Изображение выглядит как текст

Автоматически созданное описание

**Отсчет по лабораторным работам**

по дисциплине «Системное программное обеспечение GNU/Linux»

Студент гр. 5130904/40008 Лиходиевский А. М.

Руководитель доц. Шмаков В. Э.

Оглавление

[Лабораторная работа №1 3](#__RefHeading___Toc153_2440990709)

[Цель работы 3](#__RefHeading___Toc7506_2440990709)

[Последовательность выполнения работы: 3](#__RefHeading___Toc7508_2440990709)

[Вывод 9](#__RefHeading___Toc318_3492897412)

[Лабораторная Работа №2 10](#__RefHeading___Toc320_3492897412)

[Цель работы 10](#__RefHeading___Toc322_3492897412)

[Последовательность выполнения работы 10](#__RefHeading___Toc324_3492897412)

[ЛАБОРАТОРНАЯ РАБОТА №3 16](#__RefHeading___Toc906_2500951832)

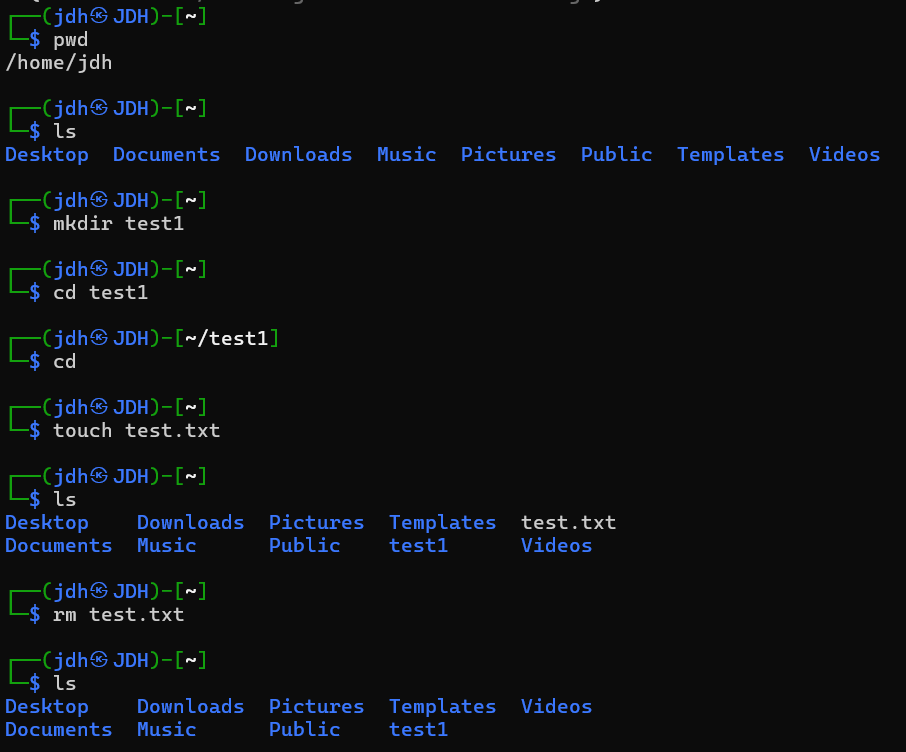
# **Лабораторная работа №1**

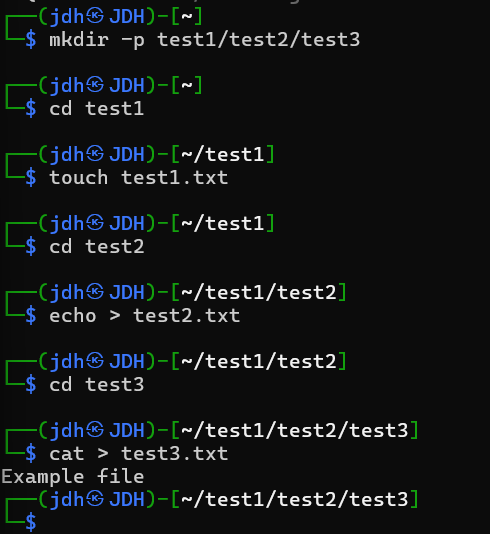
**Базовые команды ОС Linux**

## **Цель работы**

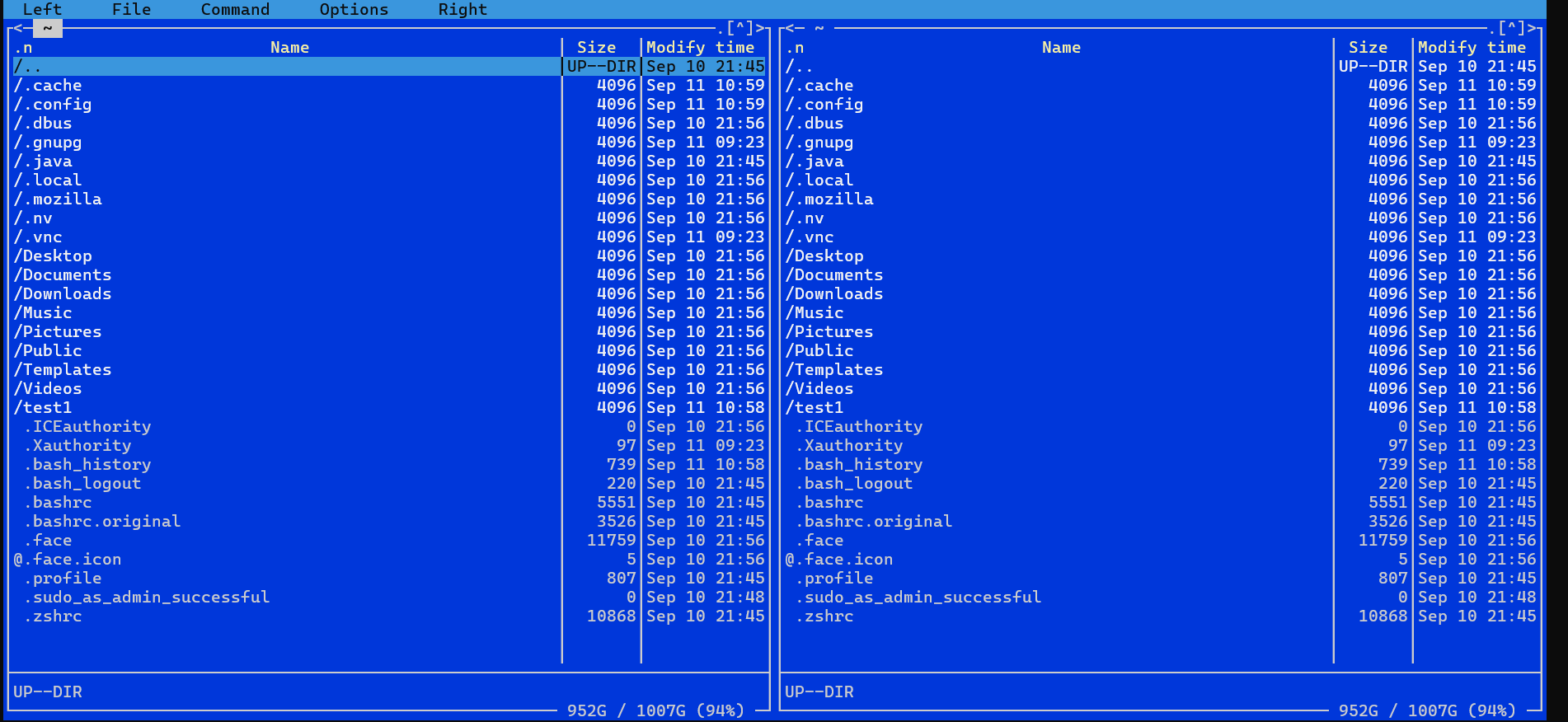
Освоение минимального набора базовых команд операционной системы Linux, знакомство с файловой системой, особенностями прав доступа, получение первичных навыков работы под Linux.

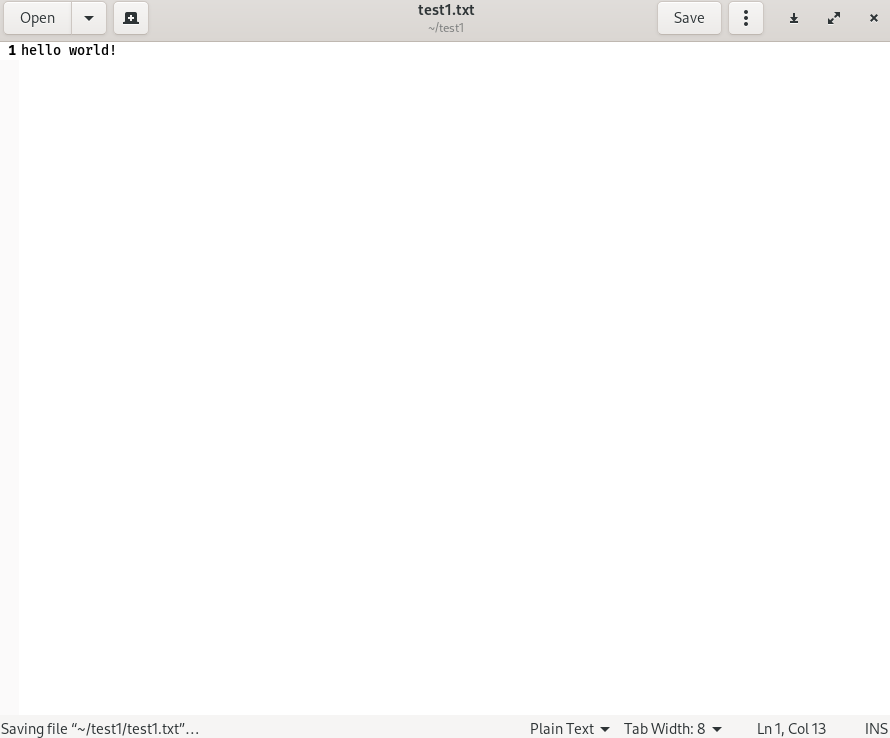
## **Последовательность выполнения работы:**

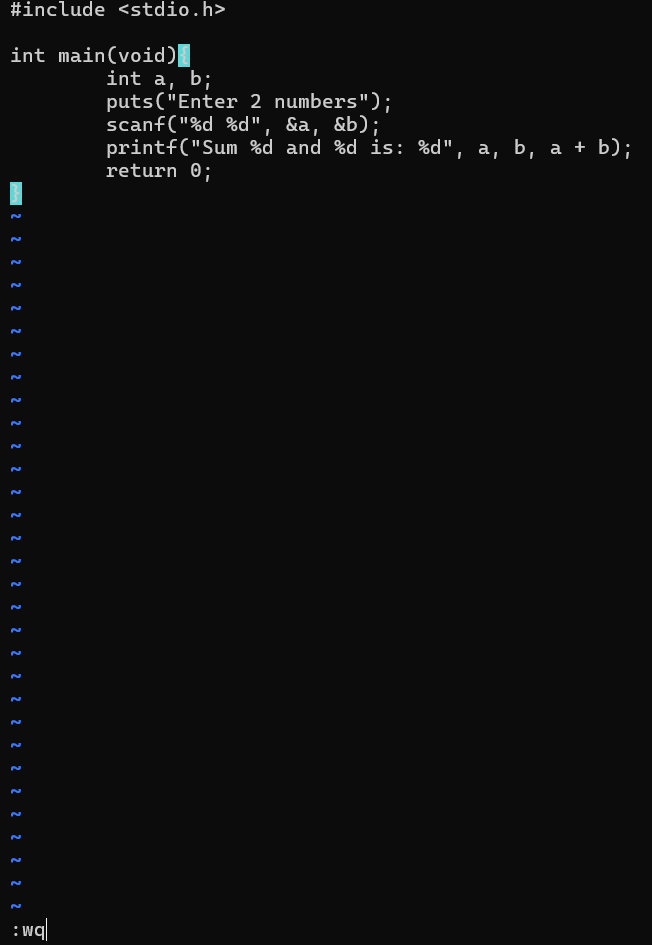
1. Запустил терминал и выполнил серию команд shell, такие, как pwd , ls , cd , mkdir , rm
2. 



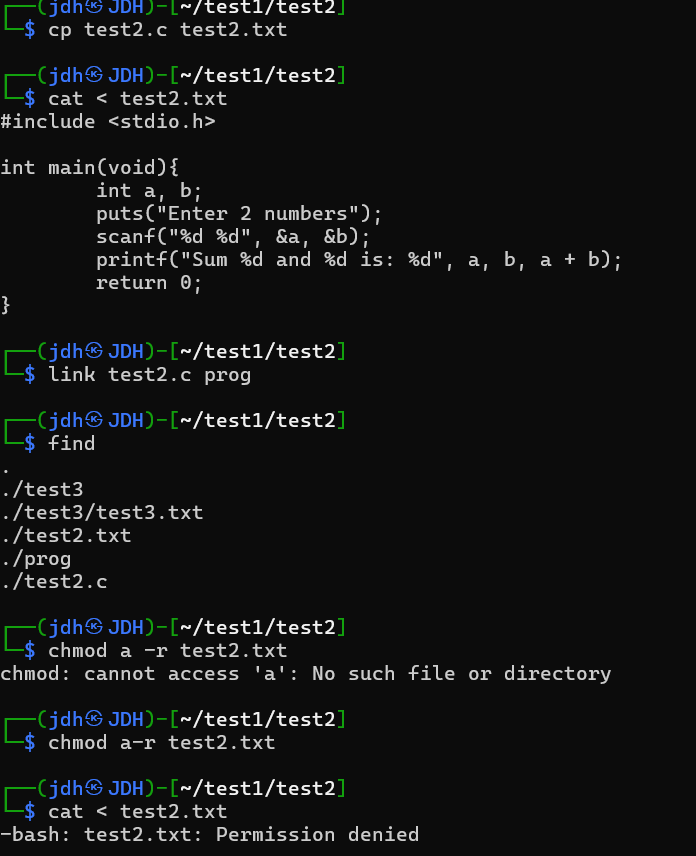
4.

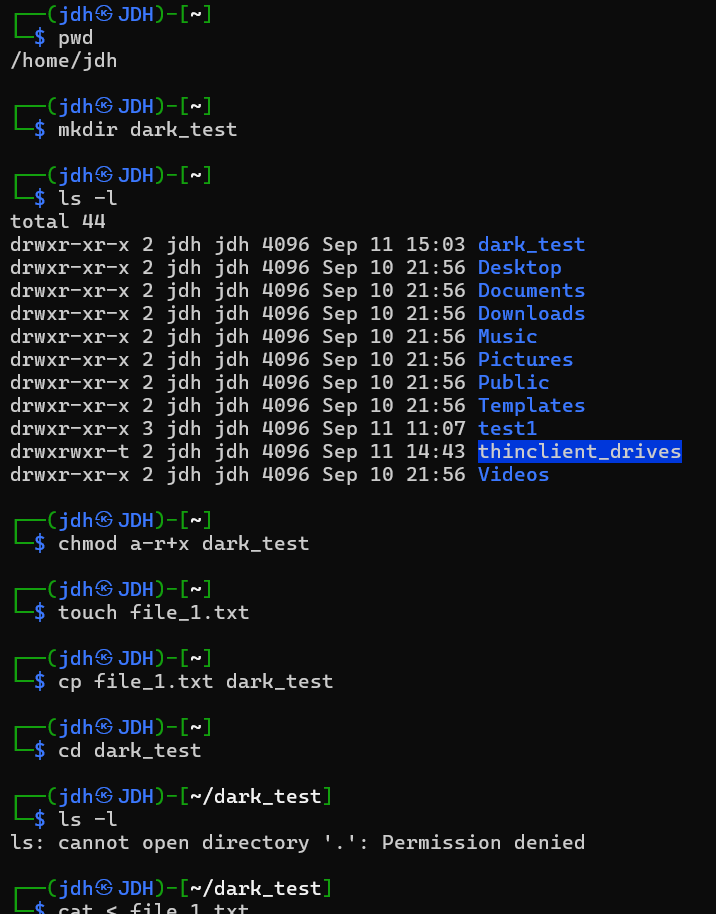






5.



6.

## **Вывод**

Было произведено освоение минимального набора базовых команд операционной системы Linux, знакомство с файловой системой, особенностями прав доступа, получение первичных навыков работы под Linux.

# Лабораторная Работа №2

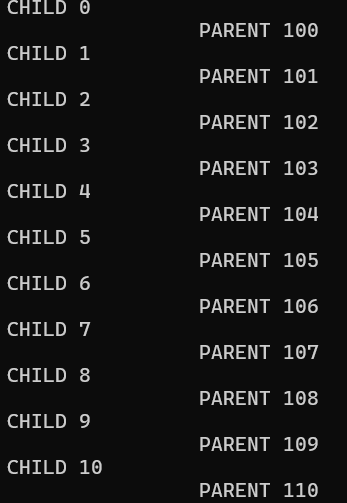
## Цель работы

Знакомство с характерной для Linux схемой порождения и завершения процессов, с отношениями типа потомок – родитель, со способами передачи информации о событии завершения процесса.

## **Последовательность выполнения работы**

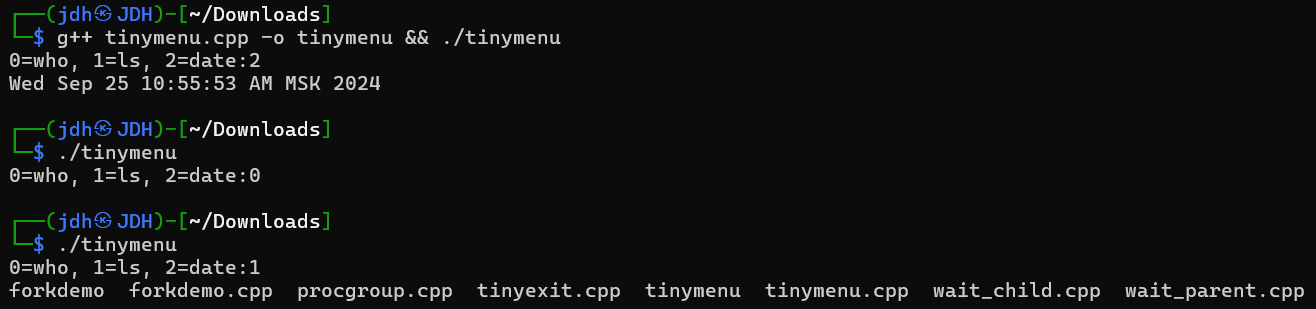
1. Запуск и компиляция файлов:

1) forkdemo.cpp



Функция fork() создаёт точную копию текущего процесса (дочерний процесс), которая выполняется параллельно родительскому, поэтому в выводе вперемешку присутствует вывод CHILD и PARENT. Причём, если запустить программу ещё раз, вывод будет отличаться (по-разному будет перемешиваться вывод CHILD и PARENT).

2) tinymenu.cpp

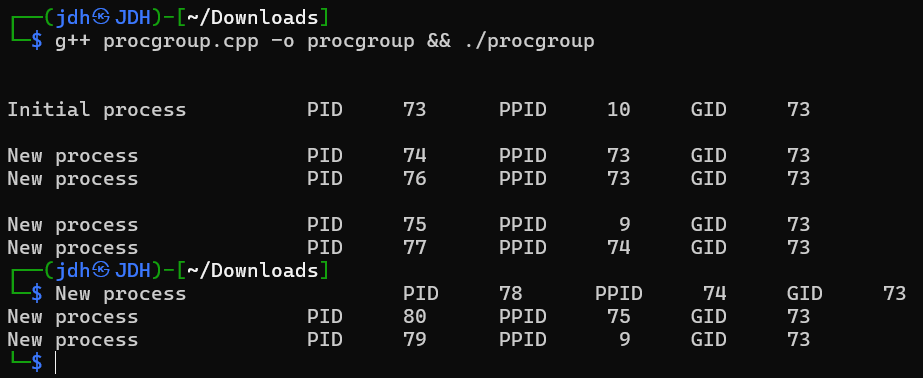


Функция execlp вызывает выбранную функцию из массива cmd.

3) tinyexit.cpp

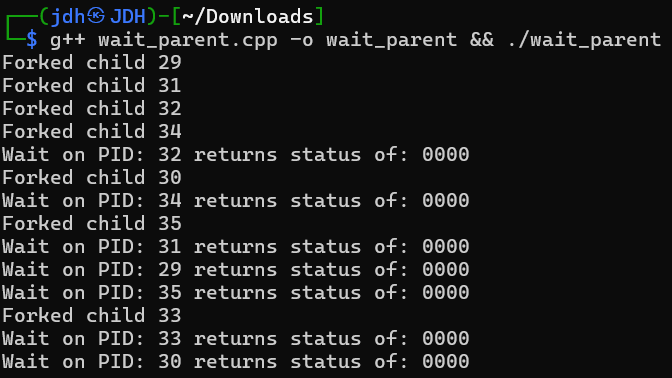
Программа (так же, как и предыдущая) выполняет указанную команду, но не завершается после каждой выполненной команды.

4) procgroup.cpp



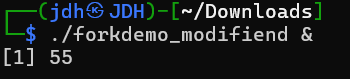
Программа генерирует копии процессов.

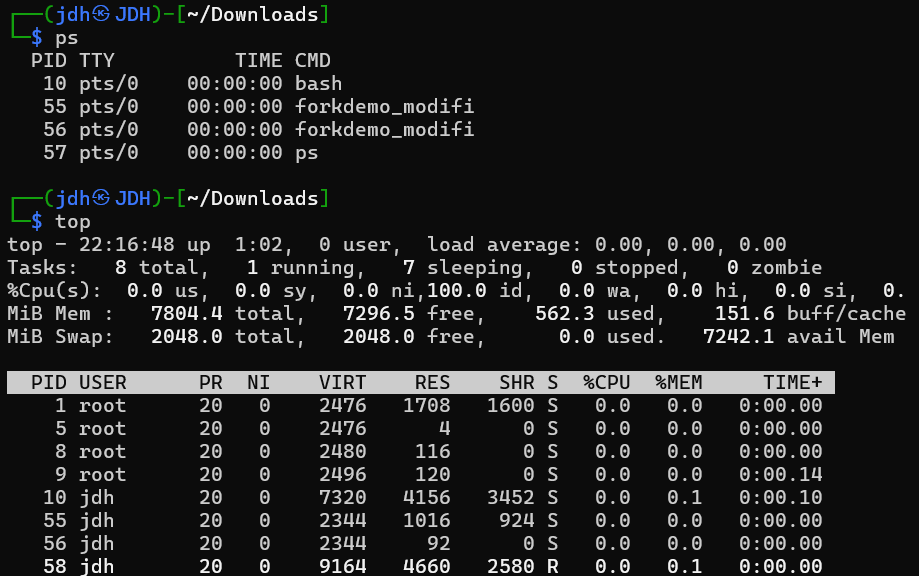
5) wait\_parent.cpp

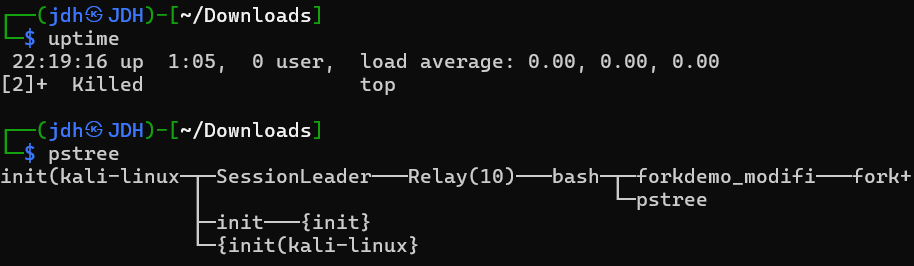


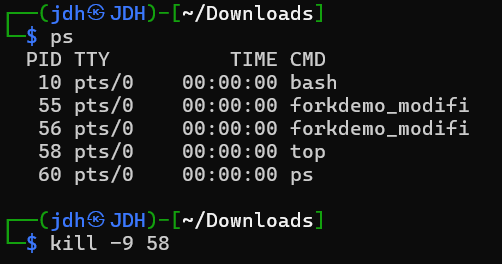
Процесс wait\_parent запускает процесс wait\_child, а затем ожидает завершения своих дочерних процессов.

3. Модифицируйте программу forkdemo.cpp (или создайте собственную), так чтобы ввод/вывод на терминал отсутствовал, а при проходе по циклу была временная задержка, например sleep (7). Запустите эту программу в фоновом режиме (background), введя при запуске символ & после пробела и зафиксировав значение PID, назначенное системой фоновому процессу при запуске. Выполните на терминале команды ps, top, uptime, pstree. Снимите свой фоновый процесс командой kill с соответствующими параметрами.









Используемые команды:

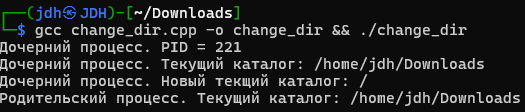
1. ps позволяет получить информацию о процессах и работать с ними
2. top выводит список работающих в системе процессов и информацию о них в отсортированном по нагрузке на процессор виде (причём в реальном времени).
3. uptime даёт информацию о: текущем времени, онлайн-пользователях, длительности работы системы и средней загрузке системы.
4. pstree показывает запущенные процессы в виде дерева.

4. Исследуйте, что произойдет, если процесс-потомок сменит

текущий каталог, будет ли изменен текущий каталог для родителя? Создайте программу, подтверждающую ответ и приведите в отчёте.

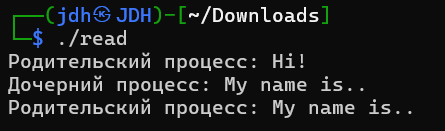
Если процесс-потомок сменит текущий каталог, текущий каталог для родителя изменяться не будет. Подтвердим предположение с помощью специально разработанного кода.

Результат запуска:



Процесс — родитель не изменил текущий каталога

5. Проиллюстрируйте как процесс-родитель и процесс-потомок разделяют один и тот же дескриптор и смещение текстового файла. Для этого составьте программу, в которой процесс-родитель должен открывать текстовый файл и запускать потомка. Потомок должен читать порцию данных из открытого файла и выводить на консоль. По завершению потомка родитель должен читать из того же файла и выводить результат на консоль. Можете использовать вызов sleep() для синхронизации доступа родителя и потомка к файлу.

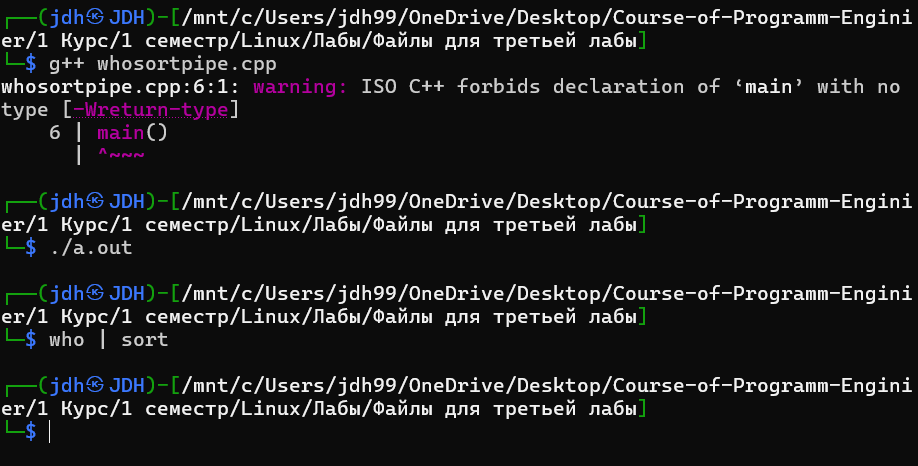


Вывод: в ходе данной лабораторной работе был получен опыт работы с процессами и программировании на Си в среде Linux

# ЛАБОРАТОРНАЯ РАБОТА №3

Программные каналы

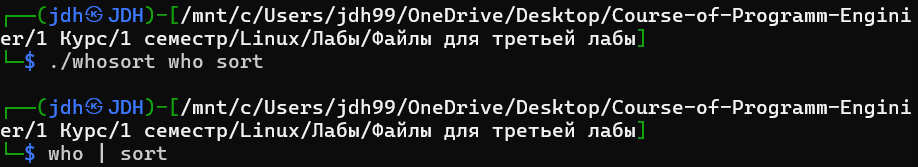
1. Скомпилируйте и выполните программу whosortpipe.cpp. Сопоставьте результат выполнения программы с выполнением этих же двух команд с терминала в конвейерном режиме (с использованием <>). Анализ результатов работы этой программы (как и всех последующих) с соответствующими скриншотами приведите в отчете.



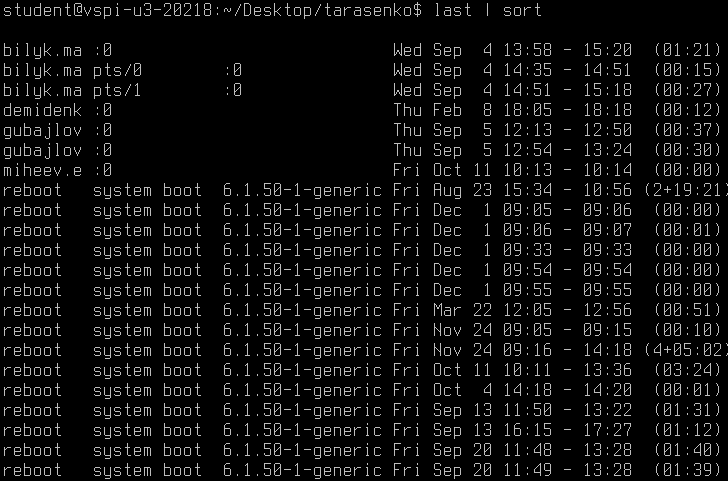
Результаты абсолютно одинаковы.

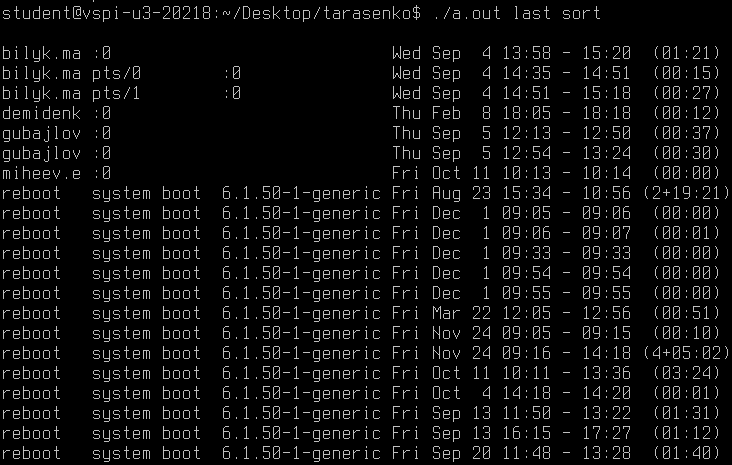
2. Программу cmdpipe.cpp запускайте после компиляции, задавая ей при стартах в качестве параметров командной строки пары команд shell для конвейеризации (who и sort ; last и sort ; last и more ; pstree и more). Сопоставьте результаты запусков программы с выполнением тех же пар командиз shell в конвейерном режиме.

2.2 Сравнение запуска с who sort:

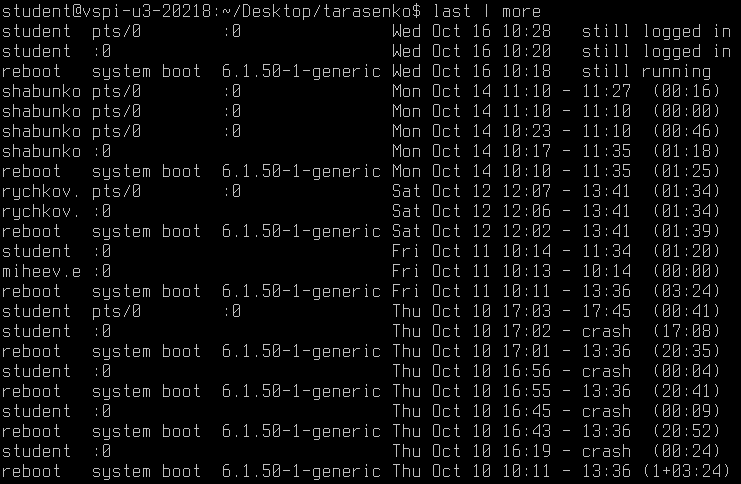


2.3 Сравнение запуска с last sort:





2.4 Сравнение запуска с last more:



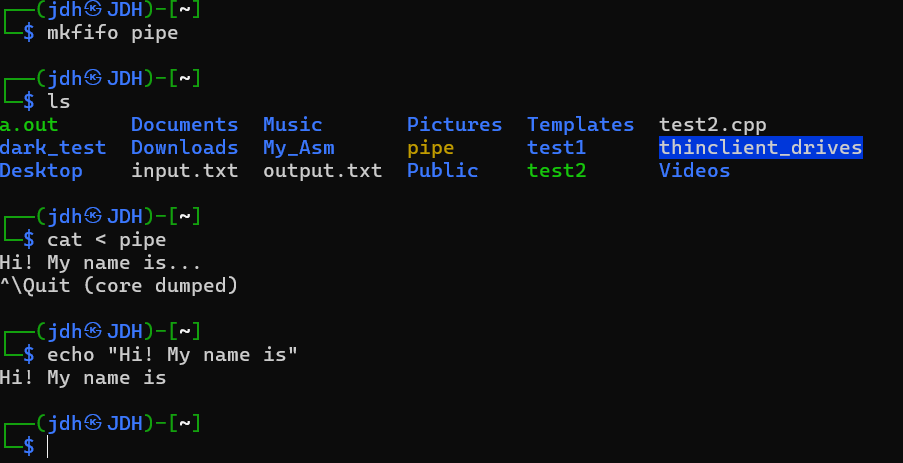
2.5 Сравнение запуска с pstree more:



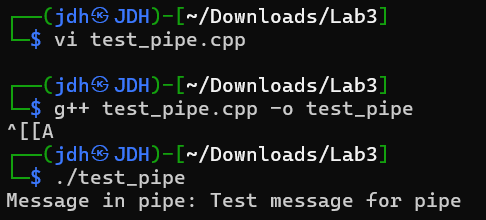


3. Создайте программный канал типа named pipe из командной строки (пример в лекциях). Используя необходимые системные вызовы организуйте канал named pipe в программе, сравните результат выполнения обмена по нему в программе с тем, чего можно достичь, создавая named pipe на терминале.

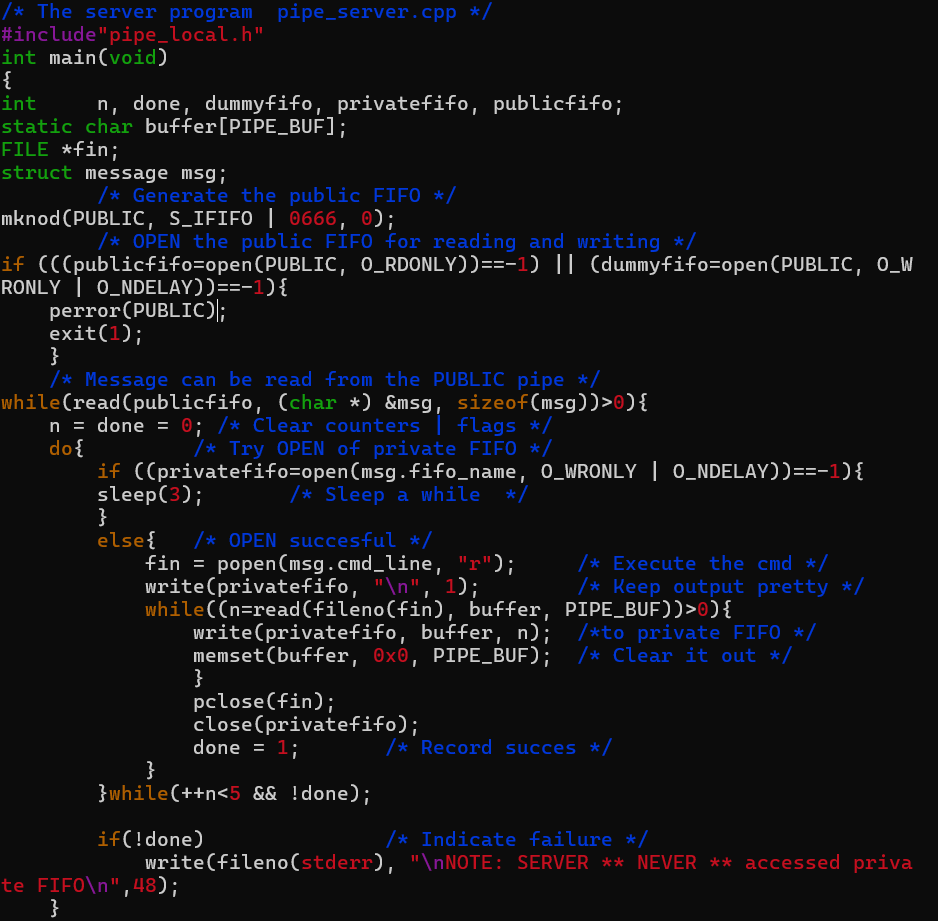
3.1 shell



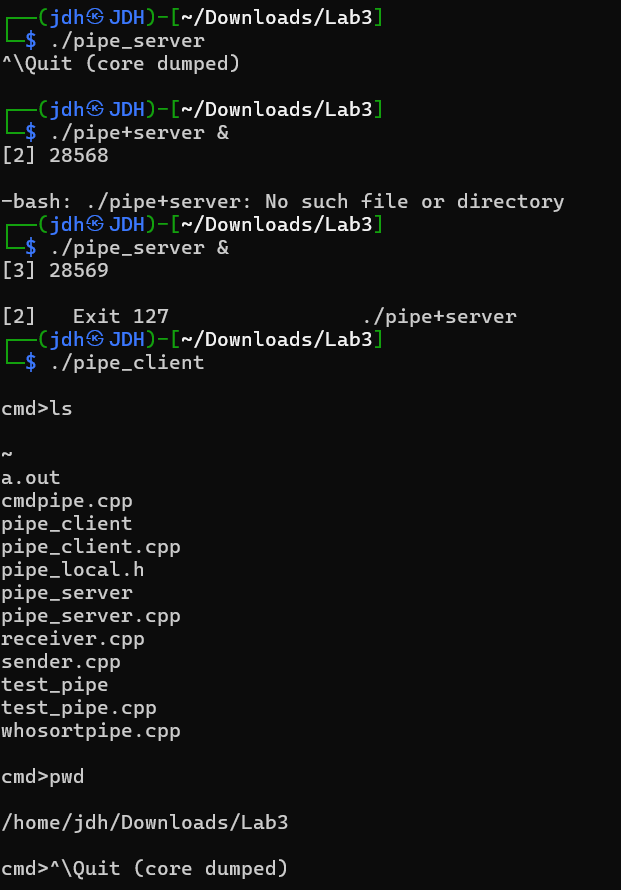
3.2



4. Разберите и выполните пример клиент-серверного взаимодействия, организованного на конвейерах различного типа. Исходный текст примера содержится в файлах pipe\_server.cpp , pipe\_client.cpp и pipe\_local.h и разобран в материалах лекций. Сервер запускается в фоновом режиме. Проанализируйте какие конвейеры используются, как они создаются, как функционирует данная системы, ее недостатки. Программа-сервер этого примера исполняет каждый командный запрос поочередно. Если какой-либо запрос потребует много времени, все остальные клиентские процессы будут ожидать обслуживания. Коды программ были переписаны, так как прежние вызывали ошибки при вызове из терминала.





******

### ***Схема взаимодействия***

1. **Клиент**:
   * Создает приватный FIFO.
   * Открывает публичный FIFO для записи.
   * Вводит команды и отправляет их серверу через публичный FIFO.
   * Открывает приватный FIFO для чтения результатов.
   * Читает и отображает результаты на stderr.
   * Закрывает и удаляет приватный FIFO перед завершением работы.
2. **Сервер**:
   * Создает публичный FIFO.
   * Открывает публичный FIFO для чтения.
   * Ожидает сообщений от клиентов.
   * При получении сообщения пытается открыть приватный FIFO клиента.
   * Выполняет команду и отправляет результаты обратно клиенту через приватный FIFO.
   * Закрывает приватный FIFO после отправки результатов.

***Анализ конвееров:******Взаимодействие происходит через два типа FIFO:***

***Публичный FIFO:*** *Используется для приема командных запросов от клиентов.*

***Приватный FIFO:*** *Создается для каждого клиента, чтобы вернуть ему результаты выполнения команды.*

***Функционирование системы:***

***1. Создание FIFO:*** *Публичный канал, создается с помощью mkfifo. Приватные каналы создаются по запросу клиента и используются для передачи результатов выполнения команды.*

***2. Клиент*** *отправляет команду на сервер через публичный канал и ждет ответа в своем приватном канале.*

***3. Сервер*** *получает команду через публичный канал, выполняет её с помощью popen(), а затем отправляет результат обратно клиенту через его приватный канал.*

***Недостатки системы:***

***Последовательная обработка —*** *сервер выполняет запросы клиентов по очереди, из-за чего медленные запросы задерживают остальные.*

***Отсутствие многозадачности —*** *сервер не обрабатывает несколько запросов одновременно.*

***Постоянное создание каналов —*** *для каждого запроса клиента создается и удаляется приватный канал.*

5.Модифицируйте программу pipe\_server.cpp из предыдущего задания так, чтобы при получении нового сообщения от очередного клиента сервер порождал очередной дочерний процесс для выполнения задачи обслуживания данного запроса (исполнения переданной от клиента команды и переправки результата обратно клиенту).

**Вывод:**

Мы использовали программные каналы для межпроцессорной коммуникации, увидели, какие недостатки и ограничения могут возникать при их применении. Также изучили простейшее клиент-серверное взаимодействие.