Лабораторные работы по курсу
Операционные системы

Лабораторная работа 1
Основы взаимодействия с командной оболочкой ОС Linux

Москва, 2023 г.

Оглавление

1. Теоретические сведения

1.1. Основы ОС Linux

UNIX – многопользовательская ОС, появившаяся в 1970-х годах. В основе данной ОС лежит принцип открытости, это система, созданная программистами и для программистов. Весь исходный код доступен пользователям, благодаря чему большое количество университетов и коммерческих организаций развивало и портировало UNIX-подобные ОС на различные вычислительные архитектуры. Принятие POSIX стандарта позволило запускать разрабатываемое ПО на различных UNIX-системах и укрепило их положение, как доминирующих. С годами многие системы отошли от первоначальных принципов: они становились громоздкими и сложными в понимании, в ответ на это была разработана ОС MINIX, которая отличалась простотой ядра и возможностью масштабирования за счет включения или выключения модулей. На базе данной ОС в 1990-х был разработан Linux, которые занял свое место среди других версий UNIX-подобных систем благодаря открытости исходных кодов, большому количеству реализованных утилит и расширенным функциям ядра. ОС Debian – это бесплатный дистрибутив Linux, портированный на большое количество архитектур: от ARM до серверных микропроцессоров.

В используемом в практикуме дистрибутиве Raspberry Pi OS (на базе Debian) для одноплатного компьютера Raspberry Pi совмещено два интерфейса: ориентированный на ввод с клавиатуры интерфейс - консоль и ориентированный на использование мыши – графический интерфейс. Причем, данное совмещение интерфейсов не требует никаких изменений в ядре системы. Именно эта гибкость сделала систему Linux такой популярной и позволила ей пережить многочисленные изменения лежащей в ее основе технологии.

Графический интерфейс пользователя системы Linux похож на первые графические интерфейсы пользователя, разработанные для UNIX в 70-х годах прошлого века и ставшие популярными благодаря компьютерам Macintosh и впоследствии — системе Windows для персональных компьютеров. Графический интерфейс пользователя создает среду рабочего стола — знакомую нам метафору с окнами, значками, каталогами, панелями инструментов, а также возможностью перетаскивания. Полную среду рабочего стола содержит администратор многооконного режима, который управляет размещением и видом окон, а также различными приложениями и создает согласованный графический интерфейс.

Популярными средами рабочего стола для Linux являются GNOME (GNU Network Object Model Environment) и KDE (K Desktop Environment). Графические интерфейсы пользователя в Linux поддерживает оконная система X Windowing System, которую обычно называют X11 (или просто X). Она определяет обмен и протоколы отображения для управления окнами на растровых дисплеях UNIX-подобных систем. X-сервер является главным компонентом, который управляет такими устройствами, как клавиатура, мышь и экран, и отвечает за перенаправление ввода или прием вывода от клиентских программ.

Реальная среда графического интерфейса пользователя обычно построена поверх библиотеки низкого уровня (xlib), которая содержит функциональность для взаимодействия с X-сервером. Графический интерфейс расширяет базовую функциональность X11, улучшая вид окон, предоставляя кнопки, меню, значки и пр. X-сервер можно запустить вручную из командной строки, но обычно он запускается во время загрузки диспетчером окон, который отображает графический экран входа в систему.

При работе на Linux-системах с помощью графического интерфейса пользователь может щелчком кнопки мыши запустить приложение или открыть файл, использовать перетаскивание для копирования файлов из одного места в другое и т. д. С одной стороны, использование графического интерфейса упрощает работу начинающего пользователя и позволяет использовать ОС именно для администрирования персонального компьютера. С другой стороны, операционные системы UNIX - интерактивная система, разработанная для одновременной поддержки множества процессов и

множества пользователей. Это разработка программистов и для программистов — чтобы использовать ее в такой среде, в которой большинство пользователей достаточно опытны и занимаются проектами разработки программного обеспечения, например, для встраиваемых систем. В таком случае удобнее использовать не графический интерфейс пользователя, а интерфейс командной строки, в котором последовательно вводятся команды, флаги и аргументы, о которых будет написано ниже.

1.2. Взаимодействие с ОС Unix

Большинство разработчиков хочет, чтобы их система была простой, элегантной и совместимой. Например, на самом нижнем уровне файл должен представлять собой просто набор байтов. Наличие различных классов файлов для последовательного и произвольного доступа, доступа по ключу, удаленного доступа и т. д. (как это реализовано на ПК) просто является помехой. А если команда ls A* означает вывод списка всех файлов, имя которых начинается с буквы «А», то команда rm A* должна означать удаление всех файлов, имя которых начинается с буквы «А», а не одного файла, имя которого состоит из буквы «А» и звездочки. Эта характеристика иногда называется принципом наименьшей неожиданности (principle of least surprise).

Другие свойства, которыми обладают UNIX-подобные операционные системы — это мощь и гибкость. Это означает, что в системе есть небольшое количество базовых элементов, которые можно комбинировать, чтобы приспособить их для конкретного приложения. Одно из основных правил системы Linux заключается в том, что каждая программа должна выполнять всего одну функцию — и делать это хорошо. То есть компиляторы не занимаются созданием листингов, так как другие программы могут лучше справиться с этой задачей.

Наконец, система не должна обладать ненужной избыточность. Так, вместо copy вполне достаточно написать cp. А чтобы получить список всех строк, содержащих строку «ard», из файла f, разработчик в операционной системе Linux вводит команду:

```
grep ard f
```

Противоположный подход, подобно графическому интерфейсу, состоит в том, что пользователь сначала запускает программу grep (без аргументов), после чего программа grep приветствует его фразой: «Здравствуйте, я grep. Я ищу шаблоны в файлах. Пожалуйста, введите ваш шаблон». Получив шаблон, программа grep запрашивает имя файла. Затем она спрашивает, есть ли еще какие-либо файлы. Наконец, она выводит резюме того, что она собирается делать, и спрашивает, все ли верно. Хотя такой тип пользовательского интерфейса может быть удобен для начинающих пользователей, он бесконечно раздражает опытных программистов. Им требуется слуга, а не нянька.

Таким образом, большинство программистов и продвинутые пользователи по-прежнему предпочитают интерфейс командной строки, называемый оболочкой (shell). Они часто запускают одно или несколько окон с оболочками из графического интерфейса пользователя и работают в них. Интерфейс командной строки оболочки значительно быстрее в использовании, существенно мощнее, прост в расширении и не грозит пользователю туннельным синдромом запястья из-за необходимости постоянно пользоваться мышью. Далее мы кратко опишем оболочку bash. Она основана на оригинальной оболочке системы UNIX, которая называется оболочкой Бурна (Bourne shell, написана Стивом Бурном, а затем в Bell Labs.), и фактически даже ее название является сокращением от Bourne Again SHell. Используется и множество других оболочек (ksh, csh и т. д.), но bash является оболочкой по умолчанию в большинстве Linux-систем.

Когда оболочка запускается, она инициализируется, а затем выводит на экран символ приглашения к вводу (обычно это знак процента или доллара) и ждет, когда пользователь введет командную строку. После того как пользователь введет командную строку, оболочка извлекает из нее первое слово, под которым подразумевается череда символов с пробелом или символом табуляции в качестве разделителя.

Пользовательский интерфейс командной строки (оболочки) Linux состоит из большого числа стандартных служебных программ, называемых также утилитами. Грубо говоря, эти программы можно разделить на шесть следующих категорий:

1. команды управления файлами и каталогами;
2. фильтры;
3. средства разработки программ, такие как текстовые редакторы и компиляторы;
4. текстовые процессоры;
5. системное администрирование;
6. разное.

По ходу изучения операционной системы Debian вам часто будет требоваться информация о том, что делает та или иная команда или системный вызов, какие у них параметры и опции, для чего предназначены некоторые системные файлы, каков их формат и т.д. Мы постарались, по мере возможности, включить описания большинства используемых в курсе команд и системных вызовов в наш текст. Однако иногда для получения более полной информации мы отсылаем читателей к UNIX Manual – руководству по операционной системе UNIX. К счастью, большая часть информации в UNIX Manual доступна в интерактивном режиме с помощью утилиты *man*.

Пользоваться утилитой *man* достаточно просто – наберите команду *man имя*, где *имя* – это *имя* интересующей вас команды, утилиты, системного вызова, библиотечной функции или файла. Попробуйте с ее помощью посмотреть информацию о команде *pwd*.

1.3. Стандартные утилиты

Стандарт POSIX определяет синтаксис и семантику около 150 программ, основном относящихся к первым трем категориям. Идея стандартизации данных программ заключается в том, чтобы можно было писать сценарии оболочки, которые работали бы на всех системах Linux. Помимо этих стандартных утилит существует еще масса прикладных программ, таких как веб-браузеры, проигрыватели мультимедийных файлов, программы просмотра изображений, офисные пакеты и т. д.

Некоторые утилиты POSIX перечислена в Таблице 1 вместе с кратким описанием. Во всех версиях операционной системы Linux есть эти программы, а также многие другие.

Таблица 1. Утилиты Linux

Команда/ утилита	Описание	Синтаксис
ls	Используется в командной оболочке Linux для вывода содержимого каталогов и информации о файлах	**ls** -опции /путь/к/папке
cd	Изменяет директорию с текущей на указанную	**cd** /путь/к/папке
cp	Копирует из файлы из одного каталога в другие	**cp** -опции /откуда /куда
mv	Переносит файлы из одного каталога в дригие	**mv** -опции /откуда /куда
rm	Удаляет файлы из каталога	**rm** -опции файл**(**ы**)**
mkdir	Создает пустой каталог	**mkdir** -опции директория
rmdir	Удаляет каталоги	**rmdir** -опции директория
chown	Передает права на файлы другим пользователям	**chown** -пользователь -опции /путь/к/файлу
chmod	Изменяет права/действия/пользователей для файла	**chmod** -опции -права /путь/к/файлу

locate	Используется для поиска файлов, расположенных на машине пользователя или на сервере. Фактически она выполняет ту же работу, что и команда *find*, однако, ведёт поиск в собственной базе данных. *find* же шаг за шагом проходит через всю иерархию директорий.	**locate** опции шаблон_для_поиска
date	Извлекает дату в различных форматах	**date** -опции -формат
tar	Архивирует и записывает вывод в файл	**tar** -опции архив.tar файлы_для_архивации
cat	Читает данные из файла или стандартного ввода и выводит их на экран	**cat** -опции файл1 файл2 ...
less	Позволяет перематывать текст вперёд/назад, осуществлять поиск в обоих направлениях, переходить сразу в конец или в начало файла	**less** -опции файл
grep	Фильтрует вывод других команд, позволяет искать по содержимому файловой системы	**grep** -опции 'поиск' /где_искать
passwd	Задает пароль пользователя	**passwd** пароль
reboot	Перезагрузка ОС	reboot
halt	Остановка ОС	halt

Прочитайте больше информации о каждой утилите используя *man*.

1.4. Регулярные выражения

Утилиты Linux-систем могут принимать несколько флагов и аргументов. Чтобы было легче указывать группы файлов, оболочка принимает регулярные выражения или, так называемые волшебные символы (magic charecters), иногда также называемые групповыми (wild cards). Например, символ «звездочка» означает все возможные текстовые строки, так что строка:

```
ls *.c
```

дает указание программе ls вывести список всех файлов, имена которых оканчиваются на «.c». Если существуют файлы x.c, y.c и z.c, то данная команда эквивалентна команде:

```
ls x.c y.c z.c
```

Другим групповым символом является вопросительный знак, который заменяет один любой символ. Кроме того, в квадратных скобках можно указать множество символов, из которых программа должна будет выбрать один. Например, команда

```
ls [ape]*
```

выводит все файлы, имя которых начинается с символов «a», «p» или «e». Такая программа, как оболочка, не должна открывать терминал (клавиатуру и монитор), чтобы прочитать с него или сделать на него вывод. Вместо этого запускаемые программы автоматически получают доступ для чтения к файлу, называемому стандартным устройством ввода (standard input), а для записи — к файлу, называемому стандартным устройством вывода (standard output), и к файлу, называемому стандартным устройством для вывода сообщений об ошибках (standard error). По умолчанию всем этим трем устройствам соответствует терминал, то есть чтение со стандартного ввода производится с клавиатуры, а запись в стандартный вывод (или в вывод для ошибок) попадает на экран. Многие Linux программы читают данные со стандартного устройства ввода и пишут на стандартное устройство вывода.

1.5. Файлы и директории

Все файлы, доступные в операционной системе *UNIX*, как и в уже известных вам операционных системах, объединяются в древовидную логическую структуру. Файлы могут объединяться в **каталоги** или **директории**. Не существует файлов, которые не входили бы в состав

какой-либо директории. Директории в свою *очередь* могут входить в состав других директорий. Допускается существование пустых директорий, в которые не входит ни один *файл*. Среди всех директорий существует только одна *директория*, которая не входит в состав других директорий – ее принято называть **корневой**. На настоящем уровне нашего незнания *UNIX* мы можем заключить, что в файловой системе *UNIX* присутствует, *по* крайней мере, два типа файлов: обычные файлы, которые могут содержать тексты программ, *исполняемый код*, данные и т.д. – их принято называть **регулярными файлами**, и директории.

Каждому файлу (регулярному или директории) должно быть присвоено имя. В различных версиях операционной системы *UNIX* существуют те или иные ограничения на построение имени файла. В стандарте *POSIX* на *интерфейс* системных вызовов для операционной системы *UNIX* содержится лишь три явных ограничения:

Нельзя создавать имена большей длины, чем это предусмотрено операционной системой (для Linux – 255 символов).

Нельзя использовать символ NUL (не путать с указателем NULL!) – он же символ с нулевым кодом, он же признак конца строки в языке С.

Нельзя использовать символ '/'.

От себя добавим, что также нежелательно применять символы "звездочка" – "*", "знак вопроса" – "?", "кавычка" – "\"", "*апостроф*" – "\`", "*пробел*" – " " и "*обратный* слэш" – "\\" (символы записаны в нотации символьных констант языка С).

Единственным исключением является корневая *директория*, которая **всегда** имеет имя "/". Эта же *директория по* вполне понятным причинам представляет собой единственный *файл*, который должен иметь уникальное имя во всей файловой системе. Для всех остальных файлов имена должны быть уникальными только в рамках той директории, в которую они непосредственно входят. В Таблице 2 приведена структура директорий ОС *Linux*.

Таблица 2. Структура директорий ОС Linux

Каталог	**Описание**
/	Корневой каталог, содержащий всю файловую иерархию.
/bin	Основные утилиты, необходимые как в однопользовательском режиме, так и при обычной работе всем пользователям (например: cat, ls, cp).
/boot	Загрузочные файлы (в том числе файлы загрузчика, ядро, initrd, System.map).
/dev	Основные файлы устройств.
/etc	Общесистемные конфигурационные файлы (имя происходит от лат. *et cetera*).
/home	Содержит домашние каталоги пользователей, которые в свою очередь содержат персональные настройки и данные пользователя.
/lib	Основные библиотеки, необходимые для работы программ из /bin и /sbin.
/media	Точки монтирования для сменных носителей, таких как CD-ROM, DVD-ROM (впервые описано в *FHS-2.3*).
/mnt	Содержит временно монтируемые файловые системы.
/opt	Дополнительное программное обеспечение.
/proc	Виртуальная файловая система, представляющая состояние ядра операционной системы и запущенных процессов в виде файлов.
/root	Домашний каталог пользователя *root*.
/run	Информация о системе с момента её загрузки, в том числе данные, необходимые для работы демонов (pid-файлы, UNIX-сокеты и т.д.).
/sbin	Основные системные программы для администрирования и настройки системы, например, init, iptables, ifconfig.
/srv	Данные для сервисов, предоставляемых системой (например, www или ftp).

/sys	Содержит информацию об устройствах, драйверах, а также некоторых свойствах ядра.
/tmp	Временные файлы (см. также /var/tmp).
/usr	*Вторичная иерархия* для данных пользователя. Содержит большинство пользовательских приложений и утилит, используемых в многопользовательском режиме. Может быть смонтирована по сети только для чтения и быть общей для нескольких машин.
/var	Изменяемые файлы, такие как файлы регистрации, временные почтовые файлы, файлы спулеров.

1.6. Процесс компиляции

Процесс компиляции представляет собой процедуру перевода текстовых файлов, в которых написан исходный код программы, в машинный код, то есть последовательность инструкций, которые будут непосредственно исполняться на процессоре. Он традиционно разделяется на четыре последовательных этапа **Указан недопустимый источник.**, показанных на Рисунке Рисунок 1:

Рисунок 1. Этапы компиляции программы

На вход компилятора подаются файлы с исходными кодами, которые могут ссылаться на функции как внутри этих файлов, так и на функции внутри внешних библиотек (в том числе на системные и библиотечные вызовы).

Первым этапом компиляции является **предобработка** исходных кодов. Она выполняется специальной подпрограммой компилятора, которая называется препроцессор. Его работа в основном связана со специальными директивами. Например, все константы, заданные в программе директивой «*#define*» будут заменены в препроцессоре на значения этой константы в тексте программы, директивы «*#include*» будут заменены текстом указанного в директиве файла и т. д.

Этап **компилирования** представляет собой преобразования исходных кодов программы из текстового формата в набор машинных инструкций на языке ассемблера. Конкретный набор используемых инструкций сильно зависит от архитектуры процессора, для которого производится компилирование. Нельзя скомпилировать исходные коды под одну архитектуру и использовать результаты компиляции на другой архитектуре.

Этап **трансляции** принимает на вход набор инструкций ассемблера и переводит его в машинный код – набор байт, которые определяют команды и операнды для процессора. Однако данный код еще невозможно исполнять на процессоре ввиду отсутствия связи между отдельными функциями и переменными.

Для определения взаимосвязей используется этап **компоновки**. Он объединяет все необходимые инструкции между собой и создает взаимосвязи между ними внутри единого исполняемого файла.

1.6.1. Компиляция программы

В *GNU Toolchain* в ОС *Linux* частичная или полная компиляция программы осуществляется с помощью вызова программы «*gcc*». В случае, если исходный код программы состоит из одного файла, наиболее удобным способом компиляции будет вызов программы *gcc* в командной оболочке со следующими атрибутами:

```
gcc main.c -o program
```

где «*main.c*» – имя файл с исходным кодом, «*program*» – название результирующего исполняемого файла программы. Параметр ***-o*** необходим для явного указания имени исполняемого файла. Если

он не указан, то исполняемый файл будет иметь название «*a.out*». Утилита *gcc* после компиляции сразу присваивает файлу программы атрибуты на исполнение.

2. Практические задания

2.1. Общие задания

2.1.1. Запустите командную строку виртуальной машины.

2.1.2. С помощью команды pwd убедитесь, что вы находитесь в директории *~/Desktop*

```
user@sab_user:~$ pwd
```

2.1.3. Создайте директорию с названием «*Номер группы_Ваша фамилия*», например: IB21_Ivanov, в которой будут сохраняться все результаты лабораторной работы. Для примера ниже показано создание директории с названием *SAB*:

```
mkdir /home/pi/SAB
```

2.1.4. Войдите в созданную вами директорию:

```
cd /home/pi/SAB
```

2.1.5. В первую очередь выведем на экран имя пользователя с помощью команды:

```
whoami
user
```

2.1.6. С помощью команды выведем текущую дату:

```
date
```

2.1.7. С помощью опции -u выведем мировое время (UTC):

```
date -u
```

2.1.8. Такого же результата можно добиться, используя опцию **utc.** Так как она состоит из нескольких символов, она должна начинаться с двух символов «–». Проверим это:

```
date --utc
```

2.1.9. Также с помощью команды date возможно сменить время системы. Для этого необходимо в параметрах указать необходимое значение. Например, чтобы вернуться в прошлое к началу лета, необходимо ввести дату и время в формате: <месяц><число><часы><минуты><год>: 0601120020. В результате будет выведена новая дата – 1 июня 2020 года 12:00:00:

```
sudo date 0601120020
```

2.1.10. В текущем каталоге создадите подкаталоги tmp_1, tmp_2, tmp_3

```
mkdir tmp_1
mkdir tmp_2
mkdir tmp_3
```

2.1.11. Не заходя в каталог tmp_1 создайте в нем файлы file_1.txt, file_2.txt, file_3.txt.

```
touch tmp_1/file_1.txt
touch tmp_1/file_2.txt
touch tmp_1/file_3.txt
```

2.1.12. Перейдите в подкаталог tmp_1 и убедитесь, что файлы были созданы.

```
cd tmp_1
ls
```

2.1.13. Запишите в созданные файлы текстовое сообщение формата «I am File N».

```
echo "I am file 1" > file_1.txt
echo "I am file 2" > file_2.txt
echo "I am file 3" > file_3.txt
```

2.1.14. Убедитесь, что запись была осуществлена

```
cat file_1.txt
```

2.1.15. Скопируйте file_1 в директорию lab_1.

```
cp file_1.txt ..
```

2.1.16. Скопируйте file_2 в созданную в п. 2.1.10 директорию tmp_2.

```
cp file_2.txt ../tmp_2
```

2.1.17. Переместите file_3 в созданную в п. 2.1.10 директорию tmp_3.

```
mv file_3.txt ../tmp_3
```

2.1.18. Вернитесь в директорию lab_1

```
cd ..
```

2.1.19. Произведите поиск слова file по всем файлам текущей директории. Убедитесь, что обнаружено только одно вхождение. Для обозначения всех файлов используйте символ *.

```
grep file *
```

2.1.20. Произведите поиск слова file по всем файлам текущей директории и поддиректорий. Убедитесь, что обнаружены все пять вхождений. Ключ -r позволяет производить рекурсивный поиск по всем поддиректориям.

```
grep file * -r
```

2.1.21. Проведите поиск по цифре 2 по всем файлам текущей директории и поддиректорий. Убедитесь, что обнаружены только два вхождения.

```
grep 2 * -r
```

2.1.22. Произведите поиск по названию файла. Убедитесь, что обнаружены пять вхождений.

```
find -name "file*"
```

2.1.23. Переместите директорию tmp_1 в директорию tmp_2 и скопируйте в неё tmp_3. Обратите внимание, что для копирования директорий необходимо использовать ключ -r.

```
mv tmp_1 tmp_2
cp -r tmp_3 tmp_2
```

2.1.24. Удалите файл file_1.txt в текущей директории. Убедитесь, что файл был удален.

```
rm file_1.txt
```

2.1.25. Удалите директории tmp_2 и tmp_3

```
rm -r tmp_2
rm -r tmp_3
```

2.1.26. Создайте файл program.c и добавьте в него следующий код программы

```
nano program.c
```

```c
#include <stdio.h>
int main()
{
        printf("Hello MIET\n");
        return 0;
}
```

2.1.27. Проверьте наличие компилятора gcc на вашей машине. Для этого выполните команду следующую команду.

```
gcc -v
```

В случае отсутствия компилятора, установите его

```
sudo apt install gcc
```

2.1.28. Скомпилируйте программу и запустите её. Убедитесь, что на экране появилось сообщение «Hello MIET».

```
gcc program.c -o program
./program
```

2.1.29. Измените код программы в соответствии с приведенным ниже листингом. Изучите её работу, скомпилируйте и запустите её. Обратите внимание, что для подключения библиотеки math необходимо добавить к команде компиляции ключ -lm.

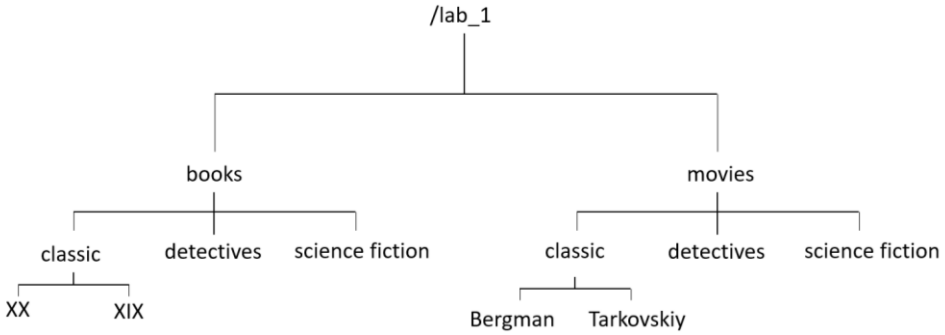
```
gcc program.c -o program -lm
./program sin 45
```

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char* argv[])
{
        double N = atof(argv[2]);
        if(strcmp(argv[1], "sin") == 0)
                printf("Sin(%lf)=%lf\n", N, sin(N/180*M_PI));
        else
                printf("Error...");
        return 0;
}
```

2.2. Индивидуальные задания

2.2.1. Создайте директории в соответствии со схемой.



2.2.2. Убедитесь, что директории вложены верно. Для этого выполните команду tree. Скриншот результата занесите в отчет.

2.2.3. Распределите файлы из списка по директориям. Для этого создайте файлы с расширением .txt для книги и .mov для кинофильма. Обратите внимание, что некоторые произведения из списка имеют экранизацию и могут быть отнесены в несколько директорий. В качестве содержимого файла укажите автора произведения/режиссера кинофильма. Для верной сортировки воспользуйтесь ресурсами интернет.

Список произведений:

A Scandal in Bohemia, And Then There Were None, Autumn Sonata, Doctor Faustus, Dune, Nostalghia, Offret, Solaris, Star Wars. Episode IV: A New Hope, The Adventure of the Speckled Band, The Glass Bead Game, The Lost World, The Magic Mountain, The Posthumous Papers of the Pickwick Club, Vanity Fair.

2.2.4. Выполните команду tree. Скриншот результата занесите в отчет. (При отсутсвии команды установите её с помощью команды *sudo apt install tree*)
2.2.5. Произведите поиск по названию произведения «Dune». Результат поиска занесите в отчет.
2.2.6. Произведите поиск по имени автора – Thomas Mann. Результат поиска занесите в отчет.
2.2.7. Модернизируйте программу из пункта 2.1.29 следующим образом.
    1. Добавьте проверку на число введенных параметров. В случае, если их число не равно 2 – выводить сообщение об ошибке и завершать работу с кодом возврата «-1».
    2. Добавьте возможность расчета косинуса угла.

3. Контрольные вопросы

1. Что такое операционная система? В чем заключаются ее основные функции?
2. Что такое дистрибутив операционной системы? Приведите примеры.
3. Возможно ли полностью отказаться от поддержки командной строки в пользу графического интерфейса? Аргументируйте свой ответ.
4. К какой архитектуре относится ядро Linux? В чем заключается её основная особенность?
5. В чем отличие файловой системы ОС Linux от файловой системы ОС Windows?
6. Объясните, что означает основная концепция Linux – «Всё есть файл»

4. Список литературы

1. Таненбаум Э. Современные операционные системы – 4-е изд. Питер, 2010. - 1120 с.
2. Карпов В.Е. Основы операционных систем: Курс лекций: Учеб. пособие / Карпов В.Е., Коньков К.А. ; Под ред. В.П. Иванникова. - М. : Интернет-университет информационных технологий, 2004. - 631 с.
3. Дейтел Г. Введение в операционные системы : В 2-х т. Т. 1 / Дейтел Г. - М. : Мир, 1987. - 359 с.
4. Г. В .Курячий, К. А. Маслинский Операционная система Linux: Курс лекций. Учебное пособие / - М. : ALT Linux; Издательство ДМК Пресс, 2016. - 348 с.