

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Московский авиационный институт
(национальный исследовательский университет)

Институт № 8
Информационных технологий и прикладной математики

Кафедра 813 «Компьютерная математика»

КУРСОВАЯ РАБОТА
по дисциплине «Операционные системы и архитектура компьютеров»
на тему: Разработка клиент-серверного программного комплекса
«MD5. База данных - библиотека»

Выполнил: студент группы М8О-211Б-20

Малков Кирилл Евгеньевич

(Фамилия, имя, отчество)

(подпись)

Принял: профессор кафедры 813

Чернова Татьяна Александровна

(Фамилия, имя, отчество)

(подпись)

Оценка:

Дата:

Москва, 2021

Содержание

I	Общая часть	4
1	Работа с процессами	4
1.1	Используемые системные объекты	4
1.2	Порождение процессов	4
1.3	Ветвящиеся процессы	5
2	Работа с файловой системой	7
2.1	Используемые системные объекты	7
2.2	Реализация ls	7
2.3	Реализация ls без папок	7
2.4	Жёсткая ссылка	9
2.5	Символическая ссылка	9
2.6	Сортировка файлов	11
3	Локальная клиент-серверная модель	12
3.1	Используемые системные объекты	12
3.2	Очередь сообщений	12
3.3	Семафоры и разделяемая память	14
4	Сетевое программирование	16
4.1	Используемые системные объекты	16
4.2	Подсчёт слов	16
4.3	Чат	17
II	Индивидуальная часть	20
5	Клиент-серверное ПО «MD5. База данных - библиотека»	20
5.1	Техническое задание	20
5.2	Формулировка задачи 1	21
5.3	Формулировка задачи 2	21
5.4	Протокол общения клиента и сервера S	21

5.5	Протокол общения серверов S и S1	22
5.6	Протокол общения серверов S и S2	22
5.7	Структура приложения	22
5.8	Основные алгоритмы	25
5.9	Руководство пользователя	32
6	Список использованных источников	38
7	Приложение	39
7.1	Порождение процессов	39
7.2	Ветвящиеся процессы	48
7.3	Реализация ls	54
7.4	Реализация ls без папок	54
7.5	Жёсткая ссылка	55
7.6	Символическая ссылка	57
7.7	Сортировка файлов	58
7.8	Очередь сообщений	59
7.9	Семафоры и разделяемая память	68
7.10	Подсчёт слов	76
7.11	Чат	85
7.12	ПО «Сетевой сервер»	89

Часть I

Общая часть

1 Работа с процессами

1.1 Используемые системные объекты

В курсовой работе использовались следующие системные объекты:

1. `fork()` и связанный с ним `waitpid()`. `fork()` позволяет создать идентичный процесс, который имеет те же данные, что и порождающий. `waitpid()` приостанавливает текущий процесс до тех пор, пока дочерний процесс не завершится, или до появления сигнала, который либо завершает текущий процесс, либо требует вызвать функцию-обработчик. Это важно для того, чтобы порождающий процесс не завершился раньше порожденного, в противном случае пока подобный процесс не удален из системы, он будет использовать слот в таблице процессов ядра.
2. `exec()`. Семейство функций `exec` заменяет текущий образ процесса новым образом процесса.

1.2 Порождение процессов

1.2.1 Техническое задание

Написать программный комплекс, который состоит из двух частей:

- а. Первая получает на вход (через командную строку) арифметический пример (вещественные числа со знаком и операции: $+$, $-$, $*$, $/$), решает его и печатает ответ.
- б. Написать программу, которая через командную строку принимает текстовый файл со строчками-арифметическими примерами (см. пункт а). Необходимо для каждой прочитанной строки создавать новый процесс, который запускает первую программу с аргументом - прочитанной строкой. Результат работы программы вместе с

примером печатается в выходной файл, имя которого передаётся также через аргументы командной строки второй программы.

1.2.2 Описание основного алгоритма

`main_b.c` принимает на вход файл, считывает все строки и для каждой делает системный вызов `fork()` и в процессе-ребёнке с помощью `execl` вызывает `main_a.c`, передавая строку из файла, для решения полученного примера.

`main_a.c` вызывает функцию `in_postfix`, которая переводит пример в постфиксную память с помощью стека, затем вызывается функция `proc_postfix`, которая, обрабатывая постфиксную запись, решает пример. Ответ выводится.

Реализация алгоритмов представлена в разделе приложения [7.1](#).

1.2.3 Руководство пользователя

На вход подаётся файл с примерами. И программа записывает решённые примеры в файл с ответами.

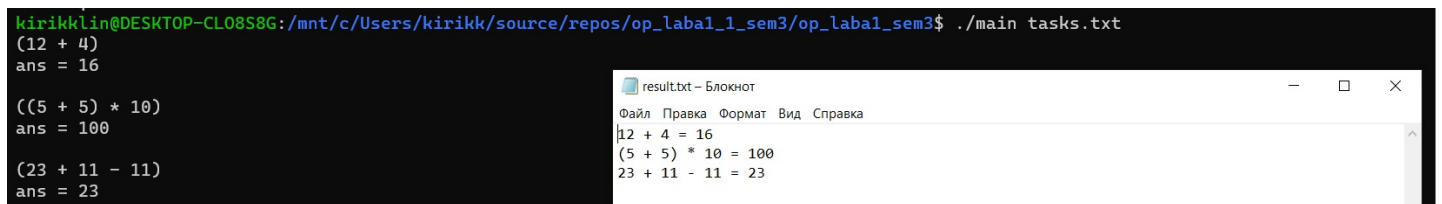


Рис. 1: Пример работы приложения "Порождение процессов".

1.3 Ветвящиеся процессы

1.3.1 Техническое задание

Написать программу генерирующую ветвящиеся процессы, по следующему правилу: каждый процесс единожды определяет случайное событие —

- 25% вероятность завершения процесса;
- 40% вероятность порождения 1 процесса;
- 25% вероятность порождения 2 процессов;

- 10% вероятность порождения 3 процессов.

Нарисуйте гистограмму распределения длин цепочек порождённых процессов, у которых выпало событие завершения процесса.

Примечание: после выбора любого варианта процесс ждёт завершения всех порождённых процессов и только тогда завершается.

1.3.2 Описание основного алгоритма

В цикле `while` в переменную `procent` записывается результат выражения `rand()%100+1`, далее с помощью условий реализовывается появление какого-либо действия с определённой вероятностью. Если процесс должен завершиться, то в файл записывается значение переменной `length_of_proc_trace`, она увеличивается в каждом добавленном процессе, пока процесс не завершится. Если выпадает необходимость создания нескольких процессов, то в переменную `create_proc` записывается количество. Далее, в цикле `for` с помощью системного вызова `fork()` создаётся определённое значение процессов.

Далее, после завершения всех процессов, получая информацию о процессах из файла строится гистограмма. Реализация алгоритмов представлена в разделе приложения [7.2](#).

1.3.3 Руководство пользователя

От пользователя требуется запустить программу, и она выведет гистограммы.



Рис. 2: Пример работы приложения "Ветвящиеся процессы".

2 Работа с файловой системой

2.1 Используемые системные объекты

Основным объектом, используемым в данной работе, является структура `dirent`, которая позволяет пользователю узнать некоторые данные о каталоге. И структура `DIR`, предоставляющая функционал работы с директориями. Также была использована структура `stat`, которая хранит информацию о файле.

2.2 Реализация `ls`

2.2.1 Техническое задание

Вывести на экран все файлы, расположенные в директории, введенной пользователем.

2.2.2 Описание основного алгоритма

Используя структуру `DIR`, откроем заданную пользователем директорию, а затем в цикле, пройдясь по каждому объекту в директории, выведем его имя, используя структуру `dirent`.

Реализация алгоритмов представлена в разделе приложения [7.3](#).

2.2.3 Руководство пользователя

Пользователю необходимо запустить программу, затем в консоли появится сообщение, которое предлагает пользователю ввести директорию. Если такой директории не существует, программа оповестит пользователя об этом.

2.3 Реализация `ls` без папок

2.3.1 Техническое задание

Вывести на экран из указанной пользователем директории все файлы, не являющиеся папками.

```
kirikklin@DESKTOP-CL08S8G:/mnt/c/Users/kirikk/source/repos/op_laba2_1_sem3/op_laba2_1_sem3$ ./main
Введите дерикторию:/mnt/d/kursach_os
.
..
.vs
a.out
client
client.cpp
database_library.txt
DBLibrary.h
file.txt
MD5.h
netserver
netserver.cpp
server_db
server_db.cpp
server_md5
server_md5.cpp
structures.h
tex
```

Рис. 3: Пример работы приложения "Реализация ls".

2.3.2 Описание основного алгоритма

Используя структуру **DIR**, откроем заданную пользователем директорию, а затем в цикле, пройдясь по каждому объекту в директории и проверяя является ли объект файлом, выведем его имя, используя структуру **dirent**.

Реализация алгоритмов представлена в разделе приложения [7.4](#).

2.3.3 Руководство пользователя

Пользователю необходимо запустить программу, затем в консоли появится сообщение, которое предлагает пользователю ввести директорию. Если такой директории не существует, программа оповестит пользователя об этом.

```
kirikklin@DESKTOP-CL08S8G:/mnt/c/Users/kirikk/source/repos/op_laba2_2_sem3/op_laba2_2_sem3$ ./main
Введите дерикторию:/mnt/d/kursach_os/
a.out
client
client.cpp
database_library.txt
DBLibrary.h
file.txt
MD5.h
netserver
netserver.cpp
server_db
server_db.cpp
server_md5
server_md5.cpp
structures.h
```

Рис. 4: Пример работы приложения "Реализация ls без папок".

2.4 Жёсткая ссылка

2.4.1 Техническое задание

Открыть указанный файл на запись. Если файл уже существует, сделать на него жёсткую ссылку (`name.x.ext`), где `x` — первый свободный номер, `ext` — текущее расширение файла.

2.4.2 Описание основного алгоритма

Алгоритм программы заключается в том, чтобы сперва найти свободное имя файла.

Для этого необходимо в бесконечном цикле перебирать числа от 1 до ∞ , переводить их в строку, а затем склеивать по шаблону, заданному в техническом задании. Проверка на существование проводится при помощи функции `access()` с флагом `F_OK`, если возвращаемое значение данной функции для нового имени файла равно 1, то файла с таким именем не существует, значит, обнаружено свободное имя. Как только было найдено подходящее имя файла, создаем на заданный пользователем файл жесткую ссылку с новым именем.

Реализация алгоритмов представлена в разделе приложения [7.5](#).

2.4.3 Руководство пользователя

Пользователю необходимо запустить программу, передав в аргумент командной строки имя файла, жёсткую ссылку, для которого надо создать. Программа выведет имя жёсткой ссылки, которая была создана.

2.5 Символическая ссылка

2.5.1 Техническое задание

Создать символическую ссылку на файл с текстом программы по указанному пути и вывести информацию по ней об атрибутах данной символической ссылки.

```

kirikklin@DESKTOP-CL08S8G:/mnt/c/labs_os/2_3$ ./main file.txt
file.1.txt
file.2.txt
file.3.txt
file.4.txt
file.5.txt
file.6.txt
file.7.txt
file.8.txt
file.9.txt
file.10.txt
free name found
Hardlink name: file.10.txt
kirikklin@DESKTOP-CL08S8G:/mnt/c/labs_os/2_3$ |

```

Рис. 5: Пример работы приложения "Жёсткая ссылка".

2.5.2 Описание основного алгоритма

Пользователь вводит путь, по которому нужно создать символическую ссылку. Далее, с помощью функции `symlink()`, создаётся символическая ссылка на файл с исходным кодом программы. Для сбора данных у ссылки используется функция `lstat()` и структура `stat`, в которой хранится информация о ссылке. Затем, выводится вся информация о ссылке из структуры `stat`.

Реализация алгоритмов представлена в разделе приложения [7.6](#).

2.5.3 Руководство пользователя

Пользователю необходимо запустить программу, передав в аргумент командной строки путь, по которому создастся символическая ссылка для файла с исходным кодом. Далее выводится информация о созданной символической ссылке.

```

kirikklin@DESKTOP-CL08S8G:/mnt/c/labs_os/2_4$ ./main symlink
1 : symlink : 3 : 46161896180692217 : UID=1000 GID=1000 Sun Jan  9 11:39:03 2022
kirikklin@DESKTOP-CL08S8G:/mnt/c/labs_os/2_4$ |

```

Рис. 6: Пример работы приложения "Жёсткая ссылка".

Выводятся следующие атрибуты ссылки:

1. Количество жёстких ссылок.

2. Имя.
3. Общий размер в байтах.
4. inode.
5. UID: идентификатор пользователя-владельца.
6. GID: идентификатор группы-владельца.
7. Время последнего изменения.

2.6 Сортировка файлов

2.6.1 Техническое задание

Вывести на экран информацию обо всех файлах в директории, отсортированную по времени последнего изменения файла.

2.6.2 Описание основного алгоритма

Программа проходит по всем файлам в директории, записывая их имена в вектор. Далее запускается функция сортировки с переданным компаратором, сравнивающим время последнего изменения двух файлов. Затем файлы из вектора выводятся с указанием атрибутов.

Реализация алгоритмов представлена в разделе приложения [7.7](#).

2.6.3 Руководство пользователя

Пользователю необходимо запустить программу, и она выведет файлы в директории отсортированном виде.

```

kirikklin@DESKTOP-CL08S8G:/mnt/d$ /mnt/c/labs_os/2_5/main
1 : kursach - ярлык (2).lnk : size=845 : 281474976710820 : UID=1000 GID=1000 : Mon Jan 11 22:37:48 2021
1 : res_testfile.txt : size=243 : 1125899906846017 : UID=1000 GID=1000 : Wed Mar 31 19:21:35 2021
1 : kod - ярлык.lnk : size=853 : 1688849860267658 : UID=1000 GID=1000 : Mon May 3 23:28:24 2021
1 : TestFile2.txt : size=4 : 2251799813688638 : UID=1000 GID=1000 : Tue May 4 12:58:38 2021
1 : world1.txt : size=502 : 3659174697242167 : UID=1000 GID=1000 : Tue May 11 11:31:53 2021
1 : MyMario.rar : size=11678725 : 4222124650663784 : UID=1000 GID=1000 : Sat May 15 12:58:55 2021
1 : kurs (1).pdf : size=360282 : 1688849860267801 : UID=1000 GID=1000 : Sun May 16 18:43:05 2021
1 : -----.pdf : size=441247 : 5348024557506473 : UID=1000 GID=1000 : Sun May 23 23:21:39 2021
1 : CourseWorkMalckow.pdf : size=441249 : 1407374883557287 : UID=1000 GID=1000 : Sun May 23 23:33:46 2021
1 : world2.txt : size=1664 : 5066549580795796 : UID=1000 GID=1000 : Thu Dec 23 14:01:19 2021
kirikklin@DESKTOP-CL08S8G:/mnt/d$ |

```

Рис. 7: Пример работы приложения "Сортировка файлов".

3 Локальная клиент-серверная модель

3.1 Используемые системные объекты

Основными системными объектами данной работы являются семафоры, очереди сообщений, разделяемая память и сокеты. В модели клиент-серверного приложения основная проблема заключается в том, что пользователей программного обеспечения может быть много, и сервер должен уметь регулировать клиентов и своевременно предоставлять доступ к ресурсам, в противном случае может возникнуть проблема "затирания" одним пользователем данных другого. Множество клиентов – это множество процессов, чтобы их синхронизировать, необходимы семафоры и очереди сообщений. Совместно с семафорами используется разделяемая память – общий сегмент в памяти системы, с которым может работать любой процесс. Как известно, изначально все процессы имеют только свою область памяти, и они не могут обращаться к данным другого процесса, разделяемая память позволяет создавать область памяти, к которой может обращаться любой процесс, подобным образом и происходит взаимодействие между клиентами.

3.2 Очередь сообщений

3.2.1 Техническое задание

С помощью очереди сообщений организовать клиент-серверную модель для решения кубического уравнения. ПО должно делать следующее

1. Пользователь в клиенте задаёт 4 коэффициента и посылает их серверу в одном

сообщении вместе со своим PID.

2. Сервер принимает сообщение и посылает три корня клиенту.
3. Клиентов может быть много. Каждый из них ждёт сообщение с типом равным своему PID.

3.2.2 Описание основного алгоритма

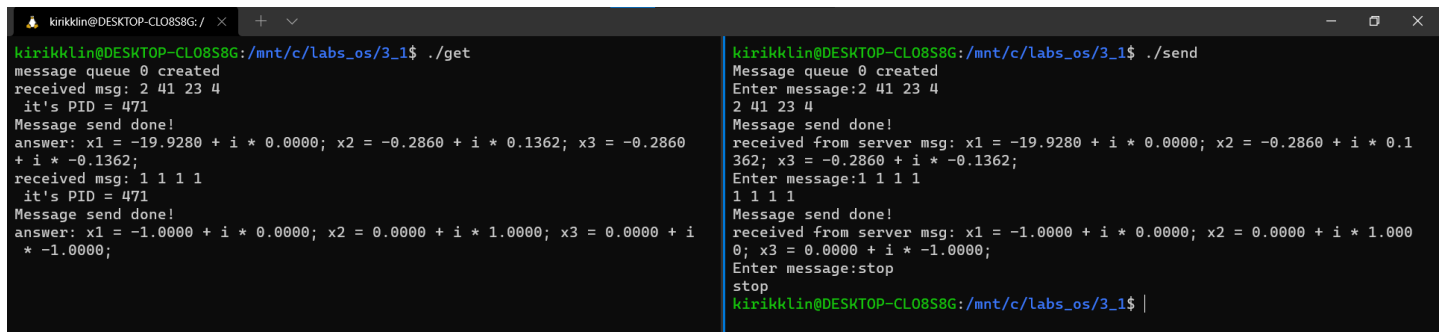
В клиенте инициализируется очередь сообщений, далее в цикле запрашиваются 4 коэффициента. Строка с коэффициентами с помощью очереди сообщений передаётся на сервер. Затем клиент ожидает получения ответа на сообщение, с таким же типом, после получения выводит результат.

Сервер получает от клиента строку с сообщением, обрабатывает её и отправляет 4 полученных коэффициента в функцию решения кубического уравнения. Затем отправляет клиенту результат.

Реализация алгоритмов представлена в разделе приложения [7.8](#).

3.2.3 Руководство пользователя

1. Запустить сервер.
2. Запустить клиент.
3. Ввести 4 коэффициента кубического уравнения.
4. Клиент выведет решение уравнения(3 корня).
5. Для завершения ввода пользователь должен ввести «stop».



```
kirikklin@DESKTOP-CL08S8G:/mnt/c/labs_os/3_1$ ./get
message queue 0 created
received msg: 2 41 23 4
it's PID = 471
Message send done!
answer: x1 = -19.9280 + i * 0.0000; x2 = -0.2860 + i * 0.1362; x3 = -0.2860
+ i * -0.1362;
received msg: 1 1 1 1
it's PID = 471
Message send done!
answer: x1 = -1.0000 + i * 0.0000; x2 = 0.0000 + i * 1.0000; x3 = 0.0000 + i
* -1.0000;

kirikklin@DESKTOP-CL08S8G:/mnt/c/labs_os/3_1$ ./send
Message queue 0 created
Enter message:2 41 23 4
2 41 23 4
Message send done!
received from server msg: x1 = -19.9280 + i * 0.0000; x2 = -0.2860 + i * 0.1
362; x3 = -0.2860 + i * -0.1362;
Enter message:1 1 1 1
1 1 1 1
Message send done!
received from server msg: x1 = -1.0000 + i * 0.0000; x2 = 0.0000 + i * 1.000
0; x3 = 0.0000 + i * -1.0000;
Enter message:stop
stop
kirikklin@DESKTOP-CL08S8G:/mnt/c/labs_os/3_1$ |
```

Рис. 8: Пример работы приложения "Очередь сообщений".

3.3 Семафоры и разделяемая память

3.3.1 Техническое задание

Реализовать клиент-сервер на семафорах и разделяемой памяти. Он должен делать следующее:

1. сервер принимает от клиента запросы в виде примера из двух вещественных чисел и бинарной операции между ними (+, *, -, /);
2. сервер возвращает ответ;
3. клиентов может быть много;
4. запросы клиенты отправляют в интерактивном режиме;
5. клиент завершается, если пользователь введёт пустую строку.

3.3.2 Описание основного алгоритма

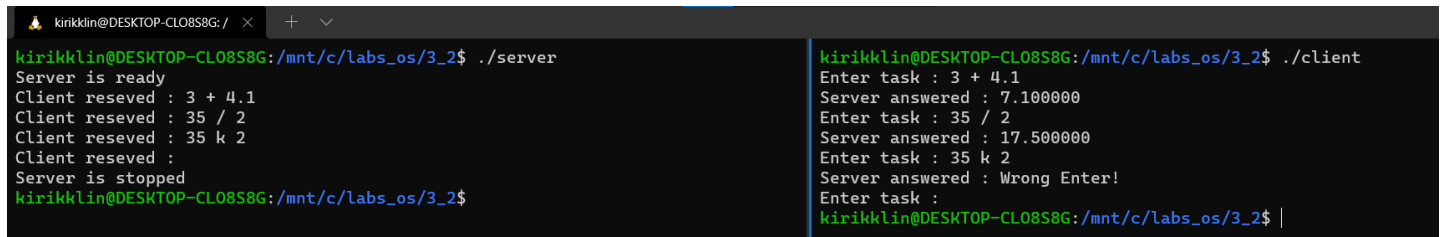
Изначально сервер создает массив из двух семафоров и разделяемую память и запускается. После этого клиент открывает семафоры и разделяемую память сервера и говорит серверу о том, что хочет подключиться, делая операцию `semop(0,-1)`. Затем клиент пишет необходимые данные и отправляет их, делая операцию `semop(1,0)`. Далее сервер проверяет, возможность обращения к разделяемой памяти, с помощью операции `semop(-1,0)`, получает данные и делает необходимые вычисления. Далее с помощью операции `semop(1,1)`, разрешает клиенту обратиться к разделяемой памяти. Клиент принимает ответ от сервера (`semop(-1,-1)`). Клиент возвращает семафоры в исходное положение, тем самым завершает работу сервера. Операция `semop()` работает

следующим образом: если в одном поле операции стоит 0, а на соответствующем поле в семафоре не 0, то процесс блокируется, также процесс блокируется, если попытаться вычесть число из 0. В остальных случаях к семафору прибавляется соответствующее значение операции.

Реализация алгоритмов представлена в разделе приложения [7.9](#).

3.3.3 Руководство пользователя

1. Запустить сервер.
2. Запустить клиент.
3. Программа запросит у пользователя ввести пример. Если введён некорректный пример, будет выведено сообщение об ошибке.
4. Для завершения ввода, требуется ввести пустую строку.



```
kirikklin@DESKTOP-CLO8S8G: /mnt/c/labs_os/3_2$ ./server
Server is ready
Client reseved : 3 + 4.1
Client reseved : 35 / 2
Client reseved : 35 k 2
Client reseved :
Server is stopped
kirikklin@DESKTOP-CLO8S8G: /mnt/c/labs_os/3_2$

kirikklin@DESKTOP-CLO8S8G: /mnt/c/labs_os/3_2$ ./client
Enter task : 3 + 4.1
Server answered : 7.100000
Enter task : 35 / 2
Server answered : 17.500000
Enter task : 35 k 2
Server answered : Wrong Enter!
Enter task :
kirikklin@DESKTOP-CLO8S8G: /mnt/c/labs_os/3_2$
```

Рис. 9: Пример работы приложения "Семафоры и разделяемая память".

4 Сетевое программирование

4.1 Используемые системные объекты

В приложении используются сокеты, с помощью которых можно установить соединение между клиентом и сервером. В работе были использованы два протокола – TCP/IP и UDP, у каждого есть свои недостатки и преимущества. у UDP, выше скорость работы, чем у TCP/IP, но не гарантируется целостность переданных сообщений, какие-то биты могут теряться во время передачи. TCP/IP следует использовать, если пользователю важен каждый бит информации и потеря хотя бы одного ведет к критическим последствиям, например, алгоритм хеширования MD5, изменение одного бита в сообщении, полностью меняет хеш. Что касается протокола UDP, то его преимущество заключается в том, что данные пересылаются быстро, но целостность информации не гарантируется, этот протокол следует использовать, например, в видеоиграх или для передачи информации по телефону. В обоих протоколах передача информации происходит по IP и порту сервера и клиента.

4.2 Подсчёт слов

4.2.1 Техническое задание

Написать сетевое приложение в виде клиента и сервера. Оно должен делать следующее:

1. Задача сервера: принимать строку текста и возвращать число слов в строке.
2. Задача клиента: запрашивать строку у пользователя, посылать строку серверу и, получив ответ сервера, печатать её на экран.
3. Сокет должен быть основан на TCP. (SOCK_STREAM или SOCK_SEQPACKET)

4.2.2 Описание основного алгоритма

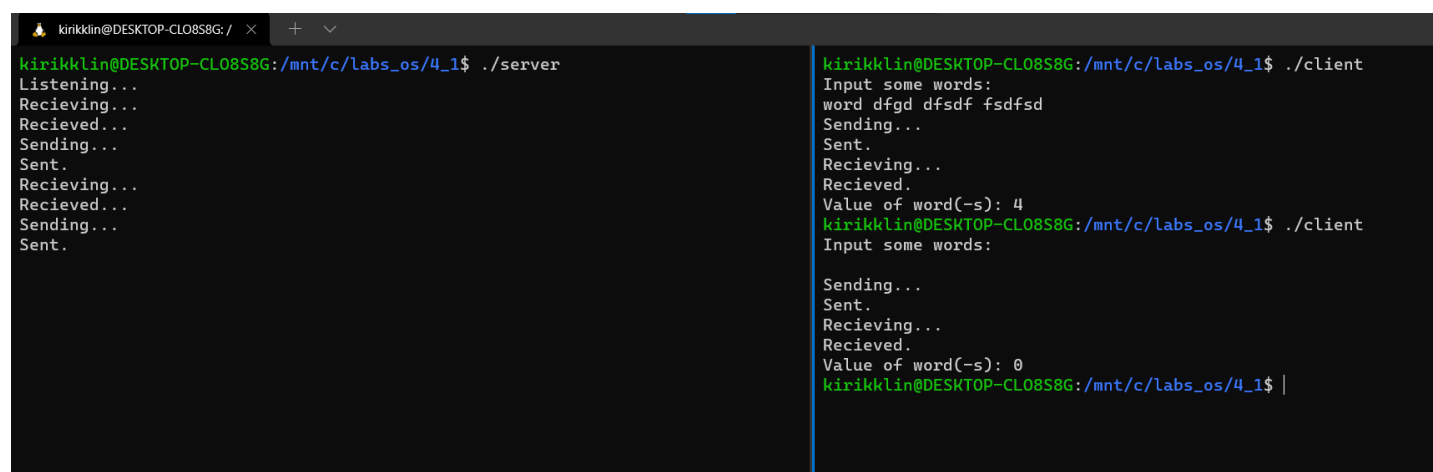
Клиент инициализирует сокеты, далее просит пользователя ввести строку с словами, отправляет её серверу, и ждёт получения сообщения от сервера. После выводит полученную от сервера информацию.

Сервер так же инициализирует сокеты. Далее в цикле переходит в слушающее состояние и при подключении пользователя создаёт новый сокет, для непосредственного общения с клиентом и делает системный вызов `fork()`. В процессе-ребёнке ждёт получения сообщения от пользователя, а родитель возвращается в слушающее состояние для подключения новых пользователей. К полученной строке применяется функция `word_counter` для подсчёта слов. Результат функции отправляется клиенту.

Реализация алгоритмов представлена в разделе приложения [7.10](#).

4.2.3 Руководство пользователя

1. Запустить сервер.
2. Запустить клиент.
3. Программа запросит у пользователя ввести строку с словами для подсчёта.
4. Клиент выведет результат, полученный от сервера.



```
kirikklin@DESKTOP-CLO8S8G: /mnt/c/labs_os/4_1$ ./server
Listening...
Receiving...
Received...
Sending...
Sent.
Receiving...
Received...
Sending...
Sent.

kirikklin@DESKTOP-CLO8S8G: /mnt/c/labs_os/4_1$ ./client
Input some words:
word dfgd dfsdf fsdfs
Sending...
Sent.
Receiving...
Received.
Value of word(-s): 4
kirikklin@DESKTOP-CLO8S8G: /mnt/c/labs_os/4_1$ ./client
Input some words:
Sending...
Sent.
Receiving...
Received.
Value of word(-s): 0
kirikklin@DESKTOP-CLO8S8G: /mnt/c/labs_os/4_1$ |
```

Рис. 10: Пример работы приложения "Подсчёт слов".

4.3 Чат

4.3.1 Техническое задание

Написать сетевое приложение — чат. Оно должен делать следующее:

1. Пользователь через аргументы задаёт 2 параметра:
 - -a IP_адрес:порт (пример -a 127.0.0.1:1024);
 - -u имя_пользователя (пример -u Ali).
2. Приложение по указанному порту отправляет вводимые пользователем сообщения и печатает принятые сообщения. Перед любым сообщением указывается логин пользователя (пример Ali: Hello!).
3. Выход из приложения выполняется по вводу строки «quit!».
4. Общение происходит через протокол UDP. (SOCK_DGRAM)

4.3.2 Описание основного алгоритма

1. Инициализируются сокеты.
2. Имя, полученное аргументом из командной строки, записывается в поле структуры, которая отправляется с помощью сокетов.
3. Далее запускается функции, возвращающая порт, полученный из аргумента командной строки, и записывающая ip сервера, в строку, переданную вторым аргументом функции.
4. Полученными данными инициализируется структура sockaddr_in my_addr, она связывается с сокетом sock.
5. Далее клиент запрашивает у пользователя порт чата друга, чтобы отправлять на него сообщения.
6. Программа совершает системный вызов fork().
7. В процессе-ребёнке клиент ждёт ввода сообщения от пользователя. Проверяет, равна ли введённая строка «quit», для завершения приложения. Если так, то цикл отправки сообщений завершается, и процессу-родителю отправляется сигнал для завершения, иначе отправляет сообщение пользователю с полученным портом, "другом".

8. В процессе-родителе запускается цикл, в котором происходит получение сообщений от "друга".
9. Сокет закрывается.

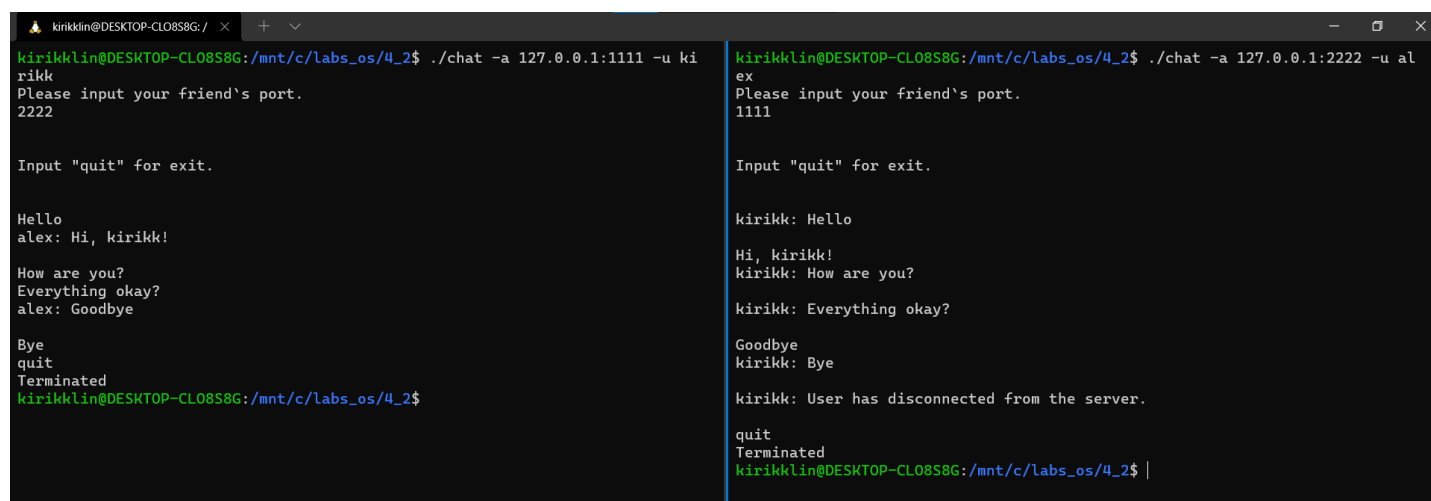
Реализация алгоритмов представлена в разделе приложения [7.11](#).

4.3.3 Руководство пользователя

Следующую инструкцию нужно выполнить для двух клиентов.

1. Пользователь должен запустить чат с аргументами:
 - (a) -a IP_адрес:порт (пример -a 127.0.0.1:1111);
 - (b) -u имя_пользователя (пример -u kirikk).

```
./chat -a 127.0.0.1:1111 -u kirikk
```
2. Программа запросит ввести порт друга, пользователь должен ввести его.
3. После подключения другого пользователя, можно писать сообщения, для выхода из чата, нужно ввести «quit».



```
kirikklin@DESKTOP-CL08S8G:/mnt/c/labs_os/4_2$ ./chat -a 127.0.0.1:1111 -u kirikk
Please input your friend's port.
2222

Input "quit" for exit.

Hello
alex: Hi, kirikk!

How are you?
Everything okay?
alex: Goodbye

Bye
quit
Terminated
kirikklin@DESKTOP-CL08S8G:/mnt/c/labs_os/4_2$

kirikklin@DESKTOP-CL08S8G:/mnt/c/labs_os/4_2$ ./chat -a 127.0.0.1:2222 -u alex
Please input your friend's port.
1111

Input "quit" for exit.

kirikk: Hello

Hi, kirikk!
kirikk: How are you?

kirikk: Everything okay?

Goodbye
kirikk: Bye

kirikk: User has disconnected from the server.

quit
Terminated
kirikklin@DESKTOP-CL08S8G:/mnt/c/labs_os/4_2$
```

Рис. 11: Пример работы приложения "Чат".

Часть II

Индивидуальная часть

5 Клиент-серверное ПО «MD5. База данных - библиотека»

5.1 Техническое задание

Необходимо реализовать клиент-серверное приложение, имеющее следующую структуру:

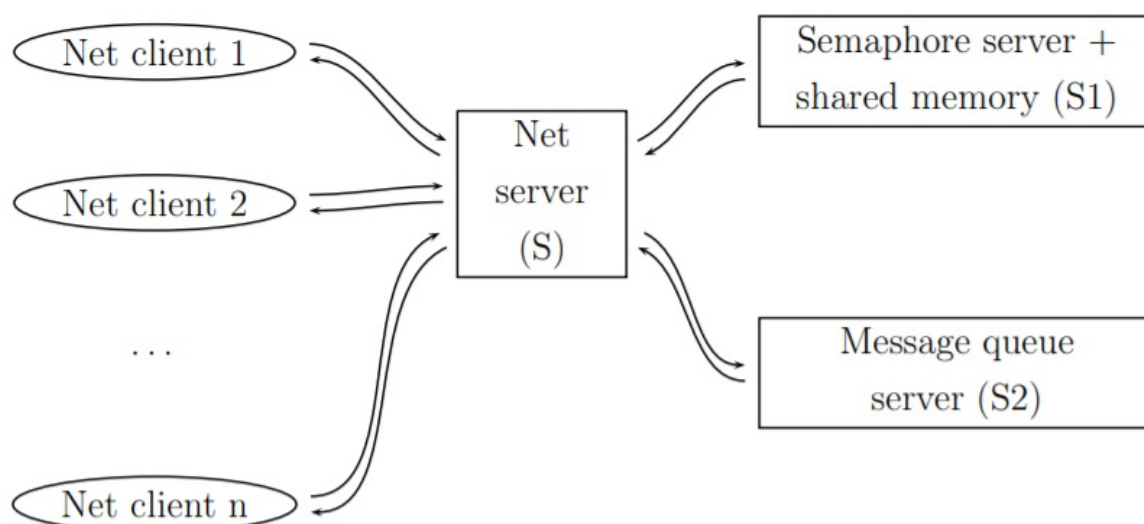


Рис. 12: Структура программного комплекса.

Сетевой сервер принимает 2 типа запросов от клиентов:

1. решение задачи «алгоритм хеширования MD5»;
2. решение задачи «Построение удалённой базы данных – библиотека.»

Используя данные, передаваемые от клиента по первому типу запроса, сетевой сервер **S** формирует запрос к серверу **S₁**, а по второму типу запроса — к серверу **S₂**. Серверы **S₁** и **S₂** формируют соответствующие задачам ответы, которые сервер **S** транслирует инициирующему клиенту.

5.2 Формулировка задачи 1

Разработать клиент-серверное сетевое приложение, реализующее следующий функционал: хеширование файла с помощью алгоритма MD5. На вход подаётся файл, далее приложение возвращает хеш.

5.3 Формулировка задачи 2

Разработать клиент-серверное сетевое приложение: удалённая база данных – библиотека. Приложение способно сохранять, удалять, печатать принимаемую от пользователя информацию. Серверная часть не принимает никаких данных. Она взаимодействует с клиентом.

Клиент принимает команду от пользователя (INSERT(), DELETE(), PRINT()). В зависимости от команды возвращает результат выполнения.

В базе данных хранятся следующие данные:

- Код книги.
- Название книги.
- ФИО автора.
- Количество книг в библиотеке.

5.4 Протокол общения клиента и сервера S

Для общения клиента и сервера используется протокол TCP, реализованный с помощью сокетов. Протокол управления передачей (TCP) — это стандарт связи, который позволяет прикладным программам и вычислительным устройствам обмениваться сообщениями по сети. Он предназначен для отправки пакетов по интернету и обеспечения успешной доставки данных и сообщений по сетям.

В клиенте программно задан IP сервера и порт. Соединение осуществляется с помощью сокетов по этим данным.

Клиент запрашивает у пользователя, какой тип приложения должен быть: алгоритм хеширования MD5 или удалённая база данных - библиотека. В зависимости от выбора в поле type, структуры MESSAGE, кладётся значение: MD5 или DATABASE,

и структура отсылается сетевому серверу. Далее, общение осуществляется в зависимости от выбранной задачи.

5.5 Протокол общения серверов S и S1

Если пользователь выбрал алгоритм хеширования MD5, то сетевой сервер с помощью семафоров устанавливает соединение с локальным сервером.

Сетевой сервер отправляет серверу на семафорах, используя разделяемую память, данные в специальной структуре SEM_STRUCT: символьную строку и переменную типа int. Возможность использования разделяемой памяти определяется значениями семафоров.

5.6 Протокол общения серверов S и S2

Центральный сервер S формирует структуру, в которой содержится массив символов и переменная типа long(используется для проверки типа сообщения), а затем с помощью очереди сообщений отправляет её локальному серверу S2.

5.7 Структура приложения

Описывается логика работы приложения, основные блоки и модули, а также взаимодействие между ними. Перечислить разработанные модули и функции. Описать последовательность вызовов функций.

Приложение состоит из клиента, сетевого сервера, сервера на семафорах и разделяемой памяти, сервера на очереди сообщений.

client.cpp:

- int main()

Структура клиента:

1. Происходит связь с сокетом, затем взаимодействие с сервером.
2. Клиент предоставляет функционал выбора задачи (MD5 или база данных).
3. Направляет полученные данные на сервер.

4. Если пользователь выбрал алгоритм, открывает переданный пользователем файл или получает от пользователя строку, которую надо захешировать.
5. Отправляет частями по 64 байта информацию из файла или строки.
6. Если пользователь выбрал базу данных, то запускается цикл общения клиента с сервером с помощью сокетов.

structures.h:

- MESSAGE – структура для обмена данными с помощью сокетов.
- SEM_STRUCT – структура для обмена данными с помощью семафоров и разделяемой памяти.
- struct msg_buf – структура для обмена данными с помощью очереди сообщений.

netserver.cpp

- int main()
- int db_msg_queue(int, MESSAGE&) – производит общение сетевого сервера и сервера на очереди сообщений.
- int md5_sem(int, MESSAGE&) – производит общение сетевого сервера и сервера на семафорах и разделяемой памяти.

Структура сетевого сервера:

1. Происходит связь с сокетом.
2. После подключения клиента происходит системный вызов fork(), и в процессе-ребёнке, в зависимости от полученных данных, вызывается общение с одним или с другим сервером, по средством вызова функций db_msg_queue или md5_sem.

DBLibrary.h:

class DBLibrary.h:

Публичная методы:

- `string new_command(const string& command);` – функция, обрабатывающая строку с командой.

Приватные:

- `int _find(string& to_find)` – функция поиска строки в файле.
- `int insert(string& text)` – добавляет новые записи в базу данных.
- `string building_string(const string& line)` – строит строчку для вывода из строки из файла.
- `int _delete(const string& line)` – удаляет запись из базы данных по имени.

`server_db.cpp`:

Структура сетевого сервера:

1. Сервер в цикле получает сообщения с помощью очереди сообщений.
2. Вызывает функцию обработки команды(`new_command`).
3. Отправляет результат сетевому серверу.

`MD5.h`:

`class MD5`:

Публичные методы:

- `MD5()` – инициализирует переменные и массив констант.
- `int new_block(char data[], uint4 len)` – хеширует блок в 512 бит.

Приватные методы:

- Функции используемые в раундах хеширования:

`static inline uint4 F(uint4 x, uint4 y, uint4 z)`

`static inline uint4 G(uint4 x, uint4 y, uint4 z)`

`static inline uint4 H(uint4 x, uint4 y, uint4 z)`

`static inline uint4 I(uint4 x, uint4 y, uint4 z)`

- `static inline uint4 CLS(uint4 x, int n)` – циклический сдвиг влево.
- `int to_uint32(const char data[], uint4 first, uint4 len)` – переводит блок из 512 бит, из символьной строки в массив `unsigned int`.
- `void to_uint32_cut(int flag)` – добавляет единичный бит и заносит битовое представление количества переданных байт в конец битового представления.
- `void hmd5()` – заносит данные в 4 тридцати двухбитные переменные.
- `void encode()` – Перезаписывает байты из четырёх переменных `unsigned int` в массив байт.
- `void hexdigest(char buf[]) const` – составляет хеш из массива байт, перезаписывая переменные в шестнадцатеричный формат, в массиве `buf`.

`server_md5.cpp`:

1. Инициализация семафоров и разделяемой памяти.
2. В цикле происходит общение с сетевым сервером, на каждой итерации вызывается функция `new_block`.
3. На последней итерации `new_block`, положит хеш в структуру для отправки сетевому серверу.

5.8 Основные алгоритмы

Опишем работу клиента.

1. Инициализируется структура `sockaddr_in addr`: указывается порт равный «1234», и `ip` сервера, в нашем случае, это локальный адрес: 127.0.0.1.
2. Сокеты соединяются с сервером.
3. Опишем логику обработки запросов блок схемой.

Опишем работу сетевого сервера.

`int main()`:

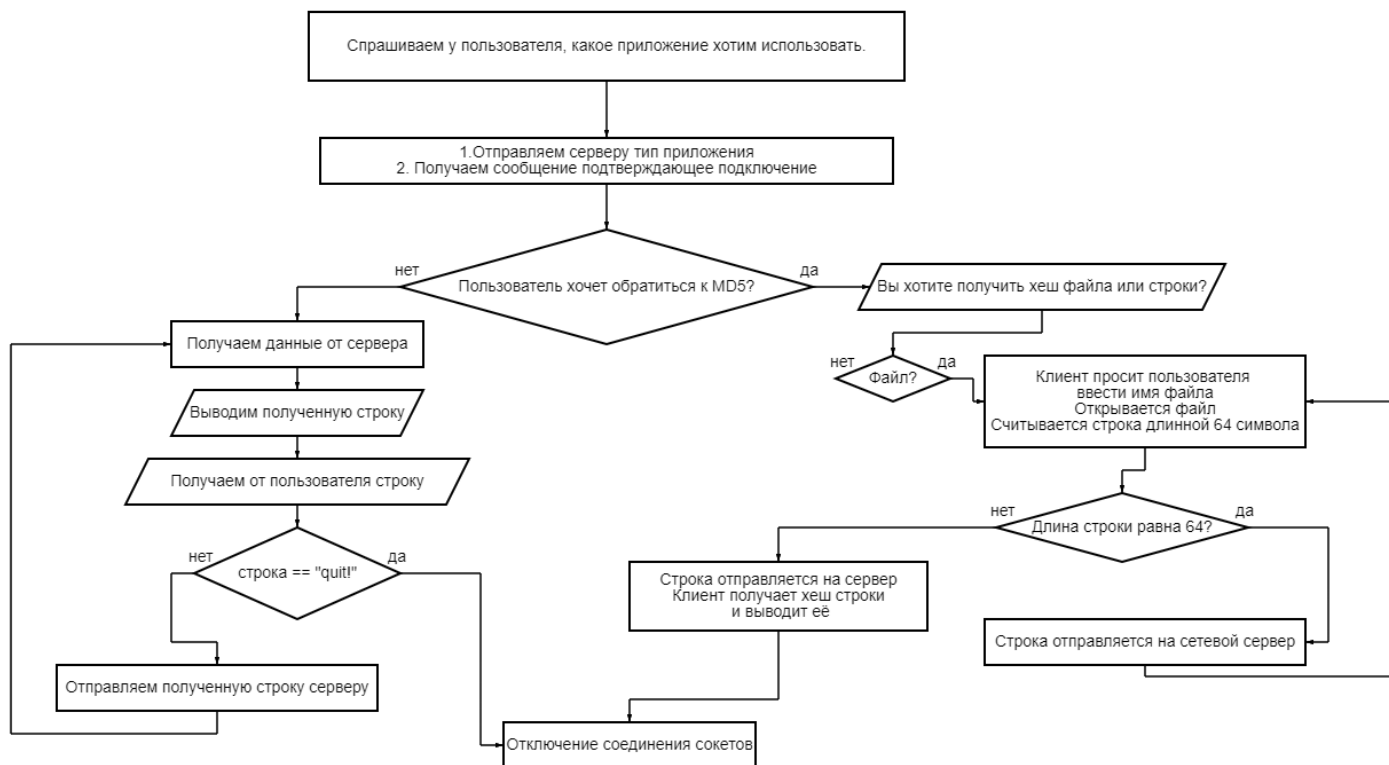


Рис. 13: Логика обработки запросов в клиенте.

1. Настройка и подключение сокетов.
2. В цикле while вызывается функция listen, она ждёт подключение новых клиентов, как только произойдёт подключение, создастся новый сокет accept_sock.
3. Далее происходит системный вызов fork(), в процессе-ребёнке будет происходить общение с каким-то из двух серверов.
4. Процесс-родитель снова переходит в слушающее состояние.
5. Процесс-ребёнок получает информацию о типе приложения. Если поле type объекта mes_struct структуры MESSAGE, равно MESSAGE::MD5, то вызывается md5_sem(), иначе db_msg_queue(), функции принимают дескриптор сокета для общения с клиентом и объект mes_struct.

int db_msg_queue(int, MESSAGE&):

1. Инициализация очереди сообщений

2. Клиенту отправляется строка, подтверждающая подключение к серверу на очереди сообщений.
3. Цикл общения с клиентом.
4. Отправляем строку, предлагающую ввод клиенту.
5. Получаем от клиента строку.
6. Проверяем, равна ли она «stop!», если равна, то выходим из цикла.
7. Отправляем строку серверу на очереди сообщений.
8. Получаем обработанную строку от сервера и отправляем клиенту.

`int md5_sem(int accepted_sock, MESSAGE& mes_struct):`

1. Инициализация семафоров и разделяемой памяти.
2. Указатель `shm_adress` указывает на область разделяемой памяти.
3. Клиенту отправляется строка, подтверждающая подключение к серверу на семафорах.
4. Цикл общения с клиентом.

В цикле:

5. Получаем от клиента строку, делаем `semop(0,-1)` для проверки доступа разделяемой памяти.
6. Кладём в память строку и количество полученных символов для сервера.
7. Делаем `semop(1,0)`, чтобы разрешить серверу считать строку.
8. Делаем `semop(-1, -1)`, чтобы проверить возможность получения данных от сервера.
9. `semop(0,1)` чтобы вернуть значения семафоров в исходное положение и разрешить сетевому серверу обращаться к разделяемой памяти.

Опишем работу сервера на очереди сообщений:

```
int main():
```

1. Инициализация очереди сообщений.

2. Цикл общения с сетевым сервером.

В цикле:

3. Получаем сообщение от сетевого сервера.

4. Вызываем метод обработки строки `new_command` и копируем результат в `msg.mtext`, для отправки серверу.

5. Отправляем данные сетевому серверу.

Опишем класс `DBLibrary`:

Класс `DBLibrary` предоставляет функционал работы с базой данных - библиотека.

Пользователь передаёт команды вида `COMMAND (data)`. Данные хранятся в файле в следующем виде: `code:title:author's name:count`:

Класс содержит следующие поля:

1. `string file_name{ "database_library.txt"}`

```
string new_command(const string& command):
```

Проверяем команду на корректность с помощью регулярного выражения.

Далее из строки получаем саму команду и с помощью конструкции `if - else` вызываем нужную функцию обработчик, передавая в неё оставшуюся часть строки – данные переданные пользователем.

```
int _find(string& to_find):
```

Циклом `while` проходим построчно по всему файлу, проверяя с помощью метода `find`, наличие слова в строке, если оно встречается, то возвращаем код книги, и в переданной строке `to_find` присваиваем представление строки из файла.

```
int DBLibrary::print(string& buf):
```

Получаем имя из строки вида `"(title)"`, с помощью регулярного выражения. Вызываем функцию `find`, если она вернула значение отличное от `-1`, то вызываем функцию `building_string`, и результат присваиваем переменной `to_print`.

```
int DBLibrary::insert(string& text)
```

1. Получим имя из строки вида "(kod, title, author, count)" , далее, если такое имя существует в файле, получаем количество таких книг. Открываем временный файл и начинаем перезаписывать в него все строки, пока не встретится запись о нужной книге. Складываем количества книг из найденной строки и кода, полученного из команды, переданной пользователем и прибавляем к строке из файла и записываем её обратно в файл.

Закрываем оба файла удаляем файл базы данных, переименовываем временный, в "database_library.txt".

Если такого имени не существует, то просто дописываем в конец файла, новую запись.

```
int DBLibrary::_delete(const string& line)
```

Удаление происходит аналогично вставке, только, если найдена строка с именем переданным пользователем, она просто не копируется во временный файл, без каких-либо действий. Если строчка была удалена, flag_to_return становится равен 1. Если flag_to_return равен 1, то возвращается 0, из этого следует, что запись была удалена, иначе возвращается -1, что сигнализирует о том, что запись не была удалена.

Опишем работу сервера на семафорах:

```
int main()
```

1. Инициализация семафоров и разделяемой памяти (семафоры инициализируются значениями (0,1)).
2. Цикл общения с сетевым сервером.
В цикле:
 3. Делаем semop(-1, 0) для проверки возможности обращения к разделяемой памяти.
 4. Вызываем функцию new_block, передавая полученные данные.
 5. Делаем semop(1,1), чтобы разрешить клиенту обратиться к разделяемой памяти.
 6. После цикла закрываем семафоры и разделяемую память.

Опишем класс MD5:

Класс содержит следующие поля:

1. `uint4 bits512[16]` – Хранит битовый набор для обработки
2. `uint4 size_byte 0` – Количество байт во всём сообщении.
3. `uint4 state[4]` – переменные, которые получают после 4х раундов хеширования.
4. `uint1 digest[16]` – хеш в байтовом представлении в порядке little-endian
5. `uint4 T[64]` – таблица констант для хеширования.
6. `uint4 S[64]` – таблица значений циклического сдвига влево, для каждого раунда.

MD5():

1. Вызывается функция инициализации, в ней инициализируется массив `state` и количество символов в сообщении.
2. Инициализируется массив констант: `T`.

`int new_block(char data[], uint4 len)` – функция обрабатывает один блок в 512 бит.

1. вызывается функция, преобразующая символьную строку в массив `unsigned int`.
2. В ранее вызванной функции в массив `bits512`, был записан битовый набор, вызывается функция `hmd5`, которая производит определённые действия с ним.
3. Если длина полученного сообщения меньше 64(это означает, что был передан последний блок данных), то если количество больше 56, это означает, что следует добавить ещё один блок, это делает функция `to_uint32_cut`(так же, если количество байт равно 0, это означает, что последний блок закончился, и в функции `to_uint32`, в конец массива, не был дописан единичный бит, как этого требует алгоритм, и её нужно добавить).
4. Значения массива `state` перезаписываются в байтовое представление.
5. Вызывается функция `hexdigest`, которая перезаписывает массив `digest` в шестнадцатеричное представление.

6. Все переменные инициализируются заново для следующих запросов клиентов.

```
int to_uint32(const char data[], uint4 first, uint4 len)
```

1. `size_byte` увеличивается на количество переданных байт.

2. Массив инициализируется нулями.

3. В массив записывается битовое представление переданной строки.

4. Если $i == 64$, возвращается 0, потому что к этому блоку ничего больше не добавить, так как он заполнен.

5. Если $i*8 < 448$, значит, ещё есть место для добавления единичного бита, и 8-ми байтов, в котырых, разместятся, биты количества байтов.

6. Иначе, если $i*8 \% 512$ не равно 0, то ещё хватает места для единичного бита, и он дописывается.

```
void MD5::to_uint32_cut(int count):
```

Массив `bits512` инициализируется нулями, далее, если `count == 0`, в начало дописывается единичный бит, и в конец массива дописывается размер входящего сообщения в битах.

Выше описанные функции выполняют первые два шага алгоритма.

Шаг 1.

Запишем длину сообщения в L . Это число целое и неотрицательное. Кратность каким-либо числам необязательна. После поступления данных идёт процесс подготовки потока к вычислениям.

Сначала к концу потока дописывают единичный бит.

Затем добавляют некоторое число нулевых бит такое, чтобы новая длина потока L' стала сравнима с 448 по модулю 512, ($L' = 512 \times N + 448$ $L' = 512 \times N + 448$). Выравнивание происходит в любом случае, даже если длина исходного потока уже сравнима с 448.

Шаг 2.

В конец сообщения дописывают 64-битное представление длины данных (количество бит в сообщении) до выравнивания. Сначала записывают младшие 4 байта,

затем старшие. Если длина превосходит $2^{64} - 1$, то дописывают только младшие биты (эквивалентно взятию по модулю 2^{64}). После этого длина потока станет кратной 512. Вычисления будут основываться на представлении этого потока данных в виде массива слов по 512 бит.

```
void MD5::hmd5():
```

Инициализируются переменные a,b,c,d, значениями из state, все вычисления будут проходить с ними.

Цикл for реализует прохождение по четырём этапам, в каждом этапе 16 раундов. Каждый раунд это определённая логическая операция с битами. Далее переменные меняются местами, и к сумме переменных «a», «f», $\langle T[i] \rangle$, применяется циклический сдвиг на «s[i]» бит, и результат прибавляется к «b», а затем это всё присваивается переменной a. После прохождения всех раундов, переменные a,b,c,d прибавляются к значениям массива state.

```
void MD5::encode():
```

Байты из массива bits512 копируются в массив digest в порядке little-endian.

```
void MD5::hexdigest(char buf[]) const:
```

Функция записывает 16-ричные представления байтов из digest, в полученный указатель char*.

Реализация алгоритмов представлена в разделе приложения [7.12](#).

5.9 Руководство пользователя

1. Запустить серверы.

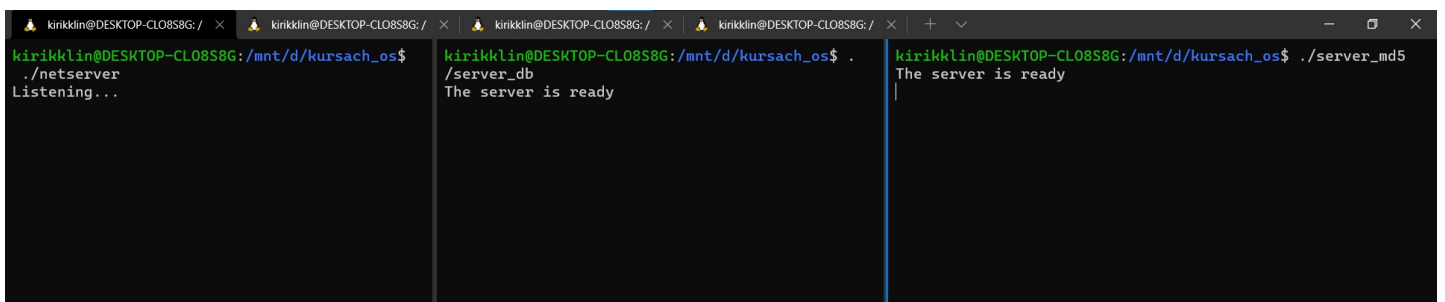
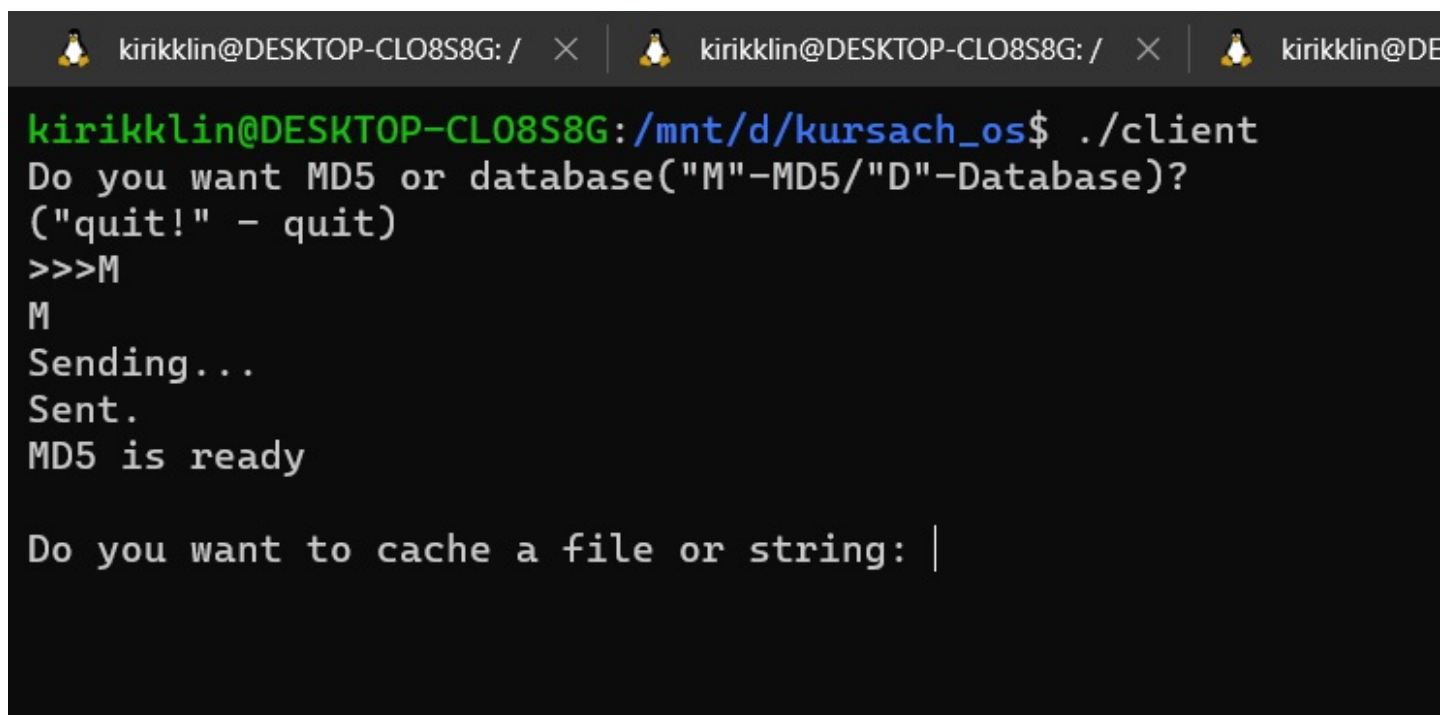


Рис. 14: Запущенные серверы.

2. Пользователь должен запустить клиента. И выбрать, какое приложение хочет использовать. Для этого надо ввести «М» или «D», также пользователь может написать «quit!» для выхода. Если будет введена некорректная строка, программа попросит повторить ввод.



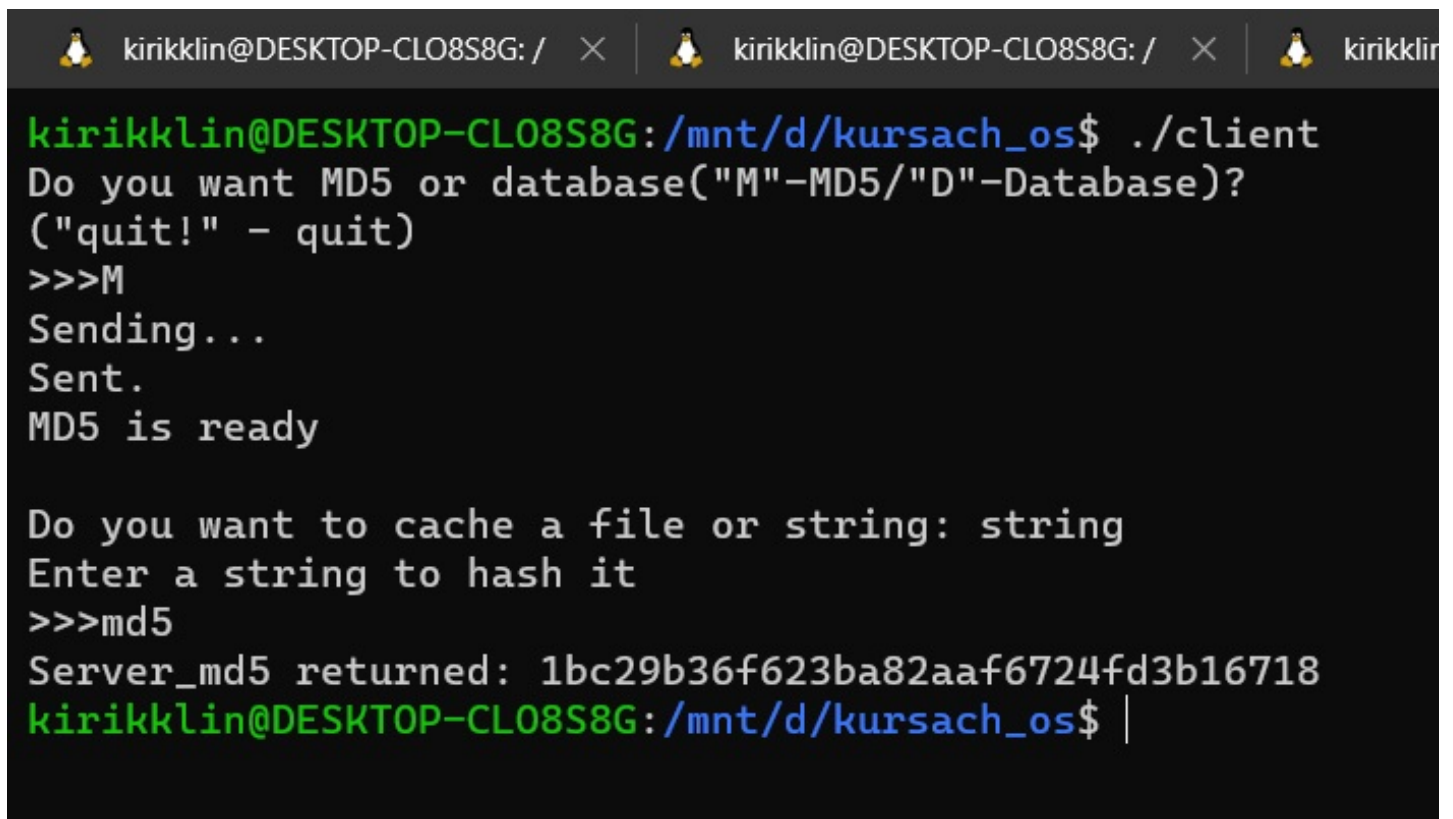
```
kirikklin@DESKTOP-CLO8S8G: /  × | kirikklin@DESKTOP-CLO8S8G: /  × | kirikklin@DE
kirikklin@DESKTOP-CLO8S8G:/mnt/d/kursach_os$ ./client
Do you want MD5 or database("M"-MD5/"D"-Database)?
("quit!" - quit)
>>>M
M
Sending...
Sent.
MD5 is ready

Do you want to cache a file or string: |
```

Рис. 15: Общение с клиентом.

Далее, общение с программой будет различаться, в зависимости от типа выбранного приложения. Рассмотрим общение с программой, если пользователь выбрал MD5:

1. Пользователь должен выбрать, он хочет захешировать файл или строку, для этого надо ввести «file» и «string» соответственно. Если ввести что-то иное, программа завершится.
2. Если пользователь ввёл «file», то программа попросит ввести его имя файла.
3. Клиент выведет хеш файла.

A terminal window with a dark background and light-colored text. The window title bar shows three tabs, each with a penguin icon and the text 'kirikklin@DESKTOP-CLO8S8G: /'. The terminal content shows a user at the prompt 'kirikklin@DESKTOP-CLO8S8G:/mnt/d/kursach_os\$' running './client'. The program asks 'Do you want MD5 or database("M"-MD5/"D"-Database)?' and the user enters 'M'. The program then says 'Sending...', 'Sent.', and 'MD5 is ready'. It then asks 'Do you want to cache a file or string: string' and the user enters 'string'. The program asks 'Enter a string to hash it' and the user enters 'md5'. The program then outputs 'Server_md5 returned: 1bc29b36f623ba82aaf6724fd3b16718'. The prompt returns to 'kirikklin@DESKTOP-CLO8S8G:/mnt/d/kursach_os\$' with a cursor.

```
kirikklin@DESKTOP-CLO8S8G: / × | kirikklin@DESKTOP-CLO8S8G: / × | kirikklin@DESKTOP-CLO8S8G: / × | kirikklin@DESKTOP-CLO8S8G: /  
kirikklin@DESKTOP-CLO8S8G:/mnt/d/kursach_os$ ./client  
Do you want MD5 or database("M"-MD5/"D"-Database)?  
("quit!" - quit)  
>>>M  
Sending...  
Sent.  
MD5 is ready  
  
Do you want to cache a file or string: string  
Enter a string to hash it  
>>>md5  
Server_md5 returned: 1bc29b36f623ba82aaf6724fd3b16718  
kirikklin@DESKTOP-CLO8S8G:/mnt/d/kursach_os$ |
```

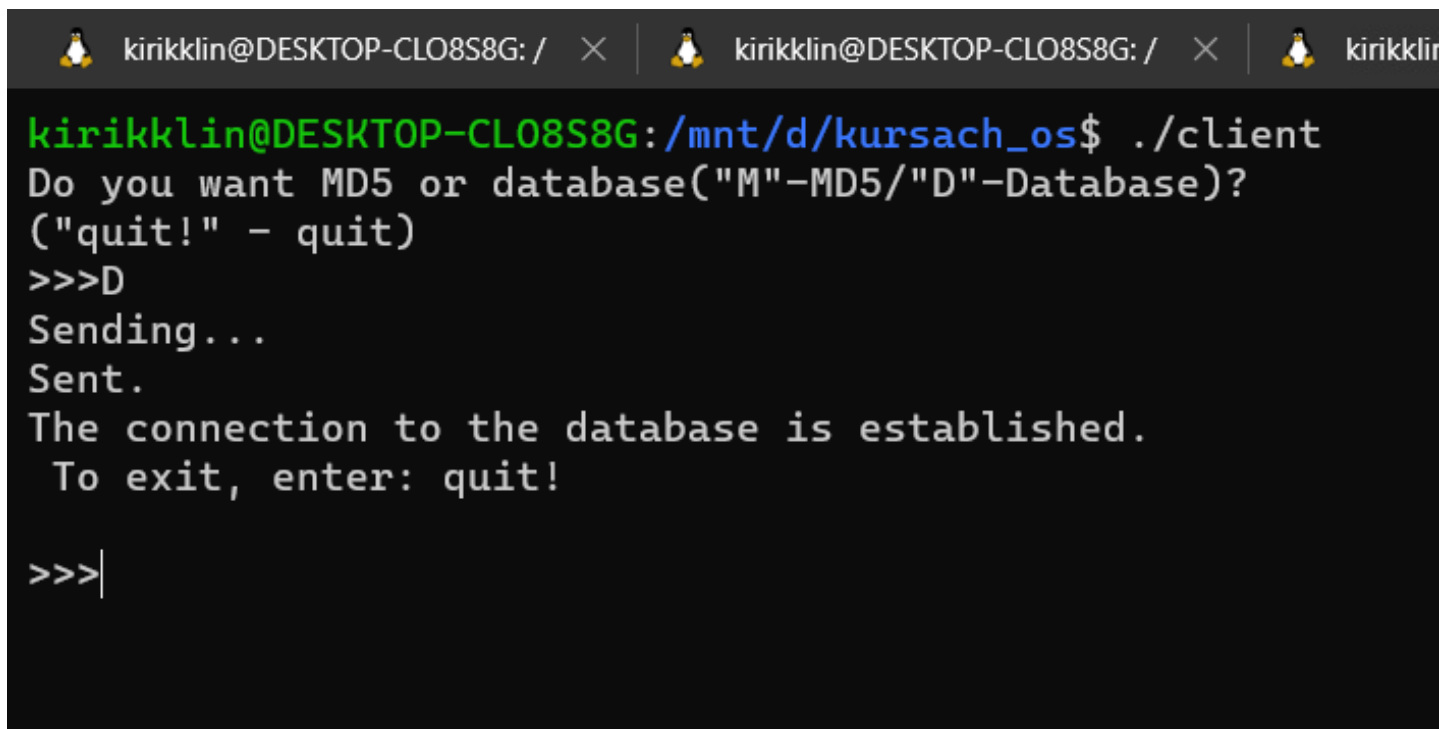
Рис. 18: Общение с клиентом.

Рассмотрим общение с программой, если пользователь выбрал базу данных:

1. Программа предлагает пользователю ввести команду.

Список команд:

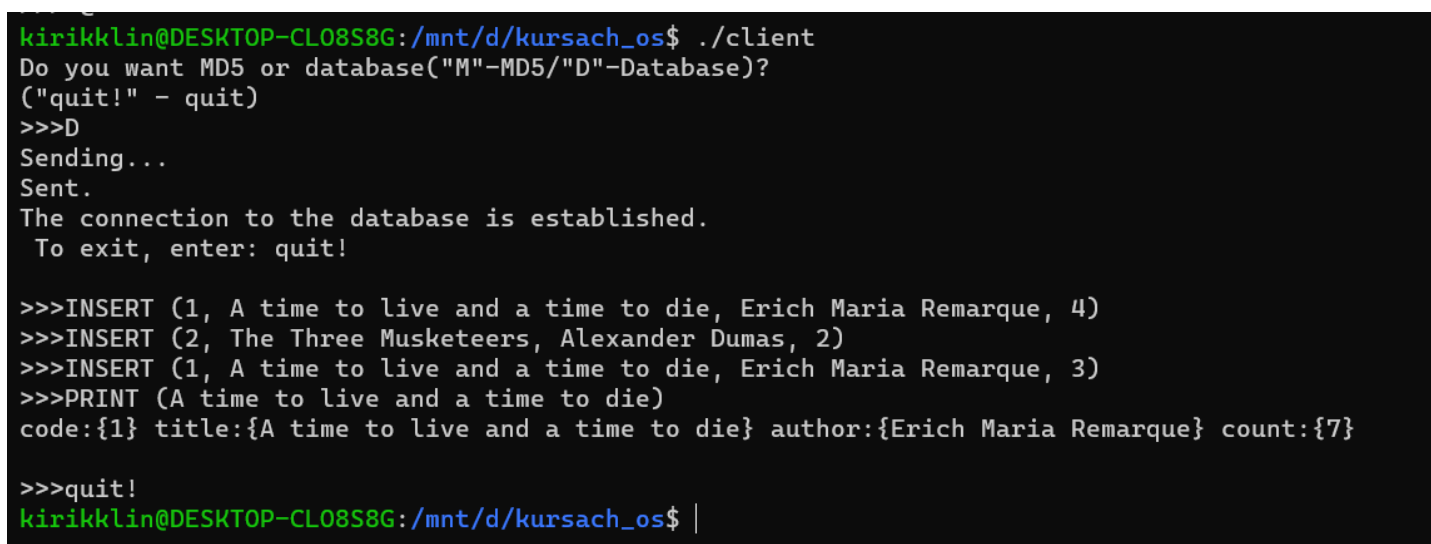
- quit! – выход.
- INSERT (code, title, author's name, count) – добавить книги в базу данных.
- PRINT (title) – вывод информации о книге.
- DELETE (title) – удаление книги из базы данных.

A terminal window with a dark background and light-colored text. The window title bar shows three tabs, each with a penguin icon and the text 'kirikklin@DESKTOP-CLO8S8G: /'. The terminal content shows a user running './client' in the directory '/mnt/d/kursach_os'. The program prompts for 'MD5 or database' and the user enters 'D'. The program sends data and confirms the database connection is established. The user then enters 'quit!' to exit.

```
kirikklin@DESKTOP-CLO8S8G: /mnt/d/kursach_os$ ./client
Do you want MD5 or database("M"-MD5/"D"-Database)?
("quit!" - quit)
>>>D
Sending...
Sent.
The connection to the database is established.
To exit, enter: quit!

>>>|
```

Рис. 19: Общение с клиентом.

A terminal window showing the same client program as in Figure 19, but with database operations. The user enters 'D' to select the database, then enters three 'INSERT' commands to add book records. The program outputs the details of the first record. Finally, the user enters 'quit!' to exit the program.

```
kirikklin@DESKTOP-CLO8S8G: /mnt/d/kursach_os$ ./client
Do you want MD5 or database("M"-MD5/"D"-Database)?
("quit!" - quit)
>>>D
Sending...
Sent.
The connection to the database is established.
To exit, enter: quit!

>>>INSERT (1, A time to live and a time to die, Erich Maria Remarque, 4)
>>>INSERT (2, The Three Musketeers, Alexander Dumas, 2)
>>>INSERT (1, A time to live and a time to die, Erich Maria Remarque, 3)
>>>PRINT (A time to live and a time to die)
code:{1} title:{A time to live and a time to die} author:{Erich Maria Remarque} count:{7}

>>>quit!
kirikklin@DESKTOP-CLO8S8G: /mnt/d/kursach_os$ |
```

Рис. 20: Общение с клиентом.

2. При последующем запуске клиент будет использовать ту же базу данных.
3. Если пользователь введёт некорректную команду, клиент сообщит ему об этом и выведет соответствующее сообщение.

```

kirikklin@DESKTOP-CL08S8G:/mnt/d/kursach_os$ ./client
Do you want MD5 or database("M"-MD5/"D"-Database)?
("quit!" - quit)
>>>D
Sending...
Sent.
The connection to the database is established.
To exit, enter: quit!

>>>PRINT (The Three Musketeers)
code:{2} title:{The Three Musketeers} author:{Alexander Dumas} count:{2}

>>>DELETE (A time to live and a time to die)
>>>PRINT (A time to live and a time to die)
There is no such name
>>>quit!
kirikklin@DESKTOP-CL08S8G:/mnt/d/kursach_os$ |

```

Рис. 21: Повторный запуск базы данных.

```

kirikklin@DESKTOP-CL08S8G:/mnt/d/kursach_os$ ./client
Do you want MD5 or database("M"-MD5/"D"-Database)?
("quit!" - quit)
>>>D
Sending...
Sent.
The connection to the database is established.
To exit, enter: quit!

>>>quit
An error has occurred in this line and it cannot be processed.

>>>delete (A time to live and a time to die)
An error has occurred in this line and it cannot be processed.

>>>PRINTF(A time to live and a time to die)
An error has occurred in this line and it cannot be processed.

>>>quit!
kirikklin@DESKTOP-CL08S8G:/mnt/d/kursach_os$ |

```

Рис. 22: Реакция на аномалию входных данных.

6 Список использованных источников

1. Таненбаум Э. С., Бос Х. Современные операционные системы (4-е издание)
2. Теренс Чан. Системное программирование на C++ для Unix.
3. Статья на сайте: <https://www.rsdn.org/article/unix/sockets.xml>
4. Статья на сайте: <https://ru.wikipedia.org/wiki/MD5>
5. Статья на сайте: <https://habr.com/ru/sandbox/26876/>

7 Приложение

7.1 Порождение процессов

Файл main_a.c

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <locale.h>
4 #include "Stack.h"
5
6 #include "proc_str.h"
7
8
9 int main(int argc, char** argv)
10 {
11     FILE* res_file = fopen(argv[1], "a");
12
13     char string[50];
14     if (argc >= 1)
15     {
16         strncpy(string, argv[0], 50);
17     }
18     else
19     {
20         printf("A: return 1\n");
21         return -1;
22     }
23     printf("(s)\n", string);
24     in_postfix(string);
25     printf("(s)\n", string);
26
27     int res = proc_postfix(string);
28     printf("ans = %d\n", res);
29
30     fprintf(res_file, "%s = %d\n", argv[0], res);
31     fclose(res_file);
32 }
```

Файл main_b.c

```
1 #include <stdio.h>
2 #include <sys/types.h>
```

```

3  #include <string.h>
4  #include <locale.h>
5  #include "Stack.h"
6
7  #include "proc_str.h"
8
9
10 int main(int argc, char** argv)
11 {
12     char buf[50];
13     pid_t pid;
14
15     int check = 0;
16
17     if (argc == 1)
18     {
19         return -1;
20     }
21
22     FILE* fp = fopen(argv[1], "r");
23
24     if (fp == NULL)
25     {
26         printf("File is not opening");
27         return -1;
28     }
29
30
31     while (!feof(fp) && check < 5)
32     {
33         check++;
34         fgets(buf, 50, fp);
35         if (feof(fp))
36             break;
37         printf("%ld\n", strlen(buf));
38         buf[strlen(buf) - 2] = '\0';
39         pid = fork();
40
41         switch (pid)
42         {
43             case -1:
44                 return -1;
45             case 0:
46                 execl("main_a", buf, argv[2], NULL);
47                 continue;
48             default:
49                 wait();

```



```

50         continue;
51         break;
52     }
53 }
54
55 printf("\n");
56 return 0;
57 }

```

Файл proc_str.h

```

1 int in_postfix(char* source_str); // 0 - success; others - own errors
2 int proc_postfix(char* string); // 0 - success; others - own errors

```

Файл proc_str.c

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <ctype.h>
4 #include "stack.h"
5 #define PRIORITI(x) ((x == '*' || x == '/') ? (3) : ((x == '(') ? (1) : (2)))
6 #define OPERATION(x) (x)
7
8 int funck(char symbol, PStack pstack, Data u_buf, int *x, int *y)
9 {
10     pop(pstack, &u_buf);
11     *y = u_buf.number;
12     pop(pstack, &u_buf);
13     *x = u_buf.number;
14
15     if (symbol == '-')
16     {
17         u_buf.number = *x - *y;
18     }
19
20     if (symbol == '+')
21     {
22         u_buf.number = *x + *y;
23     }
24 }

```

```

25
26     if (symbol == '*')
27     {
28         u_buf.number = *x * *y;
29     }
30
31
32     if (symbol == '/')
33     {
34         u_buf.number = *x / *y;
35     }
36
37     push(pstack, u_buf);
38 }
39
40 int in_postfix(char* source_str) // 0 - success; others - own errors
41 {
42     int i, j, flag;
43     Stack stack;
44     Data u_buf;
45     init(&stack);
46     stack.dataType = OPERATOR;
47     char * new_str;
48     new_str = (char*)malloc(sizeof(char) * (strlen(source_str) * 2));
49
50     flag = 0;
51     j = 0;
52     for (i = 0; i < strlen(source_str); i++)
53     {
54         if (source_str[i] == ' ' || source_str[i] == '\n' || source_str[i] == EOF)
55             continue;
56
57         if (isdigit(source_str[i]))
58         {
59             new_str[j++] = source_str[i];
60             if (source_str[i+1] == '+' || source_str[i+1] == '-' || source_str[i+1] == '*' ||
        ↪ source_str[i+1] == '/' || source_str[i+1] == ')' || source_str[i+1] == ' ' ||
        ↪ source_str[i+1] == '\0')
61                 new_str[j++] = ' ';
62             flag = 1;
63         }
64         else
65         {
66             top(&stack, &u_buf);
67             if (source_str[i] == ')')
68             {
69                 while (u_buf.oper_mvd != '(' && !isEmpty(&stack))

```

```

70     {
71         // записать проверку на то вдруг стек раньше закончится
72         new_str[j++] = u_buf.oper_mvd;
73         new_str[j++] = ' ';
74         pop(&stack, &u_buf);
75         top(&stack, &u_buf);
76     }
77     pop(&stack, &u_buf);
78
79 }
80 else
81 {
82     if (source_str[i] == '(')
83     {
84         u_buf.oper_mvd = source_str[i];
85         push(&stack, u_buf);
86         flag = 0;
87         continue;
88     }
89
90     if ((source_str[i] == '-' || source_str[i] == '+') && !flag)
91     {
92         new_str[j++] = source_str[i];
93         continue;
94     }
95
96     while ((PRIORITI(source_str[i]) <= PRIORITI(u_buf.oper_mvd) && !isEmpty(&stack)))
97     {
98         // записать проверку на то вдруг стек раньше закончится
99         new_str[j++] = u_buf.oper_mvd;
100        new_str[j++] = ' ';
101        pop(&stack, &u_buf);
102        top(&stack, &u_buf);
103        flag = 0;
104    }
105    // Пункт а:
106    if (isEmpty(&stack) || (PRIORITI(source_str[i]) > PRIORITI(u_buf.oper_mvd)) ||
107        ⇨ source_str[i] == '(')
108    {
109        u_buf.oper_mvd = source_str[i];
110        push(&stack, u_buf); // TODO: проверку всё ли хорошенько закончилось
111        flag = 0;
112    }
113 }
114 }
115 top(&stack, &u_buf);

```

```

116 while (!isEmpty(&stack))
117 {
118     new_str[j++] = u_buf.oper_mvd;
119     new_str[j++] = ' ';
120     pop(&stack, &u_buf);
121     top(&stack, &u_buf);
122 }
123 new_str[j-1] = '\0';
124 strcpy(source_str, new_str);
125 }
126
127 int proc_postfix(char* string)
128 {
129     int i = -1, j, x, y;
130     Stack stack;
131     Data u_buf;
132
133     init(&stack);
134     stack.dataType = NUMBER;
135
136     u_buf.number = 0;
137     while (string[++i] != '\0')
138     {
139         if (string[i] == ' ')
140         {
141             continue;
142         }
143         if ((string[i] == '-' || string[i] == '+') && string[i + 1] != ' ' && string[i+1] != '\0')
144         {
145             continue;
146         }
147
148         if (isdigit(string[i]))
149         {
150             u_buf.number += string[i] & 0x0F;
151             u_buf.number *= 10;
152             if (string[i + 1] == ' ' || string[i+1] == '\0')
153             {
154                 u_buf.number /= 10;
155                 j = i;
156                 while (string[j] != ' ' && j >= 0)
157                     j--;
158                 if (string[j + 1] == '-')
159                 {
160                     u_buf.number *= -1;
161                 }
162                 push(&stack, u_buf); // проверка.

```

```

163         u_buf.number = 0;
164     }
165 }
166 else
167 {
168     funck(string[i], &stack, u_buf, &x, &y);
169 }
170
171 }
172
173 top(&stack, &u_buf);
174 return u_buf.number;
175 }

```

Файл Stack.h

```

1  #define STACK_OVERFLOW  -100
2  #define STACK_UNDERFLOW -101
3  #define OUT_OF_MEMORY -102
4
5  #include "DataTypes.h"
6  #include "StackList.h"
7
8  int init(PStack stack); // 0 - success; others - own errors
9  int push(PStack stack, union Data toPush); // 0 - success; others - own errors
10 int pop(const PStack stack, PData destination); // 0 - success; others - own errors
11 int top(const PStack stack, PData destination); // 0 - success; others - own errors
12 int isEmpty(const PStack stack); // 0 - empty; 1 - empty; others - own errors;
13 int clear(PStack stack); // 0 - success; others - own errors
14 int resise(PStack stack); // 0 - success; others - own errors

```

Файл StackList.h

```

#include "DataTypes.h"

struct StackNode
{
    union Data data;
    struct StackNode* next;

```

```

};

typedef struct StackNode Node, *PNode;

struct StackBasedOnList
{
    enum DataType dataType;
    PNode top;
};

typedef struct StackBasedOnList Stack, *PStack;

```

Файл list_stack.c

```

1  #include <stdio.h>
2  #include "Stack.h"
3
4  #ifndef STACK_ON_ARRAY
5  int init(PStack stack) // 0 - success; others - own errors
6  {
7      stack->dataType = NOTHING;
8      stack->top = NULL;
9      return 0;
10 }
11
12
13 int push(PStack stack, union Data toPush) // 0 - success; others - own errors
14 {
15     PNode NewEl;
16
17     NewEl = (PNode)malloc(sizeof(Node));
18     if (NewEl != NULL)
19     {
20         NewEl->next = stack->top;
21         (stack->top) = NewEl;
22         NewEl->data = toPush;
23     }
24     else
25     {
26         printf("No memory available.\n");

```

```

27     return OUT_OF_MEMORY;
28 }
29 return 0;
30 }
31 int pop(const PStack stack, PData destination) // 0 - success; 1 - stack is empty
32 {
33     PNode tempPtr;
34     if (stack->top == NULL)
35     {
36         return 1;
37     }
38
39     tempPtr = stack->top;
40     *destination = (stack->top)->data;
41
42     stack->top = (stack->top)->next;
43     free(tempPtr);
44 }
45 int top(const PStack stack, PData destination) // 0 - success; 1 - stack is empty
46 {
47     if (stack->top == NULL)
48     {
49         return 1;
50     }
51     *destination = (stack->top)->data;
52 }
53 int isEmpty(const PStack stack) // 1 - empty; 0 - not empty; others - own errors;
54 {
55     return stack->top == NULL;
56 }
57 int clear(PStack stack) // 0 - success; others - own errors
58 {
59     free(stack);
60     return 0;
61 }
62
63 #endif

```

Файл DataTypes.h

```
#pragma once
```

```
enum DataType { NOTHING, NUMBER, OPERATOR };
```

```
union Data
{
    int number;
    char oper_mvd;
};

typedef union Data Data;
typedef Data* PData;
```

7.2 Ветвящиеся процессы

Файл main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <unistd.h>
5 #include <sys/wait.h>
6 #include <time.h>
7 #include "map.h"
8
9 int main()
10 {
11     pid_t pid = 0;
12     int procent = 0;
13     int is_working = 0;
14     int create_proc = 0;
15     int i = 0, j = 0;
16     int length_of_proc_trace = 0;
17     int children[3];
18     int k = 0;
19
20     char* name_of_file = "save.txt";
21     FILE* file_to_save;
22     if (!(file_to_save = fopen(name_of_file, "w")))
23     {
24         printf("File isn't open\n");
25         return -1;
26     }
```



```

27  fclose(file_to_save);
28
29  int main_pid = getpid();
30  printf("Main process - %d\n", main_pid);
31
32  while (!is_working)
33  {
34      is_working = 0;
35      srand(getpid());
36      procent = rand() % 100 + 1;
37
38      if (procent > 0 && procent <= 25) //конец
39      {
40          if (!(file_to_save = fopen(name_of_file, "a")))
41          {
42              printf("File isn't open\n");
43              return -1;
44          }
45          for (k = 0; k < length_of_proc_trace; k++)
46          {
47              if (k == 0)
48              {
49                  printf("Pid: %d (My parent%d) | ", getpid(), getppid());
50
51
52                  fprintf(file_to_save, "%d %d\n", getpid(), length_of_proc_trace);
53
54
55              }
56              if (k == length_of_proc_trace - 1)
57              {
58                  printf("*\n");
59                  continue;
60              }
61              printf("*");
62          }
63          fclose(file_to_save);
64          break;
65      }
66
67      else if (procent > 25 && procent <= 65) //+1
68          create_proc = 1;
69
70      else if (procent > 65 && procent <= 90) //+2
71          create_proc = 2;
72
73      else if (procent > 90 && procent <= 100) //+3

```

```

74         create_proc = 3;
75
76     for (i = 0; i < create_proc; i++)
77     {
78         switch (pid = fork())
79         {
80             case 0: //ребенок
81             {
82                 is_working = 0;
83                 length_of_proc_trace++;
84                 break;
85             }
86
87             default: //отец
88             {
89                 is_working = 1;
90
91                 children[i] = pid;
92                 if (i == create_proc - 1)
93                 {
94                     for (j = 0; j < create_proc; j++)
95                     {
96                         waitpid(children[j], 0, 0);
97
98                     }
99                 }
100                 break;
101             }
102         }
103         if (!pid)
104         {
105             break;
106         }
107     }
108
109 }
110
111 if (getpid() == main_pid)
112 {
113     map dict;
114     init(&dict);
115     unsigned int pidd, length, max_value = 0;
116
117     if (!(file_to_save = fopen(name_of_file, "r")))
118     {
119         printf("File isn't open\n");
120         return -1;

```

```

121     }
122
123     while (1)
124     {
125         read_two_digits(file_to_save, &pidd, &length);
126         if (max_value < length)
127             max_value = length;
128         if (!feof(file_to_save))
129             append(&dict, pidd, length);
130         else
131             break;
132         //printf("%d %d\n", pidd, length);
133     }
134     make_graph(&dict, max_value);
135     return 0;
136 }
137 }

```

Файл map.h

```

1  #include <stdio.h>
2
3  void read_two_digits(FILE* file, int* PID, int* length)
4  {
5      char buffer[256];
6      char c;
7      int i = 0;
8
9      while (1)
10     {
11         c = getc(file);
12         if (c != ' ' && c != EOF)
13             buffer[i] = c;
14         else
15             break;
16         i++;
17     }
18     buffer[i] = '\0';
19     *PID = atoi(buffer);
20
21     i = 0;
22     while (1)
23     {
24         c = getc(file);

```

```

25     if (c != '\n' && c != EOF)
26         buffer[i] = c;
27     else
28         break;
29     i++;
30 }
31 buffer[i] = '\0';
32 *length = atoi(buffer);
33 }
34
35 struct map
36 {
37     unsigned int name[512];
38     unsigned int length[512];
39     unsigned int size_of_array;
40 };
41
42 typedef struct map map;
43
44 void init(map* mass_of_process)
45 {
46     mass_of_process->size_of_array = 0;
47 }
48
49 void append(map* mass_of_process, unsigned int PID, unsigned int length)
50 {
51     mass_of_process->name[mass_of_process->size_of_array] = PID;
52     mass_of_process->length[mass_of_process->size_of_array] = length;
53     (mass_of_process->size_of_array)++;
54 }
55
56 void make_graph(map* mass_of_process, unsigned int max_value)
57 {
58     int max_length = 0, helper = 0;
59
60     for (int j = 0; j <= max_value; j++) // нахожу максимум (для разметки) max_value - максимальная
        ↪ длина повторения
61     {
62         helper = 0;
63         for (int i = 0; i != mass_of_process->size_of_array; i++)
64         {
65             if (j == (mass_of_process->length)[i])
66                 helper++;
67         }
68         if (max_length < helper)
69             max_length = helper;
70     }

```

```

71 //printf("%d\n", max_length);
72 int size_of_hash_table = max_value + 1;
73 unsigned int* hash_table = (unsigned int*)calloc(size_of_hash_table, sizeof(unsigned int));
74 for (int j = 0; j <= max_value; j++) // заполняем массив
75 {
76     helper = 0;
77     for (int i = 0; i != mass_of_process->size_of_array; i++)
78     {
79         if (j == (mass_of_process->length)[i])
80             helper++;
81     }
82     hash_table[j] = helper;
83 }
84
85 for (int i = 0; i != 0; i++)
86 {
87     printf("%d - %d\n", i, hash_table[i]);
88 }
89 printf("\n");
90
91 for (int i = max_length; i != 0; i--)
92 {
93     for (int j = 1; j != size_of_hash_table; j++)
94     {
95         printf("| ");
96         if (hash_table[j] >= i)
97         {
98             printf("*");
99             printf(" |");
100         }
101         else
102             printf(" |");
103     }
104     printf("\n");
105 }
106
107 for (int j = 1; j != size_of_hash_table; j++)
108 {
109     printf(" %d\t", j);
110 }
111 printf("\n");
112 free(hash_table);
113
114 }

```

7.3 Реализация ls

Файл main.c

```
1 #include <stdio.h>
2 #include <dirent.h>
3 #include <locale.h>
4
5 int main()
6 {
7     DIR* dir;
8     setlocale(LC_ALL, "Rus");
9     struct dirent* entity;
10    char directory[255]; // path
11    printf("Введите дерикторию:");
12    scanf("%s", directory);
13    dir = opendir(directory);
14
15    if (!dir)
16    {
17        printf("Неправильная дериктория!\n");
18        return 1;
19    }
20
21    while ((entity = readdir(dir)) != NULL)
22    {
23        printf("%s\n", entity->d_name);
24    }
25    closedir(dir);
26    return 0;
27 }
```

7.4 Реализация ls без папок

Файл main.c

```
1 #include <stdio.h>
2 #include <dirent.h>
3 #include <locale.h>
4
5 int main()
6 {
7     DIR* dir;
```

```

8  setlocale(LC_ALL, "Rus");
9  struct dirent* entity;
10 char directory[255]; // ". "; // path
11 printf("Введите дерикторию:");
12 scanf("%s", directory);
13 dir = opendir(directory);
14
15 if (!dir)
16 {
17     printf("Wrong Enter\n");
18     return 1;
19 }
20
21 while ((entity = readdir(dir)) != NULL)
22 {
23     if (entity->d_type == DT_REG)
24         printf("%s\n", entity->d_name);
25 }
26 closedir(dir);
27 return 0;
28 }

```

7.5 Жёсткая ссылка

Файл main.c

```

1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <sys/stat.h>
4  #include <stdlib.h>
5  #include <fcntl.h>
6  #include <unistd.h>
7
8  char* get_hardlink(const char* current_file_name)
9  {
10     int i = 0;
11     int number_in_hl = 1;
12     char* hard_link_name = malloc(sizeof(char) * 255);
13     char number_str[255];
14     while (current_file_name[i] != '.')
15     {
16         hard_link_name[i++] = current_file_name[i];
17     }

```

```

18
19     hard_link_name[i++] = current_file_name[i];
20
21     printf("1: %s\n", hard_link_name);
22
23     int start_ext_pos = i;
24     int tmp = start_ext_pos;
25     int j;
26
27     while (1)
28     {
29         j = 0;
30         i = tmp;
31         printf("2: %d\n", i);
32         start_ext_pos = tmp;
33         sprintf(number_str, "%d", number_in_hl++);
34         while (number_str[j])
35         {
36             hard_link_name[i++] = number_str[j++];
37         }
38         hard_link_name[i++] = '.';
39
40         while (current_file_name[start_ext_pos])
41         {
42             hard_link_name[i++] = current_file_name[start_ext_pos++];
43         }
44         hard_link_name[i] = '\0';
45
46         printf("3: %s\n", hard_link_name);
47         if (access(hard_link_name, F_OK) == -1)
48         {
49             //free(number_str);
50             printf("4: free name found\n");
51             return hard_link_name;
52         }
53
54     }
55 }
56
57
58
59 int main(int argc, char* argv[])
60 {
61     int file_descriptor = -1;
62     char* hard_link_name = NULL;
63
64     if (argc < 2)

```



```

65 {
66     printf("0 arguments\n");
67     return -1;
68 }
69
70 file_descriptor = open(argv[1], O_WRONLY);
71
72 if (file_descriptor == -1)
73 {
74     printf("File does not exist!\n");
75     return -3;
76 }
77
78 hard_link_name = get_hardlink(argv[1]);
79
80 printf("Hardlink name: %s\n", hard_link_name);
81
82 if (link(argv[1], hard_link_name) == -1)
83 {
84     printf("There is an error with making link of file.\n");
85     return -5;
86 }
87 free(hard_link_name);
88 if (close(file_descriptor) == -1)
89 {
90     printf("There is a problem with closing file. File descriptor: %d\n", file_descriptor);
91     return -4;
92 }
93
94 return 0;
95 }

```

7.6 Символическая ссылка

Файл main.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/stat.h>
5 #include <unistd.h>
6 #include <string.h>
7 #include <time.h>

```

```

8
9 int main(int argc, char** argv)
10 {
11     if (argc != 2)
12         return 0;
13
14     int link;
15     struct stat st;
16
17     link = symlink("4.c", argv[1]);
18
19     if (!link)
20     {
21         lstat(argv[1], &st);
22         printf("%ld : ", st.st_nlink); /* количество жестких ссылок */
23         printf("%s : ", argv[1]); // name
24         printf("%ld : ", st.st_size); /* общий размер в байтах */
25         printf("%ld : ", st.st_ino); /* inode */
26         printf(" UID=%d   GID=%d ", st.st_uid, st.st_gid); /* идентификатор пользователя-владельца */
27         ↪ /* идентификатор группы-владельца */
28         printf(ctime(&(st.st_mtime))); /* время последнего изменения */
29     }
30     else
31         printf("Can't make link\n");
32     return 0;
33 }

```

7.7 Сортировка файлов

Файл main.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/stat.h>
5 #include <unistd.h>
6 #include <string.h>
7 #include <time.h>
8
9 int main(int argc, char** argv)
10 {
11     if (argc != 2)
12         return 0;

```

```

13
14     int link;
15     struct stat st;
16
17     link = symlink("4.c", argv[1]);
18
19     if (!link)
20     {
21         lstat(argv[1], &st);
22         printf("%ld : ", st.st_nlink); /* количество жестких ссылок */
23         printf("%s : ", argv[1]); // name
24         printf("%ld : ", st.st_size); /* общий размер в байтах */
25         printf("%ld : ", st.st_ino); /* inode */
26         printf(" UID=%d   GID=%d ", st.st_uid, st.st_gid); /* идентификатор пользователя-владельца */
27         ↪ /* идентификатор группы-владельца */
28         printf(ctime(&(st.st_mtime))); /* время последнего изменения */
29     }
30     else
31         printf("Can't make link\n");
32     return 0;
33 }

```

7.8 Очередь сообщений

Файл get.c

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <time.h>
4  #include <sys/ipc.h>
5  #include <sys/msg.h>
6  #include <sys/errno.h>
7  #include <unistd.h>
8  #include <math.h>
9  #include <stdlib.h>
10
11 struct msg_buf
12 {
13     char mtext[128];
14 } msg;
15
16 struct digit
17 {

```

```

18     double first;
19     double second;
20 };
21
22 int parse_line(char* line, int* coef);
23 void do_kardano(char* answer, double a, double b, double c, double d);
24 double croot(double x);
25
26 int main()
27 {
28     int msgqid, rc;
29     key_t msgkey;
30     msgkey = ftok("/mnt/c/labs_os/3_1/send.c", 'm');
31
32     msgqid = msgget(msgkey, IPC_CREAT | 0660);
33
34     if (msgqid < 0)
35     {
36         perror(strerror(errno));
37         printf("failed to create message queue with msgqid = %d\n", msgqid);
38         return 1;
39     }
40     else
41         printf("message queue %d created\n", msgqid);
42
43     int a, b, c, d, coef[4];
44     char answer[128], line[128];
45
46     while (1)
47     {
48         rc = msgrcv(msgqid, &msg, sizeof(msg) - sizeof(long), 0, 0);
49         //printf("rc: %d\n", rc);
50         if (rc < 0)
51         {
52             perror(strerror(errno));
53             printf("msgrcv failed, rc=%d\n", rc);
54             break;
55         }
56         printf("received msg: %s it's PID = %ld\n", msg.mtext, msg.mtype);
57
58         if (strcmp(msg.mtext, "stop the server\n") == 0)
59             break;
60
61         strcpy(line, msg.mtext);
62         if (parse_line(line, coef) || coef[0] == 0)
63         {
64             strcpy(msg.mtext, "Wrong enter!");

```

```

65     }
66     else
67     {
68         a = coef[0];
69         b = coef[1];
70         c = coef[2];
71         d = coef[3];
72         do_kardano(answer, a, b, c, d);
73         strcpy(msg.mtext, answer);
74     }
75
76     rc = msgsnd(msgqid, &msg, sizeof(msg) - sizeof(long), msg.mtype);
77     if (rc < 0)
78     {
79         printf("msgsnd failed, rc = %d\n", rc);
80         return 1;
81     }
82     else
83         printf("Message send done!\n");
84
85     printf("answer: %s\n", msg.mtext);
86 }
87 return 0;
88 }
89
90 double croot(double x)
91 {
92     if (x < 0)
93         return -pow(-x, 1.0 / 3.0);
94     return pow(x, 1.0 / 3.0);
95 }
96
97 int parse_line(char* line, int* coef)
98 {
99     int i = 0;
100    int j = 0;
101
102    while (line[i] != '\n')
103    {
104        if (line[i] >= '0' && line[i] <= '9')
105        {
106            coef[j] = atoi(line + i);
107            j++;
108
109            while (line[i] >= '0' && line[i] <= '9')
110                i++;
111

```

```

112     }
113     else if (line[i] == '-')
114     {
115         coef[j] = atoi(line + i);
116         j++;
117
118         if (line[i + 1] < '0' || line[i + 1] > '9')
119             return 1;
120         i++;
121
122         while (line[i] >= '0' && line[i] <= '9')
123             i++;
124     }
125     else if (line[i] == ' ')
126         i++;
127     else
128         return 1;
129 }
130
131 if (j != 4)
132     return 1;
133 return 0;
134 }
135
136 void print_mass(int* mass, int len)
137 {
138     for (int i = 0; i != len; i++)
139         printf("%d ", mass[i]);
140     printf("\n");
141 }
142
143 void do_kardano(char* answer, double a, double b, double c, double d)
144 {
145     double p = (3.0 * a * c - b * b) / (3.0 * a * a);
146     double q = (2.0 * b * b * b - 9.0 * a * b * c + 27.0 * a * a * d) / (27.0 * a * a * a);
147     double S = (q * q / 4.0) + (p * p * p / 27.0);
148
149     double F;
150     if (q == 0)
151         F = M_PI / 2.0;
152     if (q < 0)
153         F = atan(-2.0 * sqrt(-S) / q);
154     if (q > 0)
155         F = atan(-2.0 * sqrt(-S) / q) + M_PI;
156
157     struct digit x[3];
158     for (int i = 0; i < 3; i++)

```

```

159     x[i].first = x[i].second = 0;
160     if (S < 0)
161     {
162         x[0].first = 2.0 * sqrt(-p / 3.0) * cos(F / 3.0) - b / (3.0 * a);
163         x[1].first = 2.0 * sqrt(-p / 3.0) * cos((F / 3.0) + 2.0 * M_PI / 3.0) - b / (3.0 * a);
164         x[2].first = 2.0 * sqrt(-p / 3.0) * cos((F / 3.0) + 4.0 * M_PI / 3.0) - b / (3.0 * a);
165     }
166     if (S == 0)
167     {
168         x[0].first = 2.0 * croot(-q / 2.0) - b / (3.0 * a);
169         x[1].first = -croot(-q / 2.0) - b / (3.0 * a);
170         x[2].first = -croot(-q / 2.0) - b / (3.0 * a);
171     }
172     if (S > 0)
173     {
174         double temp1 = croot((-q / 2.0) + sqrt(S)) + croot((-q / 2.0) - sqrt(S));
175         double temp2 = croot((-q / 2.0) + sqrt(S)) - croot((-q / 2.0) - sqrt(S));
176         x[0].first = temp1 - b / (3.0 * a);
177         x[1].first = -temp1 / 2.0 - b / (3.0 * a); x[1].second = sqrt(3) * temp2 / 2.0;
178         x[2].first = -temp1 / 2.0 - b / (3.0 * a); x[2].second = -sqrt(3) * temp2 / 2.0;
179     }
180
181     answer[0] = '\0';
182
183     char tmp[128];
184     sprintf(tmp, "x1 = %.4lf + i * %.4lf;", x[0].first, x[0].second);
185     strncat(answer, tmp, 128);
186
187     sprintf(tmp, " x2 = %.4lf + i * %.4lf;", x[1].first, x[1].second);
188     strncat(answer, tmp, 128);
189
190     sprintf(tmp, " x3 = %.4lf + i * %.4lf;", x[2].first, x[2].second);
191     strncat(answer, tmp, 128);
192 }

```

Файл send.c

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <time.h>
4  #include <sys/ipc.h>
5  #include <sys/msg.h>
6  #include <sys/errno.h>
7  #include <unistd.h>

```

```

8  #include <math.h>
9  #include <stdlib.h>
10
11 struct msg_buf
12 {
13     char mtext[128];
14 } msg;
15
16 struct digit
17 {
18     double first;
19     double second;
20 };
21
22 int parse_line(char* line, int* coef);
23 void do_kardano(char* answer, double a, double b, double c, double d);
24 double croot(double x);
25
26 int main()
27 {
28     int msgqid, rc;
29     key_t msgkey;
30     msgkey = ftok("/mnt/c/labs_os/3_1/send.c", 'm');
31
32     msgqid = msgget(msgkey, IPC_CREAT | 0660);
33
34     if (msgqid < 0)
35     {
36         perror(strerror(errno));
37         printf("failed to create message queue with msgqid = %d\n", msgqid);
38         return 1;
39     }
40     else
41         printf("message queue %d created\n", msgqid);
42
43     int a, b, c, d, coef[4];
44     char answer[128], line[128];
45
46     while (1)
47     {
48         rc = msgrcv(msgqid, &msg, sizeof(msg) - sizeof(long), 0, 0);
49         //printf("rc: %d\n", rc);
50         if (rc < 0)
51         {
52             perror(strerror(errno));
53             printf("msgrcv failed, rc=%d\n", rc);
54             break;

```



```

55     }
56     printf("received msg: %s it's PID = %ld\n", msg.mtext, msg.mtype);
57
58     if (strcmp(msg.mtext, "stop the server\n") == 0)
59         break;
60
61     strcpy(line, msg.mtext);
62     if (parse_line(line, coef) || coef[0] == 0)
63     {
64         strcpy(msg.mtext, "Wrong enter!");
65     }
66     else
67     {
68         a = coef[0];
69         b = coef[1];
70         c = coef[2];
71         d = coef[3];
72         do_kardano(answer, a, b, c, d);
73         strcpy(msg.mtext, answer);
74     }
75
76     rc = msgsnd(msgqid, &msg, sizeof(msg) - sizeof(long), msg.mtype);
77     if (rc < 0)
78     {
79         printf("msgsnd failed, rc = %d\n", rc);
80         return 1;
81     }
82     else
83         printf("Message send done!\n");
84
85     printf("answer: %s\n", msg.mtext);
86 }
87 return 0;
88 }
89
90 double croot(double x)
91 {
92     if (x < 0)
93         return -pow(-x, 1.0 / 3.0);
94     return pow(x, 1.0 / 3.0);
95 }
96
97 int parse_line(char* line, int* coef)
98 {
99     int i = 0;
100     int j = 0;
101

```

```

102 while (line[i] != '\n')
103 {
104     if (line[i] >= '0' && line[i] <= '9')
105     {
106         coef[j] = atoi(line + i);
107         j++;
108
109         while (line[i] >= '0' && line[i] <= '9')
110             i++;
111
112     }
113     else if (line[i] == '-')
114     {
115         coef[j] = atoi(line + i);
116         j++;
117
118         if (line[i + 1] < '0' || line[i + 1] > '9')
119             return 1;
120         i++;
121
122         while (line[i] >= '0' && line[i] <= '9')
123             i++;
124     }
125     else if (line[i] == ' ')
126         i++;
127     else
128         return 1;
129 }
130
131 if (j != 4)
132     return 1;
133 return 0;
134 }
135
136 void print_mass(int* mass, int len)
137 {
138     for (int i = 0; i != len; i++)
139         printf("%d ", mass[i]);
140     printf("\n");
141 }
142
143 void do_kardano(char* answer, double a, double b, double c, double d)
144 {
145     double p = (3.0 * a * c - b * b) / (3.0 * a * a);
146     double q = (2.0 * b * b * b - 9.0 * a * b * c + 27.0 * a * a * d) / (27.0 * a * a * a);
147     double S = (q * q / 4.0) + (p * p * p / 27.0);
148

```

```

149     double F;
150     if (q == 0)
151         F = M_PI / 2.0;
152     if (q < 0)
153         F = atan(-2.0 * sqrt(-S) / q);
154     if (q > 0)
155         F = atan(-2.0 * sqrt(-S) / q) + M_PI;
156
157     struct digit x[3];
158     for (int i = 0; i < 3; i++)
159         x[i].first = x[i].second = 0;
160     if (S < 0)
161     {
162         x[0].first = 2.0 * sqrt(-p / 3.0) * cos(F / 3.0) - b / (3.0 * a);
163         x[1].first = 2.0 * sqrt(-p / 3.0) * cos((F / 3.0) + 2.0 * M_PI / 3.0) - b / (3.0 * a);
164         x[2].first = 2.0 * sqrt(-p / 3.0) * cos((F / 3.0) + 4.0 * M_PI / 3.0) - b / (3.0 * a);
165     }
166     if (S == 0)
167     {
168         x[0].first = 2.0 * croot(-q / 2.0) - b / (3.0 * a);
169         x[1].first = -croot(-q / 2.0) - b / (3.0 * a);
170         x[2].first = -croot(-q / 2.0) - b / (3.0 * a);
171     }
172     if (S > 0)
173     {
174         double temp1 = croot((-q / 2.0) + sqrt(S)) + croot((-q / 2.0) - sqrt(S));
175         double temp2 = croot((-q / 2.0) + sqrt(S)) - croot((-q / 2.0) - sqrt(S));
176         x[0].first = temp1 - b / (3.0 * a);
177         x[1].first = -temp1 / 2.0 - b / (3.0 * a); x[1].second = sqrt(3) * temp2 / 2.0;
178         x[2].first = -temp1 / 2.0 - b / (3.0 * a); x[2].second = -sqrt(3) * temp2 / 2.0;
179     }
180
181     answer[0] = '\0';
182
183     char tmp[128];
184     sprintf(tmp, "x1 = %.4lf + i * %.4lf;", x[0].first, x[0].second);
185     strncat(answer, tmp, 128);
186
187     sprintf(tmp, " x2 = %.4lf + i * %.4lf;", x[1].first, x[1].second);
188     strncat(answer, tmp, 128);
189
190     sprintf(tmp, " x3 = %.4lf + i * %.4lf;", x[2].first, x[2].second);
191     strncat(answer, tmp, 128);
192 }

```

7.9 Семафоры и разделяемая память

Файл client.c

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <time.h>
4  #include <sys/ipc.h>
5  #include <sys/msg.h>
6  #include <sys/errno.h>
7  #include <unistd.h>
8  #include <math.h>
9  #include <stdlib.h>
10
11 struct msg_buf
12 {
13     char mtext[128];
14 } msg;
15
16 struct digit
17 {
18     double first;
19     double second;
20 };
21
22 int parse_line(char* line, int* coef);
23 void do_kardano(char* answer, double a, double b, double c, double d);
24 double croot(double x);
25
26 int main()
27 {
28     int msgqid, rc;
29     key_t msgkey;
30     msgkey = ftok("/mnt/c/labs_os/3_1/send.c", 'm');
31
32     msgqid = msgget(msgkey, IPC_CREAT | 0660);
33
34     if (msgqid < 0)
35     {
36         perror(strerror(errno));
37         printf("failed to create message queue with msgqid = %d\n", msgqid);
38         return 1;
39     }
40     else
41         printf("message queue %d created\n", msgqid);
42
43     int a, b, c, d, coef[4];
```

```

44 char answer[128], line[128];
45
46 while (1)
47 {
48     rc = msgrcv(msgqid, &msg, sizeof(msg) - sizeof(long), 0, 0);
49     //printf("rc: %d\n", rc);
50     if (rc < 0)
51     {
52         perror(strerror(errno));
53         printf("msgrcv failed, rc=%d\n", rc);
54         break;
55     }
56     printf("received msg: %s it's PID = %ld\n", msg.mtext, msg.mtype);
57
58     if (strcmp(msg.mtext, "stop the server\n") == 0)
59         break;
60
61     strcpy(line, msg.mtext);
62     if (parse_line(line, coef) || coef[0] == 0)
63     {
64         strcpy(msg.mtext, "Wrong enter!");
65     }
66     else
67     {
68         a = coef[0];
69         b = coef[1];
70         c = coef[2];
71         d = coef[3];
72         do_kardano(answer, a, b, c, d);
73         strcpy(msg.mtext, answer);
74     }
75
76     rc = msgsnd(msgqid, &msg, sizeof(msg) - sizeof(long), msg.mtype);
77     if (rc < 0)
78     {
79         printf("msgsnd failed, rc = %d\n", rc);
80         return 1;
81     }
82     else
83         printf("Message send done!\n");
84
85     printf("answer: %s\n", msg.mtext);
86 }
87 return 0;
88 }
89
90 double croot(double x)

```

```

91 {
92     if (x < 0)
93         return -pow(-x, 1.0 / 3.0);
94     return pow(x, 1.0 / 3.0);
95 }
96
97 int parse_line(char* line, int* coef)
98 {
99     int i = 0;
100    int j = 0;
101
102    while (line[i] != '\n')
103    {
104        if (line[i] >= '0' && line[i] <= '9')
105        {
106            coef[j] = atoi(line + i);
107            j++;
108
109            while (line[i] >= '0' && line[i] <= '9')
110                i++;
111
112        }
113        else if (line[i] == '-')
114        {
115            coef[j] = atoi(line + i);
116            j++;
117
118            if (line[i + 1] < '0' || line[i + 1] > '9')
119                return 1;
120            i++;
121
122            while (line[i] >= '0' && line[i] <= '9')
123                i++;
124        }
125        else if (line[i] == ' ')
126            i++;
127        else
128            return 1;
129    }
130
131    if (j != 4)
132        return 1;
133    return 0;
134 }
135
136 void print_mass(int* mass, int len)
137 {

```

```

138     for (int i = 0; i != len; i++)
139         printf("%d ", mass[i]);
140     printf("\n");
141 }
142
143 void do_kardano(char* answer, double a, double b, double c, double d)
144 {
145     double p = (3.0 * a * c - b * b) / (3.0 * a * a);
146     double q = (2.0 * b * b * b - 9.0 * a * b * c + 27.0 * a * a * d) / (27.0 * a * a * a);
147     double S = (q * q / 4.0) + (p * p * p / 27.0);
148
149     double F;
150     if (q == 0)
151         F = M_PI / 2.0;
152     if (q < 0)
153         F = atan(-2.0 * sqrt(-S) / q);
154     if (q > 0)
155         F = atan(-2.0 * sqrt(-S) / q) + M_PI;
156
157     struct digit x[3];
158     for (int i = 0; i < 3; i++)
159         x[i].first = x[i].second = 0;
160     if (S < 0)
161     {
162         x[0].first = 2.0 * sqrt(-p / 3.0) * cos(F / 3.0) - b / (3.0 * a);
163         x[1].first = 2.0 * sqrt(-p / 3.0) * cos((F / 3.0) + 2.0 * M_PI / 3.0) - b / (3.0 * a);
164         x[2].first = 2.0 * sqrt(-p / 3.0) * cos((F / 3.0) + 4.0 * M_PI / 3.0) - b / (3.0 * a);
165     }
166     if (S == 0)
167     {
168         x[0].first = 2.0 * croot(-q / 2.0) - b / (3.0 * a);
169         x[1].first = -croot(-q / 2.0) - b / (3.0 * a);
170         x[2].first = -croot(-q / 2.0) - b / (3.0 * a);
171     }
172     if (S > 0)
173     {
174         double temp1 = croot((-q / 2.0) + sqrt(S)) + croot((-q / 2.0) - sqrt(S));
175         double temp2 = croot((-q / 2.0) + sqrt(S)) - croot((-q / 2.0) - sqrt(S));
176         x[0].first = temp1 - b / (3.0 * a);
177         x[1].first = -temp1 / 2.0 - b / (3.0 * a); x[1].second = sqrt(3) * temp2 / 2.0;
178         x[2].first = -temp1 / 2.0 - b / (3.0 * a); x[2].second = -sqrt(3) * temp2 / 2.0;
179     }
180
181     answer[0] = '\0';
182
183     char tmp[128];
184     sprintf(tmp, "x1 = %.4lf + i * %.4lf;", x[0].first, x[0].second);

```

```

185     strncat(answer, tmp, 128);
186
187     sprintf(tmp, " x2 = %.4lf + i * %.4lf;", x[1].first, x[1].second);
188     strncat(answer, tmp, 128);
189
190     sprintf(tmp, " x3 = %.4lf + i * %.4lf;", x[2].first, x[2].second);
191     strncat(answer, tmp, 128);
192 }

```

Файл server.c

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <time.h>
4  #include <sys/ipc.h>
5  #include <sys/msg.h>
6  #include <sys/errno.h>
7  #include <unistd.h>
8  #include <math.h>
9  #include <stdlib.h>
10
11 struct msg_buf
12 {
13     char mtext[128];
14 } msg;
15
16 struct digit
17 {
18     double first;
19     double second;
20 };
21
22 int parse_line(char* line, int* coef);
23 void do_kardano(char* answer, double a, double b, double c, double d);
24 double croot(double x);
25
26 int main()
27 {
28     int msgqid, rc;
29     key_t msgkey;
30     msgkey = ftok("/mnt/c/labs_os/3_1/send.c", 'm');
31
32     msgqid = msgget(msgkey, IPC_CREAT | 0660);
33

```



```

34  if (msgqid < 0)
35  {
36      perror(strerror(errno));
37      printf("failed to create message queue with msgqid = %d\n", msgqid);
38      return 1;
39  }
40  else
41      printf("message queue %d created\n", msgqid);
42
43  int a, b, c, d, coef[4];
44  char answer[128], line[128];
45
46  while (1)
47  {
48      rc = msgrcv(msgqid, &msg, sizeof(msg) - sizeof(long), 0, 0);
49      //printf("rc: %d\n", rc);
50      if (rc < 0)
51      {
52          perror(strerror(errno));
53          printf("msgrcv failed, rc=%d\n", rc);
54          break;
55      }
56      printf("received msg: %s it's PID = %ld\n", msg.mtext, msg.mtype);
57
58      if (strcmp(msg.mtext, "stop the server\n") == 0)
59          break;
60
61      strcpy(line, msg.mtext);
62      if (parse_line(line, coef) || coef[0] == 0)
63      {
64          strcpy(msg.mtext, "Wrong enter!");
65      }
66      else
67      {
68          a = coef[0];
69          b = coef[1];
70          c = coef[2];
71          d = coef[3];
72          do_kardano(answer, a, b, c, d);
73          strcpy(msg.mtext, answer);
74      }
75
76      rc = msgsnd(msgqid, &msg, sizeof(msg) - sizeof(long), msg.mtype);
77      if (rc < 0)
78      {
79          printf("msgsnd failed, rc = %d\n", rc);
80          return 1;

```

```

81     }
82     else
83         printf("Message send done!\n");
84
85     printf("answer: %s\n", msg.mtext);
86 }
87 return 0;
88 }
89
90 double croot(double x)
91 {
92     if (x < 0)
93         return -pow(-x, 1.0 / 3.0);
94     return pow(x, 1.0 / 3.0);
95 }
96
97 int parse_line(char* line, int* coef)
98 {
99     int i = 0;
100    int j = 0;
101
102    while (line[i] != '\n')
103    {
104        if (line[i] >= '0' && line[i] <= '9')
105        {
106            coef[j] = atoi(line + i);
107            j++;
108
109            while (line[i] >= '0' && line[i] <= '9')
110                i++;
111
112        }
113        else if (line[i] == '-')
114        {
115            coef[j] = atoi(line + i);
116            j++;
117
118            if (line[i + 1] < '0' || line[i + 1] > '9')
119                return 1;
120            i++;
121
122            while (line[i] >= '0' && line[i] <= '9')
123                i++;
124        }
125        else if (line[i] == ' ')
126            i++;
127        else

```

```

128         return 1;
129     }
130
131     if (j != 4)
132         return 1;
133     return 0;
134 }
135
136 void print_mass(int* mass, int len)
137 {
138     for (int i = 0; i != len; i++)
139         printf("%d ", mass[i]);
140     printf("\n");
141 }
142
143 void do_kardano(char* answer, double a, double b, double c, double d)
144 {
145     double p = (3.0 * a * c - b * b) / (3.0 * a * a);
146     double q = (2.0 * b * b * b - 9.0 * a * b * c + 27.0 * a * a * d) / (27.0 * a * a * a);
147     double S = (q * q / 4.0) + (p * p * p / 27.0);
148
149     double F;
150     if (q == 0)
151         F = M_PI / 2.0;
152     if (q < 0)
153         F = atan(-2.0 * sqrt(-S) / q);
154     if (q > 0)
155         F = atan(-2.0 * sqrt(-S) / q) + M_PI;
156
157     struct digit x[3];
158     for (int i = 0; i < 3; i++)
159         x[i].first = x[i].second = 0;
160     if (S < 0)
161     {
162         x[0].first = 2.0 * sqrt(-p / 3.0) * cos(F / 3.0) - b / (3.0 * a);
163         x[1].first = 2.0 * sqrt(-p / 3.0) * cos((F / 3.0) + 2.0 * M_PI / 3.0) - b / (3.0 * a);
164         x[2].first = 2.0 * sqrt(-p / 3.0) * cos((F / 3.0) + 4.0 * M_PI / 3.0) - b / (3.0 * a);
165     }
166     if (S == 0)
167     {
168         x[0].first = 2.0 * croot(-q / 2.0) - b / (3.0 * a);
169         x[1].first = -croot(-q / 2.0) - b / (3.0 * a);
170         x[2].first = -croot(-q / 2.0) - b / (3.0 * a);
171     }
172     if (S > 0)
173     {
174         double temp1 = croot((-q / 2.0) + sqrt(S)) + croot((-q / 2.0) - sqrt(S));

```

```

175     double temp2 = croot((-q / 2.0) + sqrt(S)) - croot((-q / 2.0) - sqrt(S));
176     x[0].first = temp1 - b / (3.0 * a);
177     x[1].first = -temp1 / 2.0 - b / (3.0 * a); x[1].second = sqrt(3) * temp2 / 2.0;
178     x[2].first = -temp1 / 2.0 - b / (3.0 * a); x[2].second = -sqrt(3) * temp2 / 2.0;
179 }
180
181 answer[0] = '\0';
182
183 char tmp[128];
184 sprintf(tmp, "x1 = %.4lf + i * %.4lf;", x[0].first, x[0].second);
185 strncat(answer, tmp, 128);
186
187 sprintf(tmp, " x2 = %.4lf + i * %.4lf;", x[1].first, x[1].second);
188 strncat(answer, tmp, 128);
189
190 sprintf(tmp, " x3 = %.4lf + i * %.4lf;", x[2].first, x[2].second);
191 strncat(answer, tmp, 128);
192 }

```

7.10 Подсчёт слов

Файл client.c

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <time.h>
4  #include <sys/ipc.h>
5  #include <sys/msg.h>
6  #include <sys/errno.h>
7  #include <unistd.h>
8  #include <math.h>
9  #include <stdlib.h>
10
11 struct msg_buf
12 {
13     char mtext[128];
14 } msg;
15
16 struct digit
17 {
18     double first;
19     double second;
20 };

```

```

21
22 int parse_line(char* line, int* coef);
23 void do_kardano(char* answer, double a, double b, double c, double d);
24 double croot(double x);
25
26 int main()
27 {
28     int msgqid, rc;
29     key_t msgkey;
30     msgkey = ftok("/mnt/c/labs_os/3_1/send.c", 'm');
31
32     msgqid = msgget(msgkey, IPC_CREAT | 0660);
33
34     if (msgqid < 0)
35     {
36         perror(strerror(errno));
37         printf("failed to create message queue with msgqid = %d\n", msgqid);
38         return 1;
39     }
40     else
41         printf("message queue %d created\n", msgqid);
42
43     int a, b, c, d, coef[4];
44     char answer[128], line[128];
45
46     while (1)
47     {
48         rc = msgrcv(msgqid, &msg, sizeof(msg) - sizeof(long), 0, 0);
49         //printf("rc: %d\n", rc);
50         if (rc < 0)
51         {
52             perror(strerror(errno));
53             printf("msgrcv failed, rc=%d\n", rc);
54             break;
55         }
56         printf("received msg: %s it's PID = %ld\n", msg.mtext, msg.mtype);
57
58         if (strcmp(msg.mtext, "stop the server\n") == 0)
59             break;
60
61         strcpy(line, msg.mtext);
62         if (parse_line(line, coef) || coef[0] == 0)
63         {
64             strcpy(msg.mtext, "Wrong enter!");
65         }
66         else
67         {

```

```

68         a = coef[0];
69         b = coef[1];
70         c = coef[2];
71         d = coef[3];
72         do_kardano(answer, a, b, c, d);
73         strcpy(msg.mtext, answer);
74     }
75
76     rc = msgsnd(msgqid, &msg, sizeof(msg) - sizeof(long), msg.mtype);
77     if (rc < 0)
78     {
79         printf("msgsnd failed, rc = %d\n", rc);
80         return 1;
81     }
82     else
83         printf("Message send done!\n");
84
85     printf("answer: %s\n", msg.mtext);
86 }
87 return 0;
88 }
89
90 double croot(double x)
91 {
92     if (x < 0)
93         return -pow(-x, 1.0 / 3.0);
94     return pow(x, 1.0 / 3.0);
95 }
96
97 int parse_line(char* line, int* coef)
98 {
99     int i = 0;
100    int j = 0;
101
102    while (line[i] != '\n')
103    {
104        if (line[i] >= '0' && line[i] <= '9')
105        {
106            coef[j] = atoi(line + i);
107            j++;
108
109            while (line[i] >= '0' && line[i] <= '9')
110                i++;
111        }
112        else if (line[i] == '-')
113        {
114

```

```

115         coef[j] = atoi(line + i);
116         j++;
117
118         if (line[i + 1] < '0' || line[i + 1] > '9')
119             return 1;
120         i++;
121
122         while (line[i] >= '0' && line[i] <= '9')
123             i++;
124     }
125     else if (line[i] == ' ')
126         i++;
127     else
128         return 1;
129 }
130
131 if (j != 4)
132     return 1;
133 return 0;
134 }
135
136 void print_mass(int* mass, int len)
137 {
138     for (int i = 0; i != len; i++)
139         printf("%d ", mass[i]);
140     printf("\n");
141 }
142
143 void do_kardano(char* answer, double a, double b, double c, double d)
144 {
145     double p = (3.0 * a * c - b * b) / (3.0 * a * a);
146     double q = (2.0 * b * b * b - 9.0 * a * b * c + 27.0 * a * a * d) / (27.0 * a * a * a);
147     double S = (q * q / 4.0) + (p * p * p / 27.0);
148
149     double F;
150     if (q == 0)
151         F = M_PI / 2.0;
152     if (q < 0)
153         F = atan(-2.0 * sqrt(-S) / q);
154     if (q > 0)
155         F = atan(-2.0 * sqrt(-S) / q) + M_PI;
156
157     struct digit x[3];
158     for (int i = 0; i < 3; i++)
159         x[i].first = x[i].second = 0;
160     if (S < 0)
161     {

```

```

162     x[0].first = 2.0 * sqrt(-p / 3.0) * cos(F / 3.0) - b / (3.0 * a);
163     x[1].first = 2.0 * sqrt(-p / 3.0) * cos((F / 3.0) + 2.0 * M_PI / 3.0) - b / (3.0 * a);
164     x[2].first = 2.0 * sqrt(-p / 3.0) * cos((F / 3.0) + 4.0 * M_PI / 3.0) - b / (3.0 * a);
165 }
166 if (S == 0)
167 {
168     x[0].first = 2.0 * croot(-q / 2.0) - b / (3.0 * a);
169     x[1].first = -croot(-q / 2.0) - b / (3.0 * a);
170     x[2].first = -croot(-q / 2.0) - b / (3.0 * a);
171 }
172 if (S > 0)
173 {
174     double temp1 = croot((-q / 2.0) + sqrt(S)) + croot((-q / 2.0) - sqrt(S));
175     double temp2 = croot((-q / 2.0) + sqrt(S)) - croot((-q / 2.0) - sqrt(S));
176     x[0].first = temp1 - b / (3.0 * a);
177     x[1].first = -temp1 / 2.0 - b / (3.0 * a); x[1].second = sqrt(3) * temp2 / 2.0;
178     x[2].first = -temp1 / 2.0 - b / (3.0 * a); x[2].second = -sqrt(3) * temp2 / 2.0;
179 }
180
181 answer[0] = '\0';
182
183 char tmp[128];
184 sprintf(tmp, "x1 = %.4lf + i * %.4lf;", x[0].first, x[0].second);
185 strncat(answer, tmp, 128);
186
187 sprintf(tmp, " x2 = %.4lf + i * %.4lf;", x[1].first, x[1].second);
188 strncat(answer, tmp, 128);
189
190 sprintf(tmp, " x3 = %.4lf + i * %.4lf;", x[2].first, x[2].second);
191 strncat(answer, tmp, 128);
192 }

```

Файл server.c

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <time.h>
4  #include <sys/ipc.h>
5  #include <sys/msg.h>
6  #include <sys/errno.h>
7  #include <unistd.h>
8  #include <math.h>
9  #include <stdlib.h>

```

10


```

11 struct msg_buf
12 {
13     char mtext[128];
14 } msg;
15
16 struct digit
17 {
18     double first;
19     double second;
20 };
21
22 int parse_line(char* line, int* coef);
23 void do_kardano(char* answer, double a, double b, double c, double d);
24 double croot(double x);
25
26 int main()
27 {
28     int msgqid, rc;
29     key_t msgkey;
30     msgkey = ftok("/mnt/c/labs_os/3_1/send.c", 'm');
31
32     msgqid = msgget(msgkey, IPC_CREAT | 0660);
33
34     if (msgqid < 0)
35     {
36         perror(strerror(errno));
37         printf("failed to create message queue with msgqid = %d\n", msgqid);
38         return 1;
39     }
40     else
41         printf("message queue %d created\n", msgqid);
42
43     int a, b, c, d, coef[4];
44     char answer[128], line[128];
45
46     while (1)
47     {
48         rc = msgrcv(msgqid, &msg, sizeof(msg) - sizeof(long), 0, 0);
49         //printf("rc: %d\n", rc);
50         if (rc < 0)
51         {
52             perror(strerror(errno));
53             printf("msgrcv failed, rc=%d\n", rc);
54             break;
55         }
56         printf("received msg: %s it's PID = %ld\n", msg.mtext, msg.mtype);
57

```

```

58     if (strcmp(msg.mtext, "stop the server\n") == 0)
59         break;
60
61     strcpy(line, msg.mtext);
62     if (parse_line(line, coef) || coef[0] == 0)
63     {
64         strcpy(msg.mtext, "Wrong enter!");
65     }
66     else
67     {
68         a = coef[0];
69         b = coef[1];
70         c = coef[2];
71         d = coef[3];
72         do_kardano(answer, a, b, c, d);
73         strcpy(msg.mtext, answer);
74     }
75
76     rc = msgsnd(msgqid, &msg, sizeof(msg) - sizeof(long), msg.mtype);
77     if (rc < 0)
78     {
79         printf("msgsnd failed, rc = %d\n", rc);
80         return 1;
81     }
82     else
83         printf("Message send done!\n");
84
85     printf("answer: %s\n", msg.mtext);
86 }
87 return 0;
88 }
89
90 double croot(double x)
91 {
92     if (x < 0)
93         return -pow(-x, 1.0 / 3.0);
94     return pow(x, 1.0 / 3.0);
95 }
96
97 int parse_line(char* line, int* coef)
98 {
99     int i = 0;
100    int j = 0;
101
102    while (line[i] != '\n')
103    {
104        if (line[i] >= '0' && line[i] <= '9')

```

```

105     {
106         coef[j] = atoi(line + i);
107         j++;
108
109         while (line[i] >= '0' && line[i] <= '9')
110             i++;
111
112     }
113     else if (line[i] == '-')
114     {
115         coef[j] = atoi(line + i);
116         j++;
117
118         if (line[i + 1] < '0' || line[i + 1] > '9')
119             return 1;
120         i++;
121
122         while (line[i] >= '0' && line[i] <= '9')
123             i++;
124     }
125     else if (line[i] == ' ')
126         i++;
127     else
128         return 1;
129 }
130
131 if (j != 4)
132     return 1;
133 return 0;
134 }
135
136 void print_mass(int* mass, int len)
137 {
138     for (int i = 0; i != len; i++)
139         printf("%d ", mass[i]);
140     printf("\n");
141 }
142
143 void do_kardano(char* answer, double a, double b, double c, double d)
144 {
145     double p = (3.0 * a * c - b * b) / (3.0 * a * a);
146     double q = (2.0 * b * b * b - 9.0 * a * b * c + 27.0 * a * a * d) / (27.0 * a * a * a);
147     double S = (q * q / 4.0) + (p * p * p / 27.0);
148
149     double F;
150     if (q == 0)
151         F = M_PI / 2.0;

```

```

152     if (q < 0)
153         F = atan(-2.0 * sqrt(-S) / q);
154     if (q > 0)
155         F = atan(-2.0 * sqrt(-S) / q) + M_PI;
156
157     struct digit x[3];
158     for (int i = 0; i < 3; i++)
159         x[i].first = x[i].second = 0;
160     if (S < 0)
161     {
162         x[0].first = 2.0 * sqrt(-p / 3.0) * cos(F / 3.0) - b / (3.0 * a);
163         x[1].first = 2.0 * sqrt(-p / 3.0) * cos((F / 3.0) + 2.0 * M_PI / 3.0) - b / (3.0 * a);
164         x[2].first = 2.0 * sqrt(-p / 3.0) * cos((F / 3.0) + 4.0 * M_PI / 3.0) - b / (3.0 * a);
165     }
166     if (S == 0)
167     {
168         x[0].first = 2.0 * croot(-q / 2.0) - b / (3.0 * a);
169         x[1].first = -croot(-q / 2.0) - b / (3.0 * a);
170         x[2].first = -croot(-q / 2.0) - b / (3.0 * a);
171     }
172     if (S > 0)
173     {
174         double temp1 = croot((-q / 2.0) + sqrt(S)) + croot((-q / 2.0) - sqrt(S));
175         double temp2 = croot((-q / 2.0) + sqrt(S)) - croot((-q / 2.0) - sqrt(S));
176         x[0].first = temp1 - b / (3.0 * a);
177         x[1].first = -temp1 / 2.0 - b / (3.0 * a); x[1].second = sqrt(3) * temp2 / 2.0;
178         x[2].first = -temp1 / 2.0 - b / (3.0 * a); x[2].second = -sqrt(3) * temp2 / 2.0;
179     }
180
181     answer[0] = '\0';
182
183     char tmp[128];
184     sprintf(tmp, "x1 = %.4lf + i * %.4lf;", x[0].first, x[0].second);
185     strncat(answer, tmp, 128);
186
187     sprintf(tmp, " x2 = %.4lf + i * %.4lf;", x[1].first, x[1].second);
188     strncat(answer, tmp, 128);
189
190     sprintf(tmp, " x3 = %.4lf + i * %.4lf;", x[2].first, x[2].second);
191     strncat(answer, tmp, 128);
192 }

```

7.11 Чат

Файл client_chat

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <time.h>
4  #include <sys/ipc.h>
5  #include <sys/msg.h>
6  #include <sys/errno.h>
7  #include <unistd.h>
8  #include <math.h>
9  #include <stdlib.h>
10
11 struct msg_buf
12 {
13     char mtext[128];
14 } msg;
15
16 struct digit
17 {
18     double first;
19     double second;
20 };
21
22 int parse_line(char* line, int* coef);
23 void do_kardano(char* answer, double a, double b, double c, double d);
24 double croot(double x);
25
26 int main()
27 {
28     int msgqid, rc;
29     key_t msgkey;
30     msgkey = ftok("/mnt/c/labs_os/3_1/send.c", 'm');
31
32     msgqid = msgget(msgkey, IPC_CREAT | 0660);
33
34     if (msgqid < 0)
35     {
36         perror(strerror(errno));
37         printf("failed to create message queue with msgqid = %d\n", msgqid);
38         return 1;
39     }
40     else
41         printf("message queue %d created\n", msgqid);
42
43     int a, b, c, d, coef[4];
```

```

44 char answer[128], line[128];
45
46 while (1)
47 {
48     rc = msgrcv(msgqid, &msg, sizeof(msg) - sizeof(long), 0, 0);
49     //printf("rc: %d\n", rc);
50     if (rc < 0)
51     {
52         perror(strerror(errno));
53         printf("msgrcv failed, rc=%d\n", rc);
54         break;
55     }
56     printf("received msg: %s it's PID = %ld\n", msg.mtext, msg.mtype);
57
58     if (strcmp(msg.mtext, "stop the server\n") == 0)
59         break;
60
61     strcpy(line, msg.mtext);
62     if (parse_line(line, coef) || coef[0] == 0)
63     {
64         strcpy(msg.mtext, "Wrong enter!");
65     }
66     else
67     {
68         a = coef[0];
69         b = coef[1];
70         c = coef[2];
71         d = coef[3];
72         do_kardano(answer, a, b, c, d);
73         strcpy(msg.mtext, answer);
74     }
75
76     rc = msgsnd(msgqid, &msg, sizeof(msg) - sizeof(long), msg.mtype);
77     if (rc < 0)
78     {
79         printf("msgsnd failed, rc = %d\n", rc);
80         return 1;
81     }
82     else
83         printf("Message send done!\n");
84
85     printf("answer: %s\n", msg.mtext);
86 }
87 return 0;
88 }
89
90 double croot(double x)

```

```

91 {
92     if (x < 0)
93         return -pow(-x, 1.0 / 3.0);
94     return pow(x, 1.0 / 3.0);
95 }
96
97 int parse_line(char* line, int* coef)
98 {
99     int i = 0;
100    int j = 0;
101
102    while (line[i] != '\n')
103    {
104        if (line[i] >= '0' && line[i] <= '9')
105        {
106            coef[j] = atoi(line + i);
107            j++;
108
109            while (line[i] >= '0' && line[i] <= '9')
110                i++;
111
112        }
113        else if (line[i] == '-')
114        {
115            coef[j] = atoi(line + i);
116            j++;
117
118            if (line[i + 1] < '0' || line[i + 1] > '9')
119                return 1;
120            i++;
121
122            while (line[i] >= '0' && line[i] <= '9')
123                i++;
124        }
125        else if (line[i] == ' ')
126            i++;
127        else
128            return 1;
129    }
130
131    if (j != 4)
132        return 1;
133    return 0;
134 }
135
136 void print_mass(int* mass, int len)
137 {

```

```

138     for (int i = 0; i != len; i++)
139         printf("%d ", mass[i]);
140     printf("\n");
141 }
142
143 void do_kardano(char* answer, double a, double b, double c, double d)
144 {
145     double p = (3.0 * a * c - b * b) / (3.0 * a * a);
146     double q = (2.0 * b * b * b - 9.0 * a * b * c + 27.0 * a * a * d) / (27.0 * a * a * a);
147     double S = (q * q / 4.0) + (p * p * p / 27.0);
148
149     double F;
150     if (q == 0)
151         F = M_PI / 2.0;
152     if (q < 0)
153         F = atan(-2.0 * sqrt(-S) / q);
154     if (q > 0)
155         F = atan(-2.0 * sqrt(-S) / q) + M_PI;
156
157     struct digit x[3];
158     for (int i = 0; i < 3; i++)
159         x[i].first = x[i].second = 0;
160     if (S < 0)
161     {
162         x[0].first = 2.0 * sqrt(-p / 3.0) * cos(F / 3.0) - b / (3.0 * a);
163         x[1].first = 2.0 * sqrt(-p / 3.0) * cos((F / 3.0) + 2.0 * M_PI / 3.0) - b / (3.0 * a);
164         x[2].first = 2.0 * sqrt(-p / 3.0) * cos((F / 3.0) + 4.0 * M_PI / 3.0) - b / (3.0 * a);
165     }
166     if (S == 0)
167     {
168         x[0].first = 2.0 * croot(-q / 2.0) - b / (3.0 * a);
169         x[1].first = -croot(-q / 2.0) - b / (3.0 * a);
170         x[2].first = -croot(-q / 2.0) - b / (3.0 * a);
171     }
172     if (S > 0)
173     {
174         double temp1 = croot((-q / 2.0) + sqrt(S)) + croot((-q / 2.0) - sqrt(S));
175         double temp2 = croot((-q / 2.0) + sqrt(S)) - croot((-q / 2.0) - sqrt(S));
176         x[0].first = temp1 - b / (3.0 * a);
177         x[1].first = -temp1 / 2.0 - b / (3.0 * a); x[1].second = sqrt(3) * temp2 / 2.0;
178         x[2].first = -temp1 / 2.0 - b / (3.0 * a); x[2].second = -sqrt(3) * temp2 / 2.0;
179     }
180
181     answer[0] = '\0';
182
183     char tmp[128];
184     sprintf(tmp, "x1 = %.4lf + i * %.4lf;", x[0].first, x[0].second);

```



```

185     strncat(answer, tmp, 128);
186
187     sprintf(tmp, " x2 = %.4lf + i * %.4lf;", x[1].first, x[1].second);
188     strncat(answer, tmp, 128);
189
190     sprintf(tmp, " x3 = %.4lf + i * %.4lf;", x[2].first, x[2].second);
191     strncat(answer, tmp, 128);
192 }

```

7.12 ПО «Сетевой сервер»

Файл client.cpp

```

1  #include <iostream>
2  #include <sys/socket.h>
3  #include <sys/types.h>
4  #include <unistd.h>
5  #include <netinet/in.h>
6  #include <stdlib.h>
7  #include <string>
8  #include <sys/stat.h>
9  #include <sys/un.h>
10 #include <arpa/inet.h>
11 #include <fcntl.h>
12 #include <fstream>
13 #include <string.h>
14 #include "structures.h"
15
16 #define SIZE 4294967295
17
18 using namespace std;
19
20 int main()
21 {
22     MESSAGE mes_struct;
23     int sock = -1;
24     struct sockaddr_in addr;
25     char server_ip[12] = "127.0.0.1";
26
27     sock = socket(AF_INET, SOCK_STREAM, 0);
28     //cout << "Socket ID : " << sock << endl;
29     if (sock == -1)
30     {

```

```

31     cout << "CLIENT: There is a problem with making socket." << endl;
32     exit(-1);
33 }
34
35 addr.sin_family = AF_INET;
36 addr.sin_port = htons(1234);
37 //addr.sin_addr.s_addr = inet_addr(ip.c_str());
38 inet_pton(AF_INET, server_ip,
39     &addr.sin_addr);
40
41
42 string choice = "";
43 cout << "Do you want MD5 or database(\"M\"-MD5/\"D\"-Database)?\n(\"quit!\" - quit)" << endl <<
44     ↪ ">>>";
45 bool flag = 1;
46 while (flag)
47 {
48     cin >> choice;
49     flag = 0;
50     if (choice == "M")
51     {
52         mes_struct.type = MESSAGE::MD5;
53     }
54     else if (choice == "D")
55     {
56         mes_struct.type = MESSAGE::DATABASE;
57     }
58     else if (choice == "quit!")
59     {
60         return 0;
61     }
62     else
63     {
64         cout << "Error, try again:";
65         flag = 1;
66     }
67 }
68
69 if (connect(sock, (struct sockaddr*)&addr, sizeof(addr)) == -1)
70 {
71     cout << "CLIENT: There is a problem with connecting socket." << endl;
72     if (close(sock) == -1)
73     {
74         cout << "CLIENT: There is error with closing socket." << endl;
75     }
76     exit(-1);
77 }

```

```

77
78 cout << "Sending..." << endl;
79
80 send(sock, (MESSAGE*)&mes_struct, sizeof(mes_struct), 0);
81 cout << "Sent." << endl;
82 recv(sock, (MESSAGE*)&mes_struct, sizeof(mes_struct), 0);
83 cout << mes_struct.data << endl;
84
85 // MD5 обработка
86 if (choice == "M")
87 {
88     cout << "Do you want to cache a file or string: ";
89     cin >> choice;
90
91     if (choice == "file")
92     {
93         cout << "Enter the file name: ";
94         cin >> choice;
95         int f_hesh = open(choice.c_str(), ios::in);
96
97         mes_struct.count_byte = read(f_hesh, mes_struct.data, 64);
98         while (mes_struct.count_byte == 64)
99         {
100             send(sock, (MESSAGE*)&mes_struct, sizeof(mes_struct), 0);
101             mes_struct.count_byte = read(f_hesh, mes_struct.data, 64);
102         }
103         send(sock, (MESSAGE*)&mes_struct, sizeof(mes_struct), 0);
104         recv(sock, (MESSAGE*)&mes_struct, sizeof(mes_struct), 0); // получаем хеш
105         cout << "Server_md5 returned: " << mes_struct.data << endl;
106     }
107     else if (choice == "string")
108     {
109         cout << "Enter a string to hash it\n>>>";
110         getchar();
111         getline(cin, choice);
112
113         int i = 0;
114
115         while (i < choice.length())
116         {
117             mes_struct.data[i] = choice[i];
118             i++;
119             if (i % 64 == 0)
120             {
121                 mes_struct.count_byte = 64;
122                 send(sock, (MESSAGE*)&mes_struct, sizeof(mes_struct), 0);
123             }

```

```

124     }
125     mes_struct.count_byte = i % 64;
126     send(sock, (MESSAGE*)&mes_struct, sizeof(mes_struct), 0);
127     recv(sock, (MESSAGE*)&mes_struct, sizeof(mes_struct), 0); // получаем хеш
128     cout << "Server_md5 returned: " << mes_struct.data << endl;
129 }
130 else
131 {
132     cout << "Input error" << endl;
133 }
134 }
135 else // База данных обработка
136 {
137     getchar();
138     while (1)
139     {
140         recv(sock, (MESSAGE*)&mes_struct, sizeof(mes_struct), 0);
141         cout << mes_struct.data;
142         if (!strcmp(mes_struct.data, ">>>"))
143         {
144             string buf;
145
146             getline(cin, buf);
147
148             if (buf == "quit!")
149             {
150                 break;
151             }
152
153             if (buf.size() > BUFFSIZE)
154             {
155                 cout << "Слишком длинная строка" << endl;
156                 break;
157             }
158             strcpy(mes_struct.data, buf.c_str());
159
160             send(sock, (MESSAGE*)&mes_struct, sizeof(mes_struct), 0);
161         }
162         else
163         {
164             cout << endl;
165         }
166     }
167 }
168
169 if (shutdown(sock, 2) == -1)
170 {

```

```

171
172     cout << "CLIENT: There is error with shutdowning socket." << endl;
173     if (close(sock) == -1)
174     {
175         cout << "CLIENT: There is error with closing socket." << endl;
176     }
177     exit(-1);
178 }
179
180 if (close(sock) == -1)
181 {
182     cout << "CLIENT: There is error with closing socket." << endl;
183     exit(-1);
184 }
185 return 0;
186 }

```

Файл netserver.cpp

```

1  #include <iostream>
2  #include <sys/socket.h>
3  #include <sys/types.h>
4  #include <unistd.h>
5  #include <netinet/in.h>
6  #include <stdlib.h>
7  #include <cstring>
8  # include <sys/ipc.h>
9  #include <sys/msg.h>
10 #include <sys/sem.h>
11 #include <sys/shm.h>
12 #include "structures.h"
13
14 using namespace std;
15
16 int db_msg_queue(int, MESSAGE&);
17 int md5_sem(int, MESSAGE&);
18
19 int main()
20 {
21     int accepted_sock;
22     int sock = -1;
23     struct sockaddr_in addr;
24     MESSAGE mes_struct;
25

```

```

26  unsigned int bytes_read = 0;
27
28  sock = socket(AF_INET, SOCK_STREAM, 0);
29  if (sock == -1)
30  {
31      cout << "SERVER: There is an error with openning socket." << endl;
32      exit(-1);
33  }
34
35  addr.sin_family = AF_INET;
36  addr.sin_port = htons(1234);
37  addr.sin_addr.s_addr = htonl(INADDR_ANY);
38
39  if (bind(sock, (struct sockaddr*)&addr, sizeof(addr)) == -1)
40  {
41      cout << "SERVER: There is an error with binding sockets." << endl;
42      close(sock);
43      exit(-1);
44  }
45
46  bool stop_flag = 0;
47
48  while (1)
49  {
50      cout << "Listening..." << endl;
51      int status = listen(sock, 5);
52      if (status != 0)
53      {
54          cout << "SERVER: There is an error with listenning sockets." << endl;
55          close(sock);
56          exit(-1);
57      }
58
59      accepted_sock = accept(sock, NULL, NULL);
60      if (accepted_sock == -1)
61      {
62          cout << "SERVER: There is an error with accepting sockets." << endl;
63          close(sock);
64          exit(-1);
65      }
66      //cout << "Access Socket ID: " << accepted_sock << endl;
67
68      if (fork() == 0)
69      {
70          cout << "Recieving..." << endl;
71          close(sock);
72          bytes_read = recv(accepted_sock, (MESSAGE*)&mes_struct, sizeof(MESSAGE), 0);

```

```

73     cout << "Recieved..." << endl;
74
75     if (bytes_read <= 0)
76     {
77         cout << "SERVER: There is an error with receiving." << endl;
78         close(accepted_sock);
79         exit(-1);
80     }
81
82     if (mes_struct.type == MESSAGE::MD5) // общение с сервером вычисляющим хеш на семафорах
83     {
84         md5_sem(accepted_sock, mes_struct);
85     }
86     else if(mes_struct.type == MESSAGE::DATABASE) // база данных на очереди сообщений
87     {
88         db_msg_queue(accepted_sock, mes_struct);
89     }
90 }
91 else
92 {
93     close(accepted_sock);
94 }
95 }
96 }
97
98 int db_msg_queue(int accepted_sock, MESSAGE& mes_struct)
99 {
100     int msgqid, rc;
101     key_t msgkey;
102     msgkey = ftok("string_for_identical_keys_queue", 'm');
103     char text[128];
104     int bytes_read;
105
106     msgqid = msgget(msgkey, IPC_CREAT | 0660);
107     if (msgqid < 0)
108     {
109         perror(strerror(errno));
110         cout << "Failed to create message queue with msgqid:" << msgqid << endl;
111         strcpy(mes_struct.data, "Failed to create message queue with msgqid:\n");
112         if (send(accepted_sock, (MESSAGE*)&mes_struct, sizeof(mes_struct), 0) == -1)
113         {
114             cout << "SERVER: There is an error with sending message." << endl;
115         }
116         return 1;
117     }
118     else
119         printf("Message queue %d created\n", msgqid);

```

```

120
121 cout << "SERVER: The connection to the database is established" << endl;
122 strcpy(mes_struct.data, "The connection to the database is established.\n To exit, enter:
↪ quit!\n");
123
124 if (send(accepted_sock, (MESSAGE*)&mes_struct, sizeof(mes_struct), 0) == -1)
125 {
126     cout << "SERVER: There is an error with sending message." << endl;
127 }
128 cout << "Sent." << endl;
129
130 while (1) // цикл общения с бд
131 {
132     strcpy(mes_struct.data, ">>>"); // сделаем строку, которая предлагает ввести команду
133
134     if (send(accepted_sock, (MESSAGE*)&mes_struct, sizeof(mes_struct), 0) == -1) // отправим клиенту
135     {
136         cout << "SERVER: There is an error with sending message." << endl;
137     }
138     cout << "Sent." << endl;
139
140     cout << "Sending..." << endl;
141
142     cout << "Recieving..." << endl;
143     bytes_read = recv(accepted_sock, (MESSAGE*)&mes_struct, sizeof(MESSAGE), 0); // получим от
↪ клиента строку
144     cout << "Recieved..." << endl;
145
146     if (bytes_read <= 0)
147     {
148         cout << "SERVER: There is an error with receiving." << endl;
149         close(accepted_sock);
150         exit(-1);
151     }
152     msg.mtype = getpid();
153     strcpy(msg.mtext, mes_struct.data);
154
155     if (strcmp(text, "stop\n") == 0)
156     {
157         break;
158     }
159
160     rc = msgsnd(msgqid, &msg, sizeof(msg) - sizeof(long), 0);
161
162     if (rc < 0)
163     {
164         printf("msgsnd failed, rc = %d\n", rc);

```



```

165         return 1;
166     }
167     else
168         printf("Message send done!\n");
169
170     if (strcmp(text, "stop the server\n") == 0)
171     {
172         break;
173     }
174
175     // получим
176     rc = msgrcv(msgqid, &msg, sizeof(msg) - sizeof(long), getpid(), 0);
177     if (rc < 0)
178     {
179         perror(strerror(errno));
180         printf("msgrcv failed, rc=%d\n", rc);
181         break;
182     }
183     printf("received from server msg: %s\n", msg.mtext);
184
185     strcpy(mes_struct.data, msg.mtext);
186
187     if (mes_struct.data[0] == '\0')
188         continue;
189
190     if (send(accepted_sock, (MESSAGE*)&mes_struct, sizeof(mes_struct), 0) == -1) // отправим клиенту
191     {
192         cout << "SERVER: There is an error with sending message." << endl;
193     }
194 }
195 return 0;
196 }
197
198 int md5_sem(int accepted_sock, MESSAGE& mes_struct)
199 {
200     int semkey = ftok(".", 'm');
201
202     int bytes_read;
203
204
205     if (semkey < 0)
206     {
207         cout << "Wrong key" << endl;
208         return 1;
209     }
210
211     int shmkey = ftok(".", 'm');

```

```

212     if (shmkey < 0)
213     {
214         cout << "Wrong key" << endl;
215         return 1;
216     }
217
218     int semid = semget(semkey, 2, 0666);
219     if (semid == -1)
220     {
221         cout << "Semaphore does not exist" << endl;
222         return 1;
223     }
224
225     int shmid = shmget(shmkey, sizeof(SEM_STRUCT), 0666);
226     if (shmid == -1)
227     {
228         cout << "Shared memory does not exist" << endl;
229         return 1;
230     }
231
232     SEM_STRUCT* shm_address = (SEM_STRUCT*)shmat(shmid, NULL, 0);
233     if (shm_address == NULL)
234     {
235         cout << "Can't get access to shared memory" << endl;
236         return 1;
237     }
238
239     strcpy(mes_struct.data, "MD5 is ready\n");
240
241     if (send(accepted_sock, (MESSAGE*)&mes_struct, sizeof(mes_struct), 0) == -1)
242     {
243         cout << "SERVER: There is an error with sending message." << endl;
244     }
245
246     cout << "semid netserver: " << semid << endl;
247
248     char line[BUFFSIZE] = "123456";
249     struct sembuf operations[2];
250
251     while (1)
252     {
253         bytes_read = recv(accepted_sock, (MESSAGE*)&mes_struct, sizeof(MESSAGE), 0); // получим от
            ↪ клиента строку
254         cout << "Recieved..." << endl;
255
256         if (bytes_read <= 0)
257         {

```

```

258     cout << "SERVER: There is an error with receiving." << endl;
259     close(accepted_sock);
260     exit(-1);
261 }
262
263 //cout << "netserver received the following line:" << mes_struct.data << endl;
264
265 operations[0].sem_num = 0;
266 operations[0].sem_op = 0;
267 operations[0].sem_flg = 0;
268
269 operations[1].sem_num = 1;
270 operations[1].sem_op = -1;
271 operations[1].sem_flg = 0;
272
273
274 if (semop(semid, operations, 2) == -1) // set value
275 {
276     if (strcmp(line, "\n") != 0)
277         cout << "Can't do oper for sem" << endl;
278     break;
279 }
280
281 strcpy(shm_address->data, mes_struct.data); // copy str // копировка в server_md5.cpp
282 shm_address->count_bytes = mes_struct.count_byte;
283
284 operations[0].sem_num = 0;
285 operations[0].sem_op = 1;
286 operations[0].sem_flg = SEM_UNDO;
287
288 operations[1].sem_num = 1;
289 operations[1].sem_op = 0;
290 operations[1].sem_flg = SEM_UNDO;
291
292 if (semop(semid, operations, 2) == -1)
293 {
294     if (strcmp(line, "\n") != 0)
295         cout << "Can't do oper for sem" << endl;
296     return 1;
297 }
298
299 operations[0].sem_num = 0;
300 operations[0].sem_op = -1;
301 //operations[0].sem_flg = 0;
302
303 operations[1].sem_num = 1;
304 operations[1].sem_op = -1;

```

```

305     //operations[1].sem_flg = 0;
306
307     if (semop(semid, operations, 2) == -1) // set value
308     {
309         if (strcmp(line, "\n") != 0)
310             cout << "Can't do oper for sem" << endl;
311         return 1;
312     }
313
314     if (mes_struct.count_byte < 64)
315     {
316         strcpy(mes_struct.data, shm_address->data); // отправка хеша
317         if (send(accepted_sock, (MESSAGE*)&mes_struct, sizeof(mes_struct), 0) == -1) // отправим
318             ↪ клиенту
319         {
320             cout << "SERVER: There is an error with sending message." << endl;
321         }
322         cout << "Sent." << endl;
323         break;
324     }
325
326     operations[0].sem_num = 0;
327     operations[0].sem_op = 0;
328     //operations[0].sem_flg = IPC_NOWAIT;
329
330     operations[1].sem_num = 1;
331     operations[1].sem_op = 1;
332     //operations[1].sem_flg = IPC_NOWAIT;
333
334     if (semop(semid, operations, 2) == -1)
335     {
336         if (strcmp(line, "\n") != 0)
337             cout << "Can't do oper for sem" << endl;
338         return 1;
339     }
340     return 0;
341 }

```

Файл server_md5.cpp

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <sys/ipc.h>

```

```

4 #include <sys/sem.h>
5 #include <sys/shm.h>
6 #include <stdlib.h>
7 #include <string.h>
8 #include <iostream>
9 #include "MD5.h"
10 #include "structures.h"
11
12 using namespace std;
13
14 int main()
15 {
16     int semkey = ftok(".", 'm');
17     MD5 md5;
18     if (semkey < 0)
19     {
20         printf("Wrong key\n");
21         return 1;
22     }
23
24     int shmkey = ftok(".", 'm');
25     if (shmkey < 0)
26     {
27         cout << "Wrong key" << endl;
28         return 1;
29     }
30
31     int semid = semget(semkey, 2, 0666 | IPC_CREAT /*/ IPC_EXCL*/);
32     if (semid == -1)
33     {
34         cout << "Can't create sem" << endl;
35         return 1;
36     }
37
38     short sarray[2];
39     sarray[0] = 0;
40     sarray[1] = 1;
41     if (semctl(semid, 1, SETALL, sarray) < 0)
42     {
43         cout << "Can't do init" << endl;
44         return 1;
45     }
46
47     int shmid = shmget(shmkey, sizeof(SEM_STRUCT), 0666 | IPC_CREAT /* / IPC_EXCL*/);
48     if (shmid < 0)
49     {
50         cout << "Can't get shared memory" << endl;

```

```

51     return 1;
52 }
53
54 SEM_STRUCT* shm_address = (SEM_STRUCT*)shmat(shmid, NULL, 0);
55 if (shm_address == NULL)
56 {
57     cout << "Can't get access to shared memory" << endl;
58     return 1;
59 }
60 cout << "The server is ready" << endl;
61
62 struct sembuf operations[2];
63 struct shmid_ds shm_struct;
64 while (1)
65 {
66     operations[0].sem_num = 0;
67     operations[0].sem_op = -1;
68     operations[0].sem_flg = 0;
69
70     operations[1].sem_num = 1;
71     operations[1].sem_op = 0;
72     operations[1].sem_flg = 0;
73
74     if (semop(semid, operations, 2) == -1)
75     {
76         cout << "Can't do oper for sem" << endl;
77         break;
78     }
79     cout << "Client reseved : " << shm_address->data << endl;
80
81     cout << "data: " << shm_address->data << "count" << shm_address->count_bytes << endl;
82
83     md5.new_block(shm_address->data, shm_address->count_bytes);
84
85     operations[0].sem_num = 0;
86     operations[0].sem_op = 1;
87     operations[0].sem_flg = SEM_UNDO;
88
89     operations[1].sem_num = 1;
90     operations[1].sem_op = 1;
91     operations[1].sem_flg = SEM_UNDO;
92
93     if (semop(semid, operations, 2) == -1)
94     {
95         cout << "Can't do oper for sem" << endl;;
96         return 1;
97     }

```

```

98     }
99
100     if (semctl(semid, 1, IPC_RMID) == -1)
101     {
102         cout << "Remove id failed" << endl;
103         return 1;
104     }
105
106     if (shmdt(shm_address) == -1)
107     {
108         cout << "Shmdt failed" << endl;
109         return 1;
110     }
111
112     if (shmctl(shmid, IPC_RMID, &shmid_struct) == -1)
113     {
114         cout << "Shmctl failed" << endl;
115         return 1;
116     }
117
118     return 0;
119 }

```

Файл MD5.h

```

1  #pragma once
2  #include <iostream>
3  #include <vector>
4  #include <iterator>
5  #include <iomanip>
6  #include <sstream>
7  #include <cmath>
8
9  using namespace std;
10 class MD5
11 {
12     typedef unsigned int uint4;
13     typedef unsigned char uint1;
14
15
16 private:
17     uint4 bits512[16];
18     uint4 size_byte{ 0 };
19     uint4 state[4];

```

```

20  uint1 digest[16]; // the result
21
22  uint4 T[64];
23  uint4 S[64] = { 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,
24                  5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20,
25                  4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,
26                  6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21 };
27
28 private:
29     static inline uint4 F(uint4 x, uint4 y, uint4 z);
30     static inline uint4 G(uint4 x, uint4 y, uint4 z);
31     static inline uint4 H(uint4 x, uint4 y, uint4 z);
32     static inline uint4 I(uint4 x, uint4 y, uint4 z);
33
34     static inline uint4 CLS(uint4 x, int n);
35
36     int to_uint32(const char data[], uint4 first, uint4 len); // строку в массив unsigned int
37     void to_uint32_cut(int flag);
38     void hmd5();
39
40     void hexdigest(char buf[]) const;
41     void encode();
42
43     void _init()
44     {
45         state[0] = 0x67452301;
46         state[1] = 0xefcdab89;
47         state[2] = 0x98badcfe;
48         state[3] = 0x10325476;
49
50         size_byte = 0;
51     }
52
53 public:
54     void new_block(char data[], uint4 len);
55
56     MD5()
57     {
58         _init();
59         for (int i = 0; i < 64; i++)
60             T[i] = uint4(pow(2, 32) * abs(sin(double(i) + 1)));
61     }
62
63 };
64
65 void MD5::to_uint32_cut(int count)
66 {

```



```

67     int i;
68
69     for (i = 0; i < 16; i++)
70     {
71         bits512[i] = 0;
72     }
73     if (count == 0)
74     {
75         bits512[0] |= 1 << 31;
76     }
77     bits512[14] = (uint4)((long long)(size_byte) * 8);
78
79     bits512[15] = (uint4)((long long)(size_byte) * 8 >> 32);
80 }
81
82
83 void MD5::new_block(char data[], uint4 count_bytes)
84 {
85     to_uint32(data, 0, count_bytes);
86     hmd5();
87     if (count_bytes < 64)
88     {
89         if (count_bytes >= 56 || count_bytes == 0)
90         {
91             to_uint32_cut(count_bytes);
92             hmd5();
93         }
94         encode();
95         hexdigest(data);
96         _init();
97     }
98 }
99
100 int MD5::to_uint32(const char data[], uint4 first, uint4 len)
101 {
102     int i = 0, cnt = 0;
103     bool tmp = 1;
104     size_byte += len;
105
106     for (i = 0; i < 16; i++)
107     {
108         bits512[i] = 0;
109     }
110
111     for (i = first; i < 64 && i < len; i++)
112     {
113         bits512[cnt] |= ((uint4)data[i] << ((i % 4) * 8));

```

```

114     if ((i + 1) % 4 == 0)
115     {
116         cnt++;
117     }
118 }
119
120 if (i == 64)
121     return 0;
122
123
124 if (i * 8 % 512 < 448 && i * 8 % 512 != 0)
125 {
126     bits512[cnt] |= 1 << (((3 - (i) % 4) * 8) - 1);
127
128     bits512[14] = (uint4)((long long)(size_byte) * 8);
129
130     bits512[15] = (uint4)((long long)(size_byte) * 8 >> 32);
131     return 1;
132 }
133 else
134 {
135     if (i * 8 % 512 != 0)
136     {
137         bits512[cnt] |= 1 << ((i % 4) * 8) - 1);
138     }
139     else
140     {
141         return 2;
142     }
143     return 1;
144 }
145 }
146
147 void MD5::encode()
148 {
149
150     for (uint4 i = 0, j = 0; j < 16; i++, j += 4) {
151         digest[j] = state[i] & 0xff;
152         digest[j + 1] = (state[i] >> 8) & 0xff;
153         digest[j + 2] = (state[i] >> 16) & 0xff;
154         digest[j + 3] = (state[i] >> 24) & 0xff;
155     }
156 }
157
158 void MD5::hmd5()
159 {
160     uint4 a = state[0];

```

```

161     uint4 b = state[1];
162     uint4 c = state[2];
163     uint4 d = state[3];
164
165     uint4 f, k;
166
167     for (int i = 0; i < 64; ++i) {
168         if (i < 16) {    //F
169             f = F(b, c, d);
170             k = i;
171         }
172         else if (i < 32) {
173             f = G(b, c, d);
174             k = (5 * i + 1) % 16;
175         }
176         else if (i < 48) {
177             f = H(b, c, d);
178             k = (3 * i + 5) % 16;
179         }
180         else if (i < 64) {
181             f = I(b, c, d);
182             k = (7 * i) % 16;
183         }
184
185         unsigned int t = b + CLS(a + f + bits512[k] + T[i], S[i]);
186         unsigned int temp = d;
187         d = c;
188         c = b;
189         b = t;
190         a = temp;
191     }
192
193     state[0] += a;
194     state[1] += b;
195     state[2] += c;
196     state[3] += d;
197
198 }
199
200 inline MD5::uint4 MD5::F(uint4 x, uint4 y, uint4 z) {
201     return (x & y) | (~x & z);
202 }
203
204 inline MD5::uint4 MD5::G(uint4 x, uint4 y, uint4 z) {
205     return (x & z) | (y & ~z);
206 }
207

```

```

208 inline MD5::uint4 MD5::H(uint4 x, uint4 y, uint4 z) {
209     return x ^ y ^ z;
210 }
211
212 inline MD5::uint4 MD5::I(uint4 x, uint4 y, uint4 z) {
213     return y ^ (x | ~z);
214 }
215
216 inline MD5::uint4 MD5::CLS(uint4 x, int n) {
217     return (x << n) | (x >> (32 - n));
218 }
219
220
221 void MD5::hexdigest(char buf[]) const
222 {
223
224     for (int i = 0; i < 16; i++)
225         sprintf(buf + i * 2, "%02x", digest[i]);
226     buf[32] = 0;
227
228 }

```

Файл server_db.cpp

```

1  #include <iostream>
2  #include <sys/ipc.h>
3  #include <sys/msg.h>
4  #include <sys/errno.h>
5  #include <unistd.h>
6  #include <netinet/in.h>
7  #include <cstring>
8  #include "structures.h"
9  #include "DBLibrary.h"
10
11 int main()
12 {
13     int msgqid, rc;
14     key_t msgkey;
15     msgkey = ftok("string_for_identical_key_queue", 'm');
16     char line[BUFSIZE];
17
18     msgqid = msgget(msgkey, IPC_CREAT | 0660);
19
20     if (msgqid < 0)

```

```

21     {
22         perror(strerror(errno));
23         printf("failed to create message queue with msgqid = %d\n", msgqid);
24         return 1;
25     }
26     else
27         cout << "The server is ready" << endl;
28
29     DBLibrary db;
30
31     while (1)
32     {
33         rc = msgrcv(msgqid, &msg, sizeof(msg) - sizeof(long), 0, 0);
34         //printf("rc: %d\n", rc);
35         if (rc < 0)
36         {
37             perror(strerror(errno));
38             printf("msgrcv failed, rc=%d\n", rc);
39             break;
40         }
41         printf("received msg: %s it's PID = %ld\n", msg.mtext, msg.mtype);
42
43         strcpy(line, msg.mtext); // TODO: получили строчку, тут, наверное, стоит вызвать функцию
44         ↪ обработку
45         cout << "line: " << line << endl;
46         strcpy(msg.mtext, db.new_command(string(line)).c_str());
47
48         rc = msgsnd(msgqid, &msg, sizeof(msg) - sizeof(long), msg.mtype); // отправили обратно
49         if (rc < 0)
50         {
51             printf("msgsnd failed, rc = %d\n", rc);
52             return 1;
53         }
54         else
55             printf("Message send done!\n");
56     }
57 }

```

Файл DBLibrary.h

```

1 #include <iostream>
2 #include <regex>
3 #include <fstream>
4 #include <string>

```

```

5
6 class DBLibrary
7 {
8 public:
9     DBLibrary() { }
10
11     string new_command(const string& command);
12
13 private:
14
15     string file_name{ "database_library.txt" };
16
17     int _find(string& to_find);
18
19     int insert(string& text);
20
21     string building_string(const string& line);
22
23     int _delete(const string& line);
24
25     int print(string&);
26 };
27
28
29 int DBLibrary::_find(string& to_find)
30 {
31     ifstream file(file_name);
32     string buf;
33     while (getline(file, buf))
34     {
35         if (buf.find(to_find) != -1)
36         {
37             smatch m;
38             if (regex_search(buf, m, regex("[0-9]{1,}")))
39             {
40                 to_find = buf;
41                 file.close();
42                 return stoi(m.str());
43             }
44         }
45     }
46     file.close();
47     return -1;
48 }
49
50
51 string DBLibrary::new_command(const string& command)

```

```

52 {
53     string buf = command;
54     regex com("[A-Z]{1,}");
55     //regex inf("\\([a-zA-Z0-9 \\.,]{1,}\\)");
56     //regex field("([a-zA-Z0-9\\.]([a-zA-Z0-9 \\.,]{1,}[a-zA-Z0-9\\.]|([a-zA-Z0-9\\.]{1,2}))");
57     smatch m;
58     if (!regex_match(buf, regex("[A-Z]{1,} *\\([a-zA-Z0-9 \\.,]{1,}\\)")))
59     {
60         return "An error has occurred in this line and it cannot be processed.\n";
61     }
62
63     if (regex_search(buf, m, com))
64     {
65         if (m.str() == "INSERT")
66         {
67             buf = m.suffix();
68
69             if (insert(buf))
70             {
71                 return "An error has occurred in this line and it cannot be processed.\n";
72             }
73
74         }
75         else if (m.str() == "PRINT")
76         {
77             buf = m.suffix();
78             if (!print(buf))
79             {
80                 return buf;
81             }
82             else
83             {
84                 return "There is no such name";
85             }
86         }
87         else if (m.str() == "DELETE")
88         {
89             buf = m.suffix();
90             _delete(buf);
91         }
92         else
93         {
94             return "An error has occurred in this line and it cannot be processed.\n";
95         }
96     }
97     return "";
98 }

```

```

99
100 int DBLibrary::print(string& to_print)
101 {
102     regex kod("[0-9]{1,}");
103     regex inf("\\([a-zA-Z0-9\\.]{1,}\\)");
104     regex field("([a-zA-Z0-9\\.][a-zA-Z0-9\\.]{1,}[a-zA-Z0-9\\.])|([a-zA-Z0-9\\.]{1,2})");
105     smatch m;
106
107     if (regex_search(to_print, m, inf))
108     {
109         to_print = m.str();
110     }
111
112     if (regex_search(to_print, m, field))
113     {
114         to_print = m.str();
115         if (_find(to_print) != -1)
116         {
117             to_print = building_string(to_print);
118             return 0;
119         }
120         else
121         {
122             return -1;
123         }
124     }
125     to_print = "An error has occurred in this line and it cannot be processed.\n";
126     return -1;
127 }
128
129 int DBLibrary::_delete(const string& line)
130 {
131     regex field("([a-zA-Z0-9\\.][a-zA-Z0-9\\.]{1,}[a-zA-Z0-9\\.])|([a-zA-Z0-9\\.]{1,2})");
132     regex inf("\\([a-zA-Z0-9\\.]{1,}\\)");
133     string name;
134     string buf_line;
135     smatch m;
136     bool flag_to_return = 0;
137
138     if (regex_search(line, m, inf))
139     {
140         name = m.str();
141     }
142
143     if (regex_search(name, m, field))
144     {
145         name = m.str();

```



```

146
147     fstream file(file_name, ios::in | ios::app);
148     ofstream tmp_file("temp_file.txt");
149
150
151     while (getline(file, buf_line))
152     {
153         if (buf_line.find(name) != -1)
154         {
155             flag_to_return = 1;
156             continue;
157         }
158         tmp_file << buf_line << endl;
159     }
160     tmp_file.close();
161     file.close();
162
163     if (remove("database_library.txt"))
164     {
165         perror(strerror(errno));
166         return 1;
167     }
168     if (rename("temp_file.txt", "database_library.txt"))
169     {
170         perror(strerror(errno));
171         return 1;
172     }
173 }
174
175 if (flag_to_return)
176     return 0;
177 else
178     return -1;
179
180 }
181
182 string DBLibrary::building_string(const string& line)
183 {
184     regex r_line_in_f("[0-9]+:[a-zA-Z0-9 .]+:[a-zA-Z0-9 .]+:[0-9]+:");
185     string res = "";
186
187     if (regex_match(line, r_line_in_f))
188     {
189         res += "code:{";
190         int i = 0;
191         while (line[i] != ':')
192             res += line[i++];

```

```

193     i++;
194     res += "} title:{";
195     while (line[i] != ':')
196         res += line[i++];
197     i++;
198     res += "} author:{";
199     while (line[i] != ':')
200         res += line[i++];
201     i++;
202     res += "} count:{";
203     while (line[i] != ':')
204         res += line[i++];
205     res += "}\n";
206     return res;
207 }
208 else
209 {
210     return "An error has occurred in this line and it cannot be processed.\n";
211 }
212 }
213
214 int DBLibrary::insert(string& text)
215 {
216     regex inf("\\([a-zA-Z0-9 \\.]{1,}\\)");
217     regex field("[a-zA-Z0-9\\.][a-zA-Z0-9 \\.]{1,}[a-zA-Z0-9\\.]|([a-zA-Z0-9\\.]{1,2})");
218     smatch m;
219
220     fstream file(file_name, ios::in | ios::app);
221
222     string buf;
223     string name;
224
225     if (regex_search(text, m, inf))
226     {
227         name = m.str();
228     }
229
230     if (regex_search(name, m, field))
231     {
232         name = m.suffix();
233     }
234
235     if (regex_search(name, m, field)) // получим имя
236     {
237         buf = m.suffix();
238         name = m.str();
239     }

```

```

240
241 string buf_line;
242
243 if (_find(name) != -1)
244 {
245     regex_search(buf, m, field);
246     buf = m.suffix();
247     regex_search(buf, m, field);
248     buf = m.str(); // ceŭvac e buf count
249
250
251     ofstream tmp_file("temp_file.txt");
252
253     while (getline(file, buf_line))
254     {
255         if (buf_line.find(name) != -1)
256         {
257             string count_str = "";
258             buf_line.pop_back();
259             int i = buf_line.size() - 1;
260             while (buf_line[i] != ':')
261             {
262                 count_str.insert(count_str.begin(), buf_line[i--]);
263                 buf_line.pop_back();
264             }
265             i = stoi(buf) + stoi(count_str);
266             buf_line += to_string(i) + ":";
267         }
268         tmp_file << buf_line << endl;
269     }
270     tmp_file.close();
271     file.close();
272     //if (rename("database_library.txt", "temp_name.txt"))
273     //{
274     // perror(strerror(errno));
275     // return 0;
276     //}
277     if (remove("database_library.txt"))
278     {
279         perror(strerror(errno));
280         return 0;
281     }
282     if (rename("temp_file.txt", "database_library.txt"))
283     {
284         perror(strerror(errno));
285         return 0;
286     }

```

```

287     }
288     else
289     {
290         buf = text;
291         for (int i = 0; i < 4; i++)
292         {
293             if (regex_search(buf, m, field))
294             {
295                 file << m.str() << ":";
296                 buf = m.suffix();
297             }
298         }
299         file << endl;
300     }
301     file.close();
302     return 0;
303 }

```

Файл structures.h

```

1  #include <iostream>
2
3  using namespace std;
4  #define BUFFSIZE 128
5
6  typedef struct
7  {
8      char data[BUFFSIZE];
9      int count_byte;
10     enum type_task
11     {
12         DATABASE,
13         MD5,
14     };
15     type_task type;
16 }MESSAGE;
17
18 typedef struct
19 {
20     char data[BUFFSIZE];
21     int count_bytes;
22 } SEM_STRUCT;
23
24 struct msg_buf

```

```
25 {  
26     long mtype;  
27     char mtext[BUFFSIZE];  
28 } msg;
```
