

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ  
И ИНФОРМАТИКИ

Кафедра технологий программирования

ЛАБОРАТОРНАЯ РАБОТА 7  
ПО ДИСЦИПЛИНЕ «СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ»

Организация обмена данными между процессами

Методические указания по выполнению лабораторной работы

Минск 2023

## СОДЕРЖАНИЕ

Введение .....	3
1 Методические рекомендации.....	4
1.1 Основной учебный материал .....	4
1.2 Сборка программ с помощью системы сборки GNU Autotools ..	4
1.2.1 Утилита make и Makefile .....	7
2 Методические указания и задания.....	8
2.1 Методические указания.....	8
2.1.1 Критерии оценивания .....	8
2.1.2 Отчет по лабораторной работе.....	9
2.2 Задания.....	12
2.2.1 Задание 1.....	12
2.2.2 Задание 2.....	14
2.2.3 Задание 3.....	15

## ВВЕДЕНИЕ

Целью работы является получение навыков разработки программ на языке C с применением Linux API для организации обмена данными между процессами. Для достижения поставленной цели необходимо решить следующие задачи:

- получить навыки разработки приложений, реализующих обмен информацией между процессами средствами неименованных или именованных каналов в программах на языке C в операционных системах семейства Linux;
- получить навыки организации обмена данными между процессами средствами очередей сообщений (System V или POSIX) в программах на языке C в операционных системах семейства Linux;
- получить навыки организации обмена данными между процессами средствами разделяемой памяти с синхронизацией посредством семафоров (System V или POSIX) в программах на языке C в операционных системах семейства Linux;
- изучить методические рекомендации и обязательный теоретический материал для выполнения лабораторной работы.

## 1 Методические рекомендации

В данном разделе представлены рекомендации по выполнению лабораторной работы.

### 1.1 Основной учебный материал

Учебный материал для выполнения лабораторной работы изложен в источниках:

а) Гунько А.В. Системное программирование в среде Linux: учебное пособие / А.В. Гунько. – Новосибирск: Изд-во НГТУ, 2020. – 235 (🔗 <https://disk.yandex.ru/i/lh2bFAEfYwZ5nw>):

1) глава 6, стр. 85;

б) Иванов Н.Н. Программирование в Linux. Самоучитель. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2012. — 400 с.: ил (🔗 <https://disk.yandex.ru/i/vuhTyuX8wubLag>):

1) главы 19-24, стр. 251.

в) Jack-Benny Persson. Linux System Programming Techniques. — Packt Publishing, 2021. — 432 p. (🔗 <https://disk.yandex.ru/i/a6m3iSBTG59SqA>):

1) глава 10, стр. 251.

### 1.2 Сборка программ с помощью системы сборки GNU Autotools

*Autotools*, или *система сборки GNU*— это набор программных средств, предназначенных для поддержки переносимости исходного кода программ между UNIX-подобными системами.

Перенос кода с одной системы на другую может оказаться непростой задачей. Различные реализации компилятора языка Си могут существенно различаться: некоторые функции языка могут отсутствовать, иметь другое имя или находиться в разных библиотеках. Программист может решить эту задачу, используя макросы и директивы препроцессора, например `#if`, `#ifdef` и прочие.

Но в таком случае пользователь, компилирующий программу на своей системе, должен будет определить все эти макросы, что не так просто,


поскольку существует множество разных дистрибутивов и вариаций систем. *Autotools* вызываются последовательностью команд `./configure && make && make install` и решают эти проблемы автоматически.


Система сборки *GNU Autotools* является частью *GNU toolchain* и широко используется во многих проектах с открытым исходным кодом. Средства сборки распространяются в соответствии с GNU General Public License с возможностью использования их в коммерческих проектах.


- Autoconf;
- Automake;
- Libtool;
- Gnulib;
- Другие средства:
  - make;
  - gettext;
  - pkg-config;
  - gcc;
  - binutils;


На рисунке 1.1 представлена схема работы `autoconf` и `automake`.


Дополнительный учебный материал по применению *Autotools*, первые 5 из которых изучить перед выполнением задания 2 и 3:


а)  <https://earthly.dev/blog/autoconf/>.


б)  [https://src\\_prepare.gitlab.io/devmanual-mirror/general-concepts/autotools/index.html](https://src_prepare.gitlab.io/devmanual-mirror/general-concepts/autotools/index.html).


в)  <https://mj-7.medium.com/how-to-package-your-software-in-linux-using-gnu-au>


г)  <https://www.programmingsought.com/article/10226793563/>.


д)  <https://youtu.be/3X00d9Qyc34?si=DrQqrhio270pIoeg>.

е)  <http://www.h-wrt.com/ru/mini-how-to/autotoolsSimpleProject>.

ж)  <https://eax.me/autotools/>

и)  [https://help.ubuntu.ru/wiki/using\\_gnu\\_autotools](https://help.ubuntu.ru/wiki/using_gnu_autotools)

к)  <https://opensource.com/article/19/7/introduction-gnu-autotools>

л)  <https://www.gnu.org/software/automake/faq/autotools-faq.html>

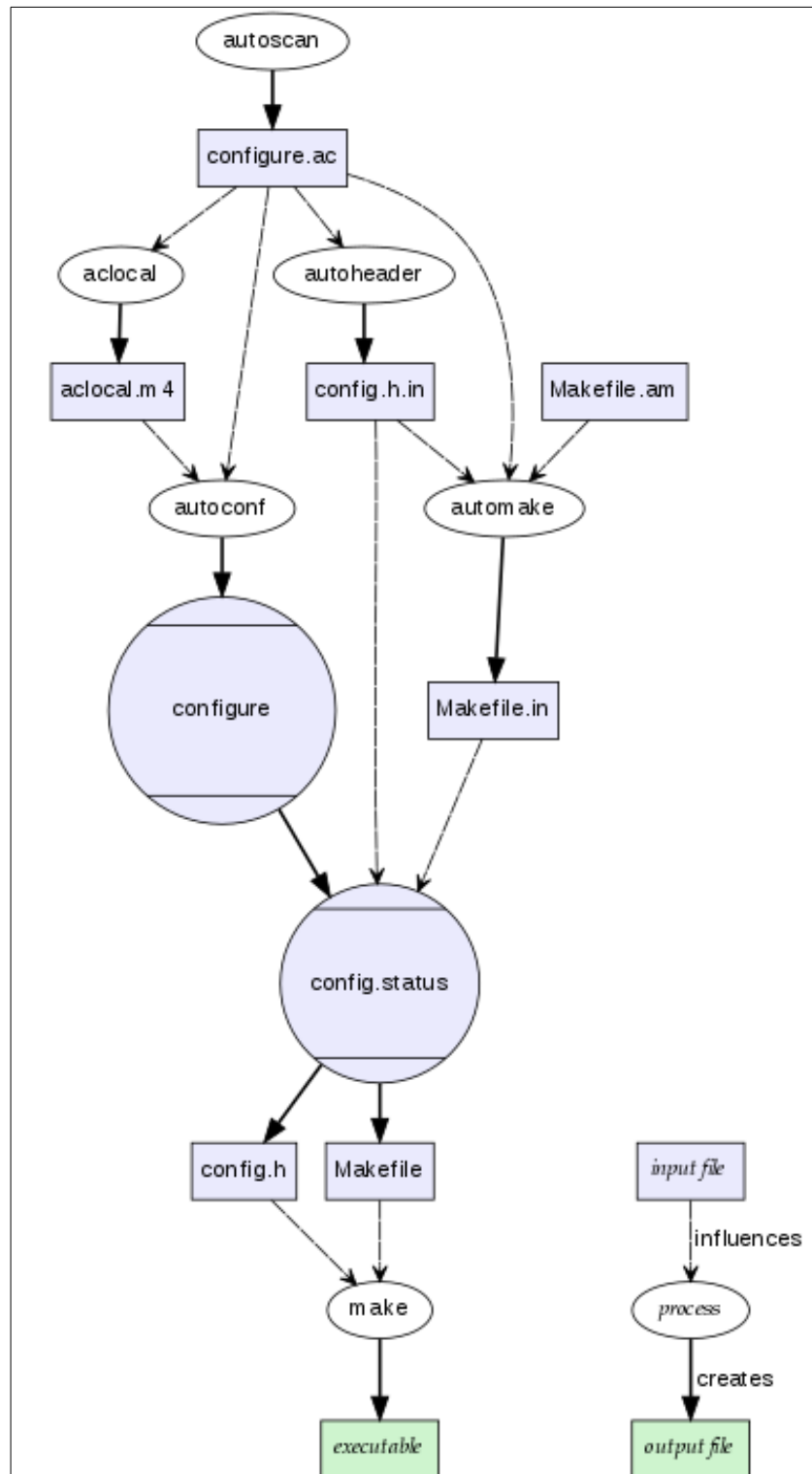





Рисунок 1.1 — Алгоритм сборки пакета с помощью Autotools

### 1.2.1 Утилита **make** и **Makefile**

Утилита **make** автоматически определяет, какие части большой программы должны быть перекомпилированы, и выполняет необходимые для этого действия. Наиболее часто **make** используется для компиляции С-программ. Отметим, что **make** можно использовать с любым языком программирования. Более того, применение утилиты **make** не ограничивается программами. Можно использовать её для описания любой задачи, где некоторые файлы должны автоматически порождаться из других всегда, когда те изменяются.

Ознакомьтесь с рекомендациями по разработке файлов **Makefile** и применению утилиты **make**:

- а)  <https://parallel.uran.ru/book/export/html/16>.
- б)  [http://linux.yaroslavl.ru/docs/prog/gnu\\_make\\_3-79\\_russian\\_manual.html](http://linux.yaroslavl.ru/docs/prog/gnu_make_3-79_russian_manual.html)
- в)  [https://www.gnu.org/software/make/manual/html\\_node/index.html](https://www.gnu.org/software/make/manual/html_node/index.html)

## 2 Методические указания и задания

### 2.1 Методические указания

Общие рекомендации по выполнению заданий лабораторной работы:

- а) Проект для любого из заданий может быть реализован в виде консольного приложения в среде ОС Ubuntu, CentOS или других средствами компилятора gcc версии не ниже 4.
- б) Проекты в заданиях 1-3 являются модификацией задания № 3 лабораторной работы № 6, в которой обмен данными между родительской и дочерними программами производится согласно одному из механизмов, требуемому в каждом задании.
- в) Проект для задания 1 может быть реализован с помощью анонимных (программных) каналов или каналов FIFO.
- г) Проект для задания 2 в качестве средства обмена использует очереди сообщений (System V или POSIX).
- д) Проект для задания 3 реализуется средствами разделяемой памяти с синхронизацией посредством семафоров (System V или POSIX).
- е) Проект для каждого из заданий должен предусматривать обработку исключительных ситуаций (отсутствие или неверное количество входных параметров, ошибки открытия входного и/или выходного файла, ошибки чтения и записи).

#### 2.1.1 Критерии оценивания

##### Оценка 4-5

Выполнено задания 1-2. Структура программы в задании 2 соответствует *КИС*, содержит **Makefile** для сборки, созданный вручную. В **Makefile** продемонстрировать использование переменных, автоматических переменных, шаблонных имен и, при необходимости, поиск пререквизитов по каталогам, абстрактные цели. Представлен отчет, исходный код проекта в **git**-репозитории. Отчет, файлы протоколов команд и меток времени без ошибок. Лабораторная работа сдана с задержкой в 1-2 недели.

##### Оценка 6-7

Выполнено задания 1-3. Структура программы соответствует *КИС* и для сборки применяется система сборки **Autotools** с автоматической генерацией



**Makefile**, а так же должен быть создан пакет проекта с помощью команды **make distcheck**.

Проект по каждому из заданий опубликовать в отдельной ветке и в отдельном каталоге. Для заданий 1 и 2 обязательны тесты. Выполнить сборку проектов по заданиям 1-2 с помощью *Github Actions*.

Представлен отчет, ответы на контрольные вопросы, исходные коды скриптов в **git**-репозитории. Отчет, исходный код может содержать незначительные ошибки. Лабораторная работа сдана с задержкой в 1 неделю.

### **Оценка 8-9**

Выполнены задания 1-3 на отличном уровне. Структура программы соответствует *КИС* и для сборки применяется система сборки **Autotools** с автоматической генерацией **Makefile**, а так же должен быть создан пакет проекта с помощью команды **make distcheck**.


Проект по каждому из заданий опубликовать в отдельной ветке и в отдельном каталоге. Для заданий 1 и 2 обязательны тесты. Выполнить сборку проектов по заданиям 1-3 с помощью *Github Actions*.

Представлен отчет, ответы на контрольные вопросы, исходные коды скриптов в **git**-репозитории. Отчет, исходный код не содержат ошибок. Лабораторная работа сдана в срок.

### **2.1.2 Отчет по лабораторной работе**

Отчет по лабораторной работе должен быть опубликован в репозитории и отвечать требованиям:

1. Отчет по лабораторной работе состоит из письменного отчета и кода программ, опубликованных в репозиторий
2. Письменный отчет содержит цель работы.
3. Письменный отчет включает вариант задания.
4. Письменный отчет добавить описание ключевых моментов реализации и тестов.
5. Исходный код программ для каждого задания опубликовать в подкаталоге **/src** соответствующего каталога проекта (задания) и в соответствующей ветке репозитория.

Ссылка на репозиторий для лабораторной работы 7 доступна в курсе  «Системное программирование».

В файле `README` в корневом каталоге проекта на *github* должна быть ссылка на отчёт и краткие сведения о выполненных заданиях (проектах).

Отчет опубликовать во внешнем хранилище или в репозитории в каталоге `/docs`. **! Отчёт должен результаты тестов по каждой программе и ответы на контрольные вопросы.**

## Требования к файлу `README` репозитория

Пример оформления файла `README` может быть таким:

```
1  # Overview
2
3  Отчет по лабораторной работе.
4
5  # Usage
6
7  // Заменить <<link>> и <<folder>> на соответствующие ссылки и названия
8  To check, please, preview report by <<link>> and script files in
   <<folder>>.
9
10 # Author
11
12 Your name and group numbers средствами. разделяемой памяти с синхронизацией
13 посредством семафоров (System V или POSIX)
```

## Защита отчета по лабораторной работе

Каждая лабораторная работа содержит тексты задач и контрольные вопросы, ответы на которые проверяются преподавателем при приёме работы у студента.

Выполнение студентом лабораторной работы и сдача её результатов преподавателю происходит следующим образом:

1. Студент выполняет разработку программ.

В ходе разработки студент обязан следовать указаниям к данной задаче (в случае их наличия). Исходные тексты программ следует разрабатывать в соответствии с требованиями к оформлению для программ на языке C.

2. Студент выполняет самостоятельную проверку исходного текста каждой разработанной программы и правильности её работы, а также свои знания по теме лабораторной работы.

Исходные тексты программ должны соответствовать требованиям к оформлению, приведённым в приложении. Недопустимо отсутствие в тексте программы следующих важных элементов оформления: спецификации программного файла и подпрограмм, а также отступов, показывающих структурную вложенность языковых конструкций.

Для проверки правильности работы программы студенту необходимо разработать набор тестов и на их основе провести тестирование программы. Тестовый набор должен включать примеры входных и выходных данных, приведённые в тексте задачи, а также тесты, разработанные студентом самостоятельно.

Самостоятельная проверка знаний по теме лабораторной работы выполняется с помощью контрольных вопросов и заданий, приведённых в конце текста лабораторной работы.

3. Студент защищает разработанные программы. Защита заключается в том, что студент должен ответить на вопросы преподавателя, касающиеся разработанной программы, и контрольные вопросы.

К защите необходимо представить исходные тексты программ, оформленных в соответствии с требованиями, и протоколы тестирования каждой программы, подтверждающие правильность ее работы.

Протокол тестирования включает в себя тест (описание входных данных и соответствующих им выходных данных), описание выходных данных, полученных при запуске программы на данном тесте, и отметку о прохождении теста. Тест считается пройденным, если действительные результаты работы программы совпали с ожидаемыми.

Пример оформления протокола тестирования программы на определение количества вхождений слов в строке представлен в таблице 2.1.


Программы, не прошедшие тестирование, к защите не принимаются. В случае неверной работы программы хотя бы на одном тесте студент обязан выполнить отладку программы для поиска и устранения ошибки.

Таблица 2.1 — Пример протокола тестирования задачи на определение количества вхождений слова в строке

№ п/п	Входные данные	Ожидаемые выходные данные	Действительные выходные данные	Тест пройден
1	foo bar foo bar	bar: 2 foo: 2	bar: 2 foo: 2	Да
2	test record created	test: 1 record: 1 created: 1	test: 1 record: 1 created: 1	Да
3	1 2 3 4 5 1 2 3 4 5 6 7 8 9	1: 2 2: 2 3: 2 4:2 5: 2	1: 2 2: 2 3: 2 4:2 5: 2	Да
4	Hello World	Hello: 1 World: 1	Hello: 1 newline World: 1	Да

## 2.2 Задания

Общие требования к проектам программ для заданий 1-3:

- Структура программы должна соответствовать модели КИС  <https://www.opennet.ru/docs/RUS/zlp/002.html>.

- Сборка выполняться с помощью утилиты **make**. При написании **Makefile** продемонстрировать использование шаблонов, переменных, префиксов **для оценки 4-5**.

- Для сборки использовать систему сборки **Autotools** с автоматической генерацией **Makefile** **для оценки 6-9**.

- Сформировать пакет с открытыми исходными кодами в формате **.tgz** (**.tar.gz**) **для оценки 6-9**.

- Продемонстрировать автоматическую сборку с *Github Actions* и результаты выполнения приложения (для оценки 6-9).

### 2.2.1 Задание 1

Проект в данном задании является модификацией проекта из задания № 3 лабораторной работы № 6. Необходимо изменить проект из задания № 3 ла-

бораторной работы № 6 и реализовать **многозадачную** версию приложения при использовании *программных каналов* или клиент-серверного приложения при использовании *каналов FIFO*. Вариант с использованием именованных каналов является заданием повышенной сложности. Для отладки использовать отладчик `gdb` ☞ <http://gun.cs.nstu.ru/ssw/gdb-html.tgz>.

Порядок выполнения задания:

1. Изучить параграф 6.2, стр. 87 (☞ <https://disk.yandex.ru/i/1h2bFAEfYwZ5nw>).

2. Изучить главу 10, стр. 251 (☞ <https://disk.yandex.ru/i/a6m3iSBTG59SqA>) и другие материалы из параграфа «Список литературы:MainContent».

3. Изучить примеры кода для анонимных (программных) каналов или именованных каналов по адресу ☞ <http://gun.cs.nstu.ru/ssw/Pipes/>.

4. Написать программу, предусматривающую обмен данными между родительской и дочерними программами или между клиентом и сервером только через каналы (анонимные или именованные FIFO соответственно), выбрав средство реализации межпроцессных коммуникаций.

5. Модифицировать и отладить в соответствии с выбранным средством коммуникации программу из задания 3 лабораторной работы № 6, реализующую порожденный (дочерний) процесс. При необходимости модифицировать ее в отдельную программу-сервер.

6. Модифицировать и отладить в соответствии с выбранным средством коммуникации программу из задания 3 лабораторной работы № 6, реализующую родительский процесс. При необходимости модифицировать ее в отдельную программу-клиент.

7. Результатом работы программы является совокупность выходных текстовых файлов, соответствующих совокупности (2 и более) входных файлов, содержащих текст, обработанный согласно вариантам, и возвращаемое значение – количество выполненных операций или –1 в случае ошибки.

8. При обработке исключительных ситуаций сообщения об ошибках выводятся на терминал.

9. Проект должен предусматривать обмен данными между родительской и дочерними программами или между клиентом и сервером только через каналы (программные или FIFO соответственно).

10. Результатом выполнения лабораторной работы считается демонстрация работы программы и обработки исключительных ситуаций преподавателю.

### 2.2.2 Задание 2

Проект в данном задании является модификацией проекта из задания № 1 текущей лабораторной работы. Необходимо изменить проект из задания № 1 и реализовать **многозадачную** версию приложения с обеспечением обмена данными между процессами средствами очередей. Вариант с использованием очередей POSIX является вариантом повышенной сложности. Для отладки использовать отладчик `gdb` ☞ <http://gun.cs.nstu.ru/ssw/gdb-html.tgz>.

Порядок выполнения задания:

1. Изучить параграф 6.3, стр. 105 и параграф 6.4, стр. 131 (☞ <https://disk.yandex.ru/i/lh2bFAEfYwZ5nw>).

2. Изучить главу 10, стр. 251 (☞ <https://disk.yandex.ru/i/a6m3iSBTG59SqA>) и другие материалы из параграфа «Список литературы:MainContent».

3. Изучить примеры кода для очередей System V (☞ <http://gun.cs.nstu.ru/ssw/SysVmsg>) или очередей POSIX по адресу ☞ <http://gun.cs.nstu.ru/ssw/PosixMsg>.

4. Написать программу, предусматривающую обмен данными между родительской и дочерними программами или между клиентом и сервером с помощью очереди сообщений, выбрав средство реализации межпроцессных коммуникаций.

5. Проект может быть реализован с применением очередей сообщений System V или POSIX. Второй способ соответствует заданию повышенной сложности.

6. Модифицировать и отладить в соответствии с выбранным средством коммуникации программу из задания № 1, реализующую порожденный (дочерний) процесс. При необходимости модифицировать ее в отдельную программу-сервер.

7. Модифицировать и отладить в соответствии с выбранным средством коммуникации программу из задания № 1, реализующую родительский

процесс. При необходимости модифицировать ее в отдельную программу-клиент.

8. Результатом работы программы является совокупность выходных текстовых файлов, соответствующих совокупности (2 и более) входных файлов, содержащих текст, обработанный согласно вариантам, и возвращаемое значение – количество выполненных операций или  $-1$  в случае ошибки.

9. При обработке исключительных ситуаций сообщения об ошибках выводятся на терминал.

10. Проект должен предусматривать обмен данными между родительской и дочерними программами или между клиентом и сервером только через очереди сообщений (System V или POSIX)

11. Результатом выполнения лабораторной работы считается демонстрация работы программы и обработки исключительных ситуаций преподавателю.

### 2.2.3 Задание 3



Проект в данном задании является модификацией проекта из задания № 2 текущей лабораторной работы. Необходимо изменить проект из задания № 2 и реализовать **многозадачную** версию приложения с обеспечением обмена данными между процессами средствами разделяемой памяти с синхронизацией посредством семафоров (System V или POSIX). Вариант с использованием очередей POSIX является вариантом повышенной сложности. Для отладки использовать отладчик `gdb` ☞ <http://gun.cs.nstu.ru/ssw/gdb-html.tgz>.

Порядок выполнения задания:

1. Изучить параграф 6.3, стр. 105 и параграф 6.4, стр. 131 (☞ <https://disk.yandex.ru/i/lh2bFAEfYwZ5nw>).

2. Изучить главу 10, стр. 251 (☞ <https://disk.yandex.ru/i/a6m3iSBTG59SqA>) и другие материалы из параграфа «Список литературы:MainContent».

3. Изучить примеры кода для разделяемой памяти (☞ <http://gun.cs.nstu.ru/ssw/SysVshm>) и семафоров System V (☞ <http://gun.cs.nstu.ru/ssw/SysVSem>).

4. Если выполняете задание повышенной сложности, то изучить примеры кода для разделяемой памяти ( <http://gun.cs.nstu.ru/ssw/PosixShm>) и семафоров POSIX ( <http://gun.cs.nstu.ru/ssw/PosixSem>).

5. Написать программу, предусматривающую, что между родительской и дочерними программами или между клиентом и сервером производится через разделяемую память, а синхронизация программ – посредством семафоров.

6. Проект может быть реализован с применением очередей сообщений System V или POSIX. Второй способ соответствует заданию повышенной сложности.

7. Проект должен предусматривать обработку исключительных ситуаций (отсутствие или неверное количество входных параметров, ошибки создания семафоров и разделяемой памяти, ошибки управления семафорами, чтения и записи из/в разделяемую память).

8. Проект должен предусматривать обмен данными между родительской и дочерними программами или между клиентом и сервером только через разделяемую память (System V или POSIX).

9. Модифицировать и отладить в соответствии с выбранным средством коммуникации программу из задания № 2, реализующую порожденный (дочерний) процесс. При необходимости модифицировать ее в отдельную программу-сервер.

10. Модифицировать и отладить в соответствии с выбранным средством коммуникации программу из задания № 2, реализующую родительский процесс. При необходимости модифицировать ее в отдельную программу-клиент.

11. Результатом работы программы является совокупность выходных текстовых файлов, соответствующих совокупности (2 и более) входных файлов, содержащих текст, обработанный согласно вариантам, и возвращаемое значение – количество выполненных операций или –1 в случае ошибки.

12. При обработке исключительных ситуаций сообщения об ошибках выводятся на терминал.

13. Результатом выполнения лабораторной работы считается демонстрация работы программы и обработки исключительных ситуаций преподавателю.



## Контрольные вопросы

1. Как именуются и где размещаются объекты IPC POSIX?
2. Опишите константы, используемые при создании и открытии объектов IPC?
3. Опишите константы, определяющие права доступа при создании объектов IPC.
4. Опишите последовательность логических операций при создании/открытии объекта IPC.
5. Опишите порядок проверки прав доступа при открытии объектов IPC.
6. Опишите отличия очереди сообщений стандарта POSIX от стандарта System V.
7. Какова структура сообщения в очереди сообщений стандарта POSIX?
8. Опишите функцию создания очереди сообщений POSIX.
9. Опишите особенности закрытия и удаления очереди сообщений POSIX.
10. Опишите функцию отправки сообщений в очередь сообщений POSIX.
11. Опишите функцию чтения сообщений из очереди сообщений POSIX.
12. Какие существуют системные ограничения на очереди сообщений POSIX?
13. Что представляют собой именованные и размещаемые в памяти семафоры POSIX?
14. Опишите функцию создания/открытия именованного семафора POSIX.
15. Опишите механизмы совместного использования разделяемой памяти для неродственных процессов.
16. Перечислите варианты использования функции `mmap`.
17. В чем заключаются преимущества и ограничения при отображении файла в память?
18. Опишите системные ограничения на объекты разделяемой памяти POSIX.