connection\_type = 1; // 0 - TCP, 1 - UDP

}

if (connection\_type == 0)

TCP\_connection();

else

UDP\_connection();

return 0;

}

void TCP\_connection()

{

// Configuring server address:

struct in\_addr local\_ip;

inet\_pton(AF\_INET, "0.0.0.0", &local\_ip); // convert our ip into local\_ip, AF\_INET for IPv4

struct sockaddr\_in local\_addr\_in; // struct for ip & port

memset(&local\_addr\_in, 0, sizeof(struct sockaddr\_in)); // memory set, fill the struct with 0

local\_addr\_in.sin\_addr = local\_ip;

local\_addr\_in.sin\_port = htons(LOCAL\_PORT); // Host To Network Short

local\_addr\_in.sin\_family = AF\_INET; // for IPv4

struct sockaddr\* local\_addr = (struct sockaddr\*)&local\_addr\_in;

int local\_addrlen = sizeof(struct sockaddr\_in); // length of struct

int connector = socket(AF\_INET, SOCK\_STREAM, 0); // socket to establish the connection. IPv4, TCP (type), protocol for sock\_stream (0 or IPPROTO\_TCP)

if (connector < 0)

{

printf("# Error creating socket-connector.\n");

return;

}

/\*

// Allows SO\_REUSEADDR for the connector,

// so this socket will be able to work on closed ports:

int opt\_val = 1;

setsockopt(connector, SOL\_SOCKET, SO\_REUSEADDR, &opt\_val);

\*/

short int bind\_error = bind(connector, local\_addr, local\_addrlen); // 0 if no errors

if (bind\_error != 0)

{

printf("# Error binding the socket: %d.\n", errno);

close(connector);

return;

}

printf("# TCP server is listening on port %d\n", LOCAL\_PORT);

while (1)

{

printf("# Listening for connections...\n");

listen(connector, MAX\_CONN); // listening for new connections, like bluetooth devices

struct sockaddr client\_addr;

int client\_addrlen = sizeof(struct sockaddr); //sockaddr\_in?

short int tcp\_socket = accept(connector, &client\_addr, &client\_addrlen); // connecting to the client using his address

// through this socket-connection we will send/receive messages

if (tcp\_socket <= 0)

{

printf("# Error establishing connection with code: %d.\n", errno);

break;

}

else

printf("# Connection established.\n");

while (1)

{

// Receiving a task from the client

char msg[31];

int msg\_len = recv(tcp\_socket, msg, sizeof(msg) - 1, 0); // receiving the [msg, max msg length, flag] from the client

if (msg\_len <= 0)

{

printf("# Error receiving the message (client disconnected): %d.\n", errno);

break;

}

msg[msg\_len] = '\0';

printf("# Received task from the client: %s\n", msg);

// Solving the task

struct SolverData data = call\_solver(msg);

double res = data.result;

unsigned short error = data.error;

if (error == 0)

{

printf("Answer is: %f\n", res);

char str\_res[31];

sprintf(str\_res, "%f", res); // double to string

str\_res[sizeof(str\_res) - 1] = '\0';

// Sending the answer to the client

if (send(tcp\_socket, str\_res, strlen(str\_res), 0) == -1) // 0 - flag

printf("# Error sending the message: %d.\n", errno);

else

printf("# Message has been delivered to the client.\n");

}

else

{

char str\_res[31];

if (error == 1)

{

printf("# Error: Wrong task format\n");

strcpy(str\_res, "# Error: Wrong task format");

}

else if (error == 2)

{

printf("# Error: Division by zero\n");

strcpy(str\_res, "# Error: Division by zero");

}

// Sending an error to the client

if (send(tcp\_socket, str\_res, strlen(str\_res), 0) == -1) // 0 - flag

printf("# Error sending the message: %d.\n", errno);

else

printf("# Message has been delivered to the client.\n");

continue;

}

}

close(tcp\_socket);

}

close(connector);

}

#define RETRY\_LIMIT 3 // limit of retry attempts for connecting to the server

#define TIMEOUT\_SEC 3

#define ACK\_MESSAGE "ACK" // ACKnowledgement message

void UDP\_connection()

{

// Configuring server's address settings:

struct in\_addr local\_ip;

inet\_pton(AF\_INET, "0.0.0.0", &local\_ip);

struct sockaddr\_in local\_addr\_in;

memset(&local\_addr\_in, 0, sizeof(struct sockaddr\_in)); // set zeros

local\_addr\_in.sin\_addr = local\_ip; // set local ip

local\_addr\_in.sin\_port = htons(LOCAL\_PORT); // set port

local\_addr\_in.sin\_family = AF\_INET; // IPv4

// Creating UDP socket:

int udp\_socket = socket(AF\_INET, SOCK\_DGRAM, 0); // AF\_INET for IPv4

if (udp\_socket < 0) {

printf("# UDP Socket creation failed.");

return;

}

// Setting nonblocking status of socket:

// fcntl(udp\_socket, F\_SETFL, O\_NONBLOCK);

// Binding the socket:

struct sockaddr\* local\_addr = (struct sockaddr\*)&local\_addr\_in;

int local\_addrlen = sizeof(struct sockaddr\_in); // length of struct

if (bind(udp\_socket, local\_addr, local\_addrlen) < 0) {

printf("# Bind failed.");

close(udp\_socket);

return;

}

printf("# UDP server is listening on port %d.\n", LOCAL\_PORT);

while (1)

{

struct sockaddr\_in client\_addr;

socklen\_t client\_addrlen = sizeof(client\_addr);

char msg[31];

// Receiving the task from the client:

printf("\n# Waiting for the task...\n");

int msg\_len = recvfrom(udp\_socket, msg, sizeof(msg), 0, (struct sockaddr\*)&client\_addr, &client\_addrlen);

if (msg\_len < 0) {

printf("# Receiving failed.\n");

continue;

}

msg[msg\_len] = '\0';

printf("# New task from the client: %s\n", msg);

// Sending the confirm msg to the client:

int sendto\_result = sendto(udp\_socket, ACK\_MESSAGE, strlen(ACK\_MESSAGE), 0, (struct sockaddr\*)&client\_addr, client\_addrlen);

if (sendto\_result == -1)

{

printf("# Error sending the ACK msg to the client (sendto error).\n");

break;

}

printf("# Confirmation has been sent to the client.\n", msg);

// Solving the task:

struct SolverData data = call\_solver(msg);

double res = data.result;

unsigned short error = data.error;

char response[31];

if (error == 0)

{

snprintf(response, sizeof(response), "%f", res);

printf("# Answer is: %f\n", res);

}

else

{

if (error == 1)

{

printf("# Error: Wrong task format\n");

strncpy(response, "# Error: Wrong task format", sizeof(response) - 1);

}

else if (error == 2)

{

printf("# Error: Division by zero\n");

strncpy(response, "# Error: Division by zero", sizeof(response) - 1);

}

response[sizeof(response) - 1] = '\0';

}

// Sending the answer to the client:

int flag\_sendto\_error = false;

printf("# Trying to send the answer to the client...\n");

int attempts = 0;

while (attempts < RETRY\_LIMIT)

{

int sendto\_result = sendto(udp\_socket, response, strlen(response), 0, (struct sockaddr\*)&client\_addr, client\_addrlen);

if (sendto\_result == -1) // how many bytes have been sent

{

printf("# Error sending the answer to the client (sendto error).\n");

flag\_sendto\_error = true;

break;

}

usleep(10); // waiting 10 ms before trying to receive the confirm msg

// Receiving the confirmation from the client:

char confirm\_msg[5];

memset(confirm\_msg, 0, sizeof(confirm\_msg));

int msg\_len = recvfrom(udp\_socket, confirm\_msg, sizeof(confirm\_msg), 0, NULL, NULL);

confirm\_msg[msg\_len] = '\0';

if (msg\_len > 0 && strncmp(confirm\_msg, ACK\_MESSAGE, msg\_len) == 0) // msg\_len > 0 and msg equals the confirmation msg (ACK\_MESSAGE)

{

printf("# Client has received the answer.\n");

attempts = 0;

break; // msg has been received, going to the next step

}

else

{

attempts++;

printf("# Client hasn't received the msg. Retrying... (%d/3)\n", attempts);

sleep(1); // waiting 1 sec

continue; // trying to receive the confirmation msg again

}

}

if (attempts >= RETRY\_LIMIT)

{

printf("# Client has been disconnected.\n");

continue; // waiting for the new task

}

if (flag\_sendto\_error == true) // error, msg hasn't been sent

continue;

}

close(udp\_socket);

}

## **Приложение 4. Исходный код программы клиента**

#include <stdio.h>

#include <stdlib.h>

#include <arpa/inet.h> // for inet\_pton, htons

#include <string.h>

#include <errno.h> // for errno

#include <unistd.h> // for close()

#include <fcntl.h> // for F\_SETFL

#include "validation.h" // checks validation of tasks

#define REMOTE\_PORT 8888 // server's port

#define REMOTE\_IP "127.0.0.1" // server's ip

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

{

bool connection\_type = 0; // 0 - TCP, 1 - UDP

int server\_port;

char \*server\_ip;

// Parcing console data:

if (argc == 3) {

server\_ip = argv[1];

server\_port = atoi(argv[2]); // convert port number to int

} else if (argc == 4) {

if (strcmp(argv[1], "-u") == 0) {

connection\_type = 1; // set UDP connection

server\_ip = argv[2];

server\_port = atoi(argv[3]);

} else {

// fprintf(stderr, "Wrong usage. Correct: %s [-u] <IP-address> <port-number>\n", argv[0]);

printf("Wrong usage. Correct: ./main [-u] <IP-address> <port-number>\n");

return 0;

}

} else {

printf("Wrong usage. Correct: ./main [-u] <IP-address> <port-number>\n");

return 0;

}

printf("# IP: %s, Port: %d\n", server\_ip, server\_port);

// Start connection:

if (connection\_type == 0)

{

TCP\_connection(server\_ip, server\_port);

}

else

{

UDP\_connection(server\_ip, server\_port);

}

return 0;

}

void TCP\_connection(char \*server\_ip, int server\_port)

{

// Configuring server address:

struct in\_addr remote\_ip;

struct sockaddr\_in remote\_addr\_in; // struct for server's ip & port

memset(&remote\_addr\_in, 0, sizeof(struct sockaddr\_in)); // fill the struct with 0

//remote\_addr\_in.sin\_addr = remote\_ip;

remote\_addr\_in.sin\_port = htons(server\_port); // Host To Network Short

remote\_addr\_in.sin\_family = AF\_INET; // for IPv4

//struct sockaddr\* remote\_addr = (struct sockaddr\*)&remote\_addr\_in;

//int remote\_addrlen = sizeof(struct sockaddr\_in); // length of struct

//inet\_pton(AF\_INET, server\_ip, &remote\_ip); // set our ip into local\_ip, AF\_INET for IPv4

// Verification of IP-address:

if (inet\_pton(AF\_INET, server\_ip, &(remote\_addr\_in.sin\_addr)) <= 0) // for IPv4

{

printf("Error: Incorrect IP-address.\n");

return 0;

}

// Creating TCP socket:

int tcp\_socket = socket(AF\_INET, SOCK\_STREAM, 0); // IPv4, TCP (type), protocol for sock\_stream (0 or IPPOTOCOL)

if (tcp\_socket < 0) {

printf("# Error creating TCP-socket with code: %d.\n", errno);

// printf("Error code: %d\n", errno); // errno - global variable for errors (socket error codes linux)

return 0;

}

// Connecting to the server:

if (connect(tcp\_socket, (struct sockaddr \*)&remote\_addr\_in, sizeof(remote\_addr\_in)) < 0)

{

printf("# Error establishing connection with code: %d.\n", errno);

close(tcp\_socket);

return 0;

}

printf("# Connection established. Enter '!exit' to exit.\n");

// Connection cycle:

while (1)

{

char msg[31];

printf("\nEnter the math task: ");

if (fgets(msg, sizeof(msg), stdin) != NULL)

msg[strcspn(msg, "\n")] = 0; // remove the newline character at the end

else

continue; // skipping iterration if empty input

if (strcmp(msg, "!exit") == 0) // comparing char massive to string

break;

// Validating the task:

remove\_spaces(msg); // removes spaces from the task

short int errors = task\_solver(msg, strlen(msg));

if (errors != 0) {

// printf("Error: wrong task's format. Code: %d.\n", errors;

continue;

}

// Sending a task to the server:

if (send(tcp\_socket, msg, strlen(msg), 0) == -1)

{

printf("Error sending the message: %d.\n", errno);

continue;

}

// Receiving the answer from the server:

char answer[31];

int answer\_len = recv(tcp\_socket, answer, sizeof(answer) - 1, 0); // answer, max answer length, flag

if (answer\_len > 0)

{

answer[answer\_len] = '\0'; // last byte is 0 because string must end with 0

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

}

else

printf("Error receiving the message: %d.\n", errno);

memset(msg, 0, sizeof(msg)); // clean msg msg for the next input

}

close(tcp\_socket);

}

#define RETRY\_LIMIT 3 // limit of retry attempts for connecting to the server

#define TIMEOUT\_SEC 3

#define ACK\_MESSAGE "ACK" // acknowledgement message

void UDP\_connection(char \*server\_ip, int server\_port)

{

// Configuring server's address settings:

struct sockaddr\_in server\_addr;

memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(server\_port); // Host To Network Short

if (inet\_pton(AF\_INET, server\_ip, &server\_addr.sin\_addr) <= 0)

{

printf("# Error: invalid IP address.\n");

return;

}

// Creating UDP socket:

int udp\_socket, attempts = 0;

if ((udp\_socket = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) // AF\_INET for IPv4

{

printf("# UDP Socket creation failed.\n");

return;

}

// Setting nonblocking status of socket:

fcntl(udp\_socket, F\_SETFL, O\_NONBLOCK);

// Interaction with the server:

int flag\_sendto\_error = false;

while (1)

{

char msg[31];

printf("\nEnter the math task: ");

if (fgets(msg, sizeof(msg), stdin) != NULL)

msg[strcspn(msg, "\n")] = 0; // removing the newline character at the end

else

{

printf("Message is empty.\n");

continue; // skipping iterration if empty input

}

if (strcmp(msg, "!exit") == 0) // comparing char massive to string

return; // exitting the program

// Validating the task:

remove\_spaces(msg); // removes spaces from the task

short int errors = task\_solver(msg, strlen(msg)); // checking errors

if (errors != 0) {

// printf("# Error: wrong task's format. Code: %d.\n", errors;

continue; // skipping iteration if error

}

// Sending a task to the server:

printf("# Trying to send the task to the server...\n");

while (attempts < RETRY\_LIMIT)

{

int sendto\_res = sendto(udp\_socket, msg, strlen(msg), 0, (struct sockaddr \*) &server\_addr, sizeof(server\_addr)); // socket, buffer, buffer\_len, flag, to whom, len to whom

if (sendto\_res == -1) // how many bytes have been sent

{

printf("# Error sending the task to the server (sendto error).\n");

flag\_sendto\_error = true;

break;

}

usleep(10); // waiting 10 ms before trying to receive the confirm msg

// Receiving the confirmation from the server:

char confirm\_msg[5];

memset(confirm\_msg, 0, sizeof(confirm\_msg));

int msg\_len = recvfrom(udp\_socket, confirm\_msg, sizeof(confirm\_msg), 0, NULL, NULL);

confirm\_msg[msg\_len] = '\0';

if (msg\_len > 0 && strncmp(confirm\_msg, ACK\_MESSAGE, strlen(ACK\_MESSAGE)) == 0) // msg\_len > 0 and msg equals the confirmation msg (ACK\_MESSAGE)

{

printf("# Server has received the task.\n");

attempts = 0;

break; // msg has been received, going to the next step

}

else

{

attempts++;

printf("# Server hasn't received the msg. Retrying... (%d/3)\n", attempts);

sleep(1); // waiting 1 sec

continue; // trying to receive the confirmation msg again

}

}

if (attempts >= RETRY\_LIMIT)

{

printf("# Connection to the server has been lost.\n");

close(udp\_socket);

return;

}

if (flag\_sendto\_error == true) // error, msg hasn't been sent

continue;

// Receiving the answer:

char answer[31];

int answer\_len = recvfrom(udp\_socket, answer, sizeof(answer), 0, NULL, NULL); // answer, max answer length, flag

if (answer\_len > 0)

{

answer[answer\_len] = '\0'; // last byte is 0 because string must end with 0

printf("%s.\n", answer);

// Sending the confirm msg to the server:

sendto(udp\_socket, ACK\_MESSAGE, strlen(ACK\_MESSAGE), 0, (struct sockaddr \*) &server\_addr, sizeof(server\_addr));

}

else

printf("# Error receiving the answer (empty).\n");

memset(msg, 0, sizeof(msg)); // clean msg msg for the next input

}

close(udp\_socket);

}

## **Приложение 5. Исходный код тестирующей программы валидатора**

#!/bin/bash

PROGRAM="validation.c"

EXECUTABLE="validation"

if [ ! -f "$PROGRAM" ]; then

echo "Error: file $PROGRAM is not found."

exit 1

fi

gcc -o $EXECUTABLE $PROGRAM

if [ $? -ne 0 ]; then

echo "Compilation error."

exit 1

fi

echo "Compilation successful."

declare -A TEST\_CASES=(

# Brackets check (correct):

["2\*3-1"]="OK"

["(2.21+3.0)\*8.3"]="OK"

["(2+6)\*2"]="OK"

["2-(8\*2.9)"]="OK"

# Operations check (correct):

["8+2+3"]="OK"

["2-5-3"]="OK"

["2\*3\*4"]="OK"

["8/4/2"]="OK"

["3/2\*7"]="OK"

["5-1+6"]="OK"

# Fractional numbers check (correct):

["3.2-4.3"]="OK"

["2.34/3.44"]="OK"

["2.0+5.1\*8.9"]="OK"

["3.01+2.45+6.89"]="OK"

["4.000+1.999+4.343"]="OK"

["2.12345/(3.93560-1.88992)"]="OK"

# Wrong brackets positions (incorrect):

["(2.0+2.0"]="WA"

[")2.0-2.0("]="WA"

[")2.001-9"]="WA"

["2.0+2)"]="WA"

["(2.3/2.0("]="WA"

[")2.3/2.3)"]="WA"

["(2.2+2.2+2.2)"]="WA"

["2.2+2.2-2.2)"]="WA"

["(2.2+1.1+3.3"]="WA"

["3.3+(3.3-3.3"]="WA"

["3+)3.3-3.3)"]="WA"

["3.0-((2.2+2.2)"]="WA"

["3.3+(2.3+9))"]="WA"

["(3.3+3.3))+2"]="WA"

["((3.4+2.3)+2"]="WA"

["(3.3+3.3)"]="WA"

["(3+3)"]="WA"

["(4+4+4+4)"]="WA"

# Wrong acceptable-symbols positions (incorrect):

["2--2"]="WA"

["2..0-2"]="WA"

[".2-0"]="WA"

["5.2.+2"]="WA"

["5.2.4+3.3"]="WA"

["3.3--3.3"]="WA"

["3++3"]="WA"

["8+3..3"]="WA"

["8+8.9+3..3"]="WA"

["0.3//2"]="WA"

["2\*\*8"]="WA"

["(3+3.3)--0"]="WA"

["8++(3.3-2.2)"]="WA"

["8.2+(2.2)"]="WA"

["2.3+.0"]="WA"

["4.444444-4."]="WA"

["1-2++2"]="WA"

["1++2-3"]="WA"

["1..9+2.9"]="WA"

["1.2+2..2"]="WA"

["9.0+2.2+3..3"]="WA"

["(8-2)++1"]="WA"

["(3.3+3.3).-2.0"]="WA"

["(8.0.+2.0)-1.1"]="WA"

["(22-2)+2+"]="WA"

["+2.2+2.3"]="WA"

["2.2.2"]="WA"

["2.2.2+2.2.2"]="WA"

# Wrong amount of numbers (incorrect):

["4+4+4+4"]="WA"

["4.0+(3.3+3.3+3.3)"]="WA"

["(3.9+3.3-2.0)-3.3"]="WA"

["1+2+3+4+5"]="WA"

["2.0"]="WA"

["2"]="WA"

# Using of non acceptable-symbols (incorrect):

["3.p+2.0"]="WA"

["3.4+2.u"]="WA"

["3.4+3.4+3.p"]="WA"

["3.4+3.4+3.3p"]="WA"

["5.5-2.2o2.2"]="WA"

["5-2-o"]="WA"

["o3.3+3.3o"]="WA"

# Exceeding the limit of math task:

["1111111111111111111111111111111"]="WA" # "1"\*31

["1111111111111111111111111+ 1"]="OK" # length<=31 (no spaces)

["1111111111111111111+11111111111"]="OK" # length==31

["11111111111111111111+11111111111"]="WA" # length==32

)

run\_test() {

local input=$1

local expected=$2

result=$(echo "$input" | ./$EXECUTABLE)

if [ "$result" == "$expected" ]; then

echo "Correct. Test '$input' (Expected: $expected, Result: $result)"

else

echo "Incorrect. Test '$input' (Expected: $expected, Result: $result)"

fi

}

echo "Starting tests..."

for input in "${!TEST\_CASES[@]}"; do

expected=${TEST\_CASES[$input]}

run\_test "$input" "$expected"

done

## **Приложение 6. Исходный код тестирующей программы калькулятора**

#!/bin/bash

PROGRAM="solver.c"

EXECUTABLE="solver"

if [ ! -f "$PROGRAM" ]; then

echo "Error: file $PROGRAM is not found."

exit 1

fi

gcc -o $EXECUTABLE $PROGRAM

if [ $? -ne 0 ]; then

echo "Compilation error."

exit 1

fi

echo "Compilation successful."

declare -A TEST\_CASES=(

# Integer numbers count:

["5+5"]=10.000000

["80-2"]=78.000000

["2\*30"]=60.000000

["2/100"]=0.020000

["12+2+10"]=24.000000

["9-10-5"]=-6.000000

["1\*20\*3"]=60.000000

["60/2/3"]=10.000000

["1/10"]=0.100000

["0/2"]=0.000000

# Fractional numbers count:

["23.02-1.3+22.2222"]=43.942200

["34.22\*22\*38.000001"]=28607.920753

["34.9990001+222.22332"]=257.222320

["3.34/3.02-22.22"]=-21.114040

["99-33.22/1.0"]=65.780000

["1 / 2.0"]=0.500000

["1.0/2.0"]=0.500000

["1.0/2"]=0.500000

["10.2 / 2.009"]=5.077153

["0.00000/10.1"]=0.000000

["20.2/(3- 3.00001)"]=-2019999.999987

["2.1 - 2.1 + 0.2"]=0.200000

["(8.0+2.0) / 2.0"]=5.000000

# Division by zero error:

["1/0"]="WA"

["2.1/0"]="WA"

["100.002 / 0"]="WA"

["20.1/(2-2)"]="WA"

["20/(4-4)"]="WA"

["20/(4.0-4)"]="WA"

["20/(4-4.0)"]="WA"

["(20.1-3.4)/0.0"]="WA"

)

run\_test() {

local input=$1

local expected=$2

result=$(echo "$input" | ./$EXECUTABLE)

if [ "$result" == "$expected" ]; then

echo "Correct. Test '$input' (Expected: $expected, Result: $result)"

else

echo "Incorrect. Test '$input' (Expected: $expected, Result: $result)"

fi

}

echo "Starting tests..."

for input in "${!TEST\_CASES[@]}"; do

expected=${TEST\_CASES[$input]}

run\_test "$input" "$expected"

done

## **Приложение 7. Результаты статического анализа**

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Cppcheck - HTML report - [project name]</title>

<link rel="stylesheet" href="style.css">

<style>

</style>

<script>

function getStyle(el, styleProp) {

var y;

if (el.currentStyle) {

y = el.currentStyle[styleProp];

} else if (window.getComputedStyle) {

y = document.defaultView.getComputedStyle(el, null).getPropertyValue(styleProp);

}

return y;

}

function toggle() {

var el = this.expandable\_content;

var mark = this.expandable\_marker;

if (el.style.display === "block") {

el.style.display = "none";

mark.textContent = "[+]";

} else {

el.style.display = "block";

mark.textContent = "[-]";

}

}

function initExpandables() {

var elements = document.querySelectorAll(".expandable");

for (var i = 0, len = elements.length; i < len; i++) {

var el = elements[i];

var clickable = el.querySelector("span");

var marker = clickable.querySelector(".marker");

var content = el.querySelector(".content");

var width = clickable.clientWidth - parseInt(getStyle(content, "padding-left")) - parseInt(getStyle(content, "padding-right"));

content.style.width = width + "px";

clickable.expandable\_content = content;

clickable.expandable\_marker = marker;

clickable.addEventListener("click", toggle);

}

}

function toggleDisplay(cb) {

var elements = document.querySelectorAll("." + cb.id);

for (var i = 0, len = elements.length; i < len; i++) {

elements[i].classList.toggle("id-filtered", !cb.checked);

}

updateFileRows();

}

function toggleSeverity(cb) {

cb.parentElement.classList.toggle("unchecked", !cb.checked);

var elements = document.querySelectorAll(".sev\_" + cb.id);

for (var i = 0, len = elements.length; i < len; i++) {

elements[i].classList.toggle("severity-filtered", !cb.checked);

}

updateFileRows();

}

function toggleTool(cb) {

cb.parentElement.classList.toggle("unchecked", !cb.checked);

var elements;

if (cb.id == "clang-tidy")

elements = document.querySelectorAll("[class^=clang-tidy-]");

else

elements = document.querySelectorAll(".issue:not([class^=clang-tidy-])");

for (var i = 0, len = elements.length; i < len; i++) {

elements[i].classList.toggle("tool-filtered", !cb.checked);

}

updateFileRows();

}

function toggleAll() {

var elements = document.querySelectorAll(".idToggle");

// starting from 1 since 0 is the "toggle all" input

for (var i = 1, len = elements.length; i < len; i++) {

var changed = elements[i].checked != elements[0].checked;

if (changed) {

elements[i].checked = elements[0].checked;

toggleDisplay(elements[i]);

}

}

}

function filterFile(filter) {

var elements = document.querySelectorAll(".fileEntry");

for (var i = 0, len = elements.length; i < len; i++) {

var visible = elements[i].querySelector("tr").querySelector("td").textContent.toLowerCase().includes(filter.toLowerCase());

elements[i].classList.toggle("text-filtered", !visible);

}

}

function filterText(text) {

filter = text.toLowerCase();

var elements = document.querySelectorAll(".issue");

for (var i = 0, len = elements.length; i < len; i++) {

var visible = false;

var fields = elements[i].querySelectorAll("td");

for (var n = 0, num = fields.length; n < num; n++) {

if (fields[n].textContent.toLowerCase().includes(filter)) {

visible = true;

break;

}

}

elements[i].classList.toggle("text-filtered", !visible);

}

updateFileRows();

}

function updateFileRows(element) {

var elements = document.querySelectorAll(".fileEntry");

for (var i = 0, len = elements.length; i < len; i++) {

var visible = elements[i].querySelector(".issue:not(.id-filtered):not(.severity-filtered):not(.tool-filtered):not(.text-filtered)");

elements[i].classList.toggle("file-filtered", !visible);

}

}

window.addEventListener("load", initExpandables);

</script>

</head>

<body>

<div id="wrapper">

<div id="header" class="header">

<h1>Cppcheck report - [project name]</h1>

<div id="filters">

<label class="checkBtn disabled"><input type="checkbox" onclick="toggleSeverity(this)" id="error"disabled/>error</label>

<label class="checkBtn disabled"><input type="checkbox" onclick="toggleSeverity(this)" id="warning"disabled/>warning</label>

<label class="checkBtn disabled"><input type="checkbox" onclick="toggleSeverity(this)" id="portability"disabled/>portability</label>

<label class="checkBtn disabled"><input type="checkbox" onclick="toggleSeverity(this)" id="performance"disabled/>performance</label>

<label class="checkBtn disabled"><input type="checkbox" onclick="toggleSeverity(this)" id="style"disabled/>style</label>

<label class="checkBtn disabled"><input type="checkbox" onclick="toggleSeverity(this)" id="information"disabled/>information</label>

|

<label class="checkBtn disabled"><input type="checkbox" onclick="toggleTool(this)" id="cppcheck"disabled/>cppcheck</label>

<label class="checkBtn disabled"><input type="checkbox" onclick="toggleTool(this)" id="clang-tidy"disabled/>clang-tidy</label>

|

<label class="severityHeader">File: <input type="text" oninput="filterFile(this.value)"/></label>

<label class="severityHeader">Filter: <input type="text" oninput="filterText(this.value)"/></label>

</div>

</div>

<div id="menu\_index">

<p><a href="index.html">Defect summary</a> </p>

<label><input type="checkbox" class="idToggle" onclick="toggleAll()" checked> Toggle all</label>

<table>

<tr><th>Show</th><th>#</th><th>Defect ID</th></tr>

<tr><td></td><td>0</td><td>total</td></tr>

</table>

<p><a href="stats.html">Statistics</a></p>

</div>

<div id="content\_index">

<table class="summaryTable">

<tr><th>Line</th><th>Id</th><th>CWE</th><th>Severity</th><th>Message</th></tr>

</tbody>

</table>

</div>

<div id="footer" class="footer">

<p>

Created by Cppcheck 2.13.0 (<a href="https://cppcheck.sourceforge.io">Sourceforge</a>, <a href="irc://irc.freenode.net/cppcheck">IRC</a>)

</p>

</div>

</div>

</body>

</html>