

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ  
И ИНФОРМАТИКИ

Кафедра технологий программирования

ЛАБОРАТОРНАЯ РАБОТА 6  
ПО ДИСЦИПЛИНЕ «СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ»

Многозадачность и многопоточность в Linux

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 .....	5
2 Методические указания и задания.....	8
2.1 Методические указания.....	8
2.1.1 Критерии оценивания .....	8
2.1.2 [Pleaseinsertintopreamble][Pleaseinsertintopreamble][Pleaseinsertintopreamb [Pleaseinsertintopreamble][Pleaseinsertintopreamble] [Pleaseinsertintopreamble][Pleaseinsertintopreamble][Pleaseinsertintopreamble] [Pleaseinsertintopreamble][Pleaseinsertintopreamble][Pleaseinsertintopreamble]	
2.1.2.1 [Pleaseinsertintopreamble][Pleaseinsertintopreamble][Pleaseinsertintoprea [Pleaseinsertintopreamble] [Pleaseinsertintopreamble][Pleaseinsertintoprea README [Pleaseinsertintopreamble][Pleaseinsertintopreamble][Pleaseinse	
2.1.2.2 Защита отчета по лабораторной работе .....	10
2.2 Задания.....	12
2.2.1 Задание 1.....	12
2.2.2 Задание 2.....	12
2.2.3 Задание 3.....	13
2.2.4 Задание 4.....	14
2.3 Варианты к заданиям 1-2.....	16
2.4 Контрольные вопросы .....	20

## ВВЕДЕНИЕ

Целью работы является получение навыков разработки многозадачных и многопоточных приложений на языке С в операционных системах Linux. Для достижения поставленной цели необходимо решить следующие задачи:

- получить навыки разработки многозадачных приложений на языке С в операционных системах семейства Linux;
- получить навыки разработки многопоточных приложений на языке С в операционных системах семейства Linux;
- изучить теоретические материалы по темам «Многозадачное программирование в Linux» и «Многопоточное программирование в Linux»;
- применить навыки к проектированию пакета приложения с реализацией библиотеки из задания 2 лабораторной работы № 5.

## 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)  Лекции 12-13

б) Гунько А.В. Системное программирование в среде Linux: учебное пособие / А.В. Гунько. – Новосибирск: Изд-во НГТУ, 2020. – 235:

1) глава 5, стр. 66;

2) глава 7, стр. 165.

в) Лав Р. Linux. Системное программирование. 2-е изд. — СПб.: Питер, 2014. — 448 с.:ил. — (Серия «Бестселлеры O'Reilly»):

1) главы 5-7, стр. 171.

### 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-autotools>
- г)  <https://www.programmersonsought.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>

### 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.

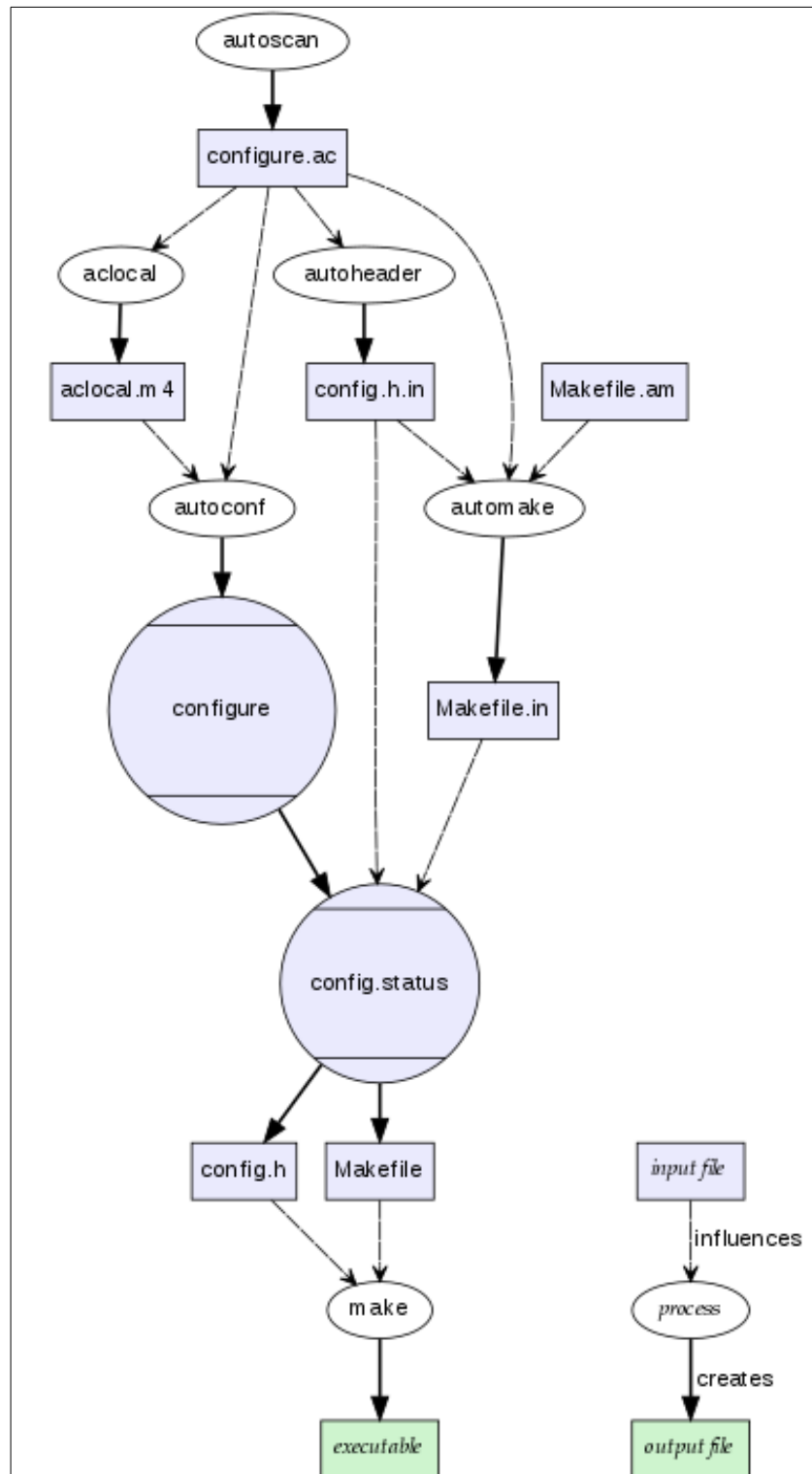





Рисунок 1.1 — Алгоритм сборки пакета с помощью Autotools

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) 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 1-3. 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, задание 3 без реализации дополнительной функции. 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 1-4 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 6 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.**

#### 2.1.2.1 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 number.
13
14 # Additional Notes
15
16 // СКОПИРОВАТЬ И ВСТАВИТЬ ССЫЛКУ НА СВОЙ РЕПОЗИТОРИЙ, НАПРИМЕР
17 https://github.com/maryiad/lab6-task1-gr16-david

```

### 2.1.2.2 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

Продолжение на след. стр.


Продолжение таблицы 2.1

4	Hello World	Hello: 1 World: 1	Hello: 1    newline World: 1	Да
---	-------------	----------------------	---------------------------------	----

## 2.2 Задания



### 2.2.1 Задание 1


Требования к структуре и сборке проекта многозадачной программы:

- 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, пререквизитов.

Выполнить следующие этапы:

1. Изучить главу 5, с. 66  <https://disk.yandex.ru/i/1h2bFAEfYwZ5nw> и главы 5-6, с. 171  <https://disk.yandex.ru/i/NSY1w0nBr-IJFg>.

2. Изучить примеры кода  <http://gun.cs.nstu.ru/ssw/Linuxprog>.

3. Написать и отладить многозадачную программу согласно варианту к текущей лабораторной работе.


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

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

6. Представить тесты, детализацию реализации в README репозитория.



### 2.2.2 Задание 2


Требования к структуре и сборке проекта программы:

- 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, пререквизитов.

Выполнить следующие этапы:

1. Изучить главу 7, с. 165  <https://disk.yandex.ru/i/1h2bFAEfYwZ5nw> и главу 7, с. 245  <https://disk.yandex.ru/i/NSY1w0nBr-IJFg>.

2. Изучить примеры кода  <http://gun.cs.nstu.ru/ssw/Threads>.


3. Написать и отладить многопоточную программу согласно варианту из задания 1 к текущей лабораторной работе.

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


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

6. Представить тесты, детализацию реализации в README репозитория.

### 2.2.3 Задание 3

Вариант соответствует варианту задания 1 лабораторной работы 5. Необходимо изменить проект из задания 2 лабораторной работы 5 и реализовать **многозадачную** версию приложения. Для отладки использовать отладчик `gdb`  <http://gun.cs.nstu.ru/ssw/gdb-html.tgz>.

Требования к проекту программы:

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

— Используется статическая библиотека из задания 2 лабораторной работы 5. Сборка выполняться с помощью утилиты `make`. При написании `Makefile` продемонстрировать использование шаблонов, переменных, пререквизитов **для оценки 4-5**.

— Используется динамическая библиотека из задания 2 лабораторной работы 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).

Порядок выполнения задания:

1. Написать программу, запускающую программу из задания № 2 лабораторной работы № 5 в количестве экземпляров, соответствующем количеству входных файлов, ждущую завершения всех экземпляров и получающую (выводящую на терминал) коды их завершения.

2. Проект использует версию библиотеки (статическую или динамическую) из задания 2 лабораторной работы 5.

3. Проект требует написания родительского приложения для запуска программы из задания 2 лабораторной работы 5 для обработки каждого из нескольких входных файлов.

4. Программа должна содержать функции порождения дочернего процесса (в нужном количестве экземпляров), запуска в каждом дочернем процессе программы из задания 2 лабораторной работы 5 с необходимыми ей аргументами командной строки, ожидания завершения каждого дочернего процесса и получения его кода завершения.


5. Результатом работы программы являются выходные текстовые файлы, содержащие текст, обработанный согласно вариантам, и возвращаемые дочерними процессами значения – количество выполненных операций или -1 в случае ошибки.

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

7. Результатом выполнения лабораторной работы считается демонстрация работы родительской программы с несколькими входными и соответствующими им выходными файлами и обработки исключительных ситуаций преподавателю.

#### 2.2.4 Задание 4

В ходе выполнения задания необходимо изменить проект из задания 3 текущей лабораторной работы и реализовать **многопоточную** версию

приложения. Для отладки использовать отладчик `gdb`  <http://gun.cs.nstu.ru/ssw/gdb-html.tgz>.

Требования к проекту многопоточной программы соответствуют требованиям задания 3 текущей лабораторной работы.

Порядок выполнения:

Порядок выполнения задания:

1. Переписать программу родительского процесса из задания № 3, запускающую потоки, реализующие функциональность программы из задания № 2 лабораторной работы № 5. Количество потоков должно соответствовать числу входных файлов. Программа должна ждать завершения всех потоков и выводить на терминал коды их завершения.

2. Проект является модификацией задания № 3, родительское приложение которой преобразуется в функцию главного потока, а дочернее – в функцию порождаемого потока, реализующую функциональность программы из задания № 2 лабораторной работы № 5 для обработки каждого из нескольких входных файлов.

3. Проект должен содержать функции порождения потока (в нужном количестве экземпляров), запуск в каждом порожденном потоке функции, реализующей функциональность из задания № 2 лабораторной работы № 5 с необходимым ей параметром, ожидания завершения каждого потока и получения его кода завершения.

4. Программа должна предусматривать синхронизацию потоков посредством мьютексов. Обмен данными между потоками может осуществляться через глобальный массив структур.

5. Проект должен предусматривать обработку исключительных ситуаций (недостаточное количество аргументов командной строки, невозможность захвата мьютекса, загрузки динамической библиотеки и импорта функции из нее).

6. Проект рекомендуется реализовать в 2 этапа:

— Преобразование дочернего процесса задания № 3 в функцию (далее оформляемую как функция порождаемого потока).

— Преобразование родительского процесса задания № 3 в функцию `main` (головного потока), содержащую все необходимые системные вызовы.

7. Результатом работы программы являются выходные текстовые файлы, содержащие текст, обработанный согласно вариантам, и возвращаемые значения – количество выполненных операций или -1 в случае ошибки.

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

9. Результатом выполнения лабораторной работы считается демонстрация работы родительской программы с несколькими входными и соответствующими им выходными файлами и обработки исключительных ситуаций преподавателю.

### **2.3 Варианты к заданиям 1-2**

Общий вариант для заданий 1 и 2. Задавать количество процессов или потоков с клавиатуры.

#### **Вариант 1**

Отсортировать в заданном каталоге (аргумент 1 командной строки) и во всех его подкаталогах файлы по следующим критериям (аргумент 2 командной строки, задаётся в виде целого числа): 1 – по размеру файла, 2 – по имени файла. Записать отсортированные файлы в новый каталог (аргумент 3 командной строки). Процедуры копирования должны запускаться в отдельном процессе для каждого копируемого файла с использованием функций `read()` и `write()`. Каждый процесс выводит на экран свой `pid`, полный путь, имя копируемого файла и число скопированных байт. Число запущенных процессов в любой момент времени не должно превышать `N` (вводится пользователем). Проверить работу программы для каталога `/usr/include/` и `N=6`.

#### **Вариант 2**

Написать программу синхронизации двух каталогов, например `Dir1` и `Dir2`. Пользователь задаёт имена `Dir1` и `Dir2` в качестве первого и второго аргумента командной строки. В результате работы программы файлы, имеющиеся в `Dir1`, но отсутствующие в `Dir2`, должны скопироваться в `Dir2` вместе с правами доступа. Процедуры копирования должны запускаться в отдельном процессе для каждого копируемого файла с использованием функций `read()` и `write()`. Каждый процесс выводит на экран свой `pid`, полный путь, имя копируемого файла и число скопированных байт. Число запущенных процессов в любой момент времени не должно превышать



N (вводится пользователем). Проверить работу программы для каталога /usr/include/ и любого другого каталога в /home/, N=6.

### **Вариант 3**

Найти в заданном каталоге (аргумент 1 командной строки) и всех его подкаталогах заданный файл (аргумент 2 командной строки). Вывести на консоль полный путь к файлу имя файла, его размер, дату создания, права доступа, номер индексного дескриптора. Вывести также общее количество просмотренных каталогов и файлов. Процедура поиска для каждого подкаталога должна запускаться в отдельном процессе. Каждый процесс выводит на экран свой `pid`, полный путь, имя и размер просмотренного файла, общее число просмотренных файлов в подкаталоге. Число запущенных процессов в любой момент времени не должно превышать N (вводится пользователем). Проверить работу программы для каталога /usr найти файл `stdio.h`, N=6.

### **Вариант 4**

Для заданного каталога (аргумент 1 командной строки) и всех его подкаталогов вывести в заданный файл (аргумент 2 командной строки) и на консоль имена файлов, их размер и дату создания, удовлетворяющих заданным условиям: 1 – размер файла находится в заданных пределах от N1 до N2 (N1,N2 задаются в аргументах командной строки), 2 – дата создания находится в заданных пределах от M1 до M2 (M1,M2 задаются в аргументах командной строки). Процедура поиска для каждого подкаталога должна запускаться в отдельном процессе. Каждый процесс выводит на экран свой `pid`, полный путь, имя и размер просмотренного файла, общее число просмотренных файлов в подкаталоге. Число запущенных процессов в любой момент времени не должно превышать N (вводится пользователем). Проверить работу программы для каталога /usr/ размер 31000 31500 дата с 01.01.1970 по текущую дату, N=6.

### **Вариант 5**

Подсчитать суммарный размер файлов в заданном каталоге (аргумент 1 командной строки) и для каждого его подкаталога отдельно. Вывести на консоль и в файл (аргумент 2 командной строки) название подкаталога, количество файлов в нём, суммарный размер файлов, имя файла с наибольшим размером. Процедура просмотра для каждого подкаталога должна запускаться в отдельном процессе. Каждый процесс выводит на экран свой `pid`, полный путь, имя и размер просмотренного файла, общее

число просмотренных файлов в подкаталоге. Число запущенных процессов в любой момент времени не должно превышать  $N$  (вводится пользователем). Проверить работу программы для каталога `/usr` и  $N=6$ .

### Вариант 6

Написать программу, находящую в заданном каталоге и всех его подкаталогах все исполняемые файлы. Диапазон (мин. – макс.) размеров файлов задаётся пользователем в качестве первого и второго аргумента командной строки. Имя каталога задаётся пользователем в качестве третьего аргумента командной строки. Программа выводит результаты поиска в файл (четвертый аргумент командной строки) в виде: полный путь, имя файла, его размер. На консоль выводится общее число просмотренных файлов. Процедура поиска для каждого подкаталога должна запускаться в отдельном процессе. Каждый процесс выводит на экран свой `pid`, полный путь, имя и размер просмотренного файла, общее число просмотренных файлов в подкаталоге. Число запущенных процессов в любой момент времени не должно превышать  $N$  (вводится пользователем). Проверить работу программы для каталога `/usr/` и размера 31000 31500 и  $N=6$ .

### Вариант 7

Написать программу нахождения массива значений функции  $y[i]=\sin(2*\pi*i/N)$   $i=[0,N-1]$  с использованием ряда Тейлора. Пользователь задаёт значения  $N$  и количество  $n$  членов ряда Тейлора. Для расчета каждого члена ряда Тейлора запускается отдельный процесс и его результат (член ряда) записывается в файл. Каждый процесс выводит на экран свой `id` и рассчитанное значение ряда. Головной процесс суммирует все члены ряда Тейлора и полученное значение  $y[i]$  записывает в файл. Проверить работу программы для значений  $N, n = [64, 5]$  и  $N, n = [32768, 7]$ .

### Вариант 8

Написать программу поиска одинаковых по их содержимому файлов в двух каталогов, например, `Dir1` и `Dir2`. Пользователь задаёт имена `Dir1` и `Dir2`. В результате работы программы файлы, имеющиеся в `Dir1`, сравниваются с файлами в `Dir2` по их содержимому. Процедуры сравнения должны запускаться с использованием функции `fork()` в отдельном процессе для каждой пары сравниваемых файлов. Каждый процесс выводит на экран свой `pid`, имя файла, число просмотренных байт и результаты сравнения. Число запущенных процессов в любой момент времени не должно пре-

вышать  $N$  (вводится пользователем). Проверить работу программы для каталога `/usr/include/` и любого другого каталога в `/home` и  $N=6$ .

### **Вариант 9**

Написать программу поиска заданной пользователем комбинации из  $m$  байт ( $m < 255$ ) во всех файлах текущего каталога. Пользователь задаёт в качестве аргументов командной строки имя каталога, строку поиска, файл результата. Главный процесс открывает каталог и запускает для каждого файла каталога отдельный процесс поиска заданной комбинации из  $m$  байт. Каждый процесс выводит на экран и в файл результата свой `pid`, полный путь и имя файла, число просмотренных в данном файле байт и результаты поиска (всё в одной строке!). Результаты поиска (только найденные файлы) по предыдущему формату записываются в выходной файл. Число запущенных процессов в любой момент времени не должно превышать  $N$  (вводится пользователем). Проверить работу программы для каталога `/usr/include/` и строки `"stdio.h"`,  $N=6$ .

### **Вариант 10**

Тоже что и в варианте 9, но включая подкаталоги. Проверить работу программы для каталога: `/usr/` размер 31000 31500 и  $N=6$ .

### **Вариант 11**

Разработать программу «интерпретатор команд», которая воспринимает команды, вводимые с клавиатуры, и осуществляет их корректное выполнение.

Для этого каждая вводимая команда должна выполняться в отдельно запускаемом процессе с использованием вызова `exec()`. Нельзя использовать вызов любого готового интерпретатора из своей программы или вызов `system()`. Для проверки работы выполнить команду: `ls -l > 1.txt`. Предусмотреть контроль ошибок и команду выхода из программы.

### **Вариант 12**

Написать программу, находящую в заданном каталоге и всех его подкаталогах все скрытые файлы. Диапазон (мин. – макс.) размеров файлов задаётся пользователем в качестве первого и второго аргумента командной строки. Имя каталога задаётся пользователем в качестве третьего аргумента командной строки. Программа выводит результаты поиска в файл (четвертый аргумент командной строки) в виде: полный путь, имя файла, его размер. На консоль выводится общее число просмотренных файлов и

суммарный их размер. Процедура поиска для каждого подкаталога должна запускаться в отдельном процессе. Каждый процесс выводит на экран свой `pid`, полный путь, имя и размер просмотренного файла, общее число просмотренных файлов в подкаталоге. Число запущенных процессов в любой момент времени не должно превышать `N` (вводится пользователем). Проверить работу программы для каталога `/home` и `/usr`, `N=6`.

### Вариант 13

Написать программу поиска одинаковых файлов по их содержимому, владельцу и группе-владельцу в двух каталогов, например, `Dir1` и `Dir2`. Пользователь задаёт имена `Dir1` и `Dir2`. В результате работы программы файлы, имеющиеся в `Dir1`, сравниваются с файлами в `Dir2` по их содержимому. Процедуры сравнения должны запускаться с использованием функции `fork()` в отдельном процессе для каждой пары сравниваемых файлов. Каждый процесс выводит на экран свой `pid`, имя файла, число просмотренных байт, владельца, группу-владельца и результаты сравнения. Число запущенных процессов в любой момент времени не должно превышать `N` (вводится пользователем). Проверить работу программы для каталога `/usr/include/` и любого другого каталога в `/home`, `N=6`.

## 2.4 Контрольные вопросы

1. Что такое `pid` и как его получить?
2. Что делает и что возвращает функция `fork()`?
3. Чем отличается дочерний процесс от родительского сразу после вызова функции `fork()`?
4. Что делает и что возвращает функция `wait()`?
5. Чем функция `waitpid()` отличается от `wait()`?
6. Каковы параметры и возвращаемое значение функции `waitpid()`?
7. Как получить код завершения дочернего процесса?
8. Перечислите макросы для анализа кода завершения дочернего процесса.
9. Какой процесс называют «зомби»?
10. Перечислите методы нормального завершения процесса.
11. Чем отличаются функции `exit()` и `_exit()`?

12. Как в дочернем процессе запустить другую программу?
13. Как определить и изменить приоритет процесса?
14. Перечислите допустимые параметры функции `kill()`.
15. Каковы особенности сигнала `SIGCHLD`? Как его перехватить?
16. Перечислите преимущества многопоточности.
17. Перечислите и охарактеризуйте модели потоков в Linux?
18. Приведите примеры шаблонов многопоточного программирования и дайте краткое описание для каждого из примеров.
19. Что такое мьютекс?
20. Какие реализации потоков используются в Linux?
21. Опишите функцию порождения потока и особенности функции, запускаемой в порожденном потоке.
22. Каковы особенности POSIX API для работы с потоками?
23. Когда и как происходит завершение потока?
24. Как сделать поток отсоединенным?
25. Опишите особенности досрочного завершения потока.
26. Каковы особенности главного потока в процессе?
27. Опишите жизненный цикл потоков.
28. Зачем нужен и как создается объект атрибутов потока?
29. Каковы особенности инициализации мьютексов и условных переменных, расположенных в разделяемой памяти?