Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**Тема работы**

**“Изучение взаимодействий между процессами”**

Студент: Прохоров Данила Михайлович Группа: М8О-208Б-20

Вариант: 16

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/dmprokhorov/OS

**Постановка задачи**

Задача: реализовать программу, в которой родительский процесс создает один дочерний процесс. Родительский процесс принимает путь к файлу и строки, которые отправляются в тот дочерний процесс, там те из них, которые оканчиваются на символы ‘;’ или ‘.’, записываются в файл, если же строки не удовлетворяют этому правилу, то они возвращаются в родительский процесс. Далее в родительском процессе сначала выводятся строки из файла (если его удалось открыть и там есть хотя бы одна строка), а потом строки, вернувшиеся из родительского процесса в дочерний.

**Общие сведения о программе**

В программе используются следующие библиотеки:

* <iostream> - для вывода информации на консоль
* <fstream> - для записи текста в файл.
* <unsitd.h> - для системных вызовов read и write в Ubuntu.
* <sys/wait.h> - для функции waitpid, когда родительский процесс ждёт дочерний.

В задании используются такие команды и строки, как:

* **int fd[2]** - создание массива из 2 дескрипторов, 0 - чтение (read), 1 - передача (write). И
* **pipe(fd)** - конвейер, с помощью которого выход одной команды подается на вход другой (также “труба”). Это неименованный канал передачи данных между родственными процессами.
* **pid\_t child = fork ()** - создание дочернего процесса, в переменной child будет храниться “специальный код” процесса (-1 - ошибка fork, 0 - дочерний процесс, >0 - родительский)
* **read(int fd, void\* buf, size\_t count) (здесь дан общий пример)** - команда, предназначенная для чтения данных, посланных из другого процесса, принимающая на вход три параметра: элемент массива дескрипторов с индексом 0 (в моей программе fd1[0], fd2[0]), указатель на память получаемого объекта (переменной, массива и т.д.), размер получаемого объекта (в байтах).
* **write(int fd, void\* buf, size\_t count) (здесь дан общий пример) -** команда, предназначенная для записи данных в другой процесс, принимающая на вход три параметра: элемент массива дескрипторов с индексом 1 (в моей программе fd1[1], fd2[1]), указатель на память посылаемого объекта (переменной, массива и т.д.), размер посылаемогообъекта (в байтах).
* **close(int fd)** - команда, закрывающая файловый дескриптор.
* **int wstatus;**

**waitpid(child, &wstatus, 0)** – команда ожидания завершения процесса с id, равным child, если она завершилась без ошибок, то в переменной wstatus будет лежать значение 0, и родительский процесс продолжит своё выполнение, в противном случае там будет лежать значение ненулевое значение, и родительский процесс аварийно завершится.

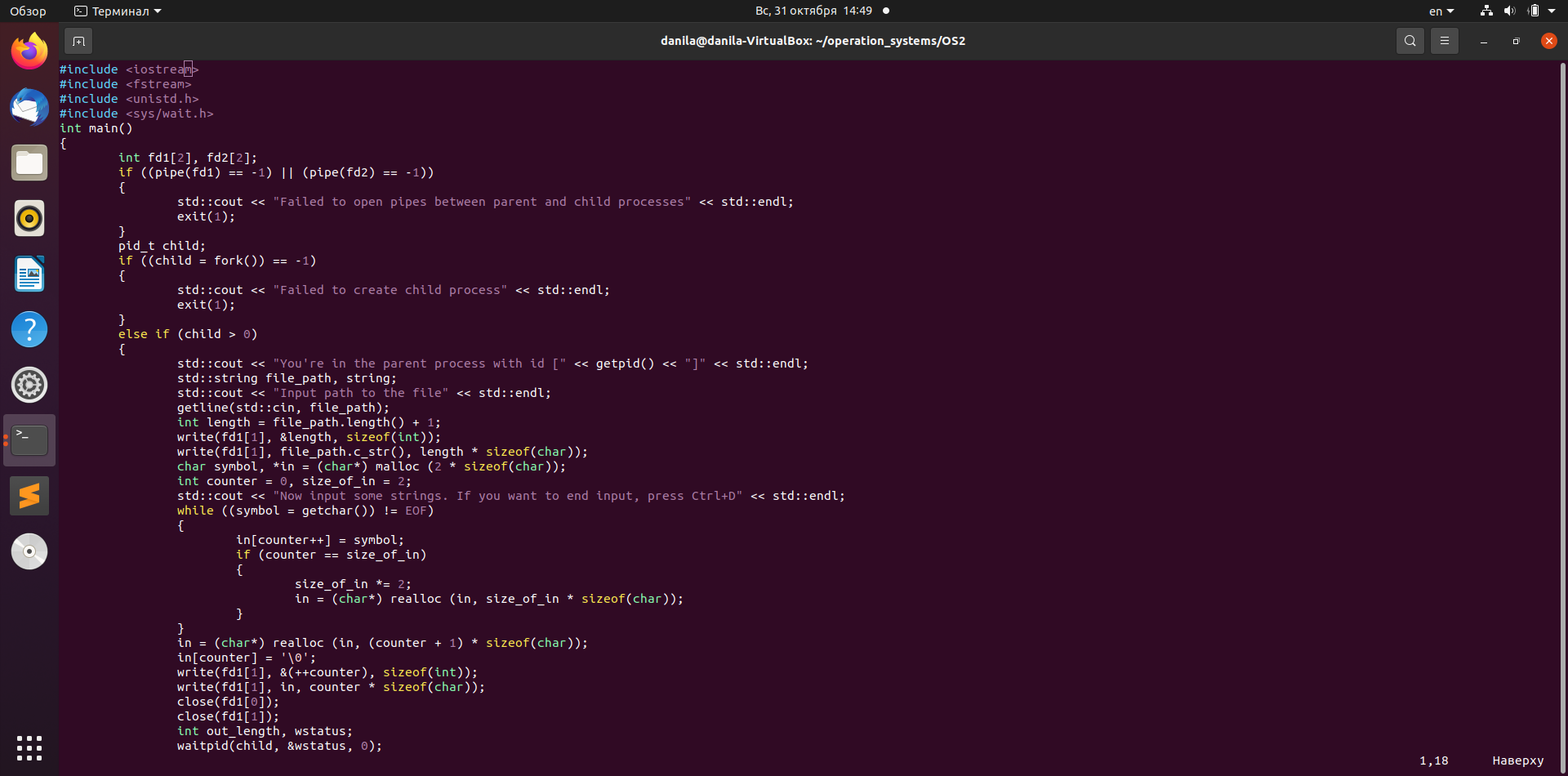
**Общий метод и алгоритм решения**

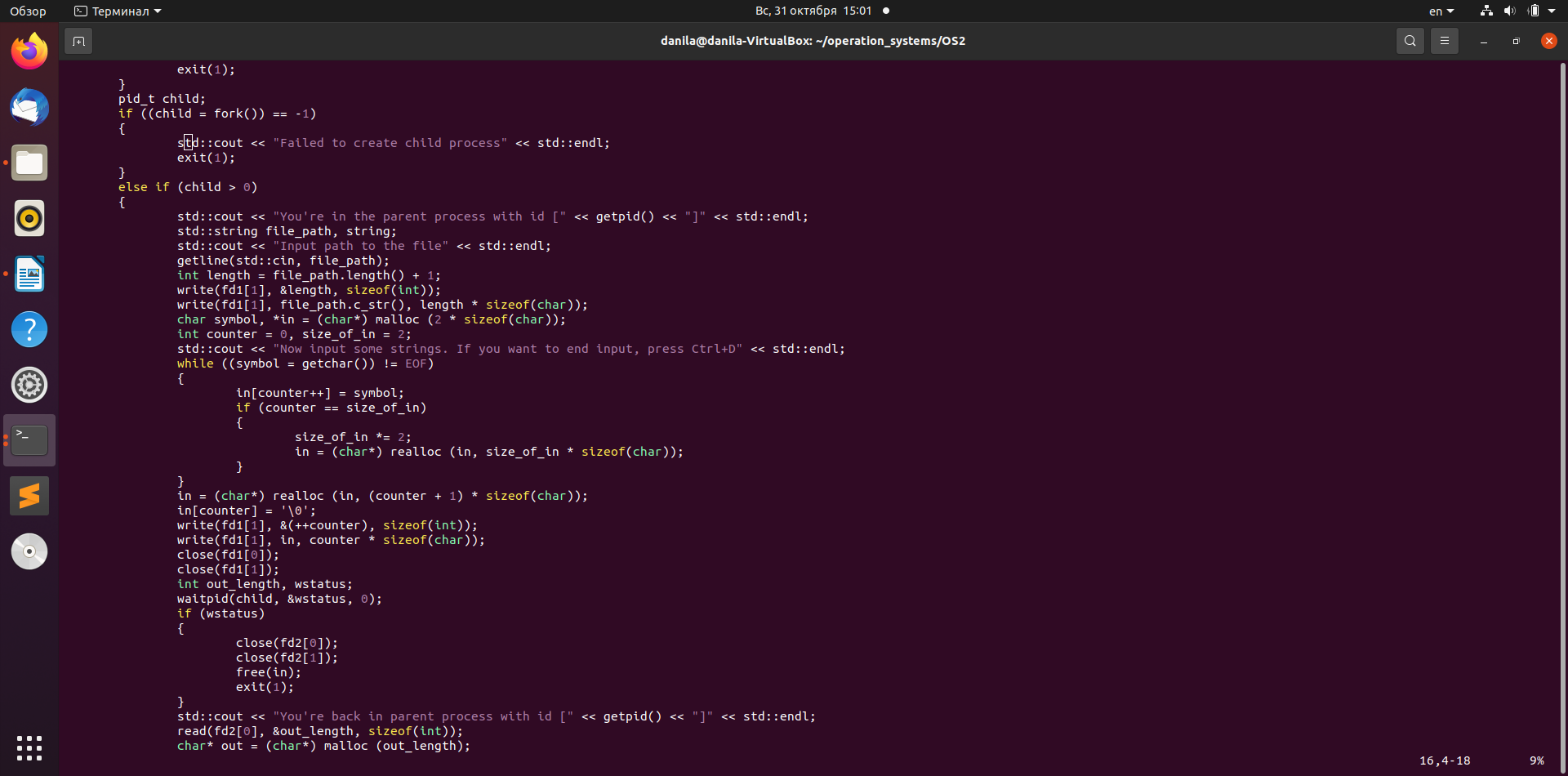
В начале программа получает на вход путь к файлу, где будут лежать нужные строки, затем пользователю предлагается ввести строки, конец ввода должен сигнализироваться символов Ctrl+D (это некоторое количество строк считывается как одна (с символами переноса строки и символом конца строки – ‘\0’). Далее программа создаёт дочерний процесс, если этого сделать не удалось, то она аварийно завершается.

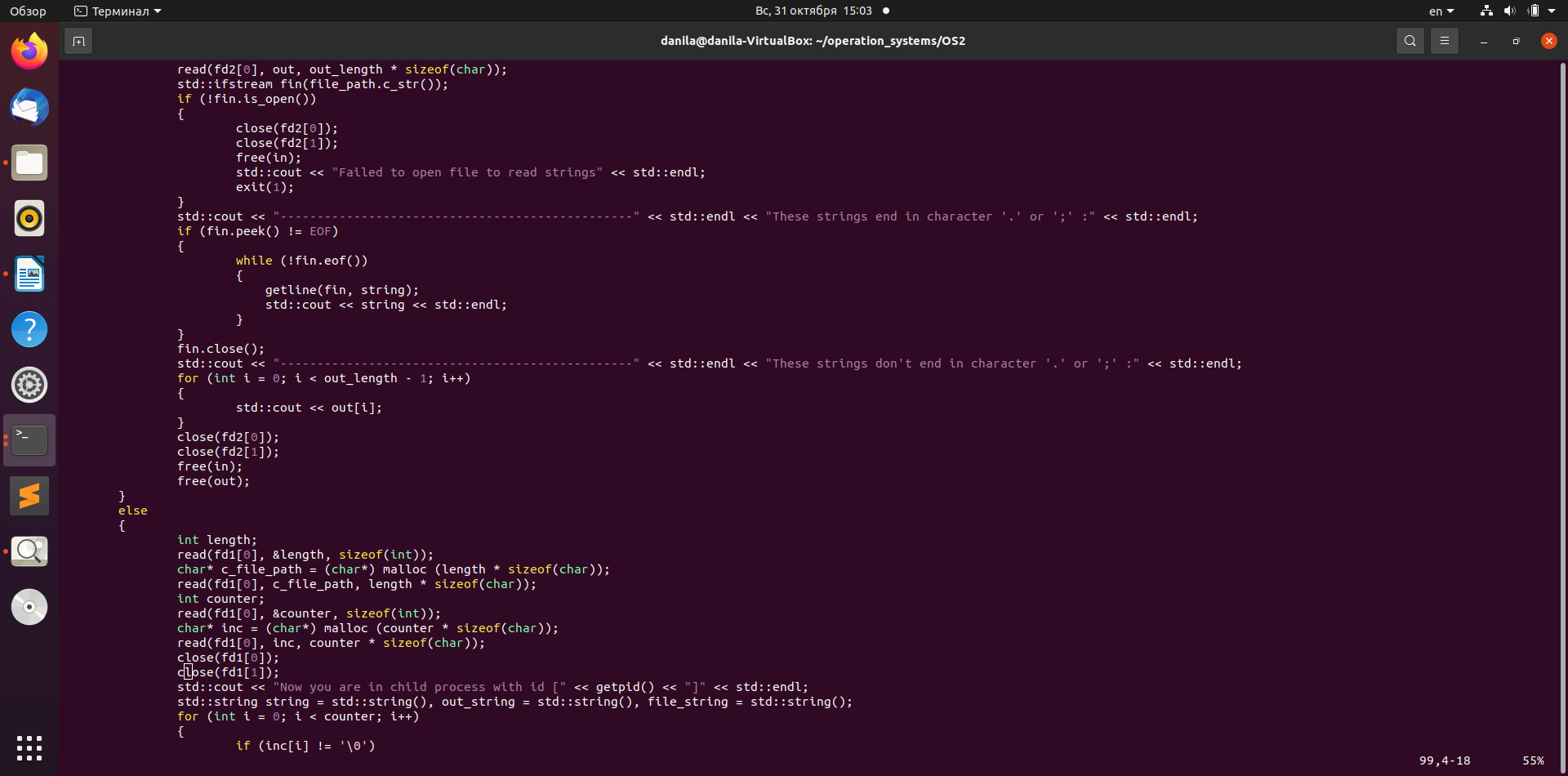
В дочернем процессе получается имя файла, а также большая исходная строка. В результате прохода по строке и некоторых операций получаются 2 строки file\_string и out\_string. Если удалость открыть файл, то file\_string записывается в файл, out\_string по неименованному каналу передаётся обратно в родительский процесс, вся память чистится, закрываются файловые дескрипторы и возвращается значение 0. Если же файл открыть не удалось, то выводится информация об ошибке открытия файла, и возвращается значение 1.

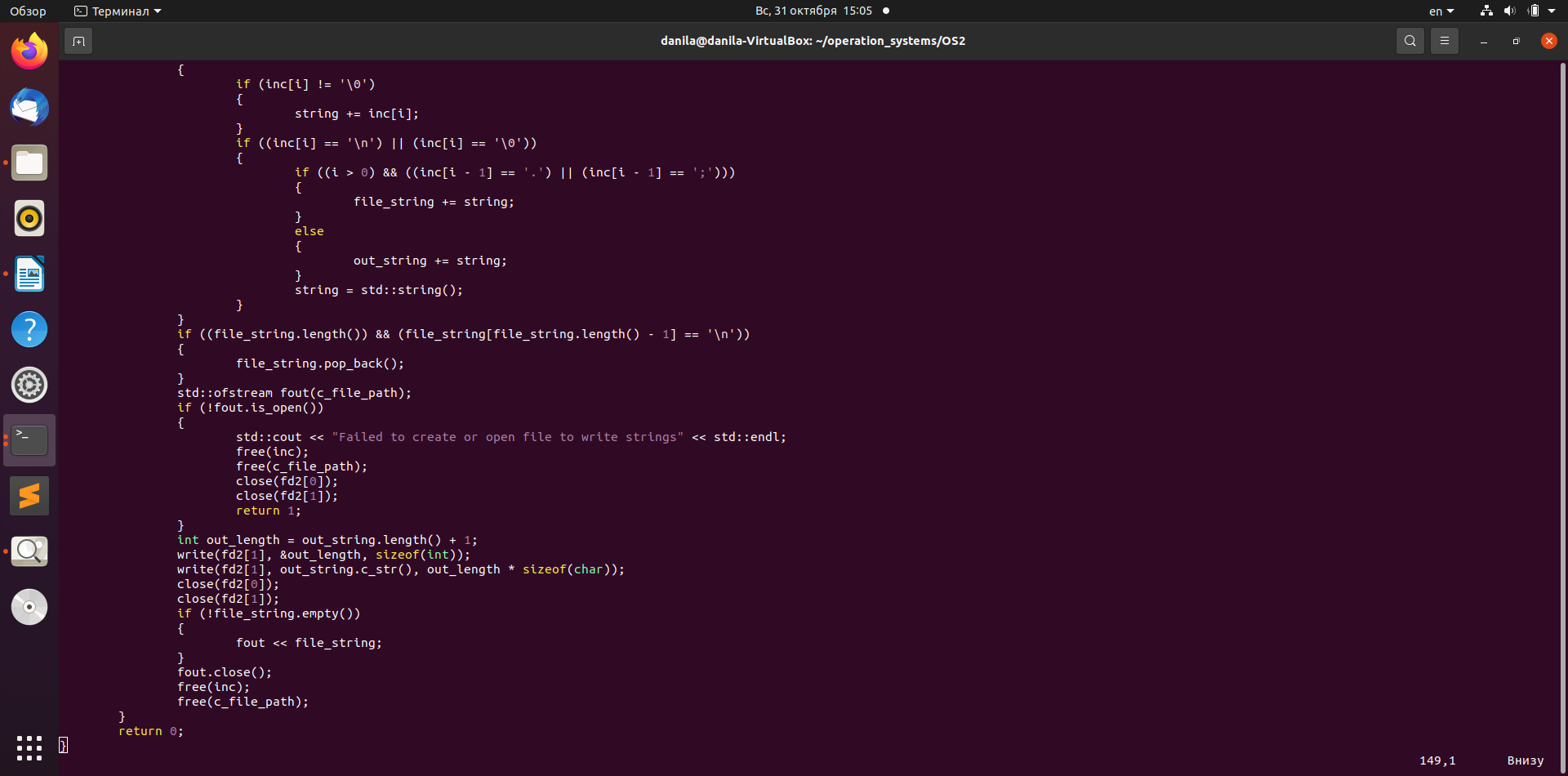
Родительский процесс сразу, как только запускается, ждёт дочерний. Если получаемое значение 1, то программа завершается, предварительно очистив всю память и закрыв файловые дескрипторы. Если же возвращаемое значение 0, то тогда программа пытается открыть файл. Если это не получилось сделать, то она аварийно завершается. В противном случае она читает из файла строки и выводит их, предварительно написав, что это строки, удовлетворяющие правилу. Дальше выводятся строки, не удовлетворяющие правилу. Далее чистится вся память и удаляются файловые дескрипторы.  
Собирается программа при помощи команды g++ task.cpp -o task, запускается при помощи команды ./task.

**Исходный код**

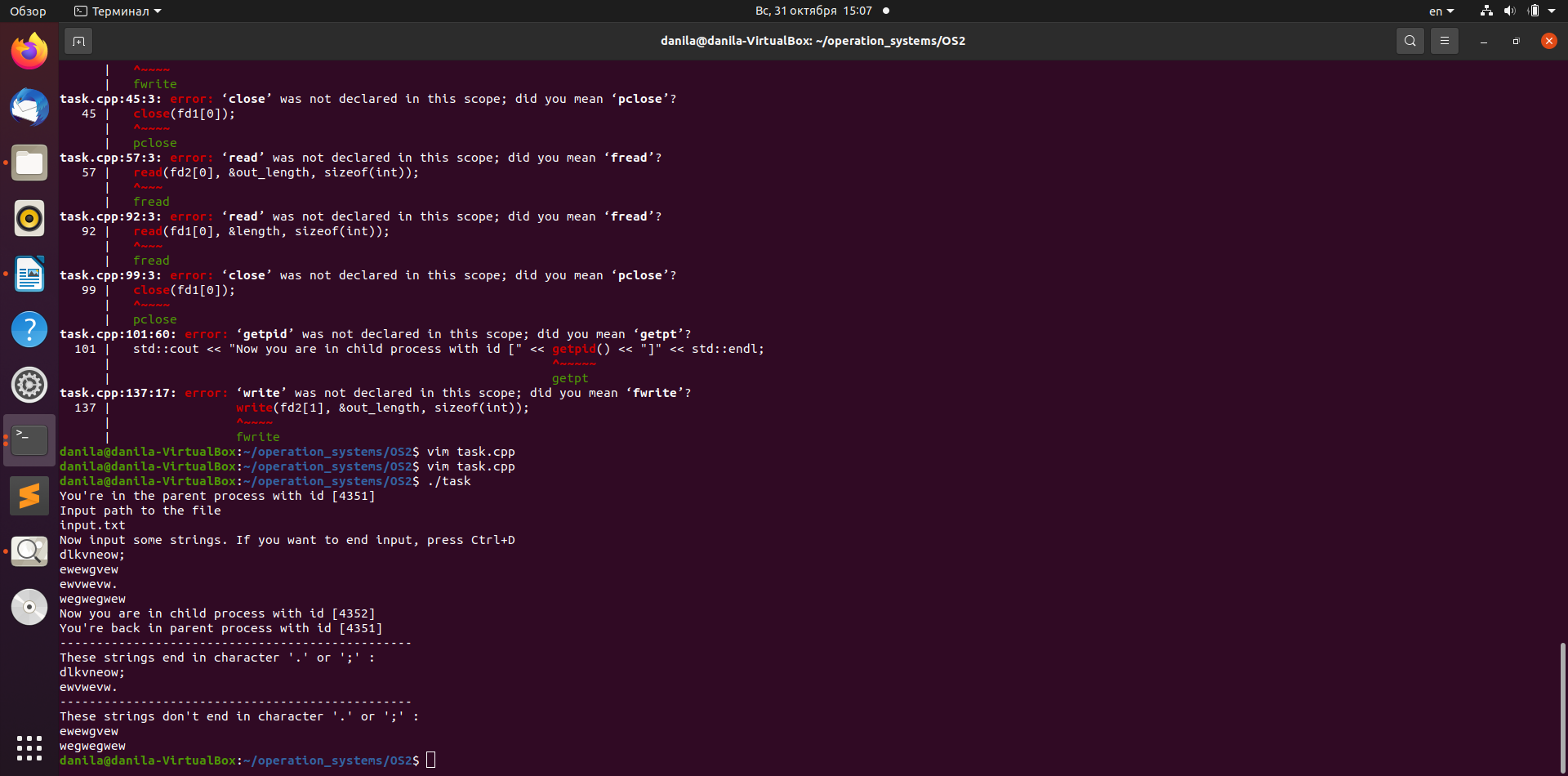
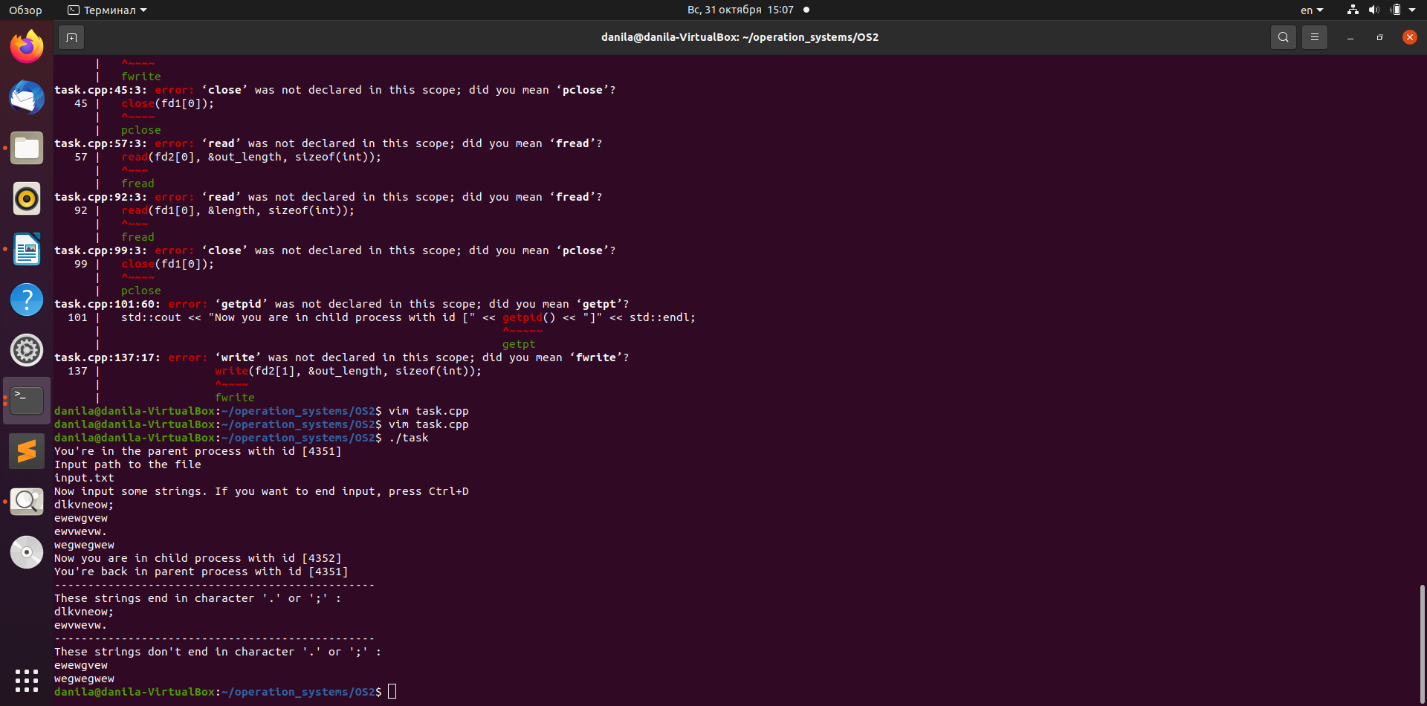




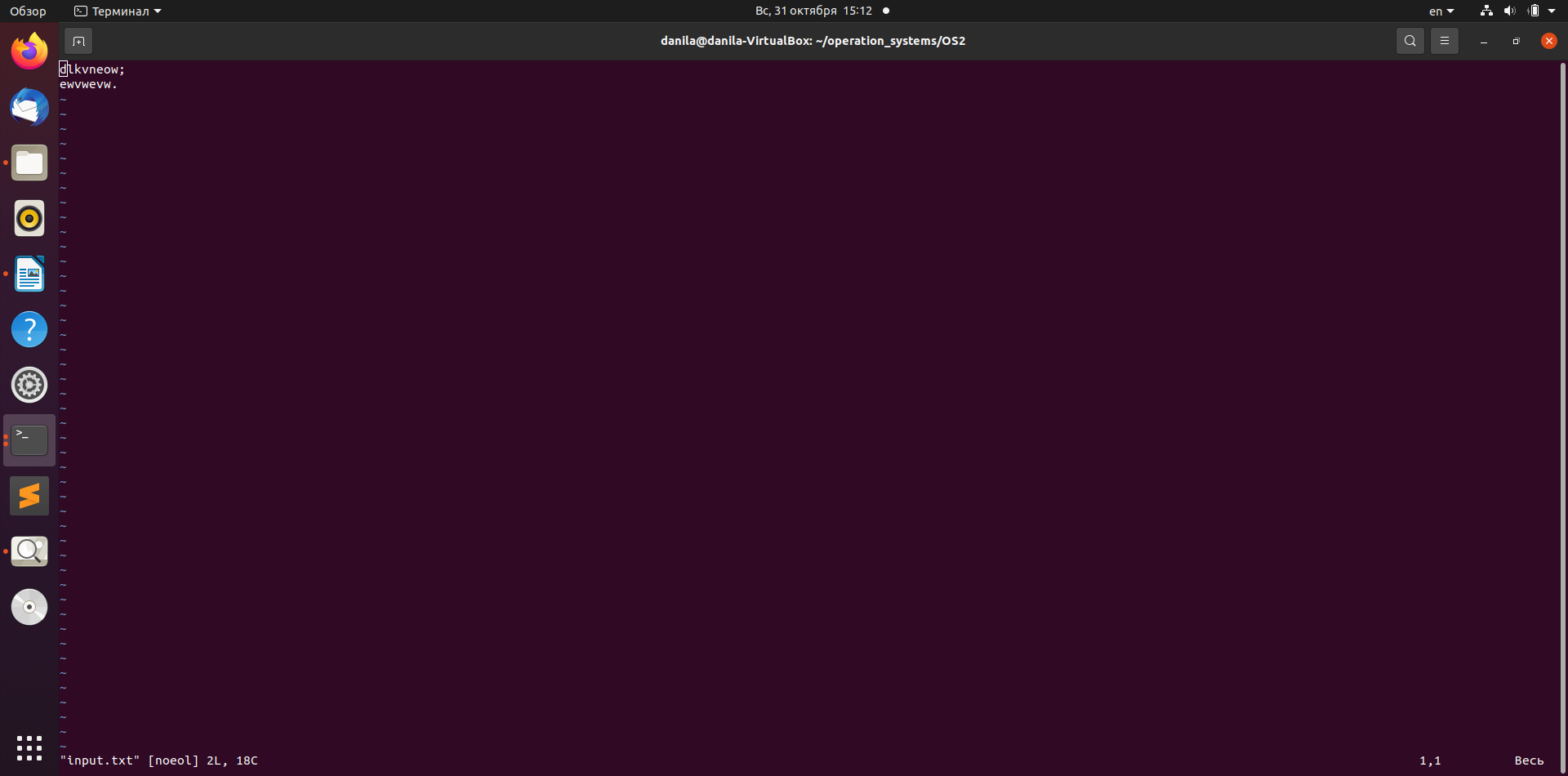




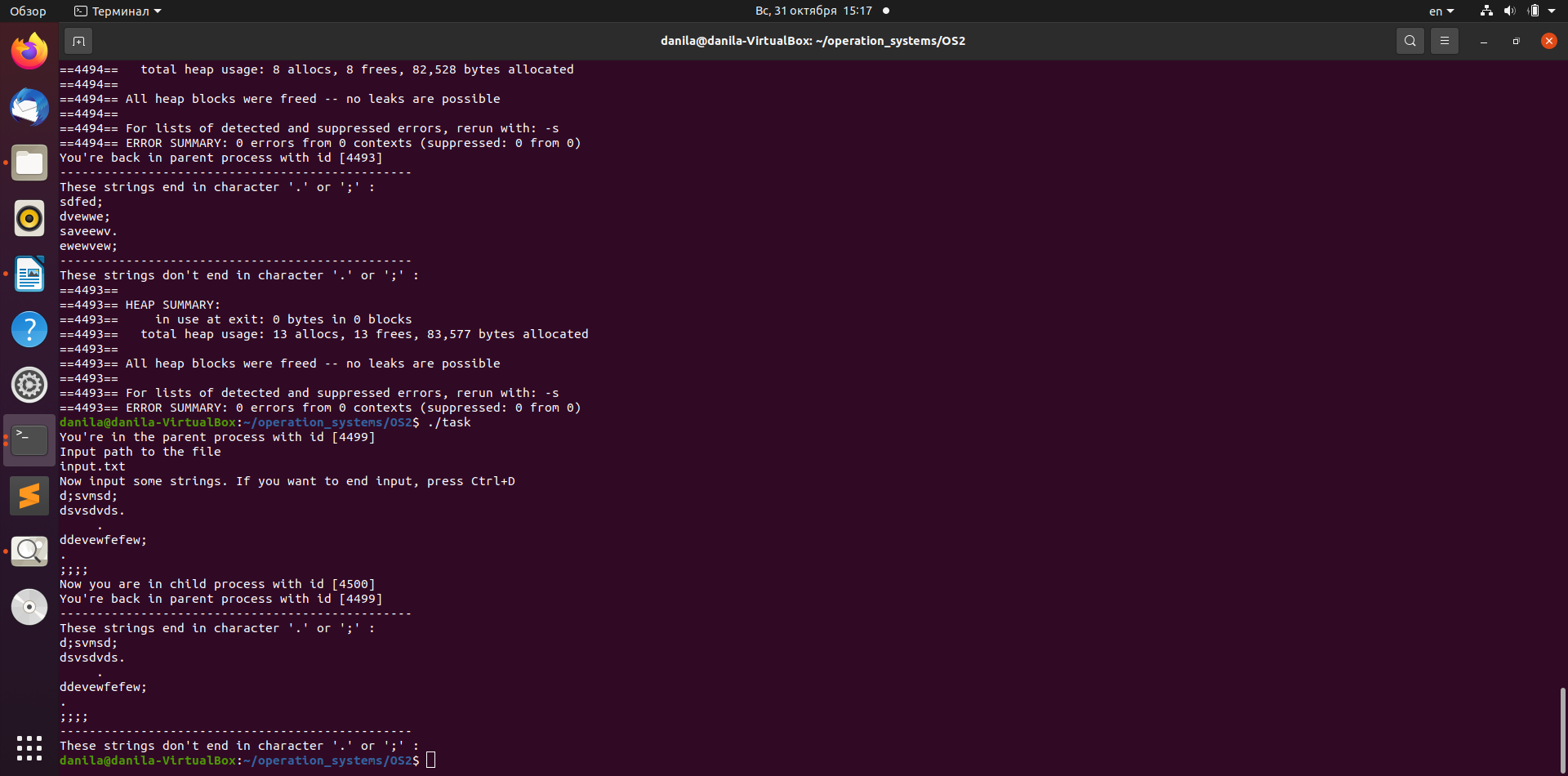
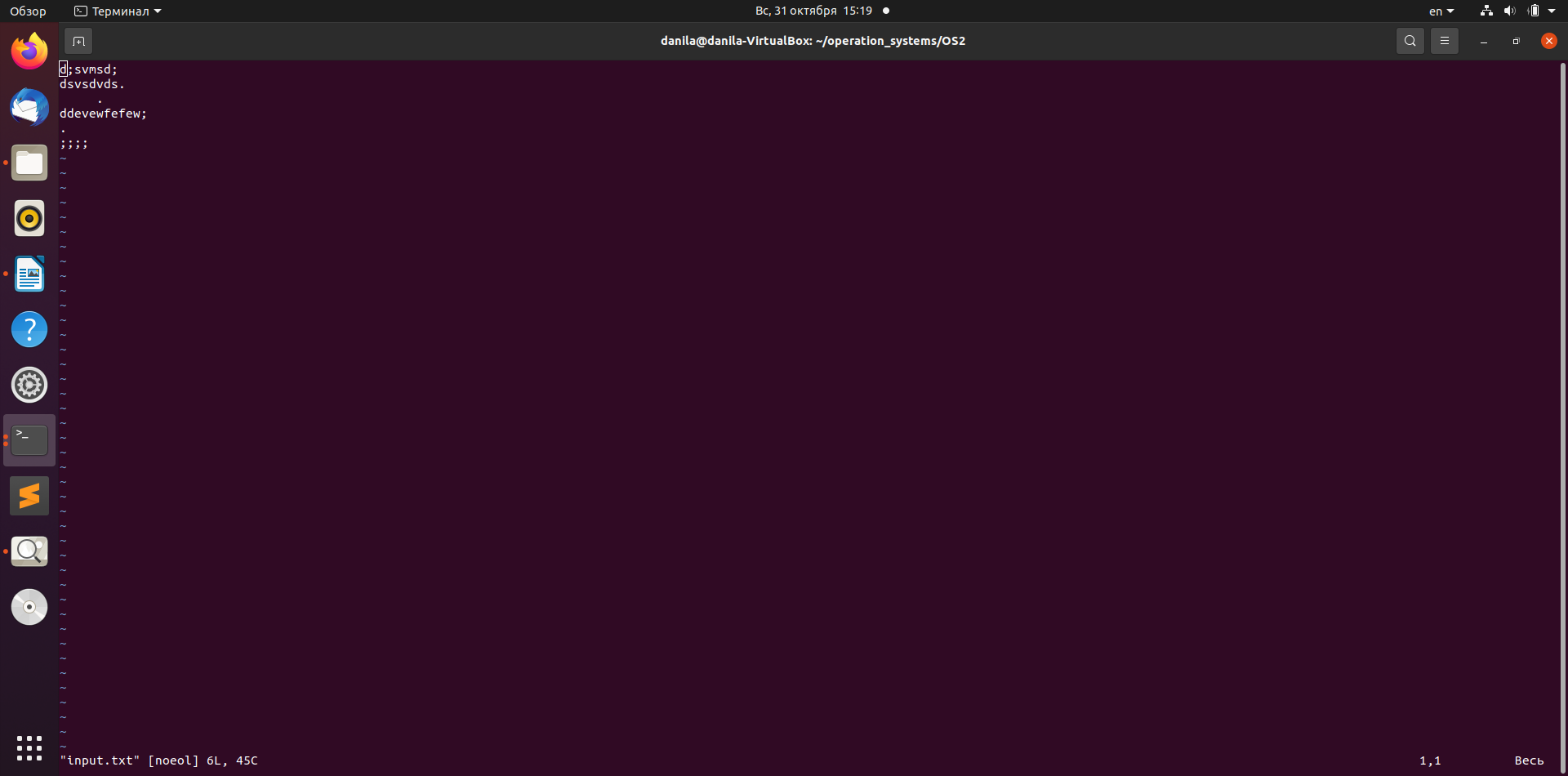
**Демонстрация работы программы**Тест 1.Вводится корректное имя файла, 2 строки оканчиваются на точку или точку с запятой, 2 нет.



Содержание файла input.txt

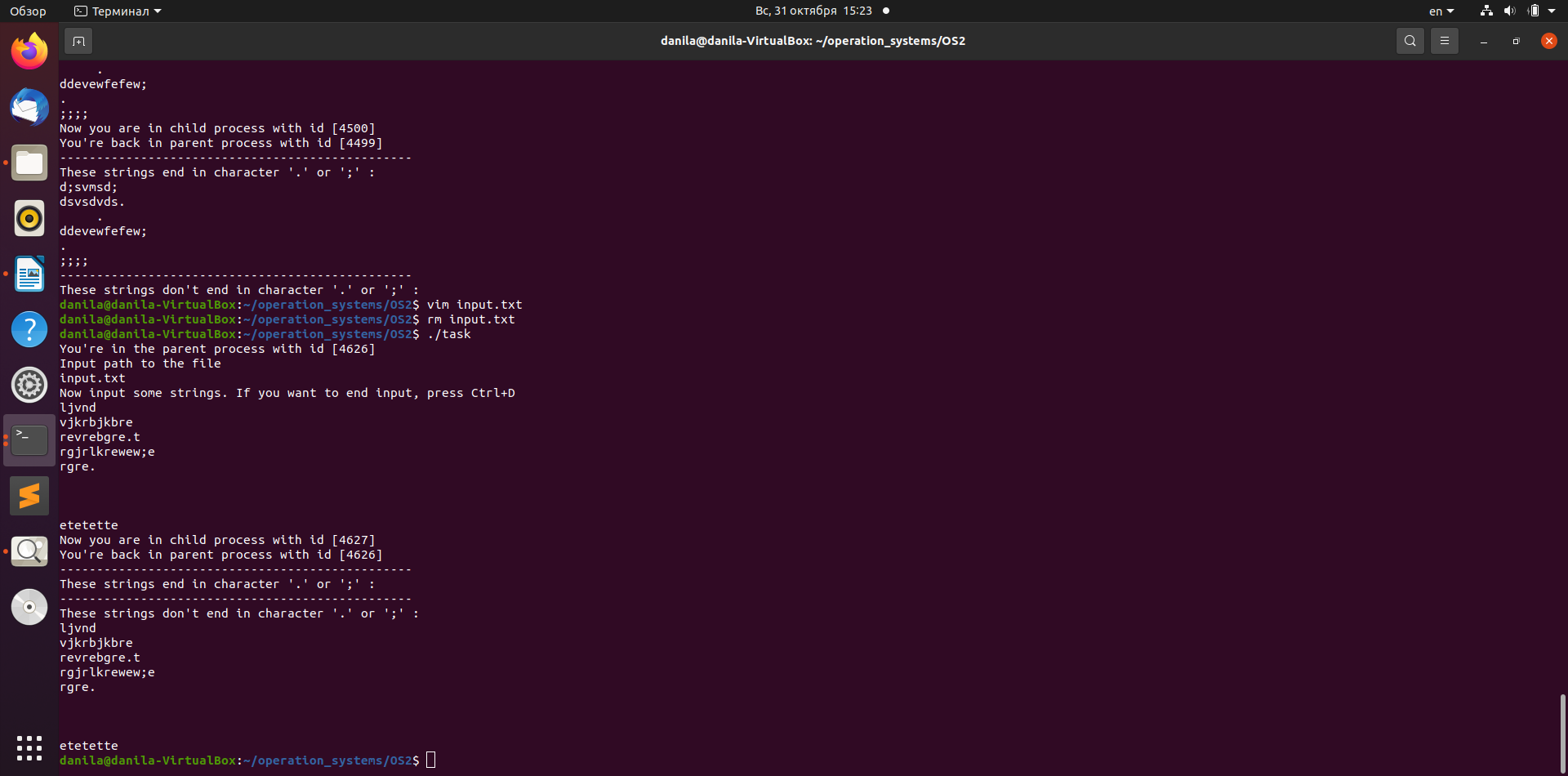
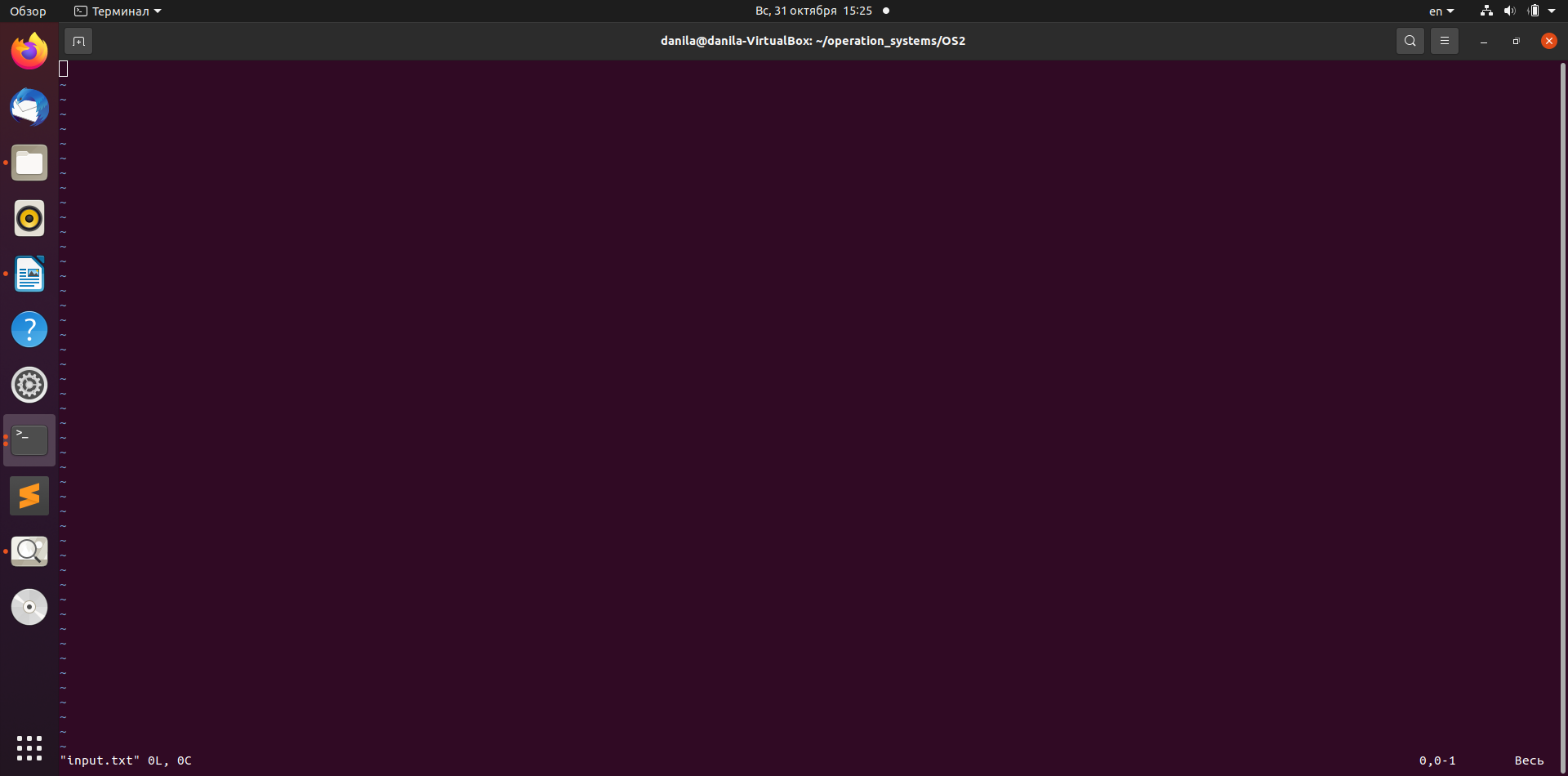


Тест 2.  
Вводится корректное имя файла, но все строки удовлетворяют правилу.

  
Содержание файла input.txt

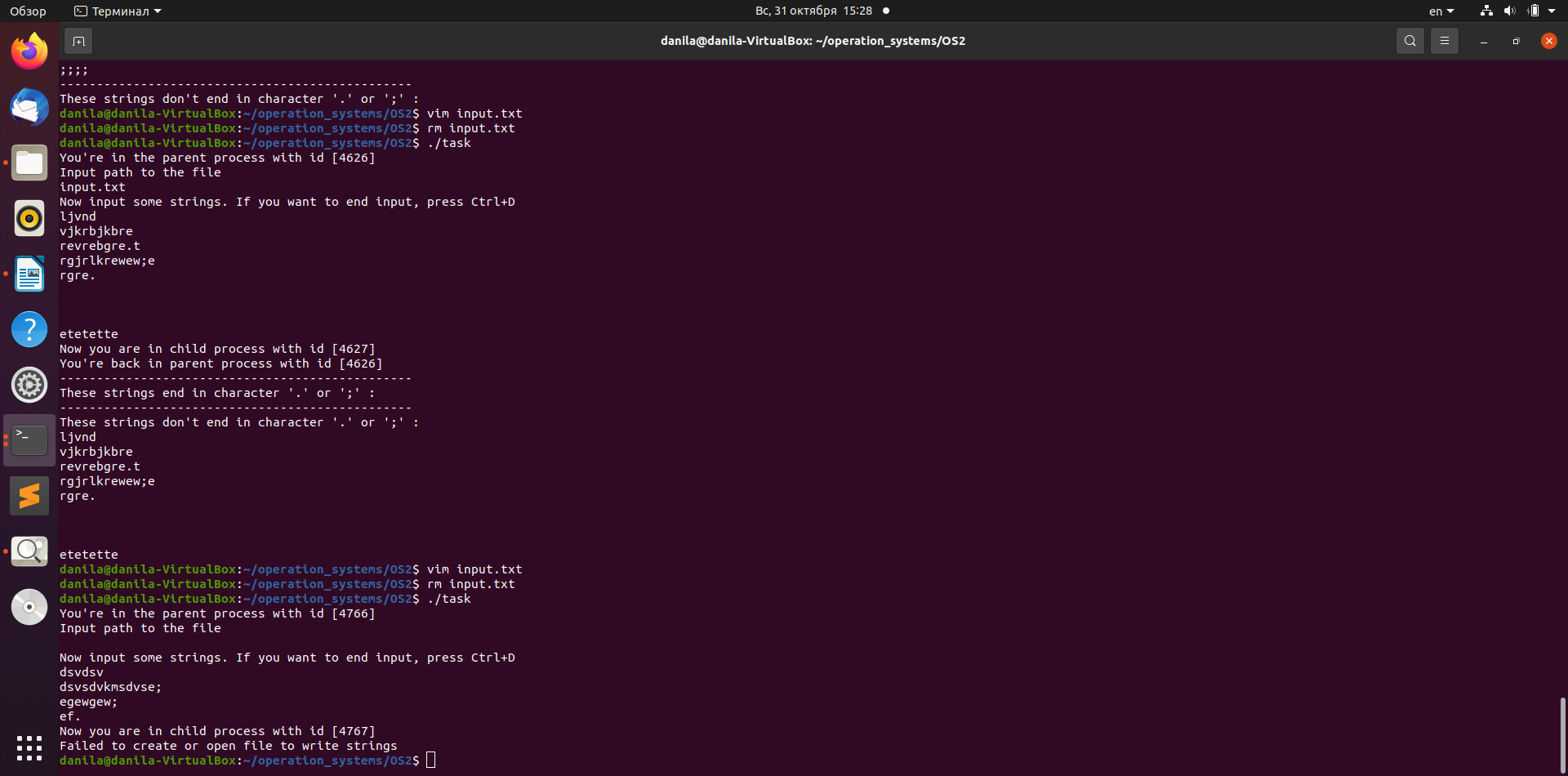
Тест 3

Вводится корректное имя файла, но все строки не удовлетворяют правилу (в строке “regre.” После точки следуют ещё 3 пробела, так что формально она имеет вид “regre. “. Пустые строки также формально не оканчиваются на точку или точку с запятой.

Содержание файла input.txt

Тест 4

Вводится некорректное имя файла (просто символ переноса строки)



**Выводы**Это была моя первая лабораторная работа по курсу “Операционные системы”. Было интересно узнать много нового и про системные вызовы, и про межпроцессное взаимодействие.