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**

**“Взаимодействие между процессами с помощью отображаемых файлов”**

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:

* <stdio.h> - для вывода информации на консоль
* <fcntl.h> - для работы с файлами
* <unsitd.h> - для системных вызовов fork, ftruncate и close в Ubuntu
* <sys/wait.h> - для функции waitpid, когда родительский процесс ждёт дочерний
* <sys/mman.h> - для вызова функций, использующихся для операций с отображаемыми файлами (mmap, mremap, munmap)
* <sys/stat.h> - для вызова функции fstat
* <string.h> - для вызова функции strcpy
* <string> - для работы со строками в формате C++

В задании используются такие команды и строки, как:

* **void \*mmap(void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset)** **–** команда, отображающая файл на физическую память (ОЗУ) и принимающая 6 аргументов:

А) Адрес начала сопоставления (у меня в программе везде NULL, то есть ядро операционной системы само выбирает адрес).

Б) Длина (в байтах).

В) Аргументы PROT (у меня везде PROT\_READ | PROT\_WRITE, то есть отображение можно прочесть, а можно и записать в него информацию).

Г) Флаги, означающие, видны ли отображения другим процессам (у меня везде есть MAP\_SHARED – обновления видны всем процессам, имеющим данное представление отображаемого файла, также если присутствует флаг MAP\_ANONYMOUS, то следующий, 5 аргумент игнорируется).

Д) Файловый дескриптор – если не указан флаг MAP\_ANONYMOUS в предыдущем, 4 аргументе, то при записи в отображаемый файл информация будет записываться в файл, дескриптор когорого указан в данном аргументе.

Е) Смещение на указанное количество байт от начальной позиции, должно быть кратно размеру страницы (у меня смещений нет, так что везде указано 0).

В случае успеха функция возвращает указатель на отображаемый файл, в противном случае – указатель типа void на -1.

* **int open(const char \*pathname, int flags, mode\_t mode) –** функция, открывающая файл и принимающая 3 аргумента:

А) Путь к файлу.

Б) Флаги (в моём случае это O\_RDWR | O\_CREAT, означающие, что в файл можно записывать, а также читать, и если файла не существовало, то создать его).

В) Параметры (в моём случае самые “разрешающие” параметры – S\_IRWXU | S\_IRWXG | S\_IRWXO, то есть файл разрешено читать и записывать в него всем пользователям).

В случае успеха вернёт число, означающее файловый дескриптор, в противном случае будет возвращено -1.

* **int munmap(void \*addr, size\_t length)** **–** функция, удаляющая отображение из заданной области и принимающая 2 аргумента:

А) Указатель на отображаемый файл.

Б) Размер отображаемого файла.

В случае успеха возвращает значение 0, в противном случае возвращает -1.

* **int ftruncate(int fd, off\_t length) –** функция, урезающая файл до указанной длины и принимающая 2 аргумента:

А) Файловый дескриптор.

Б) Новая длина файла.

В случае успеха вернётся значение 0, в противном случае -1.

* **void \*mremap(void \*old\_address, size\_t old\_size, size\_t new\_size, int flags, ... /\* void \*new\_address \*/) –** функция, изменяющая размер и возможно адрес отображаемого файла и принимающая 4 или 5 аргументов:

А) Указатель на старый адрес виртуальной памяти.

Б) Старый размер отображаемого файла.

В) Новый размер отображаемого файла.

Г) Флаги (у меня присутствует флаг MREMAP\_MAYMOVE, означающий, что если изменить размер памяти, находясь на текущем адресе нельзя, то ядро операционной системы может поменять адрес, однако в таком случае остальные указатели на данную область памяти становятся невалидными, но у меня никаких разных указателей на одинаковые куски памяти нет, поэтому я могу смело использовать этот флаг).

Д) Новый адрес памяти, если в предыдущем, 4 пункте, указан флаг MREMAP\_FIXED, то тогда можно указать новый адрес памяти, куда будет перемещено отображение файла (у меня нигде 5 аргумент функции mremap не используется).

В случае успеха функция вернёт указатель на новый участок памяти

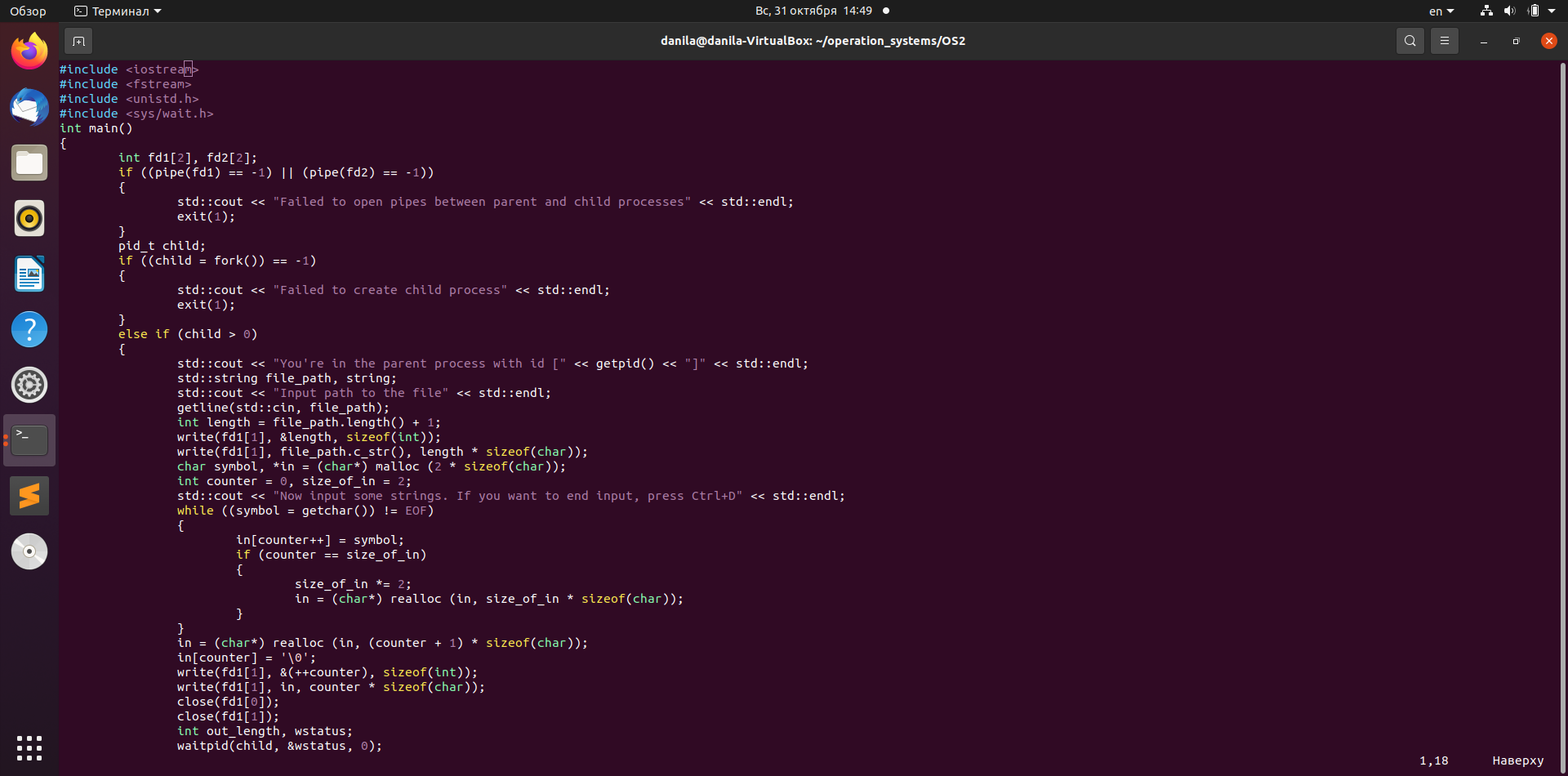
**Общий метод и алгоритм решения**

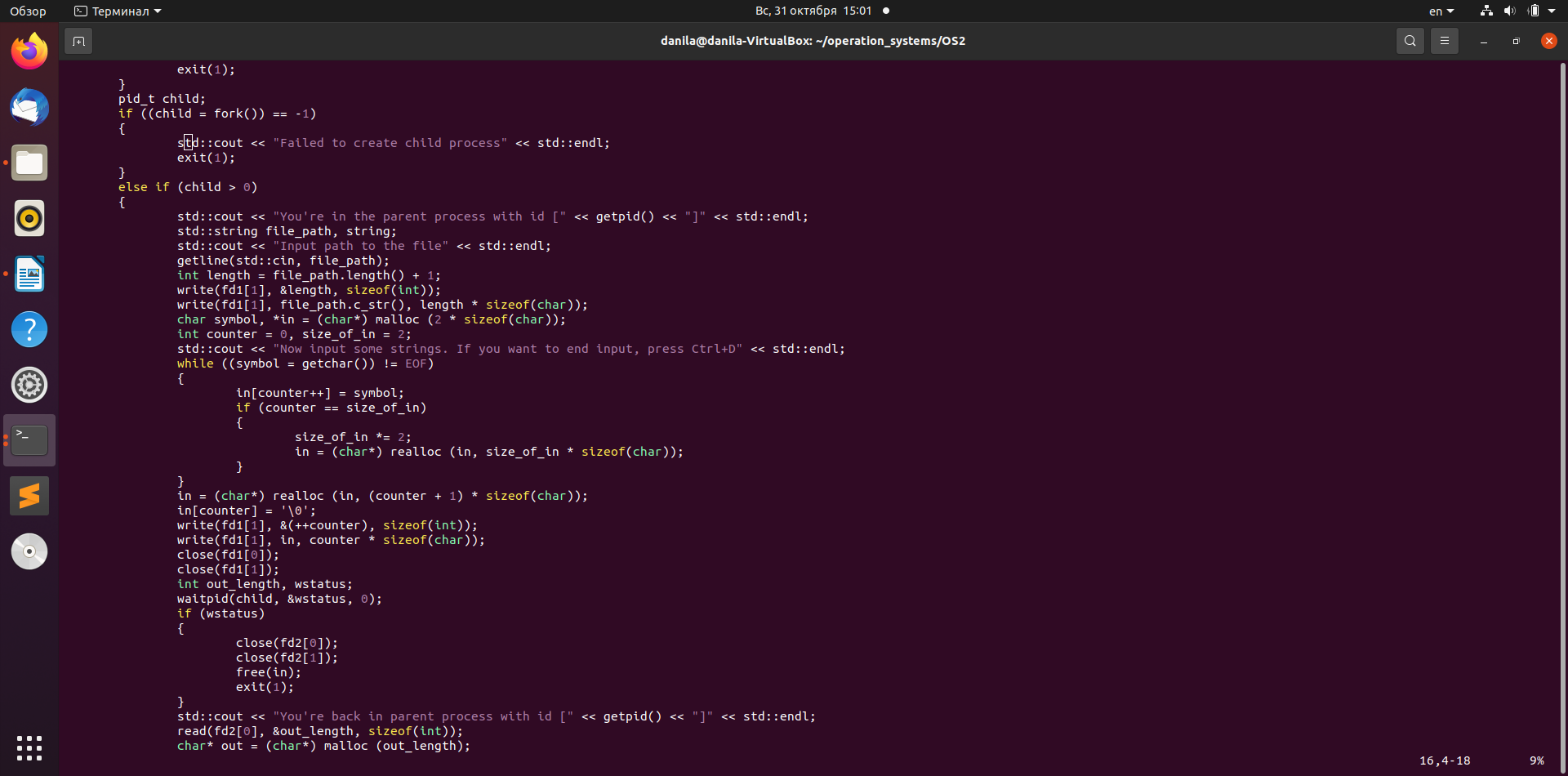
В начале программа получает на вход путь к файлу, где будут лежать нужные строки, затем пользователю предлагается ввести строки, конец ввода должен сигнализироваться символов Ctrl+D (это некоторое количество строк считывается как одна (с символами переноса строки и символом конца строки – ‘\0’). Далее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

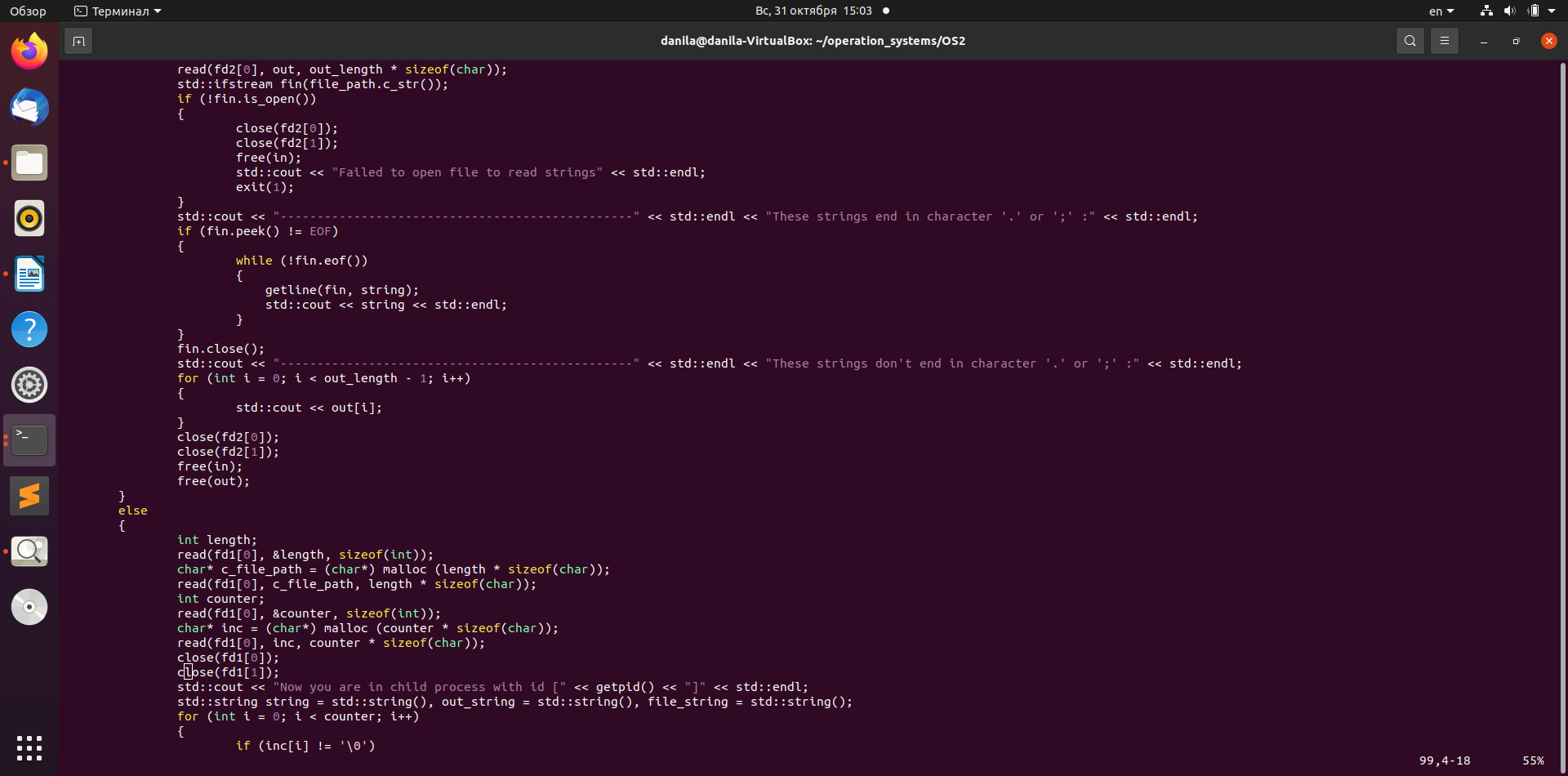
В дочернем процессе получается имя файла, а также большая исходная строка. В результате прохода по строке и некоторых операций получаются 2 строки file\_string и out\_string. Если удалость открыть файл, то file\_string записывается в файл, out\_string по неименованному каналу передаётся обратно в родительский 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.

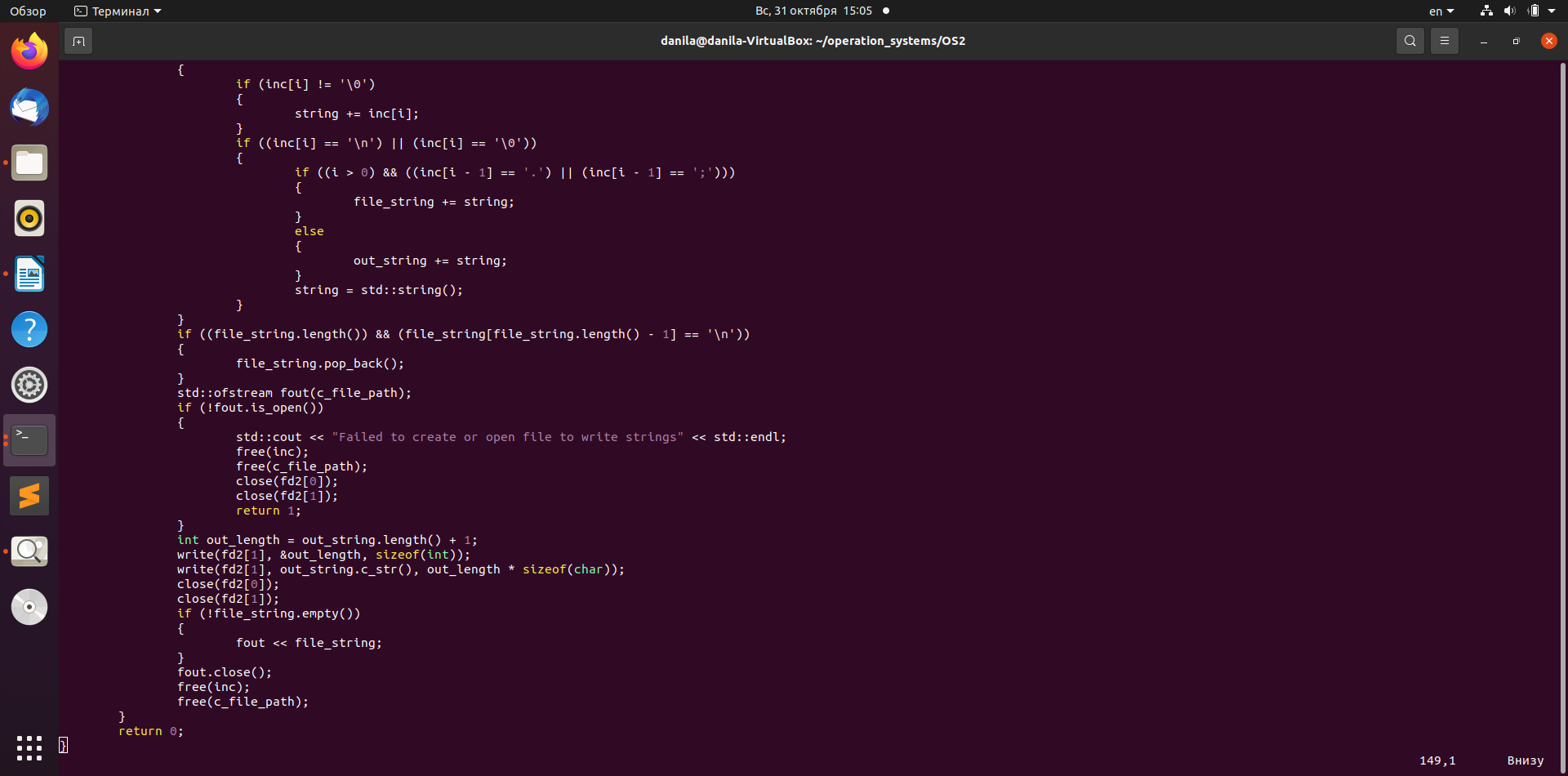
Родительский процесс сразу, как только запускается, ждёт дочерний. Если получаемое значение 1, то программа завершается, предварительно очистив всю память и закрыв файловые дескрипторы. Если же возвращаемое значение 0, то тогда программа пытается открыть файл. Если это не получилось сделать, то она аварийно 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**

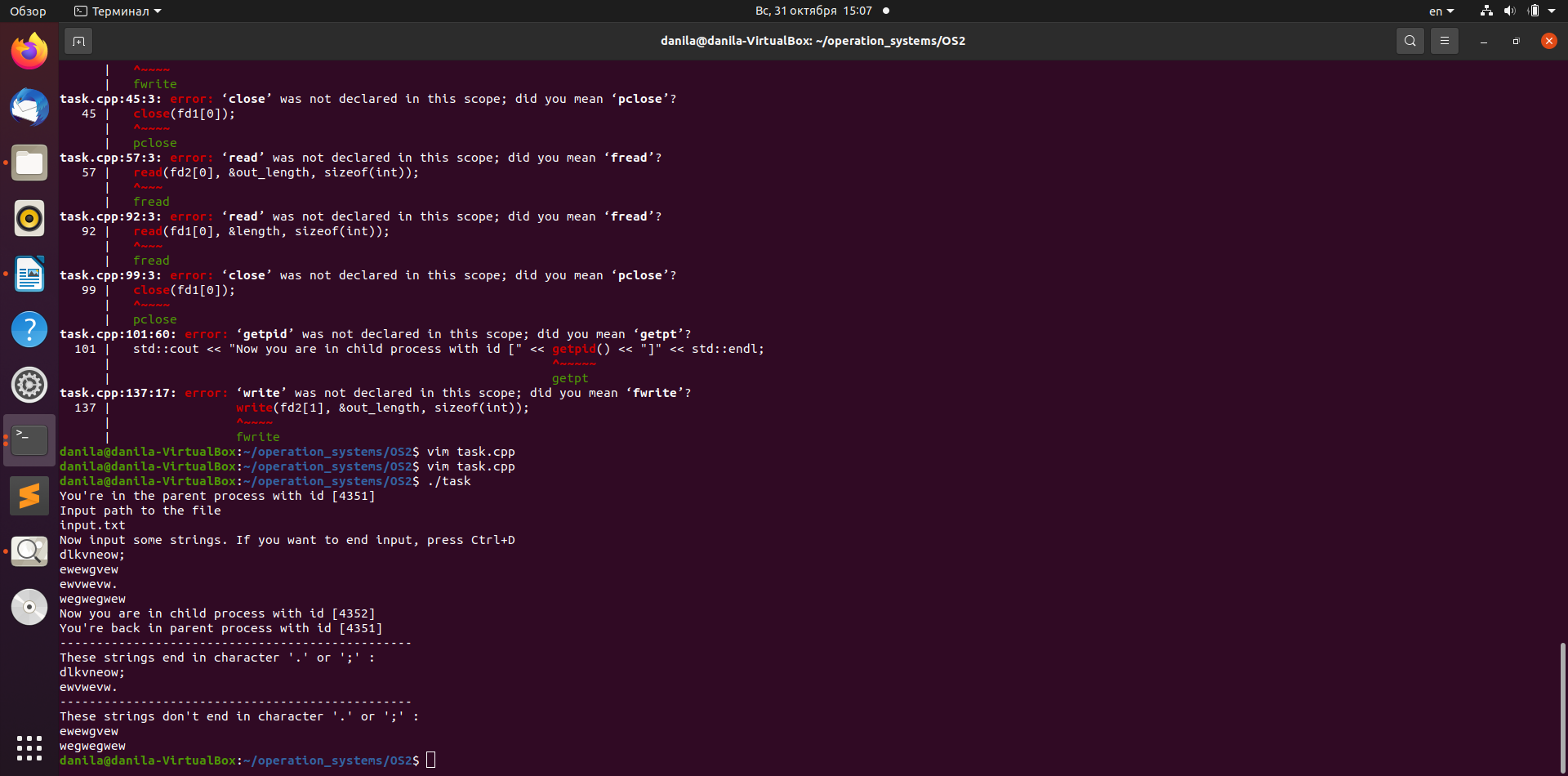
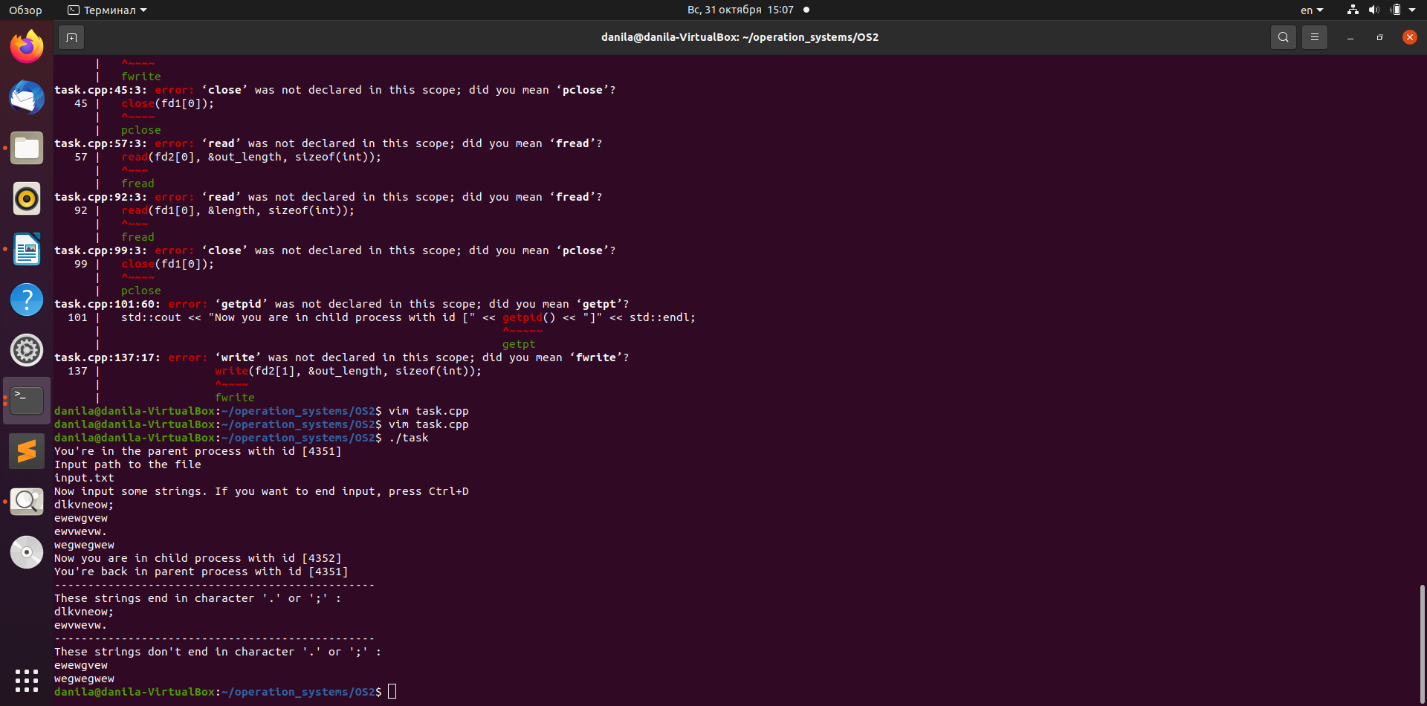




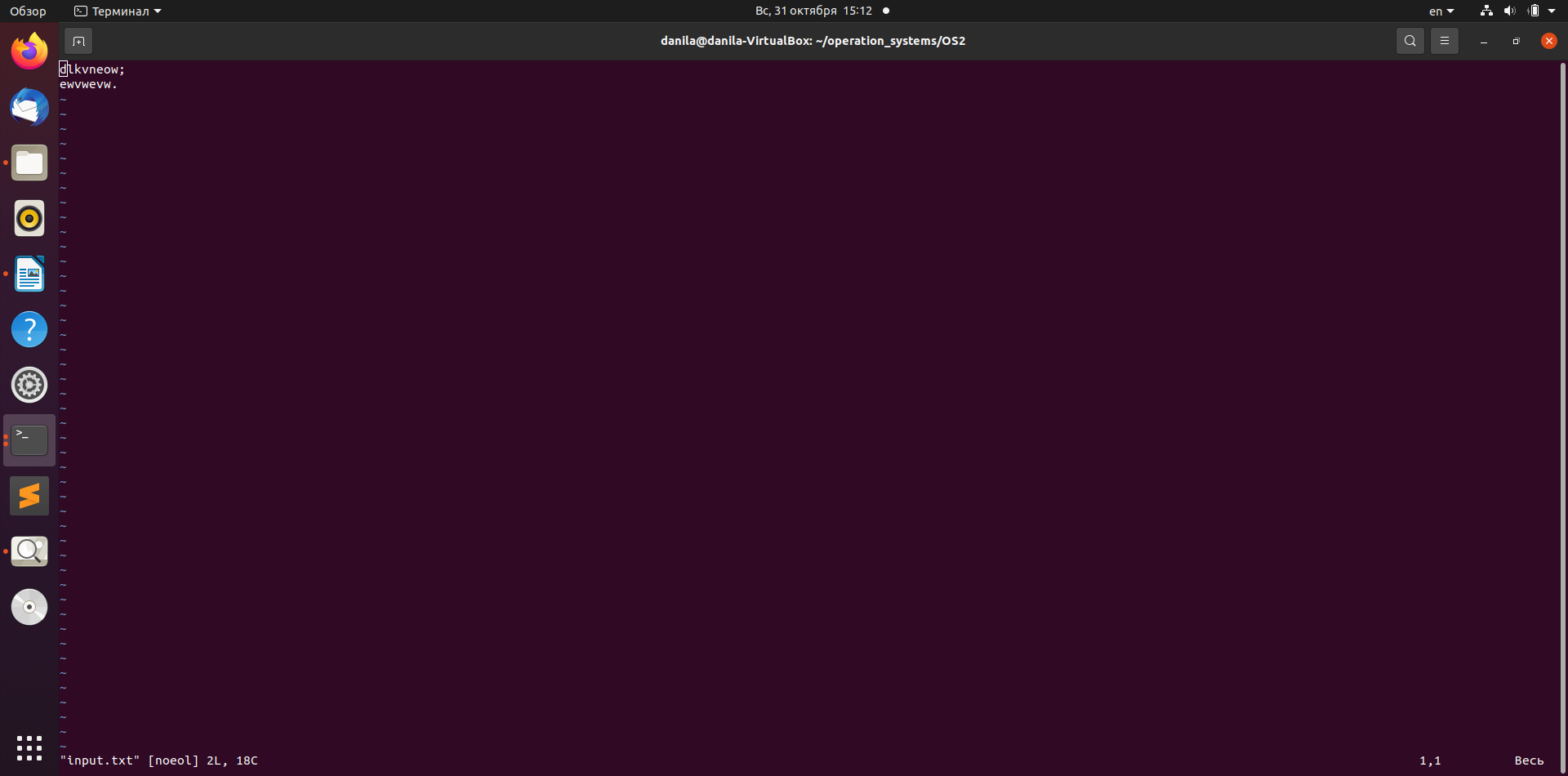




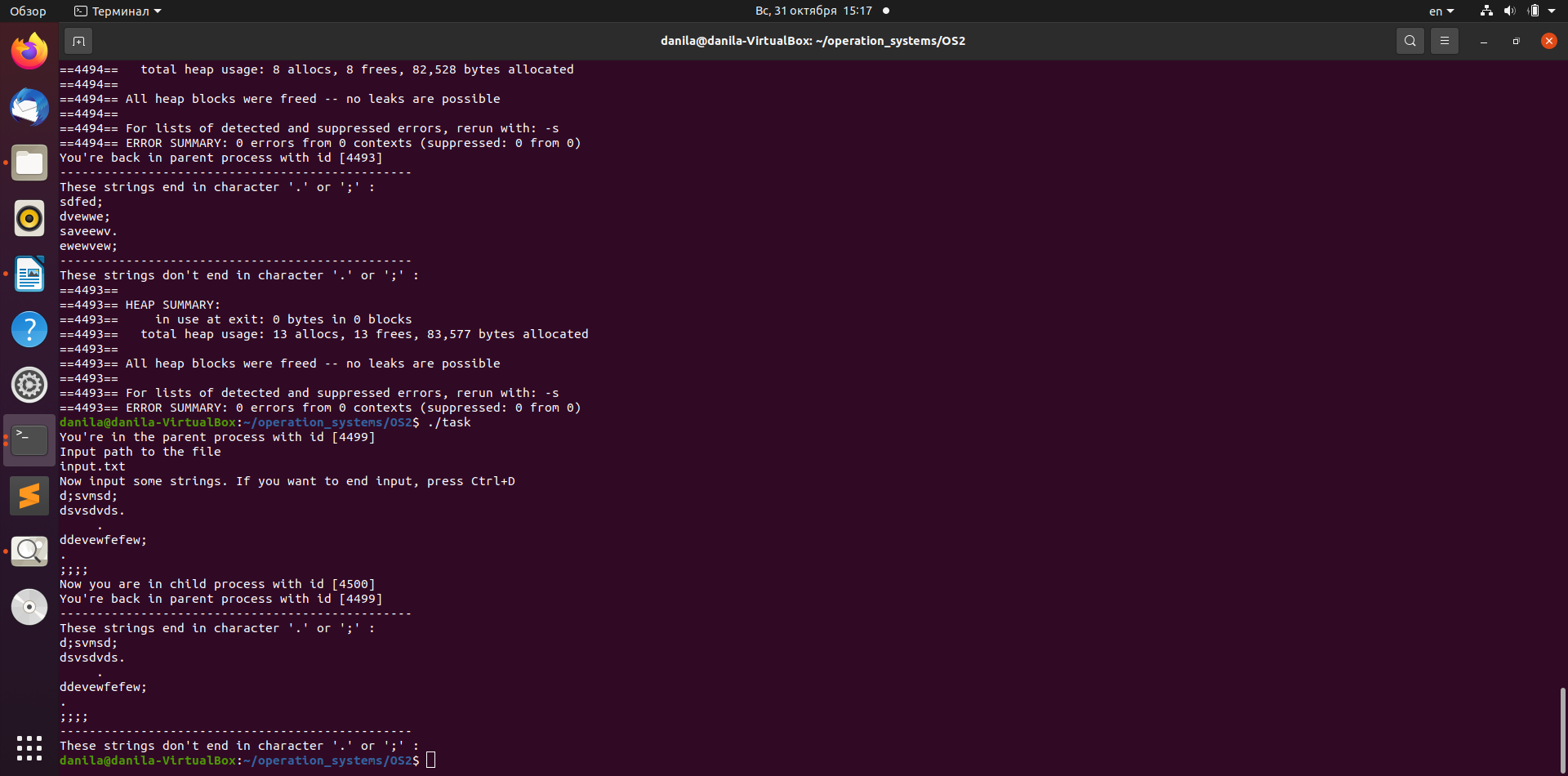
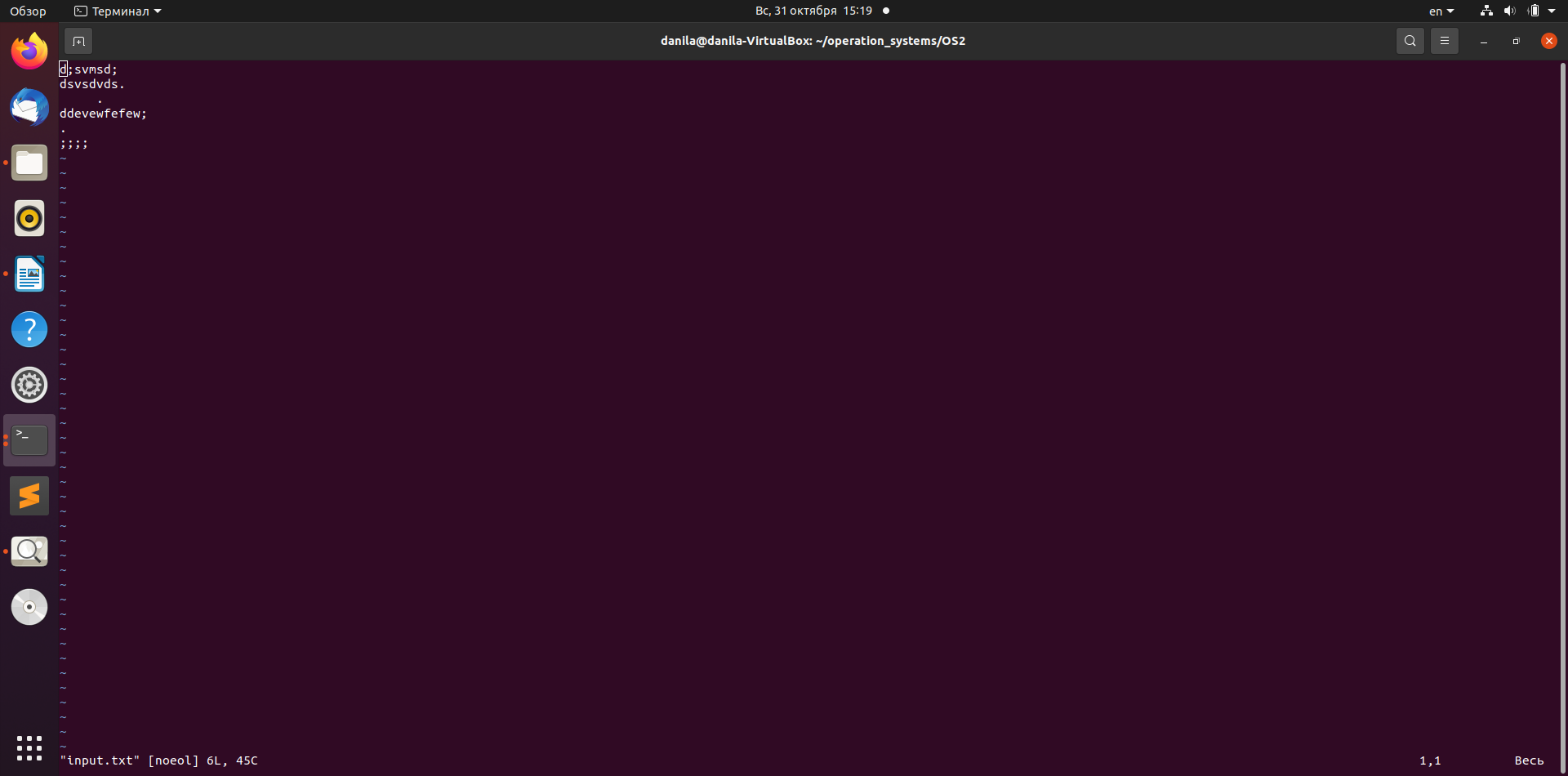
**Демонстрация работы программы**Тест 1.Вводится корректное имя файла, 2 строки оканчиваются на точку или точку с запятой, 2 нет.



Содержание файла input.txt

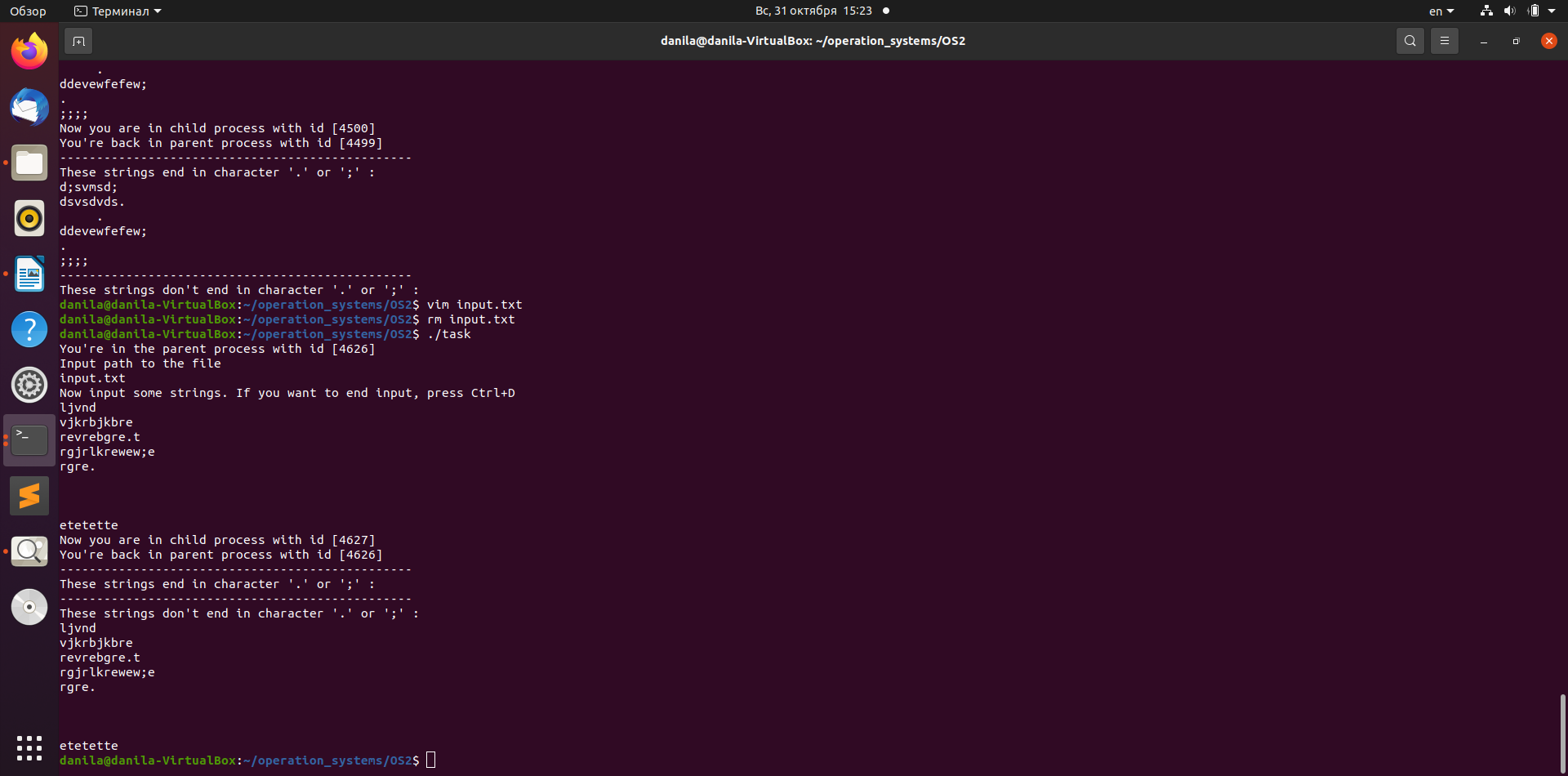
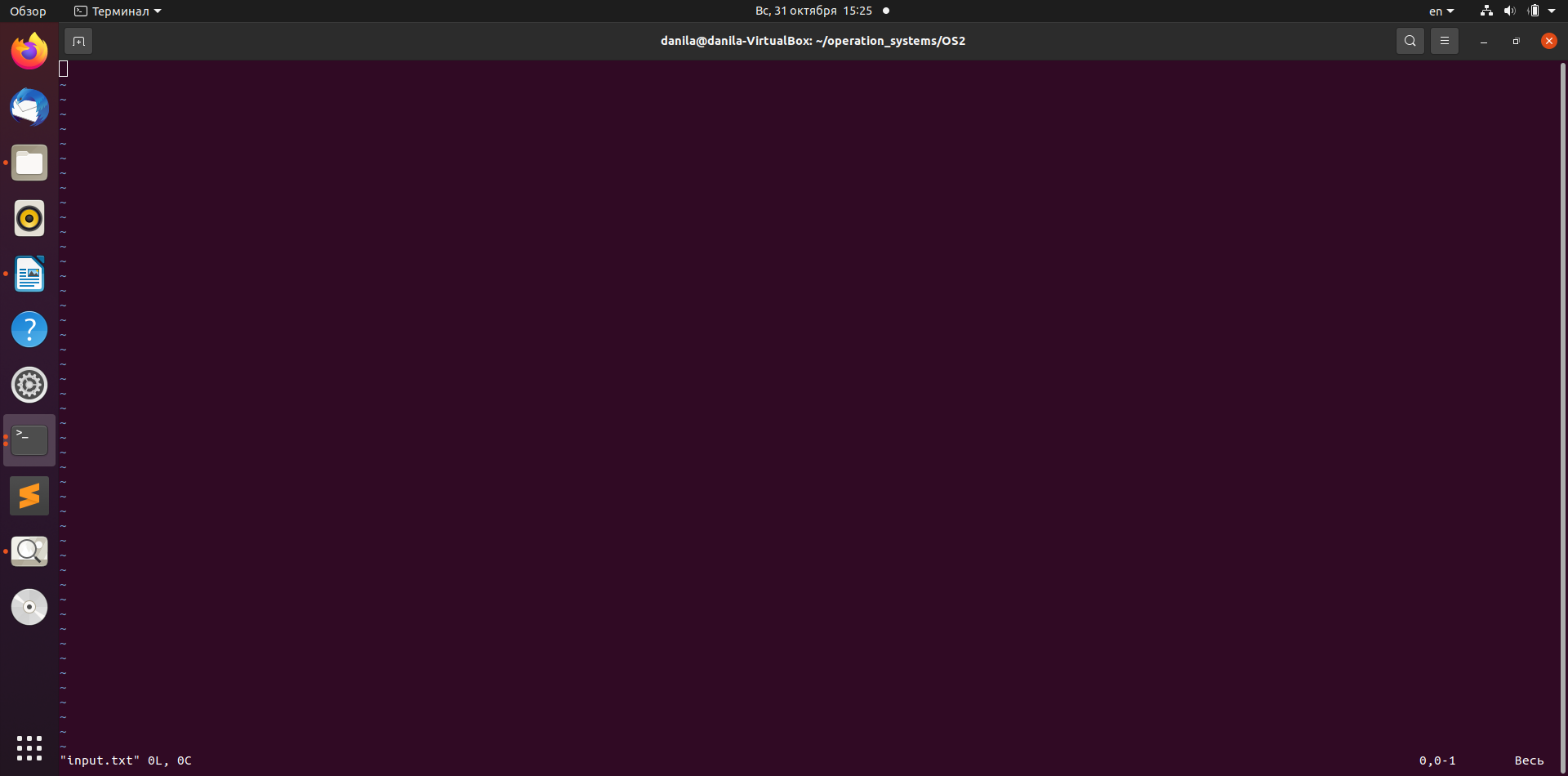


Тест 2.  
Вводится корректное имя файла, но все строки удовлетворяют правилу.

  
Содержание файла input.txt

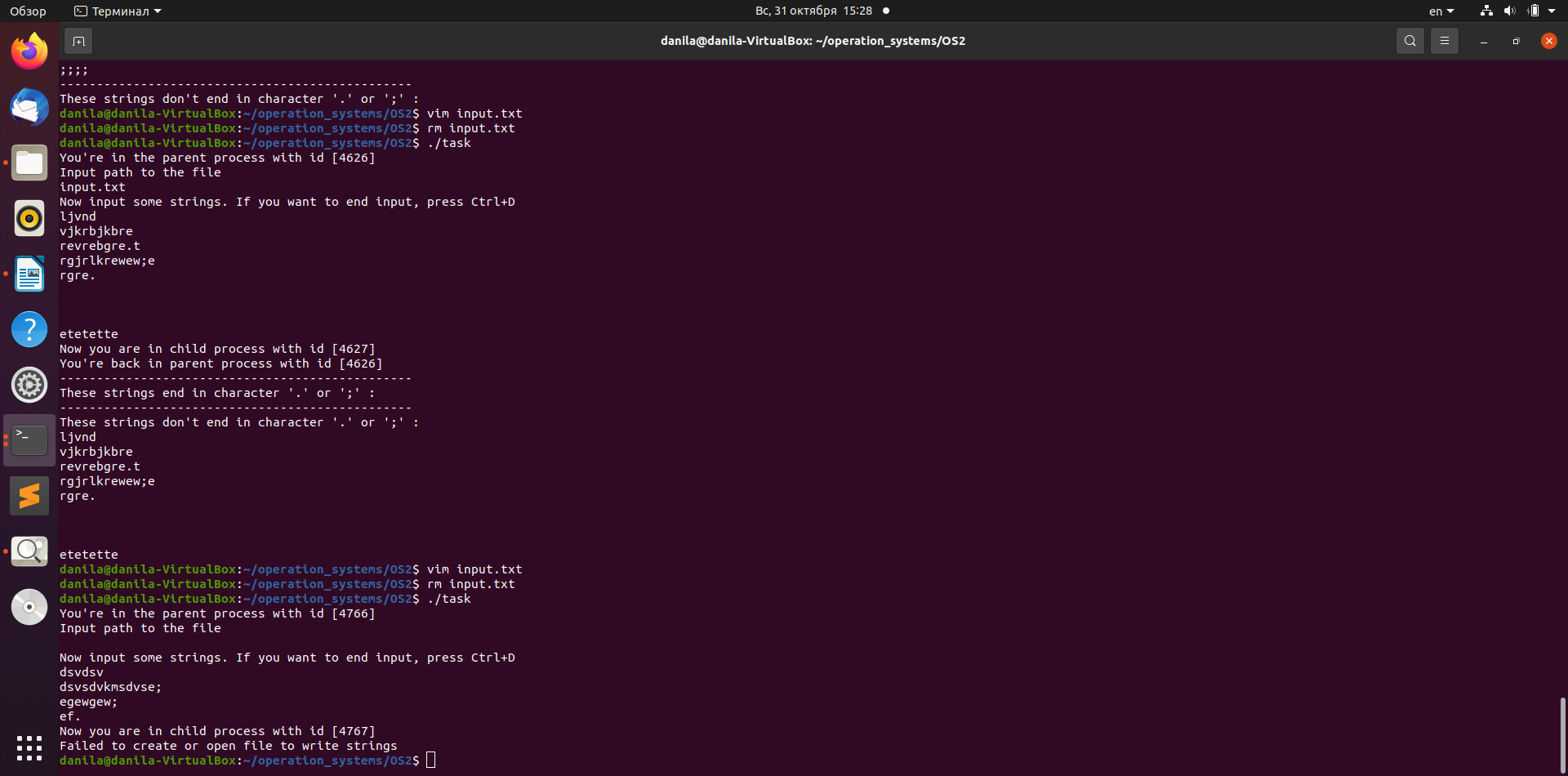
Тест 3

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.