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 ПК»

на тему: Разработка клиент-серверного приложения
«Поиск простых чисел в определенном диапозоне»

Выполнил: студент группы М8О-211Б-19

Евкарпиев Михаил Димитриевич

(Фамилия, имя, отчество)

(подпись)

Принял: доцент кафедры 813

Эйниев Эльчин Тейюбович

(Фамилия, имя, отчество)

(подпись)

Оценка: **Дата**:

Москва, 2020

Содержание

I Теоретическая **4**

1 Системные вызовы System V для семафоров **4**

2 Межсетевое взаимодействие через механизм сокетов **10**

II Практическая **13**

3 Лабараторная 1 **13**

3.1 Задача №1 . . . 13
3.2 Задача №2 . . . 13
3.3 Задача №3 . . . 14
3.4 Задача №4 . . . 15
3.5 Задача №5 . . . 15
3.6 Задача №6 . . . 16

4 Лабораторная работа №2 **17**

4.1 Используемые системные объекты . . . 17
4.2 Задача №1 . . . 17
4.3 Задача №2 . . . 18
4.4 Задача №3 . . . 19
4.5 Задача №4 . . . 19
4.6 Задача №5 . . . 21

5 Лабораторная работа №3 **23**

5.1 Используемые системные объекты . . . 23
5.2 Задача №1 . . . 23
5.3 Задача №2 . . . 24
5.4 Задача №3 . . . 26

6 Лабораторная работа №4 **28**

6.1 Используемые системные объекты . . . 28

6.2 Задача №1 29
6.3 Задача №2 31

7 Лабораторная работа №5 **33**
7.1 Используемые системные объекты 33
7.2 Задача №1 34
7.3 Задача №2 36

8 ПО «Поиск простых чисел в определенном диапозоне» **39**
8.1 Техническое задание 39
8.2 Используемые системные объекты 39
8.3 Структура приложения 40
8.4 Основные алгоритмы 41
8.5 Руководство пользователя 43

9 Список использованных источников **45**

10 Приложение **46**
10.1 Лабораторная №1 46
10.2 Лабораторная №2 52
10.3 Лабораторная №3 61
10.4 Лабораторная №4 70
10.5 Лабораторная №5 80
10.6 ПО «Поиск простых чисел в определенном диапозоне» 87

Часть I

Теоретическая

1 Системные вызовы System V для семафоров

Семафоры — это средства синхронизации выполнения множества процессов. Семафоры группируются в наборы, каждый из которых содержит один и более семафоров. Семафоры часто используются совместно с разделяемой памятью, реализуя таким образом мощный метод межпроцессного взаимодействия.

Например, семафор может быть использован для управления доступом к устройству, такому как принтер. Семафор может гарантировать, что с принтером в данный момент работает только один процесс. Это защитит от перемешивания вывода нескольких процессов на печать. Над семафором определены три операции: добавление и вычитание целочисленных значений и ожидание, пока значение семафора не станет нулевым. При вычитании действует ограничени, что значение семафора не может стать отрицательным. Если программа попытается сделать это, она будет приостановлена.

Ниже приведен обзор системных вызовов для работы с семафорами:

- semget() Этот системный вызов получает набор из одного или более семафоров. semget() возвращает идентификатор набора семафоров. Семафор однозначно определяется этим идентификатором и начинающимся с нуля индексом в наборе. Используется для создания или получения доступа к набору из одного или нескольких семафоров. При успехе, он возвращает идентификатор набора семафоров.
  Аргументы semget():

  – key Ключ доступа к набору. Похож на имя файла. В качестве ключа мо-

жет использоваться любое целое значение. Различные пользователи набора должны договориться об уникальном значении ключа. Ключ может быть создан библиотечной функцией $ftok()$. Если необходим приватный ключ, может быть использовано значение IPC_PRIVATE.

- nsems Количество семафоров в наборе. Это значение должно быть больше или равно 1. Семафор задается идентификатором набора и индексом в этом наборе. Индекс меняется от нуля до nsems−1.
- semflg Биты прав доступа и флаги, используемые при создании набора. Девять младших битов задают права доступа для хозяина, группы и других пользователей. Для набора семафоров определены права чтения и изменения. Флаги таковы:
- IPC_CREAT Если этот флаг установлен и набор не существует, или задан ключ IPC_PRIVATE, будет создан новый набор. Если же набор с таким ключом уже существует и не задан флаг IPC_EXCL, то semget() возвратит его идентификатор.
- IPC_EXCL Этот флаг используется только вместе с IPC_CREAT. Он используется для того, чтобы создать набор только тогда, когда такого набора еще не существует. Этот флаг похож на O_EXCL при создании файлов.

Следующие системные параметры, просматриваемые prctl() ограничивают вызов semget():

- process.max-sem-nsems – максимальное количество семафоров в наборе.
- project.max-sem-ids и zone.max-sem-ids – максимальное количество наборов семафоров в проекте или зоне, соответственно.

- semctl() Этот системный вызов служит следующим целям:

- Получает значение одиночного семафора из набора или всех семафоров в наборе.

- Устанавливает значение одного или всех семафоров в наборе.
- Получает информацию о состоянии набора семафоров.
- Определяет число процессов, ожидающих, пока семафор из набора не станет нулевым.
- Определяет число процессов, ожидающих, пока семафор в наборе не увеличится по сравнению с его текущим значением.
- Определяет процесс, который выполнял последнюю операцию над семафором.
- Изменяет права доступа к набору семафоров.
- Удаляет набор семафоров. Наборы семафоров, так же как файлы и очереди сообщений, должны удаляться явным образом.

Аргументы semctl():

- semid идентификатор, полученный от semget()
- semnum индекс семафора в наборе. Первый семафор в наборе имеет индекс 0.
- cmd команда.
- arg тип этого параметра зависит от команды cmd. Это может быть:
  - Целое число, задающее новое значение семафора
  - Указатель на массив беззнаковых коротких целых, используемый для установки и получения значения всех семафоров в наборе.
  - Указатель на информационную структуру semid_ds для набора семафоров.

В прототипе semctl() последний параметр указан как union. Это означает, что тип последнего параметра зависит от значения команды cmd. Ниже показано, какие типы arg используются с различными командами:

- int val;
  SETVAL. Эта команда устанавливает значение отдельного семафора в наборе.

- struct semid_ds *buf;
  IPC_STAT Эта команда копирует состояние набора семафоров в буфер buf.
- IPC_SET. Эта команда устанавливает значения хозяина, группы и прав доступа для набора семафоров.
- ushort *array;
  GETALL. Эта команда получает значения всех семафоров в наборе и помещает их в массив, на который указывает array.
- SETALL. Устанавливает все семафоры из набора в значения, которые хранятся в массиве array.
- arg не используется
- GETVAL. Эта команда получает значение семафора с индексом semnum.
- GETPID. Эта команда получает идентификатор процесса, который совершал последнюю операцию над семафором с индексом semnum.
- GETNCNT. Эта команда получает количество процессов, ожидающих увеличения значения семафора по сравнению с текущим.
- GETZCNT. Эта команда получает количество процессов, ожидающих, пока значение семафора не станет 0.
- IPC_RMID. Эта команда удаляет набор семафоров и его идентификатор. Если какие-то процессы были заблокированы в операциях с этим набором, во всех этих процессах соответствующие вызовы semop() вернут ошибку EIDRM.

- semop(). Этот системный вызов оперирует с одним или несколькими семафорами в наборе. В действительности, это набор операций над набором семафоров. Каждая операция позволяет увеличить значение семафора на заданную величину (POST или UNLOCK), уменьшить (WAIT или LOCK), или ожидать, пока значение семафора не станет нулевым. Уменьшение значения семафора может

заблокировать процесс, если вычитаемая величина меньше его текущего значения.

Набор операций выполняется атомарно, в том смысле, что при проверке возможности всех операций никакие другие операции над семафорами набора не выполняются. Если какая-то из операций приводит к блокировке, то ни одна операция из набора не выполняется и весь набор операций блокируется. Когда какой-то другой процесс изменит семафоры, снова проверяется возможность всех операций в наборе и т. д. Это исключает возможность мёртвой блокировки, но может привести к так называемой «проблеме голодания», когда набор операций блокируется на неограниченное время, при том, что для каждой отдельно взятой операции существуют интервалы времени, в течении которого она возможна.

Системный вызов semop() производит операции над одним или более семафорами из набора. Операции увеличивают или уменьшают значение семафора на заданную величину, или ожидают, пока семафор не станет нулевым.

Аргументы этого системного вызова таковы:

- semid Идентификатор набора семафоров.
- sops. Адрес массива с элементами типа struct sembuf. Каждый элемент задает одну операцию над одним семафором. По этому адресу должно размещаться nsops таких структур. Эта структура определена следующимобразом:

```
sys/sem.h:
struct sembuf {
    ushort sem_num; /* semaphore */
    short sem_op;   /* semaphore operation */
    short sem_flg;  /* operation flags */
};
```

Поля этой структуры имеют следующий смысл:

* sem_num. Индекс семафора, над которым производится операция.
* sem_op. Если это значение положительно, оно добавляется к текущему значению семафора (POST). Если sem_op отрицательно, и его абсолютное значение больше текущего значения семафора, операция обычно блокируется. Иначе, из семафора просто вычитается абсолютное значение этого поля (WAIT). Если sem_op равен нулю, операция блокируется, пока семафор не станет нулевым.
* sem_flg. Это поле задает дополнительные флаги. Нулевое значение означает, что не задано никаких флагов. Допустимы следующие флаги, объединяемые побитовым ИЛИ:
  - IPC_NOWAIT. Если установлен этот флаг, и операция должна была бы привести к блокировке, semop() возвращает неуспех. Этот флаг может быть использован для анализа и изменения значения семафора без блокировки.
  - SEM_UNDO. Если этот флаг установлен, при завершении процесса (как по exit(), так и по сигналу), операция будет отменена. Это защищает от посмертной блокировки ресурса процессом, который ненормально завершился, не успев освободить ресурс. Использование SEM_U NDO отменяет операцию над семафором, но не обеспечивает, что ресурс, защищаемый этим семафором, остался в согласованном состоянии после завершения процесса.

Кроме того, нужно иметь в виду, что отмена операций реализована в виде счетчика, в кото¬ром накапливаются все значения, добавляемые и вычитаемые процессом при операциях над данным семафором. При завершении процесса этот счетчик просто вычитается из текущего значения семафора. Из этого описания должно быть ясно, что если вы используете флаг SEM_UNDO, его необходимо использовать при всех операциях над этим семафором. Если вы указываете SEM_UNDO только при операциях LOCK, но не при UNLOCK, счетчик отмены может просто переполниться, а его применение к семафору может привести к нежелательным результатам.

При выполнении операции над несколькими семафорами, значения семафоров не меняются, пока не окажется, что все требуемые операции могут быть успешно выполнены. При этом, поведение системы в случае, когда набор операций содержит несколько разных операций над одним и тем же семафором, не стандартизовано. Разные реализации POSIX могут интерпретировать такой набор по разному.

nsops Количество структур в массиве, на который указывает sops. nsops должен всегда быть больше или равен 1. Параметр prctl() process.max-sem-ops ограничивает максимальное количество операций в одном наборе.

2 Межсетевое взаимодействие через механизм сокетов

Базовым средством построением межсетевого взаимодействия является реализация (или поддержка) операционной системой стека(ов) протоколов. Стек протоколов – совокупность протоколов, совместно реализующих задачи по передаче и приему данных, определению и извещению об ошибках, управлению потоками данных, маршрутизации, разрешению имен и так далее. В зависимости от выбранного стека протоколов изменяется набор решающихся стеком задач, скорость передачи данных, накладные расходы и так далее. Для пользователя прикладной программы или протокола седьмого уровня семиуровневой модели OSI) важны не столько конкретный стек протоколов сам по себе, сколько услуги, предоставляемые этим стеком и интерфейсы взаимодействия из прикладной программы со стеком. Требуемые услуги могут включать установление виртуального канала, установление соединения, обмен дейтаграммами и др.

Основным интерфейсом прямого взаимодействия прикладной программы со стеком протоколов для операционных систем Windows /Unix является интерфейс сокетов (socket). Сокетом могут называться следующие понятия: программный интерфейс доступа к стеку протоколов – интерфейс сокетов, более узко можно рассматривать сокет

как комбинацию IP адреса и порта получателя и IP адреса и порта отправителя. Или сокет – наименование некоторого объекта, осуществляющего сетевое взаимодействие. Прикладная программа вызывает функции ОС для управления этим объектом и для получения (отправки) данных через него. Существует несколько базовых функций для работы с сокетами:

- socket – выделяет память под объект сокета, определяет тип обмена данными (с установлением соединения или без), тип сокета (для взаимодействия через Интернет или NFS), стек протоколов, и возвращает его дескриптор, который используется потом для идентификации именно этого сокета.
- bind – «привязывает» сокет к определенному адресу локального компьютера (у компьютера может быть более одного адреса).
- listen – функция вызывается серверным процессом для начала ожидания соединения по данному сокету
- connect – функция, пытающая установить соединение с указанным адресом серверного сокета (выполняется на клиенте). Серверный сокет должен к этому времени находиться в режиме «прослушивания» – то есть для него должна быть вызвана функция listen.
- accept – «принимает» соединение от клиента серверным процессом. Возвращаемое значение является дескриптором нового сокета, по которому сервер будет обращаться именно к этому соединению. По старому сокету сервер будет ожидать запросов на соединение от новых клиентов. Таким образом сервер обслуживает нескольких клиентов. В ОС UNIX обычно вместе с этой функцией еще вызывают функцию для создания нового процесса – что приводит к распараллеливанию обработки запросов от клиентов.
- send / recv – отправка и прием данных.
- shatdown – прекращает прием / отправку сообщений для данного сокета и закрывает соединение для данного сокета (объект сокета остается в памяти)
- close – освобождает память, занятую сокетом и освобождает дискриптор.

Процесс последовательного вызова ф-ий при работе с сокетом (с установлением соединения) представлен ниже.

Для сервера: socket - bind - listen - accept - (send recv) - shutdown - close.

Для клиента: socket - connect - (send | recv) - shutdown - close

Часть II

Практическая

3 Лабараторная 1

3.1 Задача №1

3.1.1 Техническое задание

Для 2-х целых чисел вычислить наименьшее общее кратное и наибольший общий делитель по алгоритму Евклида.

3.1.2 Описание основного алгоритма

По правилам считаем НОД и м помощью НОДа ищем НОК по формуле. Реализация алгоритма представлена в подразделе 10.1.1.

3.1.3 Руководство пользователя

Enter 2 numbers: 11 20
NOD: 1
NOK:220

3.2 Задача №2

3.2.1 Техническое задание

Проверить правильность расстановки в тексте круглых квадратных и фигурных скобок.

3.2.2 Описание основного алгоритма

10.1.2

Используя 3 переменные, отвечающие каждая за свой вид скобок, прибавляя или отнимая единицу при встрече соответствующей скобки, проверяем вложенность скобок и их количество (нехватает или есть лишняя). Функция вернет сумму 3-х переменных, и если их сумма равна нулю, то все верно.

3.2.3 Руководство пользователя

{[]({}}
Correct

{[(])}
Uncorrect

3.3 Задача №3

3.3.1 Техническое задание

Для заданного целого n вычислить сумму $1^1 + 2^2 + 3^3 + ... + n^n$. Динамически определить при каком n произойдет переполнение для типов short, unsigned short, int, unsigned int, long long.

3.3.2 Описание основного алгоритма

10.1.3

Считаем сумму, сохраняя предыдущую сумму в отдельной переменной, если текущая сумма вдруг становится меньше предыдущей, то посчитанное число не помещается в текущий тип.

3.3.3 Руководство пользователя

Enter number:15
Overflow Char – V
Overflow Short Int – 3414
Overflow Int – 405071318

Your result Long Long: 11424093749340454

3.4 Задача №4

3.4.1 Техническое задание

Составить программу возведения в 3 степень (куб числа), используя закономерность:

$$13 = 1; 23 = 3 + 5 = 8; 33 = 7 + 9 + 11 = 27; 43 = 13 + 15 + 17 + 19 = 64, ...$$

3.4.2 Описание основного алгоритма

10.1.4

Используя данную закономерность, просто считаем куб числа.

3.4.3 Руководство пользователя

12

133+135+137+139+141+143+145+147+149+151+153+155=1728

3.5 Задача №5

3.5.1 Техническое задание

Числа Фибоначчи определяются соотношениями: $f_0 = f_1 = 1; f_n = f_{n_1} - 1 + f_{n_2} - 2$. Составить программу поиска первого числа Фибоначчи (и его номера), большего заданного m.

3.5.2 Описание основного алгоритма

10.1.5

Считаем числа Фибоначчи в цикле, и сравниваем получившееся числа с текущим. Как только получившиеся число становиться больше введенного пользователем, выходим из цикла и выводим получившееся число Фибоначчи.

3.5.3 Руководство пользователя

12
fibo number: 13
serial number: 8

3.6 Задача №6

3.6.1 Техническое задание

Составить программу, определяющую можно ли заданное натуральное число представить в виде суммы квадратов двух натуральных чисел. Если это возможно, то вывести эти квадраты (все возможные).

3.6.2 Описание основного алгоритма

10.1.6

В цикле проходим от нуля до корня из введенного числа с первым числом, и аналогично со вторым. Если сумма квадратов текущих чисел равны корню, то мы нашли числа для суммы квадратов и выводим их. После нахождения первой пары не останавливаемся и ищем остальные пары, или убеждаемся в их отсутсвии.

3.6.3 Руководство пользователя

Enter the number: 45
$3^2 + 6^2$

4 Лабораторная работа №2

4.1 Используемые системные объекты

Библиотека <dirent.h> – отвечает за работу с файловой системой, перемещение по папкам, удаление, добавление, изменение папок и файлов, а так же их атрибутов. В этой библиотеке лежит структура struct dirent которая помогает вывести нужные атрибуты файлов.

4.2 Задача №1

4.2.1 Техническое задание

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

4.2.2 Описание основного алгоритма

10.2.1

Используем названную выше библиотеку для открытия папки по задананному пользователем пути, после поочередно проходимся по всем файлам функцией readdir и получаем имя каждого файла, после чего выводим его на экран.

4.2.3 Руководство пользователя

.
task2.cpp
out.1.txt
task1.cpp
task5.cpp
..
task3.cpp
Hello
task4.cpp
out.txt

a.out
.
4.txt

4.3 Задача №2

4.3.1 Техническое задание

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

4.3.2 Описание основного алгоритма

10.2.2

Повторяем действия из предыдущей задачи, однако теперь при проходе по папке смотрим не только на имя но и на тип файла и если тип файла не DIR то мы выводим имя файла на экран.

4.3.3 Руководство пользователя

.
task2.cpp
out.1.txt
task1.cpp
task5.cpp
task3.cpp
task4.cpp
out.txt
a.out
4.txt

4.4 Задача №3

4.4.1 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.

4.4.2 Описание основного алгоритма

10.2.3

Открываем папку, проверяем файл на существование, если он существует, генерируем имя для жесткой ссылки (исходя из количества жестких ссылок) и расширения файла. После функцией link из библиотеки <unistd.h> создаём жесткую ссылку на этот файл используя сгенерированное имя. С помощью системной команды ls убедимся в том что ссылка создалась.

4.4.3 Руководство пользователя

Enter puth to file: out.txt
The file already existed.
File will be cleared and opened for writing. A new hard link will be created
To stop enter: stop
Enter text:
Hellow world!
stop
mikhail@mikhail-computer: /Institute/3-dsemester/OSandAK/lab2$ ls
4.txt Hello out.txt task2.cpp task4.cpp
a.out out.0.txt task1.cpp task3.cpp task5.cpp

4.5 Задача №4

4.5.1 Техническое задание

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

4.5.2 Описание основного алгоритма

10.2.4

Пользователь вводит имя файла (или полный путь к нему, если файл лежит не в текущей директории) и программа создаёт символическую ссылку на него. Использую структуру stat из библиотеки <unistd.h> получаем данные о ссылке, И выводим эти данные.

4.5.3 Руководство пользователя

Enter file puth: /home/mikhail/Institute/3-dsemester/OSandAK/lab2/out.0.txt
Enter link directory: /home/mikhail/Institute/3-dsemester/OSandAK/lab2

Устройство: 66312,
inode: 263625,
режим доступа: 16893,
количество жестких
ссылок: 3,
идентификатор пользователя-владельца: 1000,
идентификатор
группы-владельца: 1000,
тип устройства: 0,
общий размер в байтах: 4096, размер
блока ввода-вывода в файловой системе: 4096,
количество выделенных
блоков: 8
время последнего доступа: 1610111956
время последней модификации: 1610112040
время последнего изменения: 1610112040

4.6 Задача №5

4.6.1 Техническое задание

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

4.6.2 Описание основного алгоритма

10.6

Заходим в папку и выводим все файлы лежащие в ней, попутно занося имена файлов в массив, и их время последнего изменеия в другой массив на соответствующую позицу. Далее в цикле сортируем массивы по времени последнего изменения, а потом просто выводим каждый элемент уже отсортированного массива, и время.

4.6.3 Руководство пользователя

.
task2.cpp 1603107015
task1.cpp 1610111483
task5.cpp 1609598729
.. 1610108571
out.0.txt 1610111951
task3.cpp 1603132026
Hello 1610111445
task4.cpp 1609608546
out.txt 1610111951
a.out 1610112415
. 1610112415
4.txt 1603113526
Sorted:
task2.cpp 1603107015
4.txt 1603113526
task3.cpp 1603132026
task5.cpp 1609598729
task4.cpp 1609608546

.. 1610108571
Hello 1610111445
task1.cpp 1610111483
out.0.txt 1610111951
out.txt 1610111951
a.out 1610112415
. 1610112415

5 Лабораторная работа №3

5.1 Используемые системные объекты

- Из библиотеки <unistd.h> используются функции связанные с порождением процессов. Выводом PID (Personal ID) процессов, а так же ожидания родительского процесса окончания работы дочернего.
- Так же используется функция sleep() дабы избежать потерь данных при выводе и посчете количества потомков.
- Для получения PID процесса нужно использовать функцию getpid().
- Для создания процесса используется функция fork().
- Для ожидания завершения дочернего процесса нужно использовать функции wait()и waitpid().

5.2 Задача №1

5.2.1 Техническое задание

Породить количество процессов, заданное аргументом командной строки. Если аргумент не указан, то сообщить об этом. Каждый порождённый процесс печатает ФИО автора программы пишет свой PID.

5.2.2 Описание основного алгоритма

10.3.1

Считываем аргумент командной строки, если его нет, выводим ошибку. В цикле порождаем дочерние процессы (цикл от 0 до n, где n – число из аргумента командной строки), каждый процесс по тому, какое значение возвращает fork определяет, родителем или потомком он является или fork вернул ошибку, и все дети пишут свой PID и ФИО автора, а после завершаются, если родитель то ждем пока ребенок не завершится, либо выводим сообщение об ошибке fork.

5.2.3 Руководство пользователя

./a.out 5
PARENT: PID – 18532
CHILD[18533]: Mikhail Evkarpiev
CHILD[18534]: Mikhail Evkarpiev
CHILD[18535]: Mikhail Evkarpiev
CHILD[18536]: Mikhail Evkarpiev
CHILD[18537]: Mikhail Evkarpiev

5.3 Задача №2

5.3.1 Техническое задание

Программа должна принимать имя выходного файла. Если оно не указано, то весь вывод осуществляется на экран.Требуется создать иерархию процессов: Первый процесс создаёт 5 процессов. Каждый из следующих создаёт 4 процесса. Следующее поколении порождает 3 процесса. И так далее. Каждый процесс после порождения других процессов печатает номер PID и PID дочерних процессов в виде: PID: <1 child PID> <2 child PID> <3 child PID> <4 child PID> <5 child PID>. Последние процессы в цепочке ждут 10 сек и после этого завершаются. Предпоследний процесс ждёт на секунду больше. Остальные аналогично. Основной процесс после запуска всех дочерних ждёт пару секунд и выводит дерево процессов(вызовом системной функции «pstree-p»). Завершиться процесс должен также через секунду после дочерних, то есть последним. Все запущенные процессы должны быть пересчитаны и перед завершением основной процесс должен написать приветствие в виде ФИО автора программы количества всех порождённых процессов.

5.3.2 Описание основного алгоритма

10.3.2

Разбираем аргументы командной строки и устанавливаем флаг, в зависимости от которого будет осуществляться вывод на консоль или в файл. Создаём массив на 5 элементов (у каждого процесса тужа запишутся в нулевой элемент – собственный PID, в остальные – PIDы детей, пятерых или меньше). Создаём tmp.txt файл для подсчёта

количества процессов, каждый процесс запишет туда свой PID начиная с новой строки, затем пройдя по файлу и подсчитав количество строк мы выясним и количество запущеных процессов.

Далее циклами порождаем дерево процессов. Каждый заполняет массив своим PID и PID детей, записывает их в файл/консоль, потом ждёт 10 или больше (зависит от того, как близко к корню дерева текущий процесс). После чего завершается. Родительский процесс печатает в файл дерево при помощи system «pstree -p > file.txt». Далее родительский процесс считает общее число потомков, а потом выводит автора и количество созданных процессов.

5.3.3 Руководство пользователя

Возможные варианты ввода: ./a.out files.txt
./a.out
Вывод:
I am parent of all PIDs - 18580
<Parent 18604>:
<Parent 18639>: <2 child 18591> <3 child 18597>
<Parent 18633>: <3 child 18597>
<Parent 18647>:
...
<Parent 18582>: <1 child 18585>: <2 child 18591>: <3 child 18597>: <4 child 18602>:
<Parent 18583>: <1 child 18589>: <2 child 18596>: <3 child 18601>: <4 child 18606>:
<Parent 18590>: <1 child 18607>: <2 child 18613>: <3 child 18623>: <4 child 18629>:
<Parent 18586>: <1 child 18614>: <2 child 18619>: <3 child 18627>: <4 child 18634>:
Autor - Mikhail Evkarpiev. 325 processes have been created!
<Parent 18580> <1 child 18581> <2 child 18582> <3 child 18583> <4 child 18586> <5 child 18590>

5.4 Задача №3

5.4.1 Техническое задание

- В аргументах командной строки задаётся количество порождаемых процессов (например -n 10) и имя выходного файла (например -o outfile). Если не указан ключ «-n», то по умолчанию нужно породить 1 процесс. Если не указан ключ «-o»,то вывод осуществляется на экран.

- Сначала порождается указанное количество процессов, каждый из которых печатает информацию в свой файл «outfile.PID», где PID – идентификатор порождённого процесса, «outfile» – файл из параметров командной строки («out.pid», если не указан ключ «-o»).

- Каждый процесс записывает в файл ФИО автора и факториал от (PPID–PID). Основной процесс ждёт завершение всех процессов, склеивает их в выходной файл и удаляет все промежуточные, если в аргументе командной строки указан ключ «-c».

5.4.2 Описание основного алгоритма

10.3.3

Считываем аргументы командной строки и проверяем на соответствие ТЗ (проверяем корректность введенных ключей). Дальше в зависимости от текущего набора флагов создаём 1 или n процессов, и помещаем информацию о них в файл или выводим на консоль (зависит от ключа -o) а так же удаляем/не удаляем tmp-файлы, порождённые детьми (зависит от наличия ключа -c). Основной процесс ждет завершения всех дочерних и получает информацию о них из их файлов и выводит в один общий выходной файл.

5.4.3 Руководство пользователя

Ввод: ./a.out -n 3 -o test -c
18964
18965
18966
Содержимое файла test.txt:

test.18965 Mikhail (PPid: 18963 - Pid: 18965) = 2
test.18964 Mikhail (PPid: 18963 - Pid: 18964) = 1
test.18966 Mikhail (PPid: 18963 - Pid: 18966) = 6
Ввод:
./a.out -n 5
Mikhail (PPid: 18982 - Pid: 18983)! = 1
Mikhail (PPid: 18982 - Pid: 18984)! = 2
Mikhail (PPid: 18982 - Pid: 18986)! = 24
Mikhail (PPid: 18982 - Pid: 18987)! = 120

6 Лабораторная работа №4

6.1 Используемые системные объекты

Используемые библиотеки: <sys/types.h>, <sys/ipc.h>, <sys/sem.h>. Семафоры – регулировщики доступа. могут принимать целые значения от нуля и больше. Процессы могут использовать сразу несколько семафоров и работать только когда значение семафора совпадает с заданными программно значениями которые «ожидает» программа.

- semget() – API «открывает» набор семафоров, идентификатор которого задан значением аргумента key, и возвращает неотрицательный целочисленный дескриптор.
- semop() – API которым можно изменять значение одного или нескольких семафоров в наборе (указанном аргументом semfd) и/или проверять равенство их значений нулю.
- Структура sembuf состоит из sem_num, sem_op, sem_flg – первое, номер семафора из набора, второе, операция над этим семафором, третье, флаги операций.
- semctl() – этим API можно запрашивать и изменять управляющие параметры набора семафоров, указанного аргументом semfd, а также удалять семафор.
- Разделяемая память – разделяемая память позволяет множеству процессов отображать часть своих виртуальных адресов в общую область памяти. Благодаря этому любой процесс может записывать данные в разделяемую область памяти, и эти данные будут доступны для чтения и модификации другим процессам.
- shm_perm – Данные, хранящиеся в записи $structipc\_perm$.
- shm_segsz – Размер разделяемой области памяти в байтах.
- shm_lpid – Идентификатор процесса, который в последний раз подсоединялся к области.
- shm_cpid – Идентификатор процесса-создателя.

- *shm_nattch* – Число процессов, подсоединенных к области в данный момент.
- *shm_atime* – Время, когда процесс в последний раз подсоединялся к области.
- shmget() – Создание и окрытие разделяемой области памяти.
- shmat() – Подсоединение разделяемой области памяти к виртуальному адресному пространству процесса с тем, чтобы этот процесс мог читать данные из разделяемой памяти и/или записышать в нее данные.
- shmdt() – Отсоединение разделяемой области памяти от виртуального адресного пространства процесса.
- shmctl() – Запрос и изменение управляющих параметров разделяемой области памяти, а также ее удаление.
- time() – возврящает текцщее время.
- rand() – возврящает случайное число.

6.2 Задача №1

6.2.1 Техническое задание

Реализовать задачу читателей и писателей через семафоры. Суть задачи состоит в следующем:

- Существует 20 процессов: 10 читателей и 10 писателей;
- Писатели периодически пишут в файл числа от 0 до 999;
- Читатели читают файл и считают количество цифр в файле: каждый из читателей подсчитывает свою цифру;
- Спустя 30 секунд все процессы завершаются, перед этим написав свой PID, роль и количество цифр (для читателей) и сгенерированных чисел (для писателей);
- Попытка чтения или записи происходит через случайный интервал времени от 10 до 20 миллисекунд.

6.2.2 Описание основного алгоритма

10.4.1

Создаём при помощи функции fork 10 читателей и 10 писателей. Если fork вернул 0 то процесс определяет кто он и дальше действует уже исходя из этого. Главный процесс ждёт 33 секунд и удаляет семафор. Пока время начала работы минус текущее время меньше 30 секунд. Писатели проверяют семафор и если они могут записать число, они открывают файл и делают это, а потом увеличивают семафор на единицу. Читатели если могут отнять единицу, то открывают файлы и считают все числа, совпадающие с их порядковым номером, а потом отнимаю единицу от значения семафора. По истечении 30 секунд и читатели и писатели пишут кем они являлись, и сколько чисел записали/прочитали.

6.2.3 Руководство пользователя

./a.out test.txt

I am Writer, my PID = 19110, count = 1670
I am Writer, my PID = 19108, count = 1659
I am Writer, my PID = 19124, count = 1657
I am Writer, my PID = 19114, count = 1664
I am Writer, my PID = 19120, count = 1663
I am Writer, my PID = 19105, count = 1705
I am Writer, my PID = 19116, count = 1647
I am Reader, my PID = 19121, the number that was counted 8, count = 4120147
I am Reader, my PID = 19111, the number that was counted 3, count = 3847757
I am Reader, my PID = 19109, the number that was counted 2, count = 4195225
I am Reader, my PID = 19104, the number that was counted 0, count = 2490550
I am Reader, my PID = 19107, the number that was counted 1, count = 3811369
I am Writer, my PID = 19112, count = 1660
I am Reader, my PID = 19113, the number that was counted 4, count = 3933991
I am Writer, my PID = 19122, count = 1685
I am Reader, my PID = 19119, the number that was counted 7, count = 3857006
I am Reader, my PID = 19117, the number that was counted 6, count = 3942202

I am Writer, my PID = 19118, count = 1680
I am Reader, my PID = 19115, the number that was counted 5, count = 3933519
I am Reader, my PID = 19123, the number that was counted 9, count = 3752825

6.3 Задача №2

6.3.1 Техническое задание

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

- Сервер принимает от клиента запросы в виде примера из двух вещественных чисел и бинарной операции между ними;
- Сервер возвращает ответ;
- Клиентов может быть много;
- Запросы клиенты отправляют в интерактивном режиме;
- Клиент завершается, если пользователь напечатает пустую строку.

6.3.2 Описание основного алгоритма

10.4.2

Сервер:

Создаёт область разделяемой памяти с указанным ключом (ключи у сервера и клиента должны совпадать) размера double + char + double. А так же открывает набор семафоров для управления разделяемой памятью. Далее сервер блокирует себя семафором и ждёт «хода» клиента. Когда клиент разблокировал сервер, последний разблокирует разделяемую память и считывает оттуда операнды и операцию и считает значение выражения. После он помещает ответ в разделяемую память начиная сначала. И блокируется до следующего выражения (пока его семафор опять не разблокируют).

Клиент:

Запрашивает у пользователя два операнда и операцию. После получает доступ к разделяемой памяти. И записывает туда double + char + double, закрывает разделяемую

память, а после переключает семафоры для разблокировки сервера и собственной блокировки. Как только блокировка с клиента сервером снята, клиент разблокирует для чтения разделяемую память и считывает первый double, в который и сервер положил результат операции. После память обнуляется и пользователю предлагают ввести ещё одно выражение. Так продолжается до ввода пустой строки.

6.3.3 Руководство пользователя

```
    ./client
Enter expression(A <op> B):
Enter A: 1
Enter <op>: +
Enter B: 100
Result = 101
Enter expression(A <op> B):
Enter A: 1
Enter <op>: -
Enter B: 100
Result = -99
Enter expression(A <op> B):
Enter A: 1
Enter <op>: *
Enter B: 99
Result = 99
Enter expression(A <op> B):
Enter A: 50
Enter <op>: /
Enter B: 25
Result = 2
Enter expression(A <op> B):
Enter A:
Enter <op>:
Enter B:
Input string is empty. Completing programm...
```

7 Лабораторная работа №5

7.1 Используемые системные объекты

Системный вызов pipe содержащийся в библиотеке <unistd.h> создает коммуникационный канал между двумя взаимосвязанными процессами (например, между родительским и порожденным процессами или между двумя процессами-братьями, порожденными одним родительским процессом). В частности, эта функция создает файл канала, который служит в качестве временного буфера и используется для того, чтобы вызывающий процесс мог записывать и считывать данные другого процесса.

Реализация очередей сообщений в UNIX System V.3 и V.4 аналогична реализации UNIX-файлов.

- msgget() – API открывает очередь сообщений, идентификатор которой совпадает со значением аргумента key, и возвращает положительный целочисленный дескриптор.

  Флаги:

  - IPC_PRIVATE – создаёт приватную очередь сообщений/семафор/сокет и т. д. Другие процессы не смогуть получить к ним доступ.
  - IPC_CREAT – создаёт новую очередь к которой в теории могут подключится и сторонние процессы.
  - IPC_EXCL – используется в связки с предыдущим для гарантированного создания новой очереди/сокета и пр. И исключает открытие уже существующих.

- msgsnd() – передает сообщение (на которое указывает msgPtr) в очередь, обозначенную дескриптором msgfd. Структура сообщения обычно содержит: long mtype – тип сообщения, char mtext – текст сообщения. Помимо mtype содержание элементов сообщения может быть произвольным (то есть при необходимости туда можно положить и несколько переменных и указатель и пр.).

- msgrcv() – принимает сообщение типа mtype из очереди сообщений, обозначенной дескриптором msgfd. Полученное сообщение хранится в объекте, на который указывает аргумент msgPtr. Значение аргумента len – это максимальный размер (в байтах) текста сообщения, которое может быть принято данным вызовом. mtype используется для идентификации приёма сообщения. Функция одним из аргументов получает целое число которое значит следующее:
  - 0 – принять из очереди самое старое сообщение любого типа.
  - Целое положительное число – принять самое старое сообщение указанного типа.
  - Целое отрицательное число – принять сообщение, тип которого меньше абсолютно.
  - Значение mtype или равен ему. Если таких сообщений в очереди несколько, принять то, которое является самым старым имеет наименьшее значение типа.
- msgctl() – помощью этого API можно запрашивать управляющие параметры очереди сообщений, обозначенной аргументом msgfd, изменять информацию в управляющих параметрах очереди, удалять очередь из системы.

  Варианты значения флага cmd:
  - IPC_STAT – копировать управляющие параметры очереди в объект, указанный аргументом mbuf Ptr.
  - IPC_SET – заменить управляющие параметры очереди параметрами, содержащимися в объекте, на который указывает аргумент mbuf Ptr.
  - IPC_RMID – удалить очередь из системы.

7.2 Задача №1

7.2.1 Техническое задание

Реализовать с помощью pipe параллельные вычисления числа .

- Через аргументы командной строки задаётся количество процессов, которые будут применяться для расчётов, и количество разбиений интервала [0; 1).

- Каждый процесс считает число в части интервала от 0 до 1, согласно своему номеру. (Например для 4 процессов первый имеет интервал от [0; 0,25), второй -– [0,25; 0,5), третий -– [0,5; 0,75), четвёртый – [0,75; 1).
- Для расчёта применяется формула численного интегрирования для подсчёта интеграла от $4/(1 + x_2)$. (Можно использовать формулу прямоугольников).
- Каждый дочерний процесс через pipe посылает подсчитанное число родителю. Он всё суммирует.
- Родитель печатает результат и показывает известное число π.

7.2.2 Описание основного алгоритма

10.5.1

Число можно вычислить при помощи интеграла $\int_0^1 \frac{4}{1+x^2}dx$. Считываем с командной строки количество процессов которые будут заниматься вычислениями, а так же количество разбиений. Так же в отдельную функцию мы выносим вычисление интеграла методом прямоугольников. Далее создаём pipe и порождаем дочерние процессы по количеству первого аргумента командной строки.

Если процесс является дочерним, он заходит в соответствующее условие (определяем по тому, что вернуло API fork). Дочерний процесс получил от родительского количество разбиений для своего отрезка и текущее значение i (порядковый номер отрезка). Тогда он при помощи формулы $[\frac{i}{parts}, \frac{i+1}{parts})$, где parts – количество процессов, находит границы своего отрезка для интегрирования и отправляет его в функцию. После того как функция вернула значение интеграла на заданном отрезке, дочерний процесс закрывает pipe на чтение и открывает на запись, записывает получившееся значение и закрывает pipe уже для записи, а потом завершается. Этот алгоритм алгоритм выполняют все дочерние процессы. Родительский процесс после порождения всех дочерних открывает pipe на чтение и считывает решения интеграла всех детей на указанных отрезках. Далее он суммирует все присланные значения и выводит ответ на консоль.

7.2.3 Руководство пользователя

./a.out 4 1000
3.1415926431731246993 ./a.out 6 200
3.1415925378490516806

7.3 Задача №2

7.3.1 Техническое задание

С помощью очереди сообщений организовать клиент-серверную модель для решения кубического уравнения.

- Пользователь в клиенте задаёт 4 коэффициента и посылает их серверу в одном сообщении.
- Сервер принимает сообщение и посылает три корня клиенту.
- Клиентов может быть много. Каждый из них имеет свой тип сообщений, для определения, куда посылать ответ.

7.3.2 Описание основного алгоритма

10.5.2

Сервер:

В коде сервера реализована функция поиска корней кубического уравнения при помощи метода Кардано. При старте сервера он пишет в консоль свой PID и создаёт очередь сообщений. После он находится в бесконечном цикле (может быть прерван только пользователем), в котором проверяет, пришло ли что-то на его mtype. Если сообщение на сервер пришло, то сервер вынимает из него нужные данные (элементы массива, которые являются коэффициентами кубического уравнения) и отправляет их в функцию вычисления корней по Кардано. Получив искомые корни, сервер помещает их в массив из трёх элементов (для ответа). После он определяет mtype – элемент переменной prodpid где указан адрес процесса который послал запрос. Когда сообщение сформировано, сервер его отправляет и возвращается к своему циклу проверки приходящих сообщений.

Клиент:
Клиентская программа запускается, печатает свой PID. Затем запрашивает у пользователя четыре коэффициента кубического уравнения и помещает их в массив отправляемого сообщения. После клиент присваивает свой PID «обратному адресу» (чтобы сервер знал, куда отсылать ответ). А также присваивает mtype ключ сервера. После этого вычисляется длинна сообщения, и оно отправляется в очередь. Процесс клиент считывает из очереди сообщений все сообщения mtype равным собственному PID. Клиент извлекает данные о корнях уравнения из массива и выводит их на консоль, после чего завершается.

7.3.3 Руководство пользователя

./server
Server: my pid is 19230 ./client
Client: my pid is 19242
Enter ratio of cubic equation:
Enter koef 'a': 1
Enter koef 'b': 2
Enter koef 'c': 3
Enter koef 'd': 4
$x_0 = -1.6506291914$
$x_1 = -0.1746854043 + i * 1.546868887231396$
$x_2 = -0.1746854043 + i * -1.546868887231396$

./client
Client: my pid is 19249
Enter ratio of cubic equation:
Enter koef 'a': 11
Enter koef 'b': 2
Enter koef 'c': -7
Enter koef 'd': 3
$x_0 = -1.0428303673$
$x_1 = 0.4305060928 + i * 0.276026305694758$

$x_2 = 0.4305060928 + i * -0.276026305694758$

8 ПО «Поиск простых чисел в определенном диапозоне»

8.1 Техническое задание

На вход подается два числа левая и правая граница интервала в котором будет осуществляться поиск простых чисел, либо – что означает завершение клиента.

8.2 Используемые системные объекты

В системе BSD UNIX 4.2 были впервые введены гнезда (sockets), которые предоставляют независимые от протокола услуги по организации сетевого интерфейса и способны обеспечить работоспособность методов IPC в масштабах локальной сети. В частности, гнезда могут работать как с протоколом TCP (Transmission Control Protocol), так и с протоколом UDP (User Datagram Protocol). Обращаться к гнезду можно по IP-адресу машины и номеру порта. Заданный таким образом адрес уникален в масштабах всей Internet, так как для каждой машины комбинация адреса и номера порта уникальна. Следовательно, два процесса, выполняемых на отдельных машинах, могут взаимодействовать друг с другом через гнезда.

Различают гнезда с установлением соединения (то есть адреса гнезд отправителя и получателя выясняются заранее, до передачи сообщений между ними) и без установления соединения (адреса гнезд отправителя и получателя передаются с каждым сообщением, посылаемым из одного процесса в другой).

Гнездо каждого типа поддерживает один или несколько транспортных протоколов, однако в любой UNIX-системе для любого гнезда всегда указывается протокол по умолчанию. Протокол по умолчанию для виртуального канала – TCP, а для дейтаграммы – UDP.

- socket() – функция создает для указанного пользователем домена гнездо заданного типа и с указанным протоколом. Аргумент domain определяет правила

именования гнезда и формат адреса, используемые в протоколе.

- bind() – функция присваивает гезду имя. Гнездо обозначается аргументом sid, значение которого, возвращенное функцией socket, представляет собой дескрипор гнезда. Аргумент addr_p указывает на структуру, содержащую имя, которое должно быть присвоено гнезду.

- listen() – вызов функции listen переводит сокет из состояния CLOSED в состояние LISTEN.

- connect() – Эта функция вызывается в клиентском процессе для установления соединения с серверным гнездом.

- accept() – Эта функция вызьвается в серверном процессе для установления соединения с клиентским гнездом (которое делает запрос на установление соединения посредством вызова функции connect)

- send() or sendto() – Эта функция передает содержащееся в аргументе buf сообщение длиной, len байтов в гнездо, заданное аргументом sid и соединенное с данным гнездом. Sendto отличается возможностью указать гнездо получателя.

- recv() or recvfrom() – Эта функция принимает сообщение через гнездо, указанное в аргументе sid. Принятое сообщение копируется в буфер buf, а максимальный размер buf задается аргументом len. Recvfrom отличается возможностью узнать «адрес» отправителя.

- shutdown() – Данная функция закрывает соединение между серверным и клиентским гнездами.

8.3 Структура приложения

Функции которые содержит сервер:

- int main(): обмен информацией с клиентом.

- bool $is\_simple(intx)$: проверка принятого числа на простоту.

- int search(int A, int B, vector <int> ivector): поиск всех простых чисел в заданном диапазоне.

Функции которые содержит клиент:

- int main(): запрашивает у пользователя данные и отсылает серверу, ждет ответа от сервера и выводит его.

Логика работы:

Сервер:
Запускается первым. Сервер создаёт сокет и в бесконечном цикле ждет подключения клиента по виртуальному каналу. Как только соединение установлено сервер посылает сообщение клиенту, что сервер установил подключение. Далее сервер ожидает два сообщения от клиента с введенными пользователем границами или завершающего символа. Левая и правая границы помещаются в соответствующие переменные. После чего попадаем в цикл с условием: пока сообщение не содержит завершающий символ. В этом цикле происходит поиск простых чисел, проверка корректности введенных границ и посылается ответ клиенту. Так до тех пор пока клиент не отключится. Когда клиент отключается ожидаем следующего подключения.

Клиент:
Клиент имеет смысл запускать только после старта сервера. Клиент создаёт сокет и пытается подключится к серверу, когда ему это удаётся, выводит сообщение об этом. Далее попадаем в цикл с улсовем: пока не введен завершающий символ. В цикле просит пользователя ввести границы интервала либо завершающий мисвол. После чего посылает введенные границы серверу, и ожидает ответа. Выводит полученное сообщение от сервера. Так до тех пор пока не введен завершающий символ.

8.4 Основные алгоритмы

10.6

Алгоритм нахождения простых чисел:

- проверка корректности введенных границ.

- если правая граница оказалась меньше 2, то простых чисел нет.
- в цикле проверяем все числа из интервала, путем нахождения хотя бы одного делителя большего 2, но меньше или равного $\sqrt{x}$.

Сервер:
bool $is\_simple(intx)$:

- A – левая граница интервала.
- B – правая граница интервала.
- ivector – вектор в который записываются простые числа.

int search(int A, int B, vector <int> ivector):

- x – проверяемое число.

int main() :

- server – дескриптор через который происходит общение клиента с сервером.
- client – номер сокет-дескриптора.
- bufsize – начальный размер буфера, в который записываеются сообщения.
- portNum – Номер порта.
- counter – счетчик для отслеживания кол-ва отправленных сообщений.
- left_border – левая граница интервала.
- right_border – правая граница интервала.
- res – переменная в которую помещается результат работы функции search().
- isExit – флаг для выхода из цикла.
- buffer – динамический массив, в который записываются сообщения для общения клиетна с сервером.

- str, strbuf1, strbuf2 - вспомогательные строки, для записи сообщения в buffer.
- vec – вектор с простыми числами из интервала.

Создаётся сокет и ему присваивается адрес и порт, затем сервер ожидает запроса на соединение от клиентов в бесконечном цикле, после соединения отправляет сообщение клиенту об успехе соединения. Сервер ожидает два сообщения от клиента, после их получения входит в цикл с условием: пока клиент не получили сообщение с завершабщим символом от клиента. В цикле вызываем функцию для поиска простых чисел в заданном интервале $is\_simple(intx)$, по описанному выше алгоритму. Если функция вернула не -1 и не 0, то в заданном интервале есть простые числа, которые в данныей момент записаны в векторе vec. Определяем размер массива в котором должен поместиться ответ клиенту. Делаем это с помощью вычитания левой границы из правой и сравниваем результат с bufsize, если он больше, то необходмо выделить больше памяти для массива buffer, иначе ответ может не поместиться полностью. Далее числа из вектора записываем в массив. Сервер отправляет сообщение с ответом клиенту. И ждем новых сообщений от клиента. Так пока клиент не пришлет завершающий символ, после чего сервер опять будет ожидать запроса на соединение

Клиент:

int main(): После запуска клиентского приложения, клиет создаёт свой сокет, подсоединяет его к адресу и порту (bind) а также запрашивает соединения с сервером. Как только соединение установлен клиент выводит сообщение об этом, послк чего попадает в цикл с условием: пока не введена завершающая команда. В цикле просит пользователя ввести два числа, после чего, отправляет их серверу. Осуществляется проверка того что ответ поместиться в buffer (проверка осуществляется так же как и на сервере). После чего получает ответ от сервера и выводит его. Когда пользователь введет #, выведется сообзение о прерывании соединения с сервером и клиент завершится.

8.5 Руководство пользователя

Пример работы:

./server

./client

Socket created successfully...
Connection to the server 127.0.0.1
Awaiting confirmation from the server...

=> Enter to terminate the connection
Server: enter left and right border
Client: 1 30
Server: 2 3 5 7 11 13 17 19 23 29
Server: enter left and right border
Client: -10 0
Server: There is not a single prime number in the given interval
Server: enter left and right border
Client: 10 5
Server: Wrong borders. Left border bigger than right border
Server: enter left and right border
Client: 20 2000
Server: 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997 1009 1013 1019 1021 1031 1033 1039 1049 1051 1061 1063 1069 1087 1091 1093 1097 1103 1109 1117 1123 1129 1151 1153 1163 1171 1181 1187 1193 1201 1213 1217 1223 1229 1231 1237 1249 1259 1277 1279 1283 1289 1291 1297 1301 1303 1307 1319 1321 1327 1361 1367 1373 1381 1399 1409 1423 1427 1429 1433 1439 1447 1451 1453 1459 1471 1481 1483 1487 1489 1493 1499 1511 1523 1531 1543 1549 1553 1559 1567 1571 1579 1583 1597 1601 1607 1609 1613 1619 1621 1627 1637 1657 1663 1667 1669 1693 1697 1699 1709 1721 1723 1733 1741 1747 1753 1759 1777 1783 1787 1789 1801 1811 1823 1831 1847 1861 1867 1871 1873 1877 1879 1889 1901 1907 1913 1931 1933 1949 1951 1973 1979 1987 1993 1997 1999
Server: enter left and right border

Client: #
Server: #
=> Connection terminated.
Goodbye

9 Список использованных источников

- Linux Ubuntu Manual: Socket. – URL:

  https://www.opennet.ru/man.shtml?topic=forkcategory=2russian=2

- Linux Ubuntu Manual: Semaphores. – URL:

  https://www.opennet.ru/man.shtml?topic=socketcategory=2russian=2

- Теренс, Ч. . Системное программирование на C++ для Unix: учебное пособие / Ч. . Теренс. : Издательская группа BHV, 1997. 299 с.

- Таненбаум, Э. Современные операционные системы: учебное пособие / Э. Таненбаум. СПб: Питер, 2018. 1120.

- Записи лекций и презентации с них, лектор Мокряков А.В., конспект Евкарпиев М.Д.

10 Приложение

Здесь приводится код. Каждый файл в своём подразделе.

10.1 Лабораторная №1

10.1.1 Задание №1

Файл task1.cpp

```
#include <iostream>
using namespace std;

int main(){
        int a, b, res;
        cout << "cin 2 numbers: ";
        cin >> a >> b;
        res = a * b;
        while (a != 0 && b != 0){
                (a > b) ? a = a % b : b = b % a;
        }
        res /= (a + b);
        cout << "NOD: " << a + b << "\nNOK:" << res;
        return 0;
}
```

10.1.2 Задание №2

Файл task2.cpp

```
#include <iostream>
#include <string>
using namespace std;

int bracket(string s){
```

```
    int len = s.length();
    int a = 0, b = 0, c = 0;
    char last = ' ';
    for (int i = 0; i < len; i++) {
        if (s[i] == '(') {
            a++; last = '(';
        }
        else if (s[i] == '[') {
            b++; last = '[';
        }
        else if (s[i] == '{') {
            c++; last = '{';
        }
        else if (s[i] == ')') {
            a--;
            if (last == '[' || last == '{') return -1;
            last = ')';
        }
        else if (s[i] == ']') {
            b--;
            if (last == '(' || last == '{') return -1;
            last = ']';
        }
        else if (s[i] == '}') {
            c--;
            if (last == '(' || last == '[') return -1;
            last = '}';
        }
    }
    return a + b + c;
}

int main(){
```

```
    string s;
    cin >> s;
    if (bracket(s) == 0) {
        cout << "Correct";
    }
    else cout << "Uncorrect";
    return 0;
}
```

10.1.3 Задание №3

Файл task3.cpp

```
#include <iostream>
#include <math.h>
#include <array>

using namespace std;

int main(){
    char f1 = 0, prevf1 = 0;
    short int f2 = 0, prevf2 = 0;
    int f3 = 0, prevf3 = 0;
    long long f4 = 0, prevf4 = 0;
    int n, i, a1 = 0, a2 = 0, a3 = 0, a4 = 0;
    long long res = 0;
    cout << "Cin number:";
    cin >> n;
    for (i = 0; i < n; i++) {
        prevf1 = f1;
        f1 += pow(i, i);
        if (f1 < prevf1) {
            cout << "Overflow Char - " << i << endl;
```

```
            a1 = 1;
            break;
        }
    }
    for (i = 0; i < n; i++) {
        prevf2 = f2;
        f2 += pow(i, i);
        if (f2 < prevf2){
            cout << "Overflow Short Int -- " << i << endl;
            a2 = 1;
            break;
        }
    }
    for (i = 0; i < n; i++) {
        prevf3 = f3;
        f3 += pow(i, i);
        if (f3 < prevf3){
            cout << "Overflow Int -- " << i << endl;
            a3 = 1;
            break;
        }
    }
    for (i = 0; i < n; i++) {
        prevf4 = f4;
        f4 += pow(i, i);
        if (f4 < prevf4){
            cout << "Overflow Long Long -- " << i << endl;
            a4 = 1;
            break;
        }
    }
    cout << "\n";
    if (a1 == 0) cout << "Your result Char: " << f1 * 1 << endl;
```

```
    if (a2 == 0) cout << "Your result Short Int: " << f2 << endl;
    if (a3 == 0) cout << "Your result Int: " << f3 << endl;
    if (a4 == 0) cout << "Your result Long Long: " << f4 << endl;
    return 0;
}
```

10.1.4 Задание №4

Файл task4.cpp

```
#include <iostream>
using namespace std;

int main() {
    int n, m = 0, t;
    cin >> n;
    t = n * (n - 1) + 1;
    for (int i = 1; i <= n; i++){
        cout << t;
        if (i < n)
            cout << '+';
        m = m + t;
        t = t + 2;
    }
    cout << "=" << m;
}
```

10.1.5 Задание №5

Файл task5.cpp

```
#include <iostream>
using namespace std;
```

```
int main(){
    int m, i = 0, j = 1, k, n = 1;
    cin >> m;
    if (m <= 0) {
        cout << "fibo number: " << i << "\nserial number: " << n;
        return 0;
    }

    while (j <= m) {
        k = j;
        j = i + j;
        i = k;
        n++;
    }
    n+=1;
    cout << "fibo number: " << j << "\nserial number: " << n;
}
```

10.1.6 Задание №6

Файл task6.cpp

```
#include <iostream>
using namespace std;

int main(){
    int n, cnt = 0;
    cout << "Enter the number: ";
    cin >> n;
    for (int i = 1; i * i <= n; i++) {
        for (int j = i; i * i + j * j <= n; j++) {
            if (i * i + j * j == n) {
```

```
                cout << i << "^2 + " << j << "^2" << endl;
                cnt = 1;
            }
        }
    }
    if (cnt == 0) cout << "NO";
    return 0;
}
```

10.2 Лабораторная №2

10.2.1 Задание №1

Файл task1.cpp

```
#include <iostream>
#include <dirent.h>
#include <cstring>
#include <sys/stat.h>
using namespace std;

int main(){
    DIR* dir;
    int time;
    struct dirent *ent;
    struct stat buf;
    char c[100];
    cin >> c;

    dir = opendir(c);

    if (dir){
            while ((ent = readdir(dir)) != NULL){
                    stat(ent->d_name, &buf);
```

```
                    time = buf.st_mtime;
                    cout << ent->d_name /*<<" "<<time*/ << endl;
            }
            closedir(dir);
    }
    else {
        cout<<"Error";
        exit(-1);
        }
    return(0);
}
```

10.2.2 Задание №2

Файл task2.cpp

```
#include <stdio.h>
#include <iostream>
#include <dirent.h>
#include <cstring>
using namespace std;

int main(){
    DIR* d;
    int iNum = 0;
    struct dirent *dir;
    char c[100];
    cin >> c;

    d = opendir(c);

    if (d){
        while ((dir = readdir(d)) != NULL){
```

```
            if(dir->d_type != DT_DIR)
                    cout << dir->d_name << endl;
        }
        closedir(d);
    }
    return 0;
}
```

10.2.3 Задание №3

Файл task3.cpp

```
#include <iostream>
#include <string>
#include <cstring>
#include <fstream>
#include <cstdio>
#include <algorithm>
#include <sys/stat.h>
#include <unistd.h>
using namespace std;

int main(){
        cout<<"Cin puth to file: ";
        struct stat buff;
        string file_path, str;
        int i = 99, j=0;
        char c[100];
        cin>>file_path;
        //strcpy(c, file_path);
        ifstream file(file_path);
        ofstream fout;
```

```
if(!file.is_open()){
        cout<<"There is no such file\nNew file will be created";
        fout.open(file_path);
if(!fout){
    cout<<"Error: file not ctreated";
    exit(-1);
        }

        cout<<"To stop enter: stop\nEnter text:\n";
        do {
                getline(cin, str);
                if (str == "stop")
                        break;
                fout<<str<<endl;
        } while(1);

        fout.close();
        return 0;
}

 file.close();

 cout<<"The file already existed.\nFile will be cleared\
 and opened for writing. A new hard link will be created\n";
 fout.open(file_path, ios_base::out | ios_base::trunc);
 if(!fout){
        cout<<"Error: file not ctreated";
        exit(-1);
}

cout<<"To stop enter: stop\nEnter text:\n";
do {
        getline(cin, str);
```

```
            if (str == "stop")
                    break;
            fout<<str<<endl;
    } while(1);
fout.close();

string file_name, ext, tmp, file_name_tmp, link_path;
string dir_path = file_path;

while(dir_path.back() != '/' && dir_path.length()) {
    tmp.push_back(dir_path.back());
    dir_path.pop_back();
}

while(tmp.length()) {
    file_name.push_back(tmp.back());
    tmp.pop_back();
}
file_name_tmp = file_name;
while(file_name_tmp.back() != '.' && file_name_tmp.length()) {
    ext.push_back(file_name_tmp.back());
    file_name_tmp.pop_back();
}

if(file_name_tmp.back() == '.')
    file_name_tmp.pop_back();

if(file_name_tmp.length()){
    file_name = file_name_tmp;
    reverse(ext.begin(), ext.end());
}
else
    ext.clear();
```

```
    int num = 0;
    link_path = dir_path + file_name + "." + to_string(num) + "." + ext;
    if(!ext.length())
        link_path.pop_back();

    FILE* link_file;
    /*if(!dir_path.length())
        dir_path.push_back('.');*/

    link_file = fopen(dir_path.c_str(), "r");
    while(link_file) {
        fclose(link_file);
        num++;
        link_path = dir_path + file_name + "." + to_string(num) + "." + ext;

        if(!ext.length())
            link_path.pop_back();

        link_file = fopen(link_path.c_str(), "r");
    }
    //stat(file_path, &buff);
    //cout<<"scale hard linfs before: "<<buff.st_nlink<<endl;
    link(file_path.c_str(), link_path.c_str());
    //stat(file_path, &buff);
    //cout<<"after: "<<buff.st_nlink<<endl;
}
```

10.2.4 Задание №4

Файл task4.cpp

```
#include <iostream>
#include <dirent.h>
#include <string.h>
#include <cstring>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdlib.h>
using namespace std;

int main() {
    string file_path, link_dir;
    struct stat buff;
    time_t last_modif;
    cout<<"Cin file puth: ";
    cin >> file_path ;
    cout<<"Cin link directory: ";
    cin>> link_dir;

    symlink(file_path.c_str(), link_dir.c_str());
    FILE* link_file;
    link_file = fopen(link_dir.c_str(), "r");
    char c;
    cout<<endl;

    lstat(link_dir.c_str(), &buff);

    cout<<"Устройство: "<<buff.st_dev<<", inode: "<<buff.st_ino<<\
    ", режим доступа: "<<buff.st_mode<<\
    ", количество жестких ссылок: "<<buff.st_nlink<<\
    ", идентификатор пользователя-владельца: "<<buff.st_uid<<\
```

```
    ", идентификатор группы-владельца: "<<buff.st_gid<<\
    ", тип устройства: "<<buff.st_rdev<<\
    ", общий размер в байтах: "<<buff.st_size<<\
    ", размер блока ввода-вывода в файловой системе: "<<buff.st_blksize<<\
    ", количество выделенных блоков: "<<buff.st_blocks<<endl;
    last_modif = buff.st_atime;
    cout<<"время последнего доступа: "<<last_modif<<endl;
    last_modif = buff.st_mtime;
    cout<<"время последней модификации: "<<last_modif<<endl;
    last_modif = buff.st_ctime;
    cout<<"время последнего изменения: "<<last_modif<<endl;
    if ((buff.st_mode & S_IFMT) == S_IFLNK)
        cout<<"Является символической ссылкой"<<endl;

    fclose(link_file);
    return 0;
}
```

10.2.5 Задание №5

Файл task5.cpp

```
#include <iostream>
#include <dirent.h>
#include <cstring>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
using namespace std;

int main(){
    DIR* dir;
    int i = 0, l=0, tmp, last_modif[100];
```

```
struct stat buff;
struct dirent *ent;
char c[100], names[100][20], chr[20];
cin >> c;

dir = opendir(c);

if (dir) {
    while ((ent = readdir(dir)) != NULL){
        strncpy(names[i], ent->d_name, 20);
        cout<<names[i]<<" ";
        stat(ent->d_name, &buff);
        last_modif[i] = buff.st_mtime;
        cout<<last_modif[i]<<endl;
        i++;
    }
    closedir(dir);
}
else {
        cout<<"Error";
        exit(-1);
}

while (l < i) {
    if (l == 0 || last_modif[l - 1] <= last_modif[l])
        ++l;
    else {
        tmp = last_modif[l];
        last_modif[l] = last_modif[l - 1];
        last_modif[l - 1] = tmp;
        strcpy(chr, names[l]);
        strcpy(names[l], names[l - 1]);
        strcpy(names[l - 1], chr);
```

```
            --l;
        }
    }
    cout<<"Sorted:"<<endl;
    for(l=0;l<i;l++){
        cout<<names[l]<<" ";
        cout<<last_modif[l]<<endl;
    }
        return 0;
}
```

10.3 Лабораторная №3

10.3.1 Задание №1

Файл task1.cpp

```
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
using namespace std;

int main(int argc, char **argv){
  if (argc > 2 ){
          perror("[ERROR]: wrong input!");
          exit(-1);
  }
  if (argc == 1){
          cout<<argv[0]<<"\nNo arguments\n";
          return 0;
```

```
  }

  int counter = atoi(argv[1]);
  if (counter == 0){
    cout<<"You entered 0\n";
    return 0;
  }
  pid_t *pid = new pid_t [counter];

  cout<<"PARENT: PID -- "<< getpid()<<endl;

  for(int i=0;i<counter;i++){
    switch(pid[i]=fork()) {
              case -1:
            perror("fork");
            exit(0);
              case 0:
            cout<<"CHILD["<<getpid()<<"]: Mikhail Evkarpiev\n";
            exit(0);
              default:
            //cout<<"PARENT: Я жду, пока потомок не вызовет exit()...\n";
            wait(0);
          }
  }
  delete [] pid;
  return 0;
}
```

10.3.2 Задание №2

Файл task2.cpp

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

```c
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>
#include <sys/types.h>

int main(int argc, char *argv[])
{
    FILE* f;
    int i, j, m, n, p, s, time = 10, read = 0, k = 5, counter = 0;
    pid_t pid, cpid1, cpid2, cpid3, cpid4;
    int *pp;

    pp = (int *)malloc(6 * sizeof(int));

    if(argc > 2){
        perror("Arguments error");
        return -1;
    }

    if(!(f = fopen("tmp.txt", "w"))){
            printf("Error to open file");
            exit (-1);
    }
    pid = getpid();
    pp[0] = pid;
    printf("I am parent of all PIDs - %d\n\n", pid);
        for(i = 0; i < k; i++){
            pid = fork();
            pp[i + 1] = pid;
            if(pid == 0){
                cpid1 = getpid();
                pp[0] = cpid1;
                for(j = 0; j < k - 1; j++){
                    cpid1 = fork();
                    pp[j + 1] = cpid1;
                    if(cpid1 == 0){
                            cpid2 = getpid();
                            pp[0] = cpid2;
                            for(m = 0; m < k - 2; m++){
                                cpid2 = fork();
```

```
pp[m + 1] = cpid2;
if(cpid2 == 0){
    cpid3 = getpid();
    pp[0] = cpid3;
    for(n = 0; n < k - 3; n++){
        cpid3 = fork();
        pp[n + 1] = cpid3;
        if(cpid3 == 0){
            cpid4 = getpid();
            pp[0] = cpid4;
            for(p = 0; p < k - 4; p++){
                    cpid4 = fork();
                    pp[p + 1] = cpid4;
                    if(cpid4 == 0){
                        sleep(time);
                        printf("<Parent %d>: ", pp[0]);
                        fprintf(f,"%d ", pp[0]);
                        for(s = 1; s < 6; s++){
                                if(pp[s] != 0)
                                        printf(\
                        "<%d child %d> ", s, pp[s]);
                            }
                        printf("\n");
                        exit(0);
                    }
                }
                sleep(time+1);
                printf("<Parent %d>: ", pp[0]);
                fprintf(f,"%d ", pp[0]);
                for(s = 1; s < 6; s++){
                        if(pp[s] != 0)
                                printf(\
                               "<%d child %d> ", s, pp[s]);
                    }
                printf("\n");
                exit(0);
        }
    }
    sleep(time+2);
    printf("<Parent %d>: ", pp[0]);
```

```
                                fprintf(f,"%d ", pp[0]);
                                for(s = 1; s < 6; s++)
                                    {
                                        if(pp[s] != 0)
                                                printf("<%d child %d> ", s, pp[s]);
                            }
                            printf("\n");
                            exit (0);
                        }
                    }
                    sleep(time+3);
                    printf("<Parent %d>: ", pp[0]);
                    fprintf(f,"%d ", pp[0]);
                    for(s = 1; s < 6; s++)
                        {
                            if(pp[s] != 0)
                                    printf("<%d child %d> ", s, pp[s]);
                        }
                            printf("\n");
                            exit (0);
                }
            }
            sleep(time+4);
            printf("<Parent %d>: ", pp[0]);
            fprintf(f,"%d ", pp[0]);
            for(s = 1; s < 6; s++){
                if(pp[s] != 0)
                        printf("<%d child %d>: ", s, pp[s]);
            }
            printf("\n");
            exit (0);
                }
    }
    printf("\n");
    system("pstree -p > file.txt");
    for (i = 0; i<5; i++)
            wait(0);
    sleep(2);

if(!(f = fopen("tmp.txt", "r"))){ printf("[ERROR] Can't open file"); exit (-1);}
```

```
    while (!feof(f)){
      fscanf(f,"%d ", &read);
      if (read !=0)
      counter++;
    }

    printf("\nAutor - Mikhail Evkarpiev. %d processes have been created!\n", counter);
    printf("<Parent %d> ", pp[0]);
        for(s = 1; s < 6; s++)
        {
            if(pp[s] != 0)
                printf("<%d child %d> ", s, pp[s]);
        }
        printf("\n");
        if(remove("tmp.txt") == -1) perror ("Remove Error");

        fclose(f);
return 0;
}
```

10.3.3 Задание №3

Файл task3.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <cstdlib>
#include <sys/wait.h>
#include <iostream>
#include <dirent.h>
using namespace std;

int main(int argc, char* argv[]){
    int i = 0, n = 0;
    int pid1, ppid, fact_base, fact = 1, flag = 1;
    FILE* f;
    FILE* tmp;
```

```
string pid_string, filename, copy;
char buf [BUFSIZ], line [BUFSIZ];
DIR* d;
struct dirent *dir;


if (argc>6) perror ("Wrong args");
if (argc == 2)
        if(!strncmp(argv[1], "-n", 2)){
                cout<<"Wrong argument\n";
                exit(-5);
        }
if(argc == 4)
if(!strncmp(argv[3], "-o", 2)){
        cout<<"Wrong argument\n";
        exit(-6);
}

char *flags_c = (char*)malloc(4*sizeof(char));
if (argc == 6){
        strncat(flags_c, argv[1], strlen(argv[1]));
        strncat(flags_c, argv[3], strlen(argv[3]));
        strncat(flags_c, argv[5], strlen(argv[5]));
}
else if (argc == 5){
        strncat(flags_c, argv[1], strlen(argv[1]));
        strncat(flags_c, argv[3], strlen(argv[3]));
        strncat(flags_c, argv[4], strlen(argv[4]));
    }
else if (argc == 4){
        strncat(flags_c, argv[1], strlen(argv[1]));
        strncat(flags_c, argv[3], strlen(argv[3]));
    }
else if (argc == 3){
        strncat(flags_c, argv[1], strlen(argv[1]));
        strncat(flags_c, argv[2], strlen(argv[2]));
}
else if (argc == 2){
        strncat(flags_c, argv[1], strlen(argv[1]));
}
```

```
if ((flags_c[1] == 'n')||(flags_c[2] == 'n')||(flags_c[3] == 'n')) {
    n = atoi(argv[2]);
  if ((flags_c[1] == 'o')||(flags_c[2] == 'o')||(flags_c[3] == 'o'))
    filename.assign(argv[4], strlen(argv[4]));
    else flag = 0;
  for (i = 0; i < n; i++){
    pid1 = fork();
    sleep(1);
    if (pid1 == 0) break;
  }
}
else {
    if ((flags_c[1] == 'o')||(flags_c[2] == 'o')||(flags_c[3] == 'o'))
              filename.assign(argv[2], strlen(argv[2]));
    else flag = 0;
    pid1 = fork();
}


if (pid1 == 0){
    if ((flags_c[1] == 'o')||(flags_c[2] == 'o')||(flags_c[3] == 'o')){
        pid1 = getpid();
        fact_base = getpid() - getppid();
        cout << pid1 <<endl;
        pid_string = to_string(pid1);
        filename = filename + "." + pid_string;
        const char * filename_c = filename.c_str();
        const char*
        if(!(f = fopen(filename_c, "w"))){
            printf("Error to open file %d", pid1);
            exit (-1);
        }

    for (i = 1; i <= fact_base; i++)
        fact = fact*i;

    fprintf(f,"%s ",filename.c_str());
    fprintf(f," Mikhail (PPid: %d - Pid: %d) = %d", getppid(), getpid(), fact);
```

```
    //fclose(f);
    }
    else{
        pid1 = getpid();
        fact_base = getpid() - getppid();
        for (i = 1; i <= fact_base; i++)
            fact = fact*i;
        //sleep(1);
        cout << filename.c_str() << " "; // имя файла
        printf(" Mikhail (PPid: %d - Pid: %d)! = %d\n", getppid(), getpid(), fact);
    }
}
else{
    wait(0);
    sleep(3);
    if (flag == 1){
    //cout<<flags_c;
    //cout<<filename.c_str();
    if(!(f = fopen("Outfile", "w"))){
        printf("[ERROR] Can't open file %d", pid1);
        exit (-2);
    }
    else
        d = opendir(".");
    if (d){
        while ((dir = readdir(d)) != NULL)
            {
                {
                    strcpy(buf, dir->d_name);
                    if (!strncmp(buf,filename.c_str(),strlen(filename.c_str()))){
                        if(!(tmp = fopen(buf, "r"))){
                           printf("[ERROR] Can't open file '%s'", buf);
                           exit (-4);
                        }
                        fscanf(tmp,"%[^\n]",line);
                        fprintf(f,"%s \n", line);
                        fclose(tmp);
                        if ((flags_c[1] == 'c')||(flags_c[3] == 'c')\
                        ||(flags_c[5] == 'c')) {
                                //cout<<buf;
```

```
                                if(remove(buf) == -1) perror ("Remove Error");
                                //cout<<"1";
                            }
                        }
                    }
                }
            closedir(d);
        }
         if (rename("Outfile", filename.c_str()) == -1) perror ("Rename error");
        }
    }
    free(flags_c);
    return 0;
}
```

10.4 Лабораторная №4

10.4.1 Задание №1

Файл `task1.cpp`

```
#include <iostream>
#include <fstream>
#include <time.h>
#include <random>
#include <sys/types.h>
#include <sys/sem.h>
#include <sys/ipc.h>
#include <unistd.h>

using namespace std;

#define KEY 55
#define SEMSIZE 2

union semun{
```

```
    int val;
    semid_ds *mbuf;
    ushort *array;
    seminfo *ibuf;
} arg;

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

    if(argc != 2){
        perror("[ERROR]: arguments");
        exit(-1);
    }
    ofstream file(argv[1]);
    if(!file.is_open()){
        perror("[ERROR]: file");
        exit(-2);
    }
    file.close();

    int sem_deskid;
    pid_t pid;
    int ppid = getpid();
    int flag, id;
    long long int count;
    long long int st_time = time(NULL);
    ushort arr[BUFSIZ];
    srand(time(NULL));
    arg.array = arr;

    sembuf sopS[1] = {{1, 1, 0}};
    sembuf sopr1[2] = {{0, 1, 0}, {1, -1, IPC_NOWAIT}};
    sembuf sopr2[1] = {{1, 1, 0}};
    sembuf sopr3[1] = {{0, -1, 0}};
```

```
sembuf sopw1[1] {{1, -1, IPC_NOWAIT}};
sembuf sopw2[1] {{0, 0, 0}};
sembuf sopw3[1] = {{1, 1, 0}};

if((sem_deskid = semget(KEY, SEMSIZE, IPC_CREAT | 0777)) == -1){
    perror("Semget error");
    exit(-3);
}
if(semop(sem_deskid, sopS, 1) == -1){
    perror("Semop error");
    exit(-4);
}

for(int i = 0; i < 20; i++){
    pid = fork();
    if(pid == -1){
        perror("Fork");
        exit(-5);
    }
    else if(pid == 0){
        flag = i % 2;
        id = i / 2;
        count = 0;
        break;
    }
}

if(getpid() == ppid){
    sleep(33);
    if (semctl(sem_deskid, 0, IPC_RMID, 0) == -1){
        perror("Semctl error");
        exit(-6);
    }
```

```
    return 0;
}

while((time(0) - st_time) < 30){
    usleep(rand() % 10000 + 10000);
    if(flag){
        if (semop(sem_deskid, sopw1, 1) != -1) {
            semop(sem_deskid, sopw2, 1);
            ofstream File(argv[1], ios::app);
            File << random() % 1000 << endl;
            count++;
            if(!File.is_open()) File.close();
            semop(sem_deskid, sopw3, 1);
        }
    }
    else{
        if (semop(sem_deskid, sopr1, 2) != -1) {
            semop(sem_deskid, sopr2, 1);
            ifstream File(argv[1], ios::app);
            char buf;
            while(File >> buf){
                if((buf - '0') == id) count++;
            }
            if(!File.is_open()) File.close();
            semop(sem_deskid, sopr3, 1);
        }
    }
}
if (flag == 1) {cout << "I am " << "Writer" << ", my PID = "\
<< getpid() << ", count = "<< count << endl;}
else {cout << "I am " << "Reader" << ", my PID = " << getpid() << \
", the number that was counted " << id << \
", count = "<< count << endl;}
```

```
    return 0;
}
```

10.4.2 Задание №2

Файл server.cpp

```
#include <iostream>
#include <sys/types.h>
#include <sys/sem.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>

using namespace std;

#define SEMKEY 78
#define SHMKEY 87
#define SEMSIZE 2

union semun{
    int val;
    semid_ds *mbuf;
    ushort *array;
    seminfo *ibuf;
}arg;

int main (){
    ushort arr[BUFSIZ];
    arg.array = arr;
    shmid_ds sbuf;
    double *operand1, *operand2;
    char *operation, *shm_ptr;
    int shm_desk;
    int sem_desk;

    sembuf sopS[1] = {{1, 1, 0}};
    sembuf sop1[2] {{0, -1, 0}, {1, 0, 0}};
    sembuf sop2[2] {{0, 0, 0 | IPC_NOWAIT}, {1, 0, 0 | IPC_NOWAIT}};
```

```
sembuf sop3[2] {{0, 1, 0 | IPC_NOWAIT}, {1, 1, 0 | IPC_NOWAIT}};

if((shm_desk = shmget(SHMKEY, (sizeof(double) * SEMSIZE + sizeof(char)),\
0777)) != -1){
    if (shmctl(shm_desk, IPC_RMID, &sbuf) == -1){
        perror("Shmctl error");
        exit(-1);
    }
}
if((shm_desk = shmget(SHMKEY, (sizeof(double) * SEMSIZE + sizeof(char)),\
0777 | IPC_CREAT)) == -1){
    perror("Shmget error");
    exit(-2);
}
if(shmctl(shm_desk, SHM_LOCK, &sbuf) == -1){
    perror("Shmctl error");
    exit(-3);
}


if((sem_desk = semget(SEMKEY, SEMSIZE, 0777)) != -1){
    if(semctl(sem_desk, 0, IPC_RMID, 0) == -1){
        perror("Semctl error");
        exit(-4);
    }
}
if((sem_desk = semget(SEMKEY, SEMSIZE, \
IPC_CREAT | 0777)) == -1){
    perror("Semget error");
    exit(-5);
}
if(semop(sem_desk, sopS, 1) == -1){
    perror("Semop error");
    exit(-6);
}
while(1){
    if(semop(sem_desk, sop1, 2) == -1){
        perror("Semop fail");
        exit(-7);
    }
```

```
        shm_ptr = (char*)shmat(shm_desk, 0, 0);
        if(shm_ptr == ((char*)-1)){
            perror("Shmat error");
            exit(-8);
        }

        operand1 = (double*)shm_ptr;
        operation = (char*)(operand1 + sizeof(double));
        operand2 = (double*)(operation + sizeof(char));

        if(*operation == '+')
            *operand1 = *operand1 + *operand2;
        else if(*operation == '-')
            *operand1 = *operand1 - *operand2;
        else if(*operation == '*')
            *operand1 = *operand1 * *operand2;
        else if(*operation == '/')
            *operand1 = *operand1 / *operand2;

        if(shmdt(shm_ptr) == -1){
            perror("Shmdt error");
            exit(-9);
        }
        if (semop(sem_desk, sop2, 2) == -1 || \
        semop(sem_desk, sop3, 2) == -1){
            perror("Semop error");
            exit(-10);
        }
        sleep(1);
    }
    return (0);
}
```

Файл `client.cpp`

```
#include <iostream>
#include <string>
#include <sys/types.h>
#include <sys/sem.h>
#include <sys/ipc.h>
#include <sys/shm.h>
```

```
#include <unistd.h>

using namespace std;

#define SEMKEY 78
#define SHMKEY 87
#define SEMSIZE 2

union semun{
    int val;
    semid_ds *mbuf;
    ushort *array;
    seminfo *ibuf;
}arg;

int is_double(string str){
    char *error_code = nullptr;
    double value = strtod(str.c_str(), &error_code);
    if(error_code == str.c_str() || *error_code != '\0')
            return 0;
    return 1;
}

int is_operation(string str){
    if(str.length() != 1)
            return 0;
    if(str[0] != '+' && str[0] != '-' && str[0] !=\
    '/' && str[0] != '*')
            return 0;
    return 1;
}

int main(){
    shmid_ds sbuf;
    double *operand1, *operand2;
    char *operation, *error_code;
    string tmpd1, tmpd2, tmpc;
    char *shm_ptr;
    int shm_desk, flag;
    int sem_deskid;
```

```
ushort arr[BUFSIZ];
arg.array = arr;

sembuf sop1[2] {{0, 0, 0}, {1, -1, 0}};
sembuf sop2[2] {{0, 1, 0 | IPC_NOWAIT}, {1, 0, 0 | IPC_NOWAIT}};
sembuf sop3[2] {{0, -1, 0}, {1, -1, 0}};
sembuf sop4[2] {{0, 0, 0 | IPC_NOWAIT}, {1, 1, 0 | IPC_NOWAIT}};

while(1){
    flag = 0;
    while(!flag){
        cout << "Enter expression(A <op> B): " << endl;
        cout << "Enter A: ";
        getline(cin, tmpd1);
        cout << "Enter <op>: ";
        getline(cin, tmpc);
        cout << "Enter B: ";
        getline(cin, tmpd2);

        if(tmpd1.empty()){
                cout << "Input string is empty. Completing programm..." << endl;
                return 0;
        }

        if(is_double(tmpd1) && is_operation(tmpc) && is_double(tmpd2)) flag = 1;
        if(!flag) cout << "Incorrect input!" << endl;

    }
    if((shm_desk = shmget(SHMKEY, (sizeof(double) * SEMSIZE + sizeof(char)),\
    0777 | IPC_CREAT)) == -1){
        perror("Shmget error");
        exit(-1);
    }

    if((sem_deskid = semget(SEMKEY, SEMSIZE, 0777)) == -1){
        perror("Semget error");
        exit(-2);
    }
    if (semop(sem_deskid, sop1, 2) == -1){
        perror("Semop error");
```

```
    exit(-3);
}
if (shmctl(shm_desk, SHM_UNLOCK, &sbuf) == -1){
    perror("Shmctl error");
    exit(-4);
}

shm_ptr = (char*)shmat(shm_desk, 0, 0);
if(shm_ptr == ((char*)-1)){
    perror("Shmat error");
    exit(-5);
}
operand1 = (double*)shm_ptr;
operation = (char*)(operand1 + sizeof(double));
operand2 = (double*)(operation + sizeof(char));

*operand1 = stod(tmpd1);
*operation = tmpc[0];
*operand2 = stod(tmpd2);

if(shmdt(shm_ptr) == -1){
    perror("Shmdt error");
    exit(-6);
}
if (shmctl(shm_desk, SHM_LOCK, &sbuf) == -1){
    perror("Shmctl error");
    exit(-7);
}
if (semop(sem_deskid, sop2, 2) == -1 || semop(sem_deskid, sop3, 2) == -1){
    perror("Semop error");
    exit(-8);
}
if (shmctl(shm_desk, SHM_UNLOCK, &sbuf) == -1){
    perror("Shmctl error");
    exit(-9);
}
shm_ptr = (char*)shmat(shm_desk, 0, 0);
if(shm_ptr == ((char*)-1)){
    perror("Shmat error");
    exit(-10);
```

```
        }

        operand1 = (double*)shm_ptr;
        cout << "Result = " << *operand1 << endl;

        if(shmdt(operand1) == -1){
            perror("Shmdt error");
            exit(-11);
        }
        if (shmctl(shm_desk, SHM_LOCK, &sbuf) == -1){
            perror("Shmctl error");
            exit(-12);
        }
        if (semop(sem_deskid, sop4, 2) == -1){
            perror("Semop error");
            exit(-13);
        }
    }
}
```

10.5 Лабораторная №5

10.5.1 Задание №1

Файл `task1.cpp`

```
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
using namespace std;

double function(double x){
        return (4.0/(1+x*x));
}
```

```
double integr(double a, double b, int n){
        double s = (function(a) + function(b))/2;
        double h = (b-a)/n;
        for(int j = 1; j < n; j++)
                s += function(j*h + a);
        double res = h*s;
        return res;
}

int main(int argc, char **argv)
{
        if (argc < 3){
                  perror("[ERROR]: wrong arguments!");
                  exit(-1);
          }

        pid_t pid;
        int fifo[2];
        int i, parts, n;
        double curr_pi, pi=0, k, step;

        parts = atoi(argv[1]);
        n = atoi(argv[2]);
        step = 1/n;

        if(pipe(fifo) == -1){
                perror("[ERROR]: pipe!");
                exit(-1);
        }

        for(i=0;i<parts;i++){
                pid = fork();
                if(!pid){
                        curr_pi = integr(i*(1.0/parts), (i+1)*(1.0/parts), n);
                        close(fifo[0]);
                        write(fifo[1], &curr_pi, sizeof(curr_pi));
                        close(fifo[1]);
                        exit(0);
                }
        }
```

```
        if(pid){
                double y=0;
                close(fifo[1]);
                while(read(fifo[0], &y, sizeof(y))){
                        pi+=y;
                        //cout<<"y = "<<y<<endl;
                }
                cout.precision(20);
                cout<< pi <<endl;
                close(fifo[0]);
        }
        return 0;
}
```

10.5.2 Задание №2

Файл server.cpp

```
#include<iostream>
#include<math.h>
#include<stdlib.h>
#include<unistd.h>
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/types.h>
#include<sys/msg.h>
#include<cstring>
using namespace std;

#ifndef MSGMAX
#define MSGMAX 1024
#endif

typedef struct{
        double first_compl;
        double second_compl;
}Elem;

typedef struct{
```

```
        long mtype;
        int procpid;
        int ratio[4];
        Elem ans[3];
}mbuf;

double servroot(double x){
        if(x < 0) return -pow(-x, 1.0/3.0);
        return pow(x, 1.0/3.0);
}

int Method_of_Kardan(double a, double b, double c, double d, Elem* x){
        if (a == 0){
                return -1;
        }

        double p = (3.0*a*c - b*b)/(3.0*a*a);
        double q = (2.0*b*b*b - 9.0*a*b*c + 27.0*a*a*d)/(27.0*a*a*a);
        double s = q*q/4.0 + p*p*p/27.0;

        double  f;
        if (q == 0)
                f = M_PI/2.0;
        else if(q < 0)
                f = atan(-2.0*sqrt(-s)/q);
        else f = atan(-2.0*sqrt(-s)/q) + M_PI;

        for(int i=0;i<3;i++){
                x[i].first_compl = x[i].second_compl = 0.0;
        }

        if (s<0){
                x[0].first_compl = 2.0*sqrt(-p/3.0)*cos(f/3.0) -\
                b/(3.0*a);
                x[1].first_compl = 2.0*sqrt(-p/3.0)*cos(f/3.0 +\
                2.0*M_PI/3.0)\
                - b/(3.0*a);
                x[2].first_compl = 2.0*sqrt(-p/3.0)*cos(f/3.0 +\
                4.0*M_PI/3.0)\
                - b/(3.0*a);
```

```
        }
        else if(s == 0){
                x[0].first_compl = 2.0*servroot(-q/2.0) - b/(3.0*a);
                x[1].first_compl = -servroot(-q/2.0) - b/(3.0*a);
                x[2].first_compl = -servroot(-q/2.0) - b/(3.0*a);
        }
        else {
                double tmp1, tmp2;
                tmp1 = servroot(-q/2.0 + sqrt(s)) + servroot(\
                -q/2.0 - sqrt(s));
                tmp2 = servroot(-q/2.0 + sqrt(s)) - servroot(\
                -q/2.0 - sqrt(s));
                x[0].first_compl = tmp1 - b/(3.0*a);
                x[1].first_compl = -tmp1/2.0 - b/(3.0*a);
                x[1].second_compl = sqrt(3)*tmp2/2.0;
                x[2].first_compl = -tmp1/2.0 - b/(3.0*a);
                x[2].second_compl = -sqrt(3)*tmp2/2.0;
        }
        return 0;
}

int main(){
        mbuf mobj;
        Elem x[3];
        int pid = getpid();
        int res;
        cout<<"Server: my pid is " << pid<<endl;

        int mqueueId;
        if((mqueueId = msgget(100, IPC_CREAT | 0666)) == -1)
                {perror("msgget"); exit(-1);}
        while(1){
                if((res = msgrcv(mqueueId, &mobj, MSGMAX, 16, \
                IPC_NOWAIT | MSG_NOERROR)) > 0){
                        if (Method_of_Kardan(mobj.ratio[0], mobj.ratio[1]\
                        , mobj.ratio[2], \
                        mobj.ratio[3], x) == -1){
                                perror("Invalid 'a'\n");
                                mobj.ans[0].first_compl = mobj.ans[0].second_compl\
                                = 0.0;
```

```
                                mobj.ans[1].first_compl = mobj.ans[1].second_compl\
                                = 0.0;
                                mobj.ans[2].first_compl = mobj.ans[2].second_compl\
                                = 0.0;
                                mobj.mtype = mobj.procpid;
                                if(msgsnd(mqueueId, &mobj, sizeof(mbuf), IPC_NOWAIT))
                                    perror("msgsnd");
                        }
                        else {
                                mobj.ans[0] = x[0];
                                mobj.ans[1] = x[1];
                                mobj.ans[2] = x[2];
                                mobj.mtype = mobj.procpid;
                                if(msgsnd(mqueueId, &mobj, sizeof(mbuf), IPC_NOWAIT))
                                    perror("msgsnd");
                        }
                }
                //else if(res != 0) {perror("msgrcv"); break;}
        }
        msgctl(mqueueId, IPC_RMID, 0);
        return 0;
}
```

Файл client.cpp

```
#include<iostream>
#include<math.h>
#include<stdlib.h>
#include<unistd.h>
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<cstring>
using namespace std;

#ifndef MSGMAX
#define MSGMAX 1024
#endif

typedef struct{
        double first_compl;
```

```
        double second_compl;
}Elem;

typedef struct{
        long mtype;
        int procpid;
        int ratio[4];
        Elem ans[3];
}mbuf;

int main(){
        int a, b, c, d;
        int pid = getpid();
        mbuf mobj;
        //int len=0;

        cout<<"Client: my pid is "<<pid<<endl;

        while(1){
                cout<<"Enter ratio of cubic equation:\nEnter koef 'a': ";
                cin>>a;
                cout<<"\nEnter koef 'b': ";
                cin>>b;
                cout<<"\nEnter koef 'c': ";
                cin>>c;
                cout<<"\nEnter koef 'd': ";
                cin>>d;
                cout<<endl;

                int mqueueId = msgget(100, IPC_CREAT | 0666);

                mobj.ratio[0] = a;
                mobj.ratio[1] = b;
                mobj.ratio[2] = c;
                mobj.ratio[3] = d;

                mobj.procpid = pid;
                mobj.mtype = 16;
```

```
                if (msgsnd(mqueueId, &mobj, MSGMAX, IPC_NOWAIT)) perror("msgsnd");

                sleep(1);
                if(msgrcv(mqueueId, &mobj, MSGMAX, pid, IPC_NOWAIT \
                | MSG_NOERROR) > 0){
                        for(int i=0;i<3;i++){
                                if(mobj.ans[i].second_compl == 0.0)
                                printf("x_%d = %.10lf\n", i, mobj.ans[i].first_compl);
                                else printf("x_%d = %.10lf + i*%.15lf\n", i,\
                                mobj.ans[i].first_compl, mobj.ans[i].second_compl);
                        }
                        if(a == 0){printf("invalid a\n");}
                }
                break;
        }
        return 0;
}
```

10.6 ПО «Поиск простых чисел в определенном диапозоне»

Файл server.cpp

```
#include <iostream>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <string>
#include <vector>
#include <cmath>
#include <unistd.h>
using namespace std;

bool is_simple(int x){
        for (int i=2;i<=sqrt(x);i++){
                if (x%i==0) return false;
```

```
        }
        return true;
}

int search(int A, int B, vector <int> &ivector){
        if (B<2 && A<=B) return 0;
        if (A>B) return -1;
        int i;
        A >= 2 ? i = A : i = 2;
        for(i; i<=B;i++){
                if (is_simple(i)) ivector.push_back(i);
        }
        if (ivector.empty()) return 0;
        return 1;
}

int main()
{
    int client, server;
    int bufsize = 1024;
    int portNum = 1500;
    int counter = 0;
    int left_border;
    int right_border;
    int res;
    bool isExit = false;
    char* buffer = new char[bufsize];
    string str, strbuf1, strbuf2 = " ";
    vector <int> vec;

    struct sockaddr_in adr;
    socklen_t adr_len;
    pid_t pid;

    if ((client = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        cout << "\nError establishing socket..." << endl;
        exit(1);
    }

    cout << "\nSocket server has been created..." << endl;
```

```
adr.sin_family = AF_INET;
adr.sin_addr.s_addr = htons(INADDR_ANY);
adr.sin_port = htons(portNum);

if ((bind(client, (struct sockaddr*)&adr,sizeof(adr))) < 0) {
    cout << "\nError binding connection..." << endl;
    return -1;
}

adr_len = sizeof(adr);
cout << "Looking for clients..." << endl;
listen(client, 1);

while ((server = accept(client,(struct sockaddr *)&adr,&adr_len))\
> 0) {
    strcpy(buffer, "Server connected...\n");
    send(server, buffer, bufsize, 0);

    cout << "Client: ";
    counter = 0;
    do {
        recv(server, buffer, bufsize, 0);
        //cout << buffer << endl;
        counter++;
        counter == 1? left_border = atoi(buffer) :\
        right_border = atoi(buffer);
        if (*buffer == '#') isExit = true;
    } while (counter != 2 && *buffer != '#');


    do {
        //cout << "\nServer: ";
        res = search(left_border, right_border, vec);

        /*for(int i=0;i<vec.size();i++){
                cout<<vec[i]<<" ";
        }*/

        if (res == -1) strcpy(buffer, "Wrong borders. \
```

```
    Left border bigger than right border");
    else if(res == 0) strcpy(buffer, "There is not\
    a single prime number in the given interval");
    else {
            if (right_border - left_border > 1024){
                (right_border >= 2) && (right_border > left_border) \
                ? bufsize = abs((right_border - left_border)*2)\
                : bufsize = 1024;
                delete [] buffer;
                buffer = new char[bufsize];
        }
            for(int i=0;i<vec.size();i++){
                     strbuf1 = to_string(vec[i]);
                      str = str + strbuf1 + strbuf2;
              }
              //if (str.lenght() >= 1024){
                      //while(str.lenght)
              //}
            strcpy(buffer, (char*)str.c_str());
            str = "";
    }

    send(server, buffer, bufsize, 0);
    vec.clear();
    memset(buffer, 0, sizeof(buffer));

    cout << "Client: ";
    counter = 0;
    do {
            recv(server, buffer, bufsize, 0);
            cout << buffer << " ";
            counter++;
            counter == 1? left_border = atoi(buffer) :\
            right_border = atoi(buffer);
            if (*buffer == '#') isExit = true;
    } while (counter != 2 && *buffer != '#');
} while (!isExit);

cout << "\n=> Connection terminated... " << inet_ntoa(adr.sin_addr);
close(server);
```

```
        cout << "\nGoodbye..." << endl;
        isExit = false;
        //break;
    }

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

Файл client.cpp

```
#include <iostream>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <math.h>
#include <stdlib.h>
#include <unistd.h>

using namespace std;

int main()
{
    char a;
    int client;
    int portNum = 1500;
    int bufsize = 1024;
    int counter = 0;
    int left_border;
    int right_border;
    char* buffer = new char[bufsize];
    bool isExit = false;
    char* ip = "127.0.0.1";

    struct sockaddr_in direc;

    if ((client = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        cout << "\nError creating socket..." << endl;
```

```
    exit(0);
}

cout << "\nSocket created successfully..." << endl;
direc.sin_family = AF_INET;
direc.sin_port = htons(portNum);
inet_pton(AF_INET, ip, &direc.sin_addr);

if (connect(client,(struct sockaddr *)&direc, sizeof(direc)) == 0)
    cout << "Connection to the server " <<\
    inet_ntoa(direc.sin_addr) << endl;

cout << "Awaiting confirmation from the server..." << endl;
recv(client, buffer, bufsize, 0);

cout << "\n=> Enter # to terminate the connection\n" << endl;

do {
    cout << "Server: enter left and right border\nClient: ";
    counter = 0;
    do {
        memset(buffer, 0, sizeof(buffer));
        cin >> buffer;

        send(client, buffer, bufsize, 0);
        counter++;
        counter == 1? left_border = atoi(buffer) : \
        right_border = atoi(buffer);
        if (*buffer == '#') {
            isExit = true;
            break;
        }
    } while (counter != 2);

    if (right_border - left_border > 1024){
            (right_border >= 2) && (right_border > left_border) \
            ? bufsize = abs((right_border - left_border)*2) : \
            bufsize = 1024;
            delete [] buffer;
            buffer = new char[bufsize];
```

```
        }
        cout << "Server: ";
        recv(client, buffer, bufsize, 0);
        cout << buffer;
        if (*buffer == '#') isExit = true;
        cout << endl;

    } while (!isExit);
    cout << "=> Connection terminated.\nGoodbye\n";

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