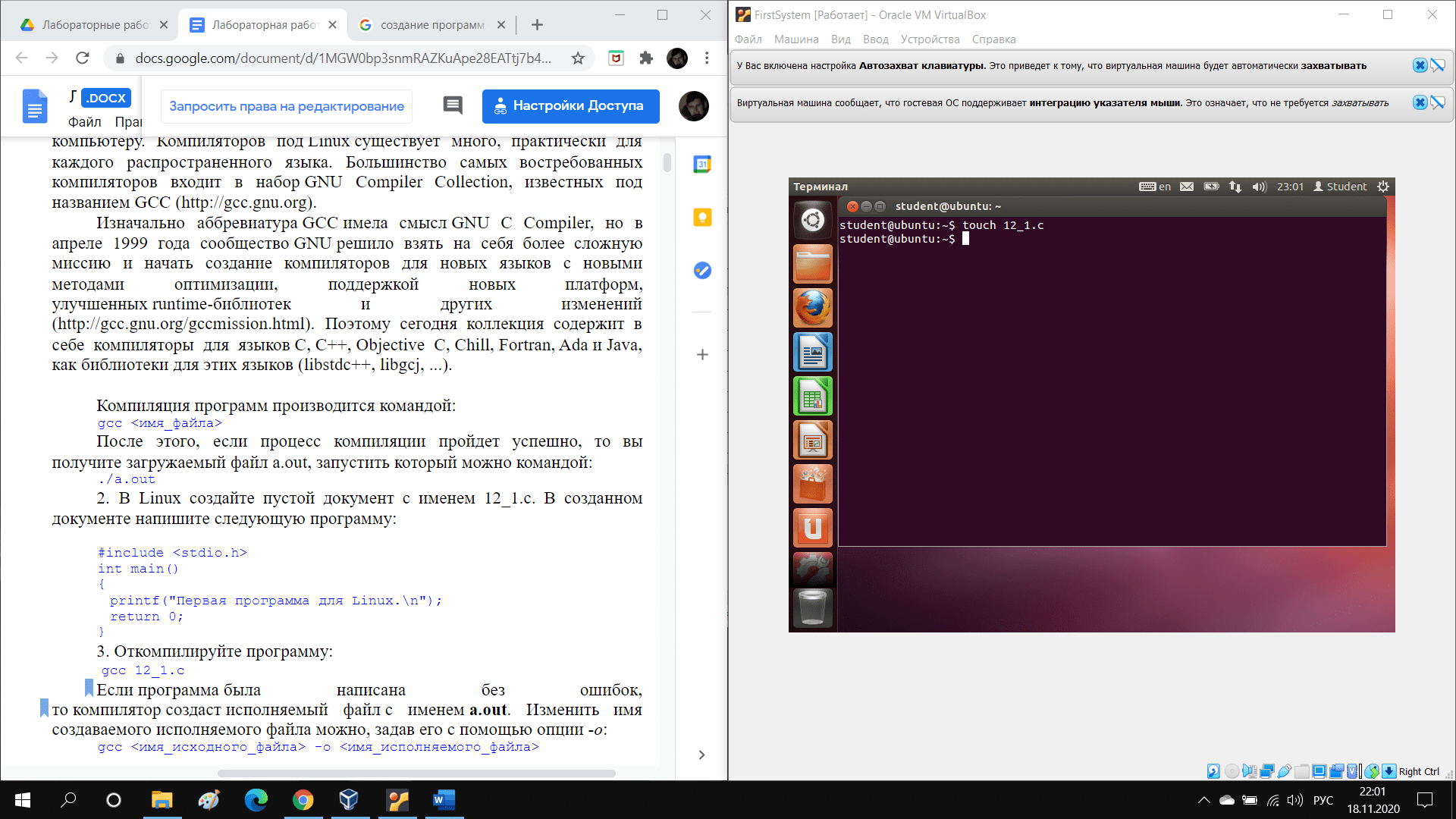
Id()Отчет по лабораторной работе №9

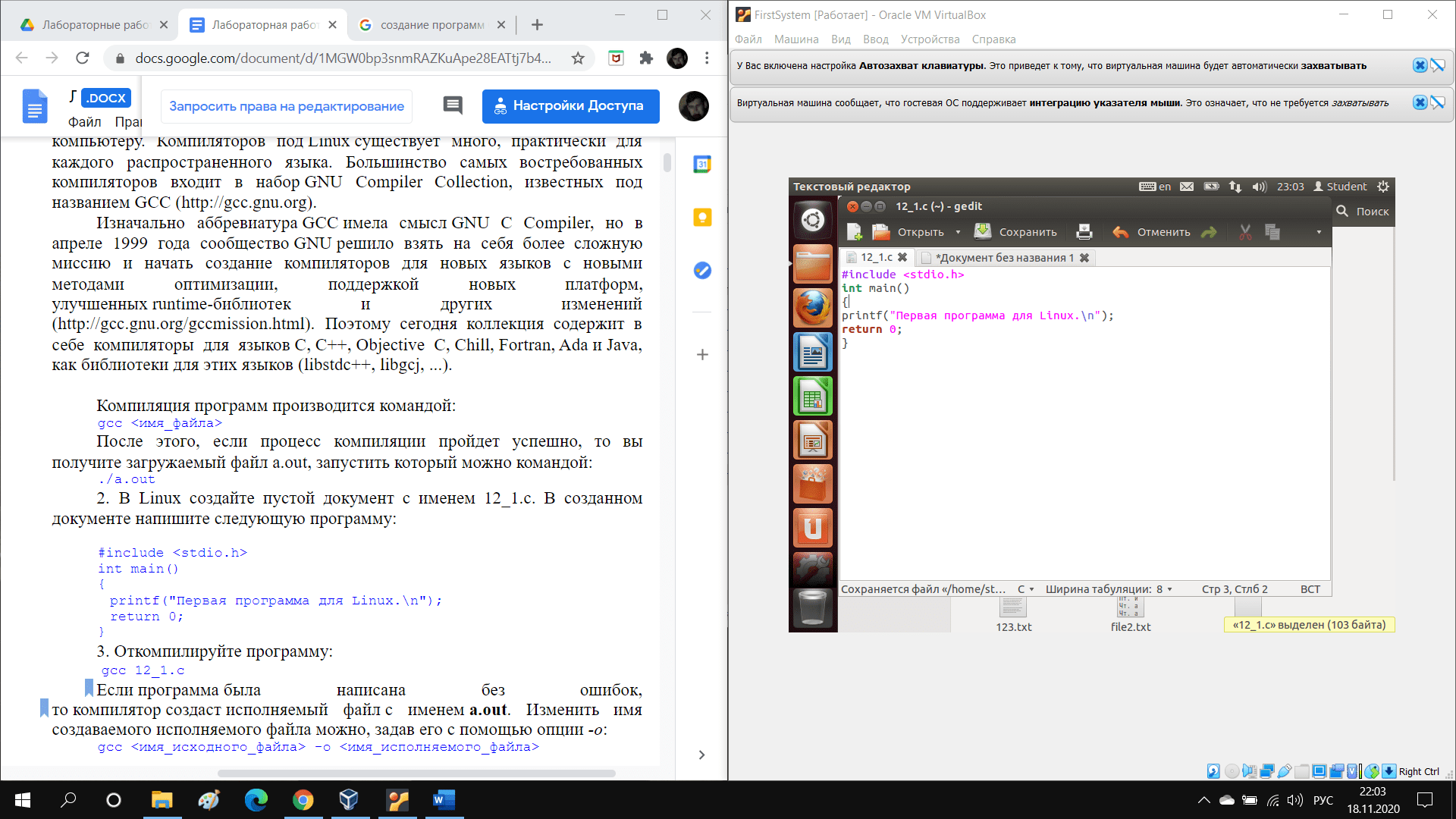
Выполнил: Новиков Даниил

Группа: ФИб-3301

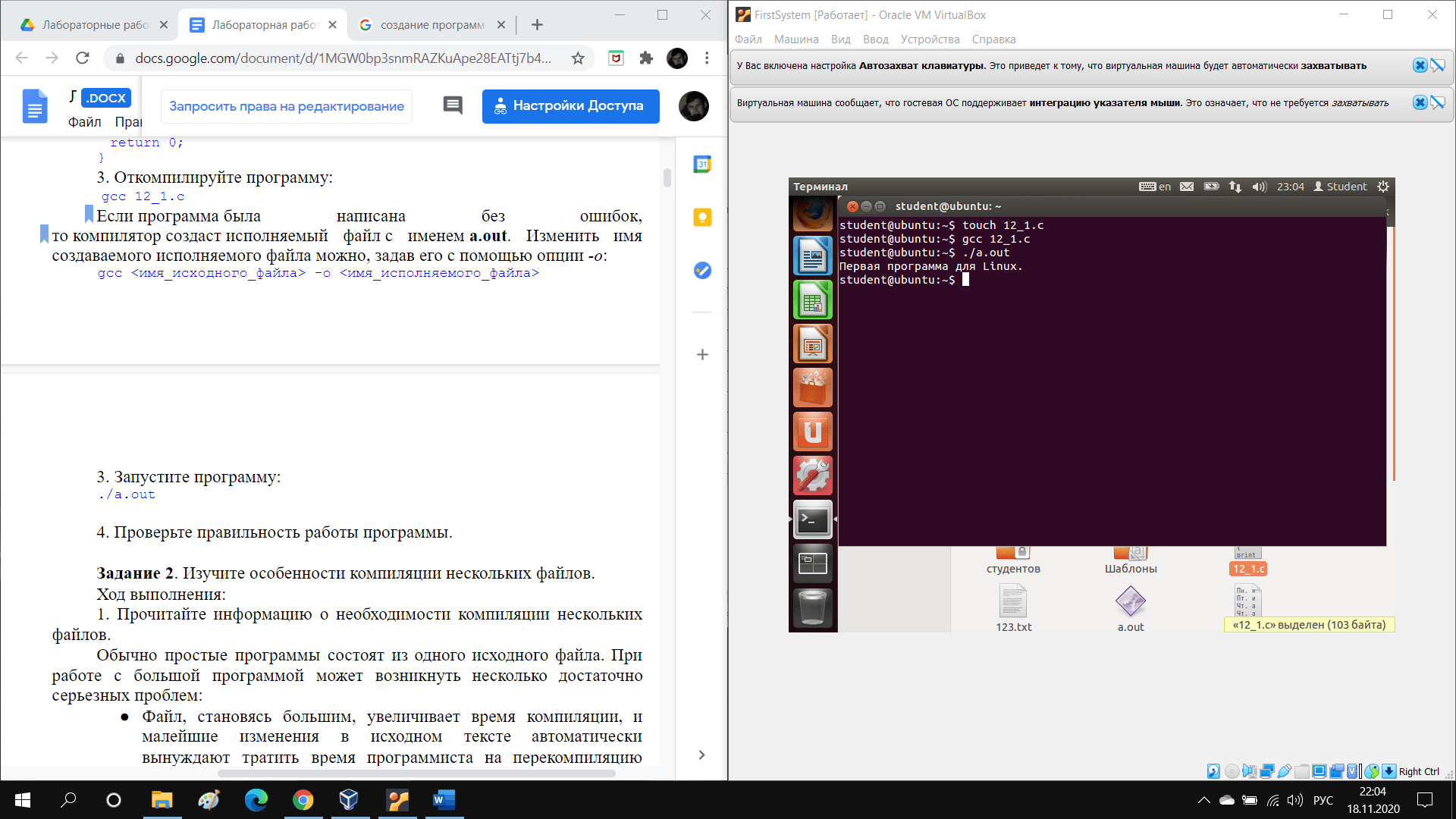
**Задание 1.**

2.



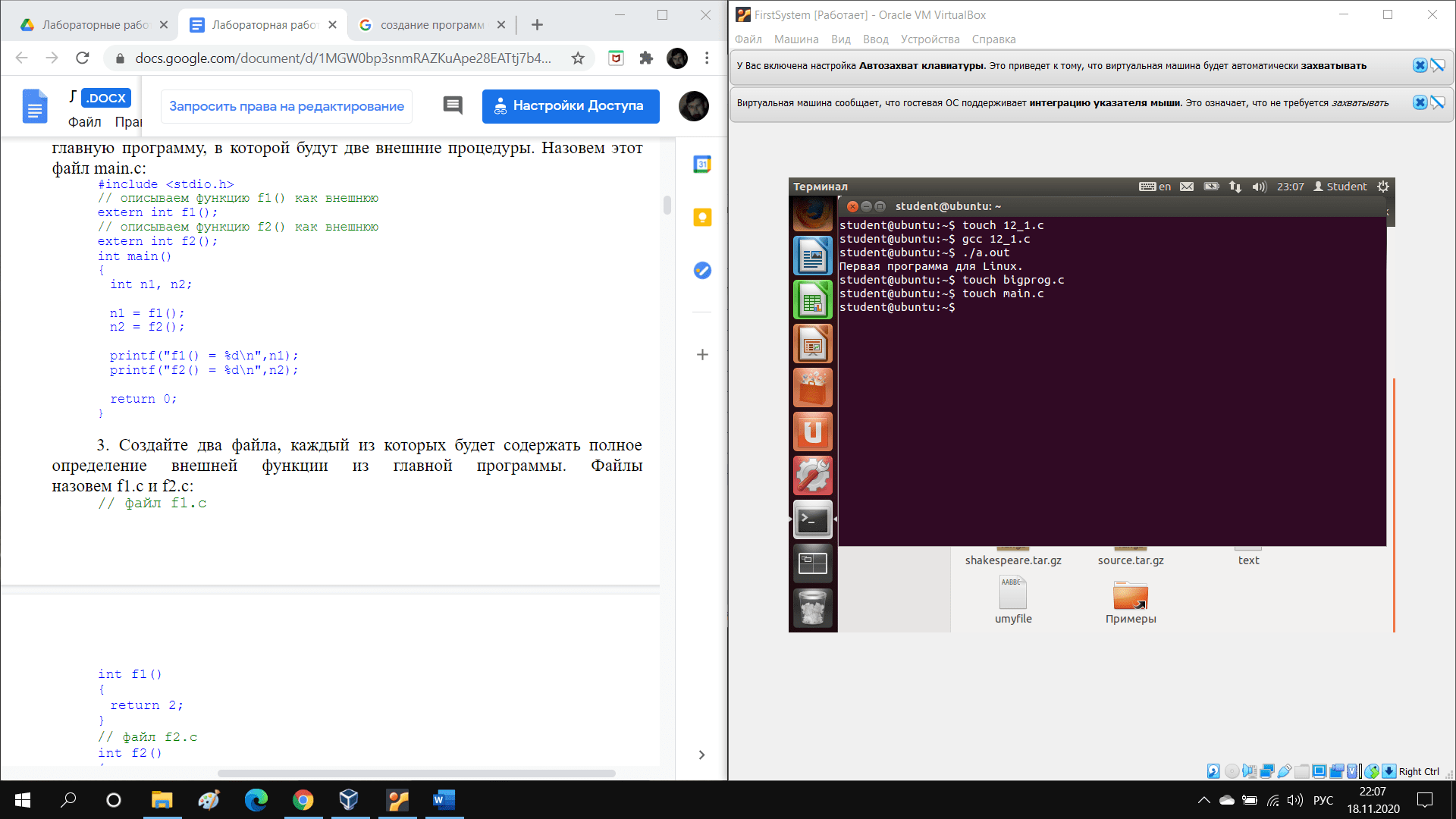


3.



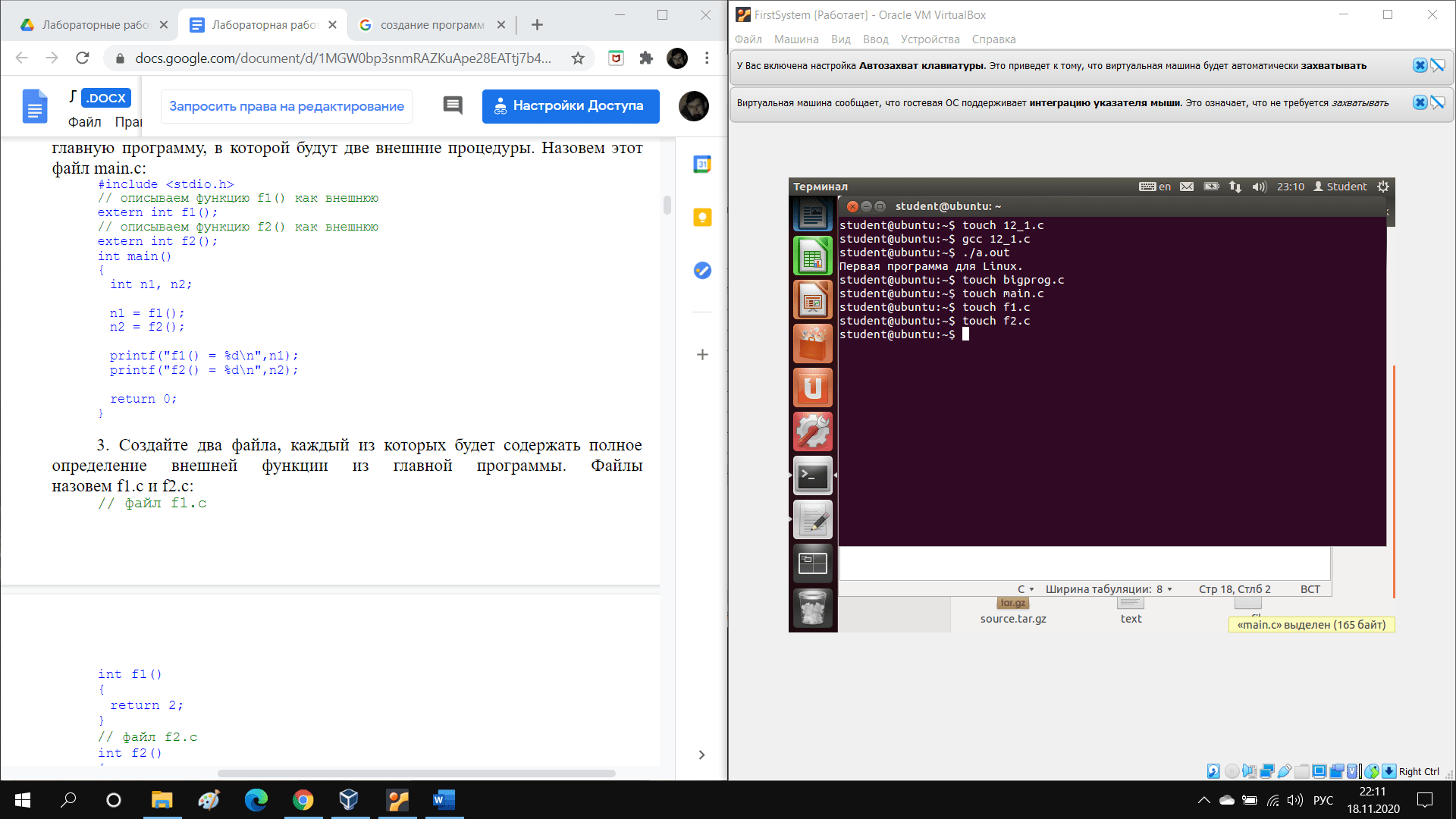
**Задание 2.**

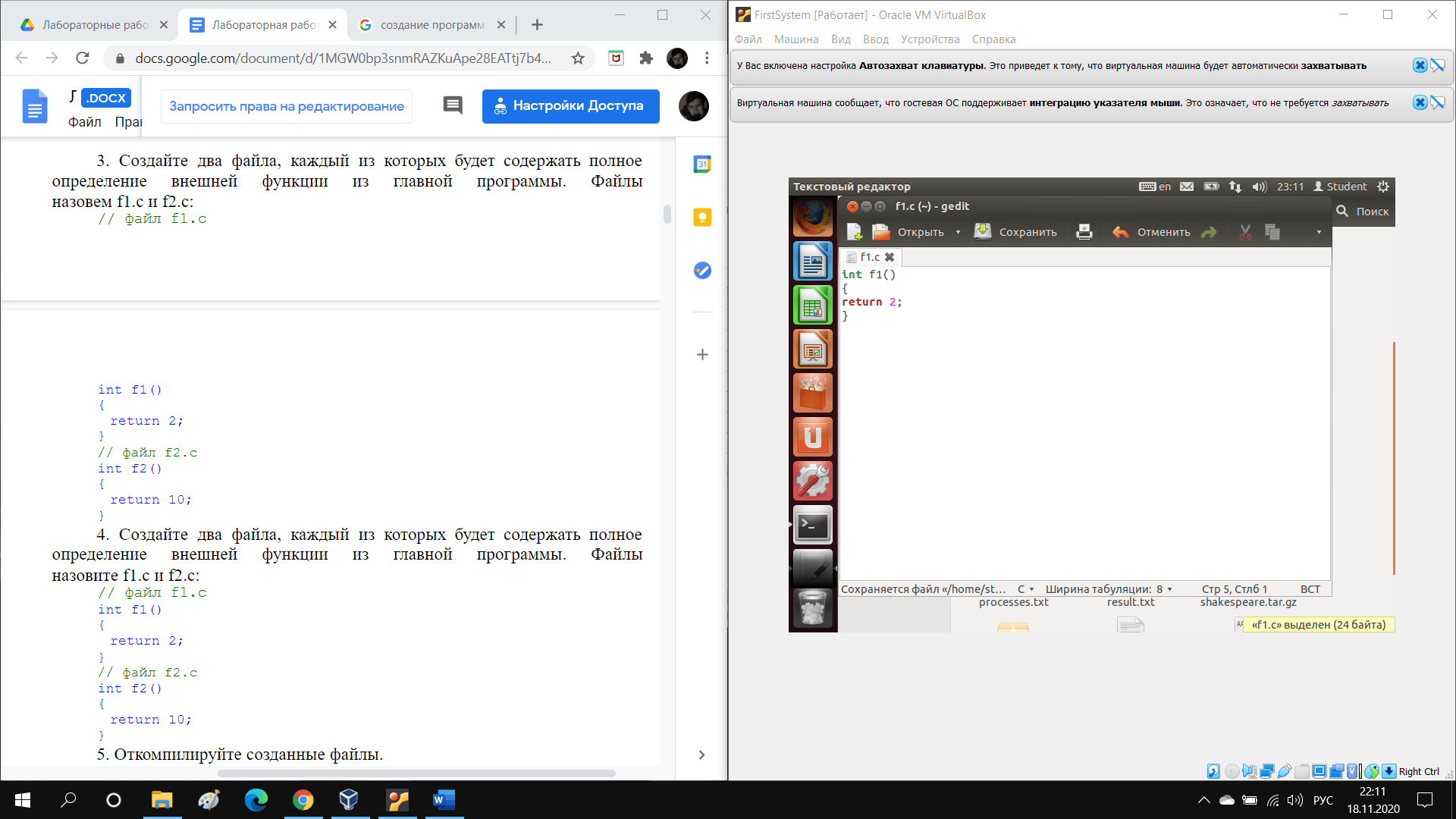
2. Создал файл главной программы:

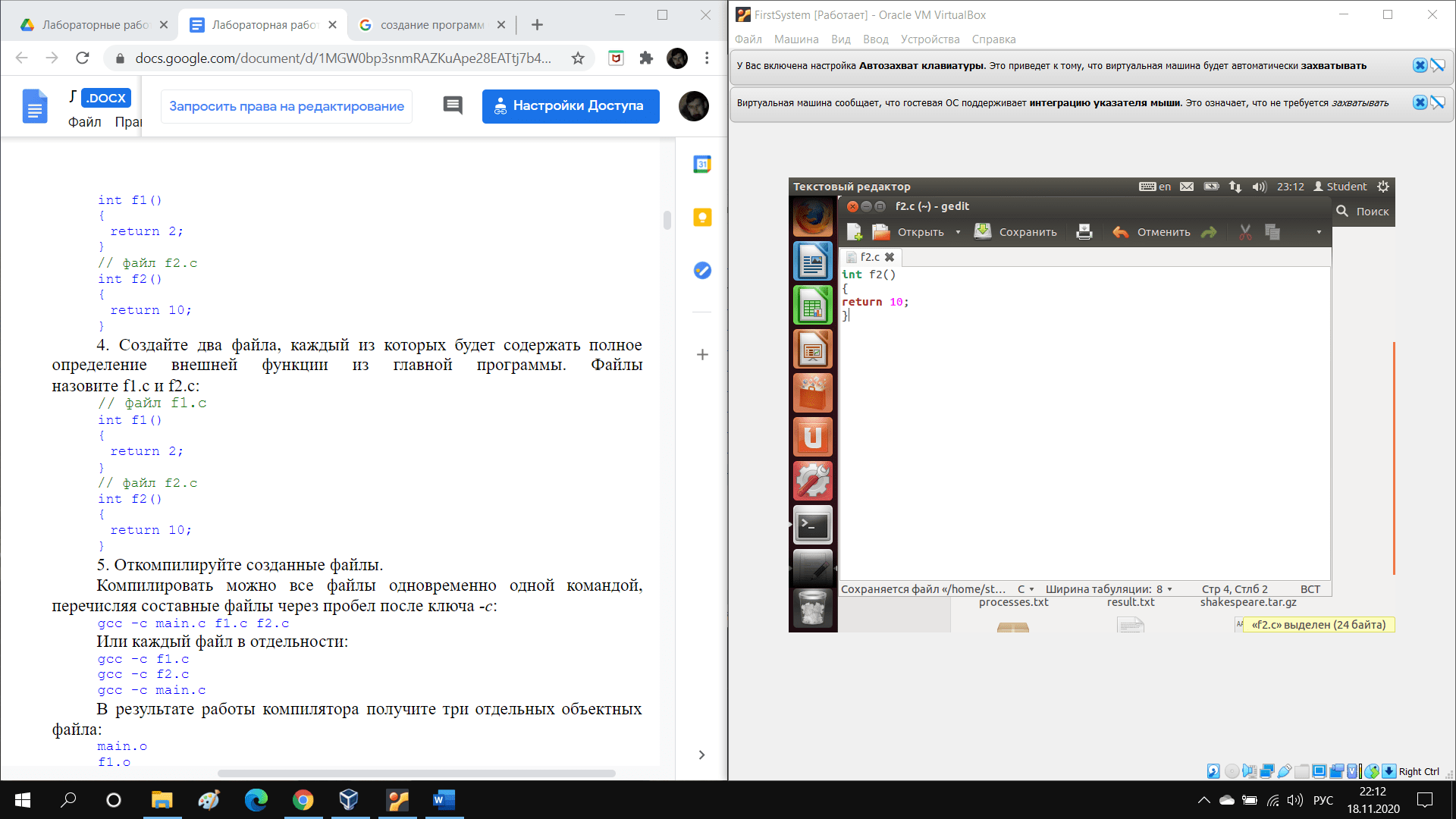




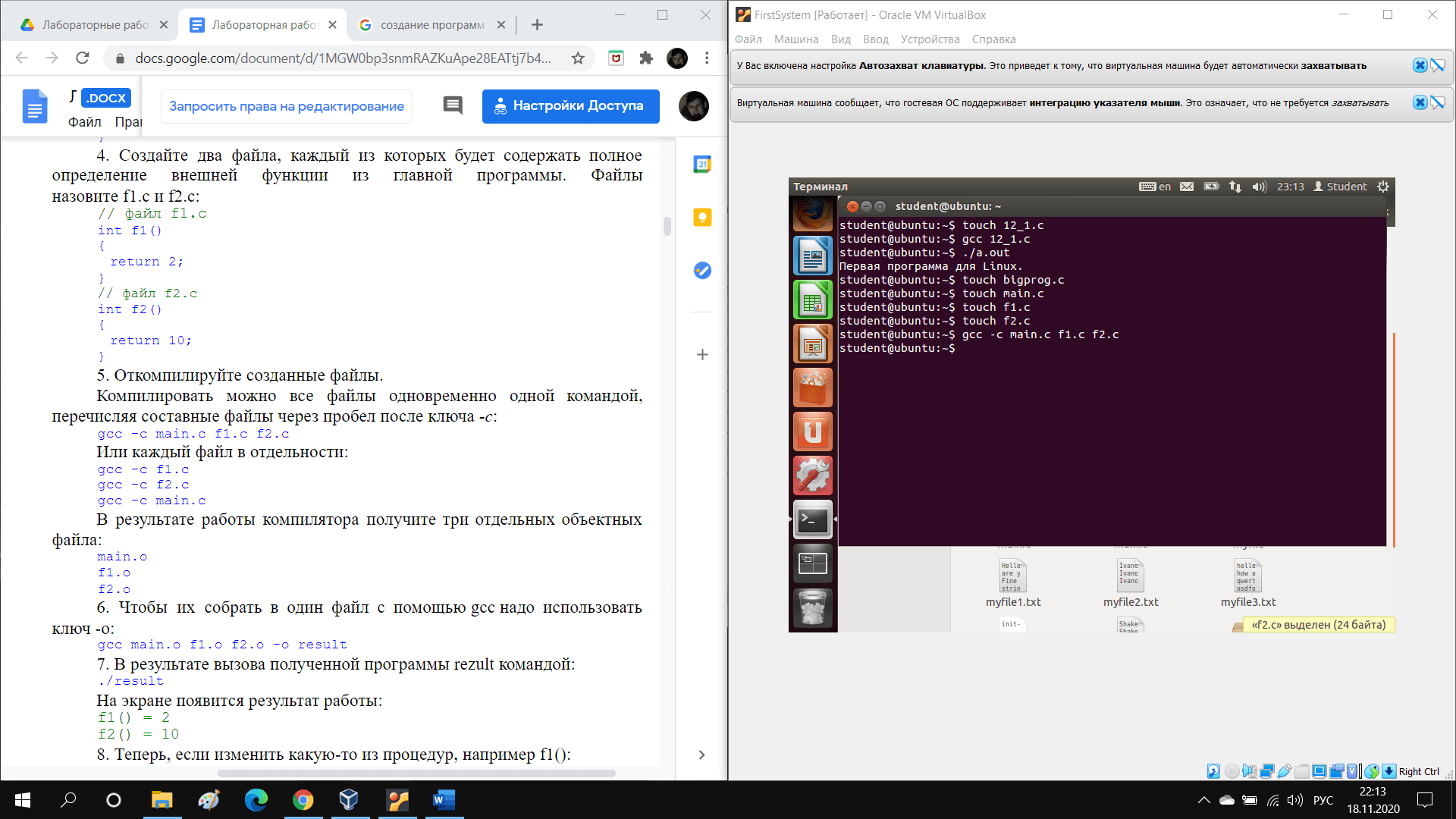
3. Создадим 2 файла, в которых будет определение функций:



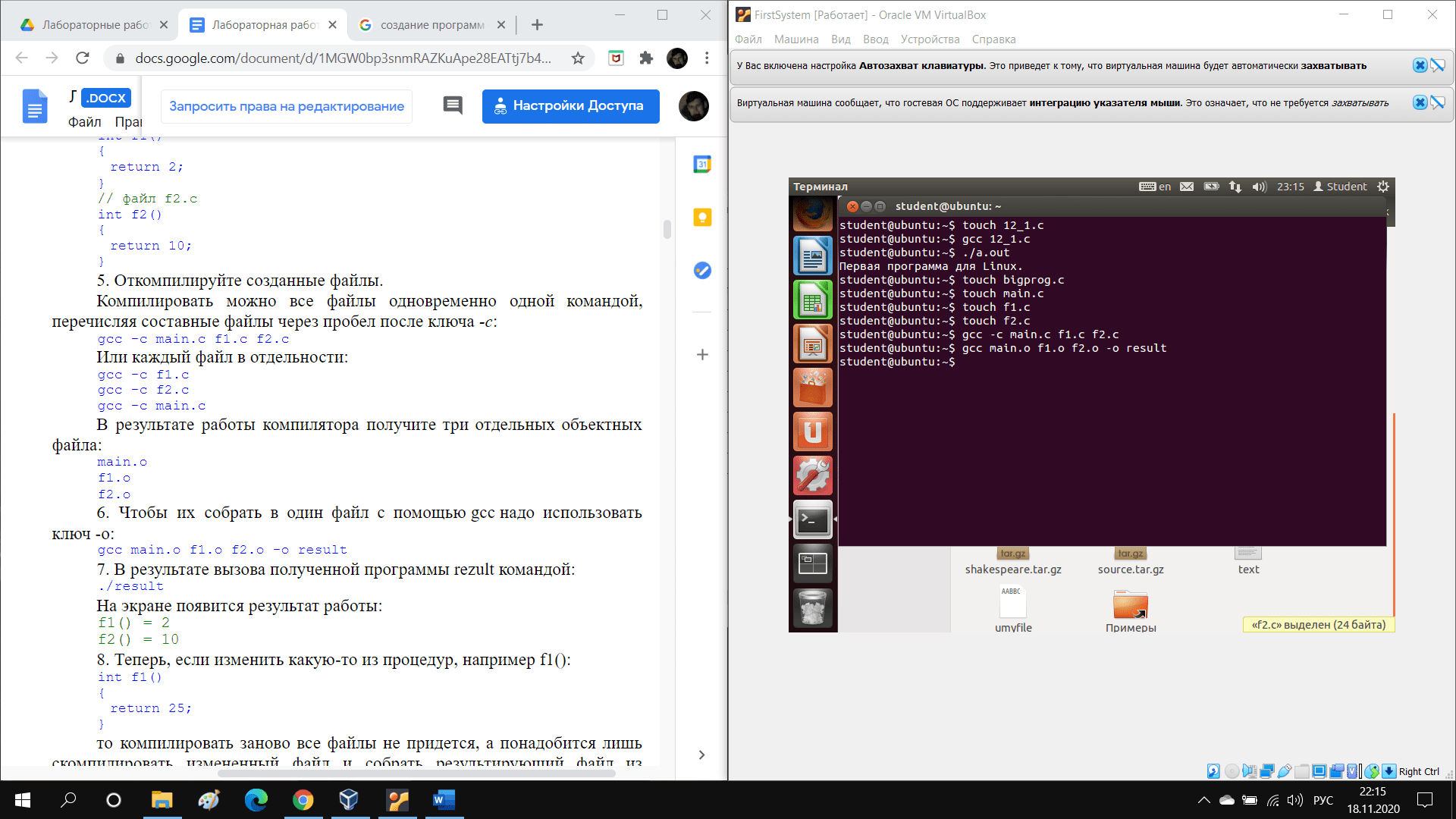




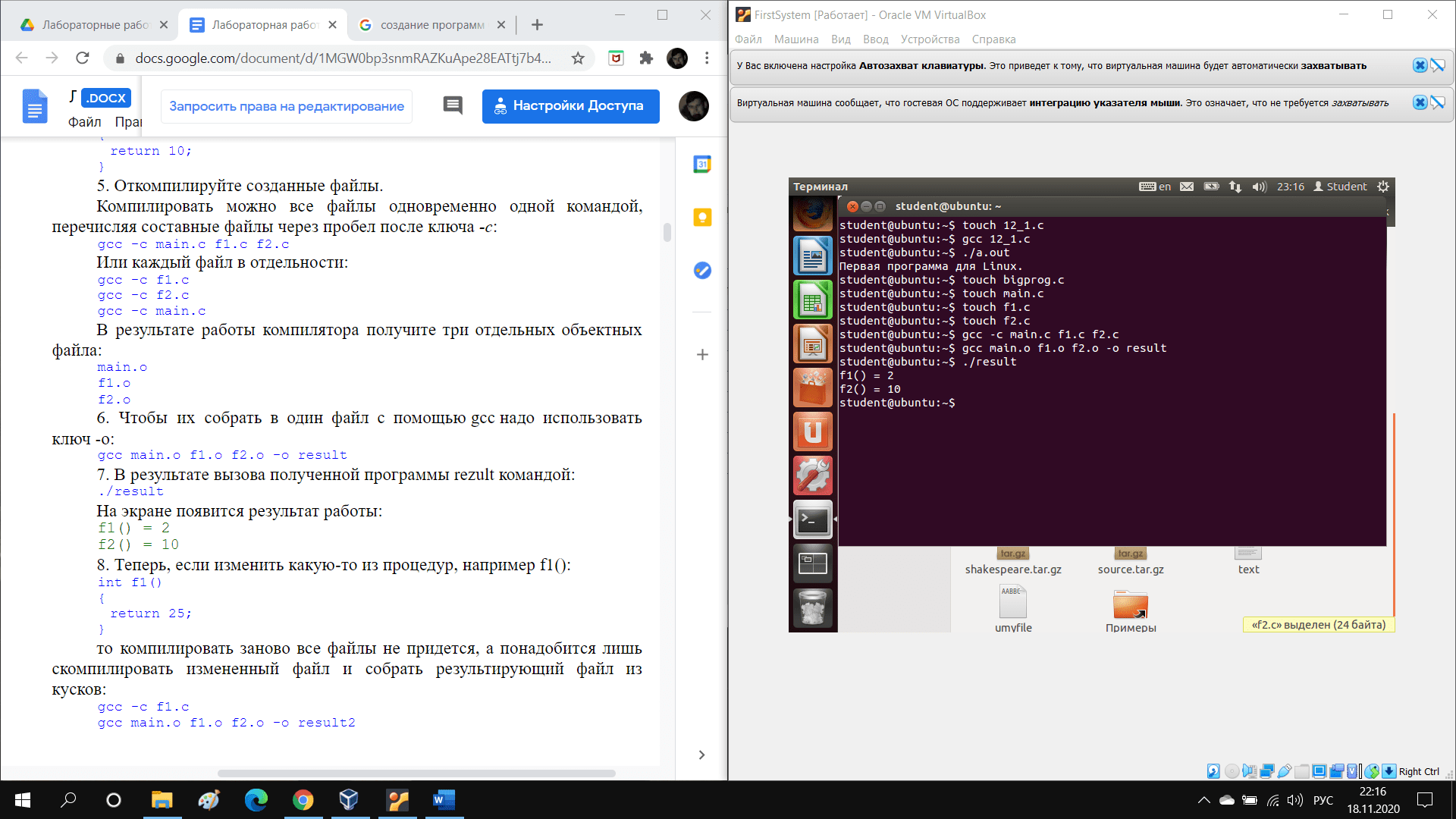
4. Откомпилируем созданные файлы:



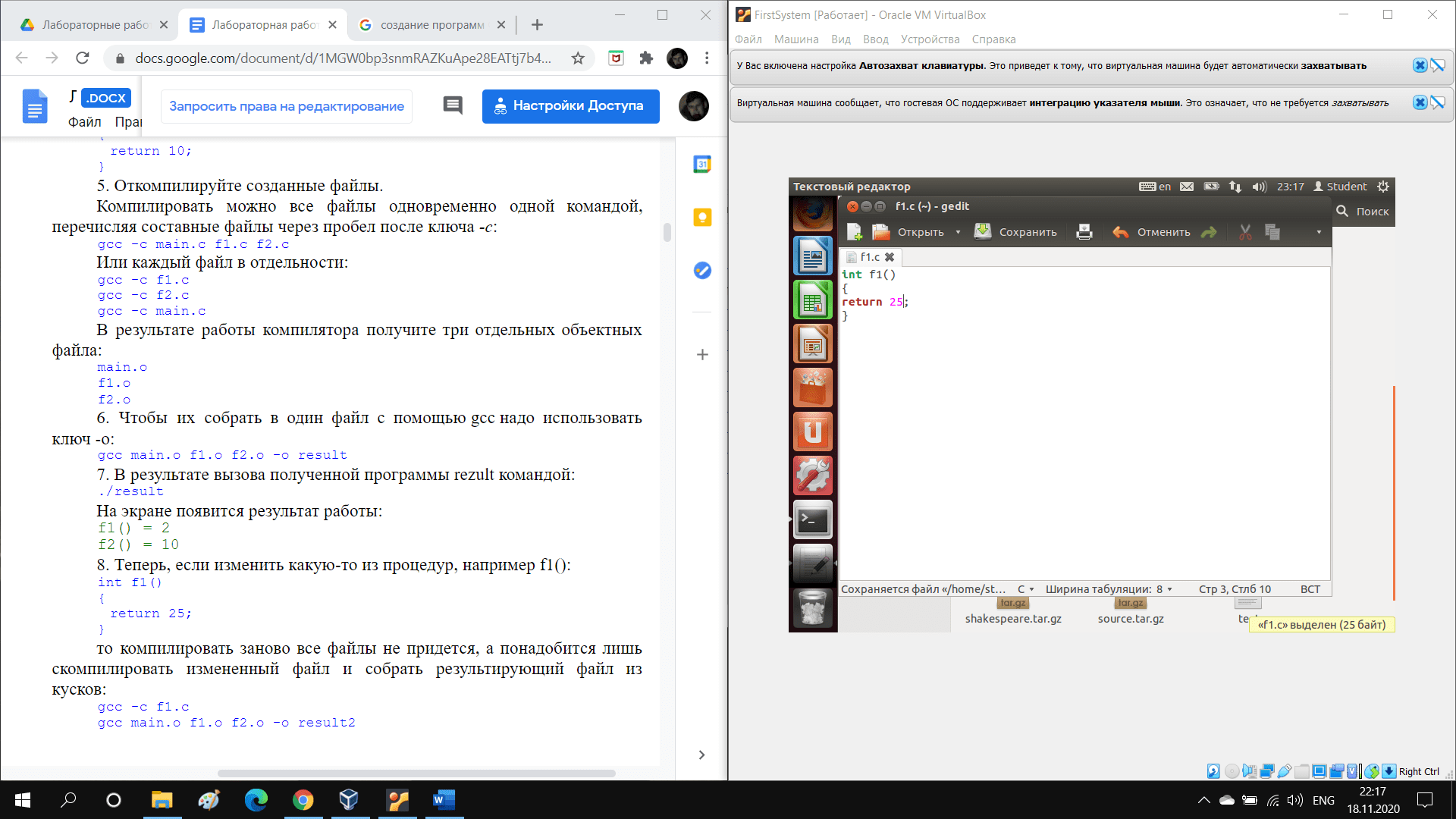
5. Соберем в один файл:



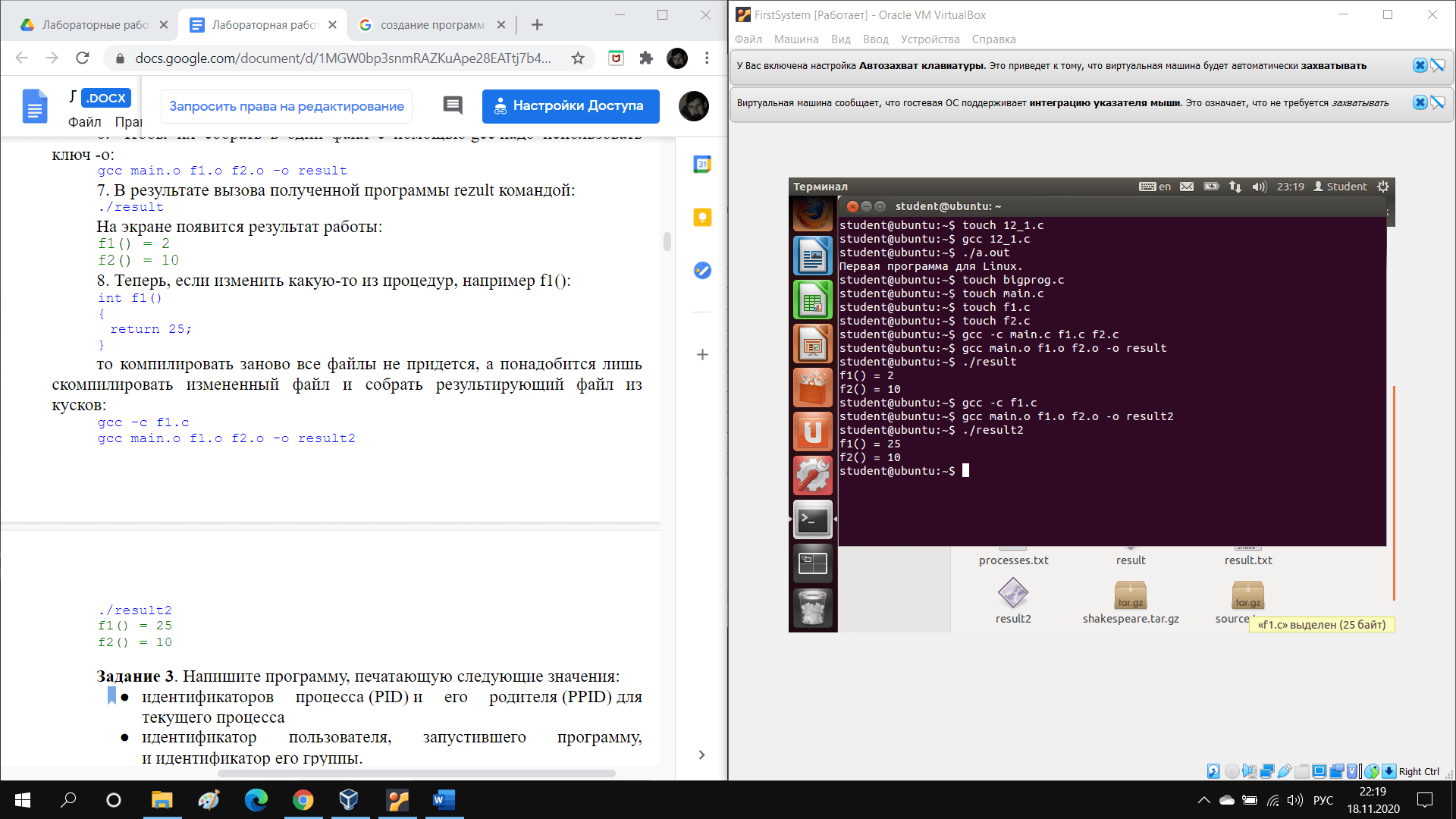
6. Запустим программу:



8. Изменим процедуру f1():

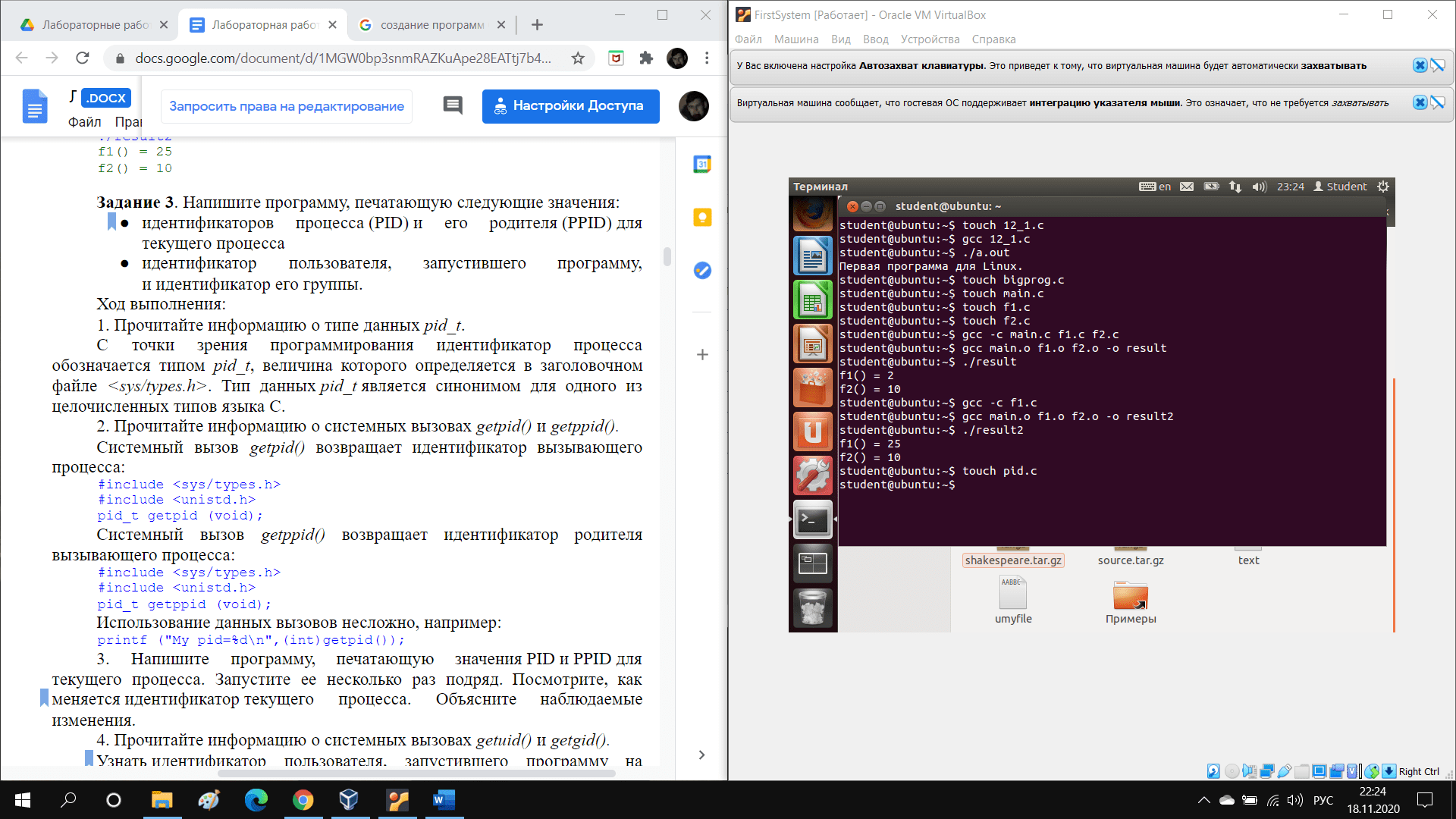


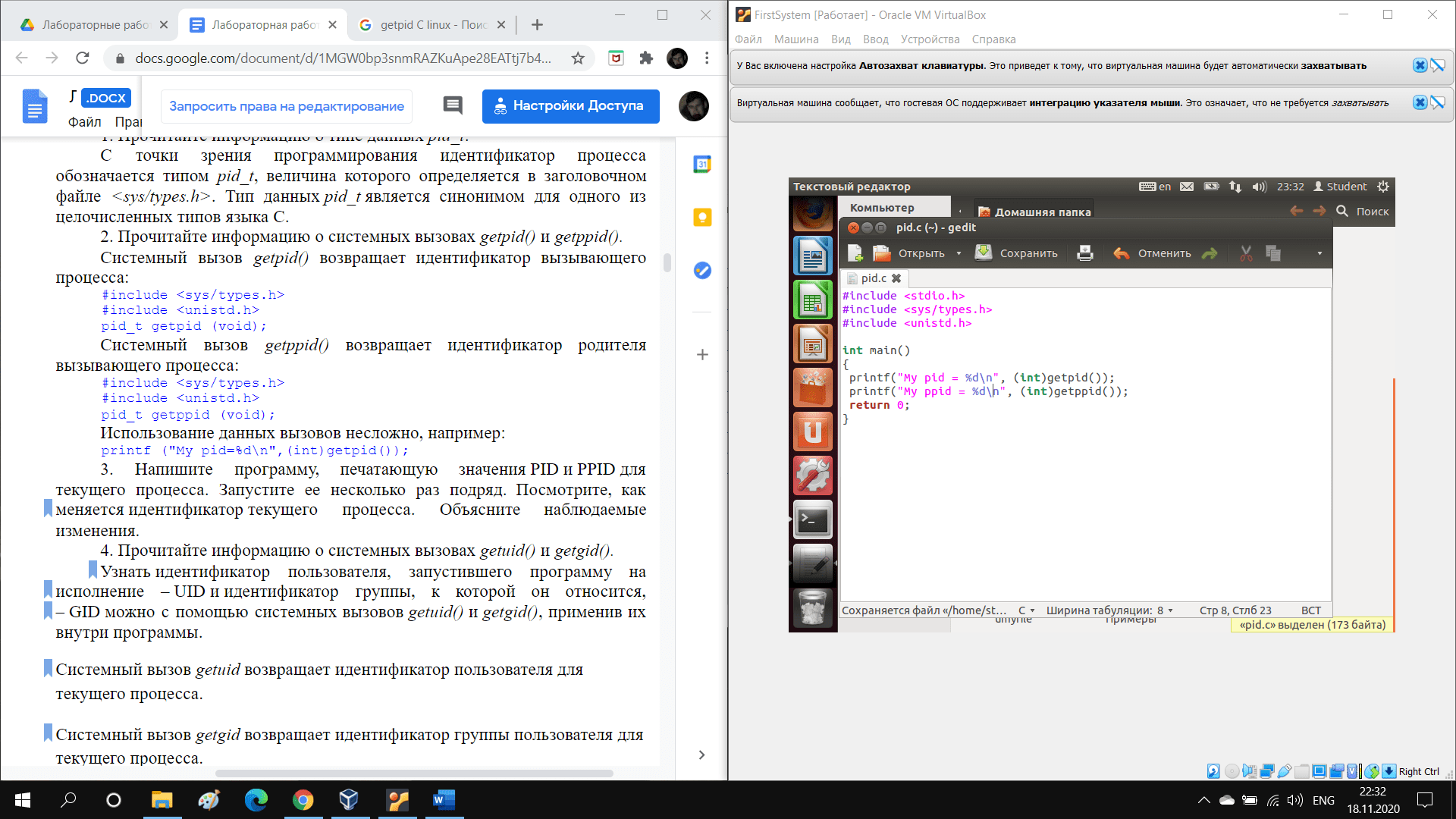
Откомпилируем файл процедуры и соберем результирующий файл:

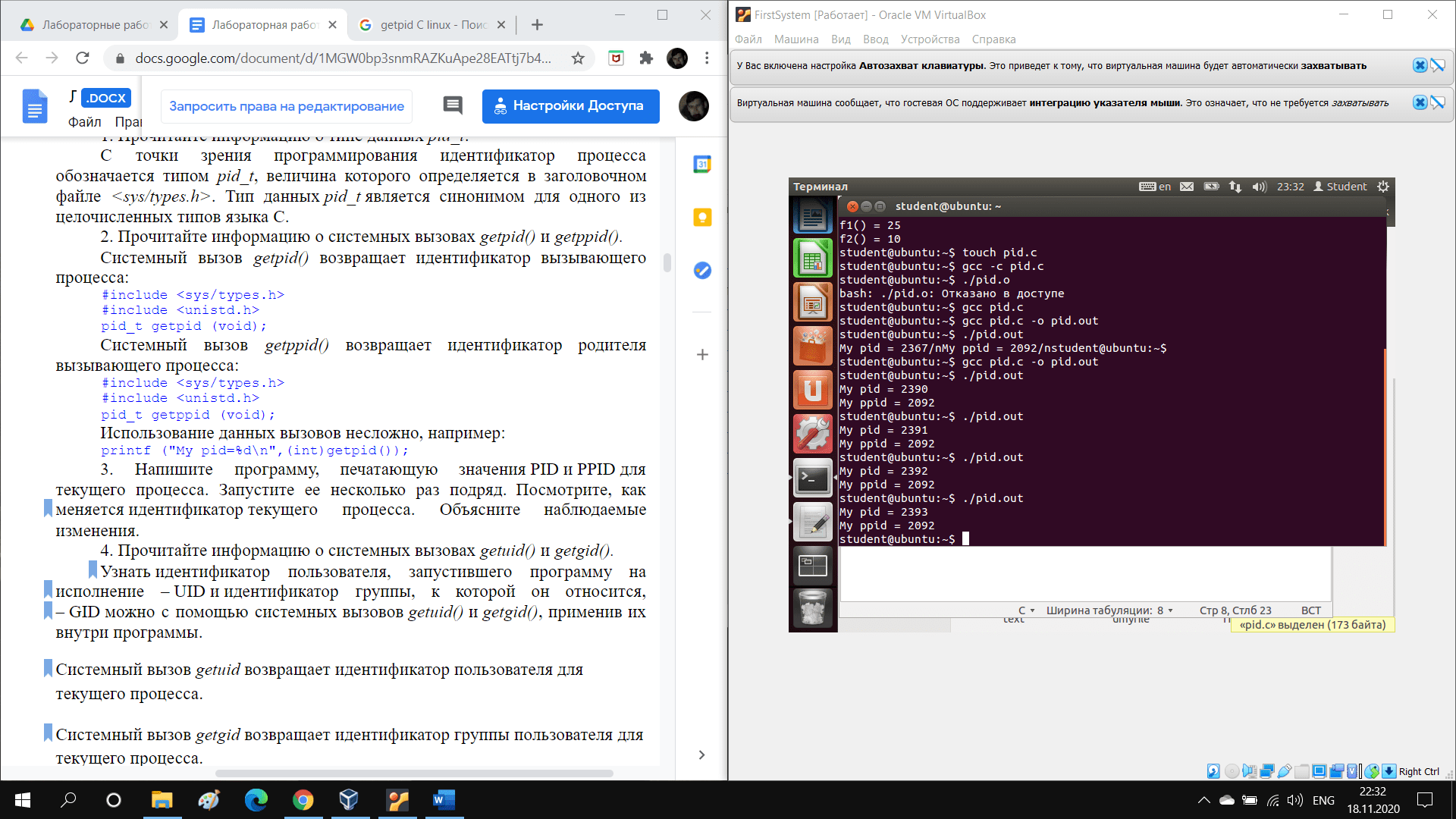


**Задание 3.**

3. Создадим файл программы pid.c:

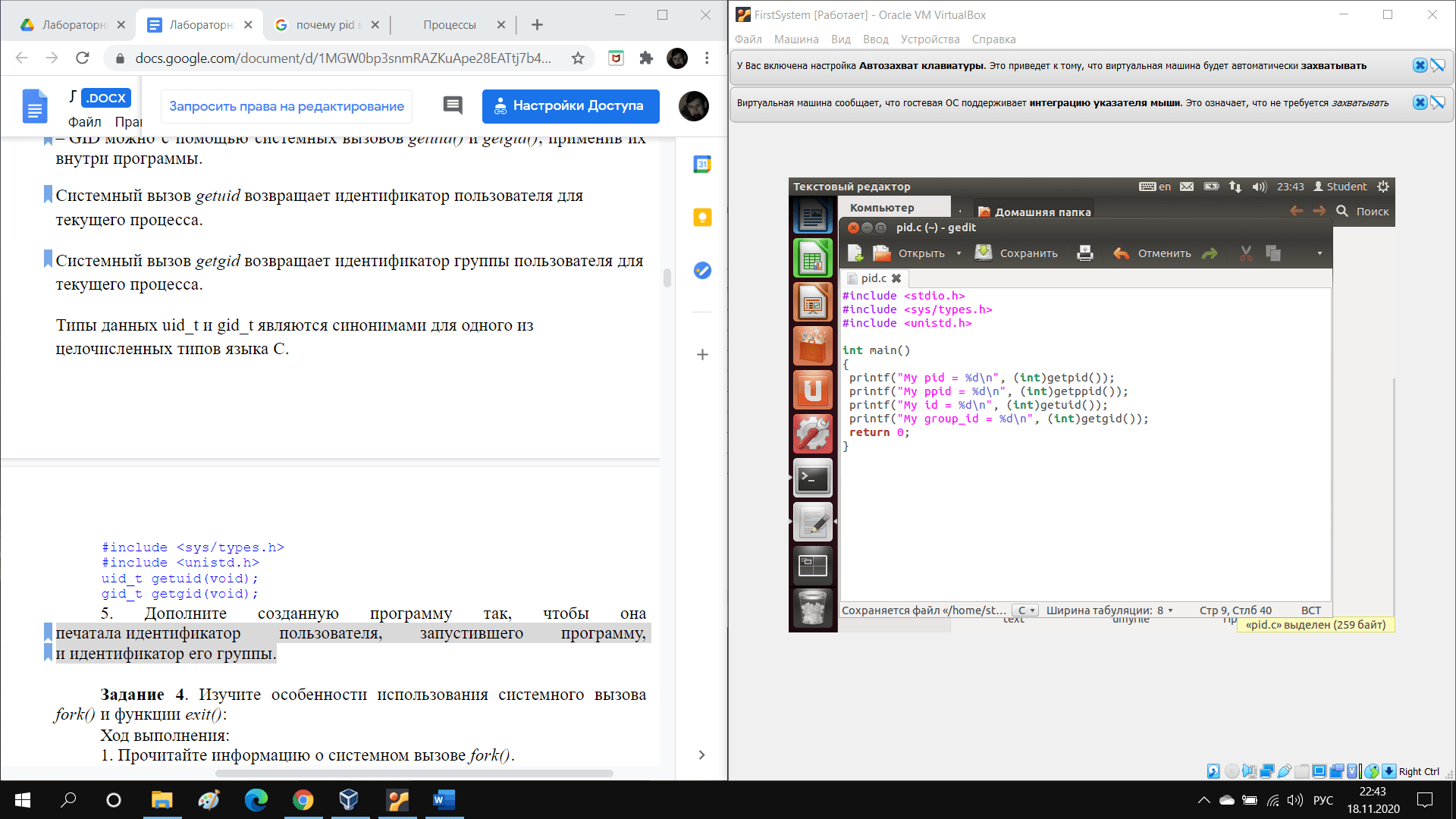


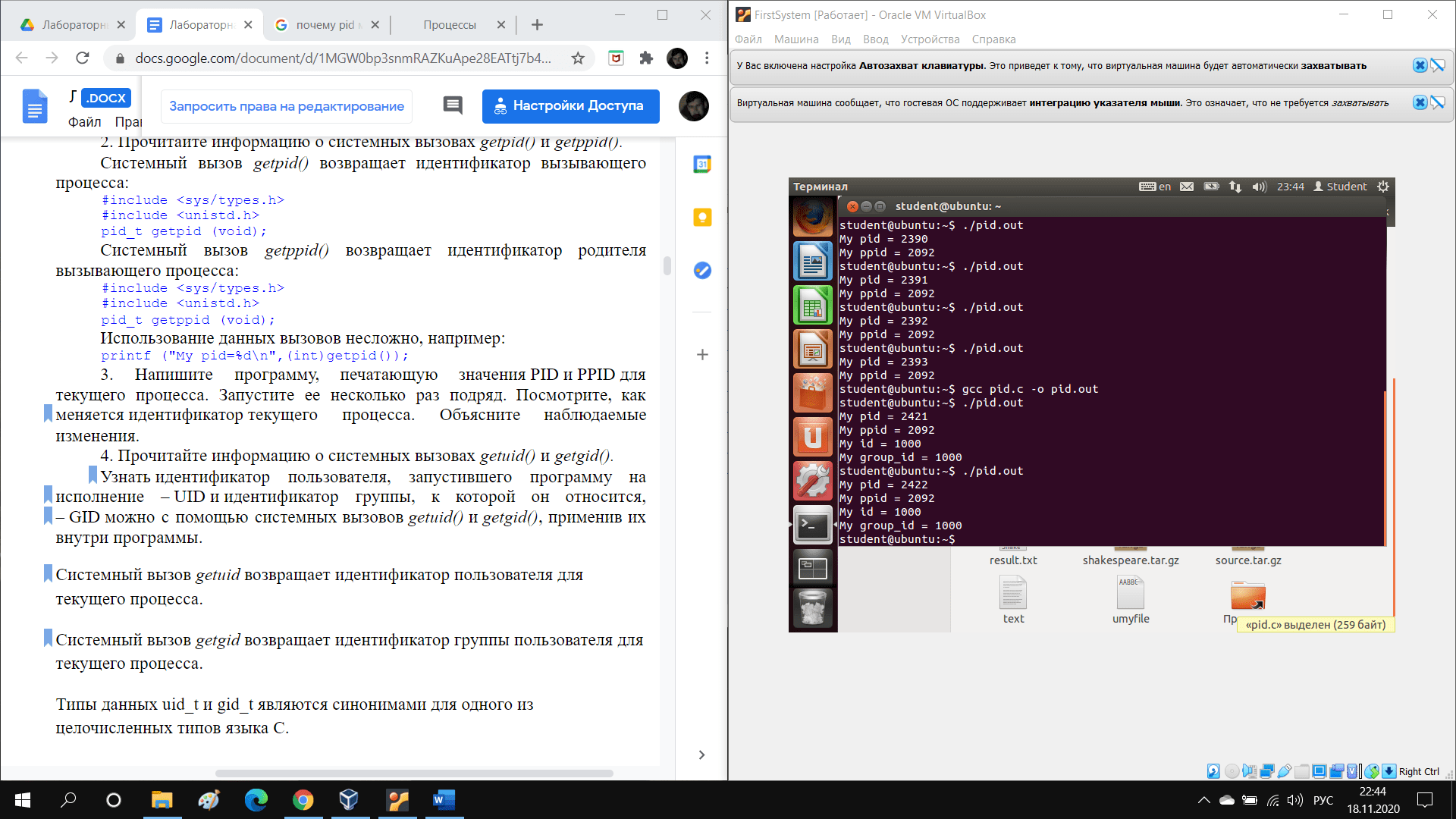




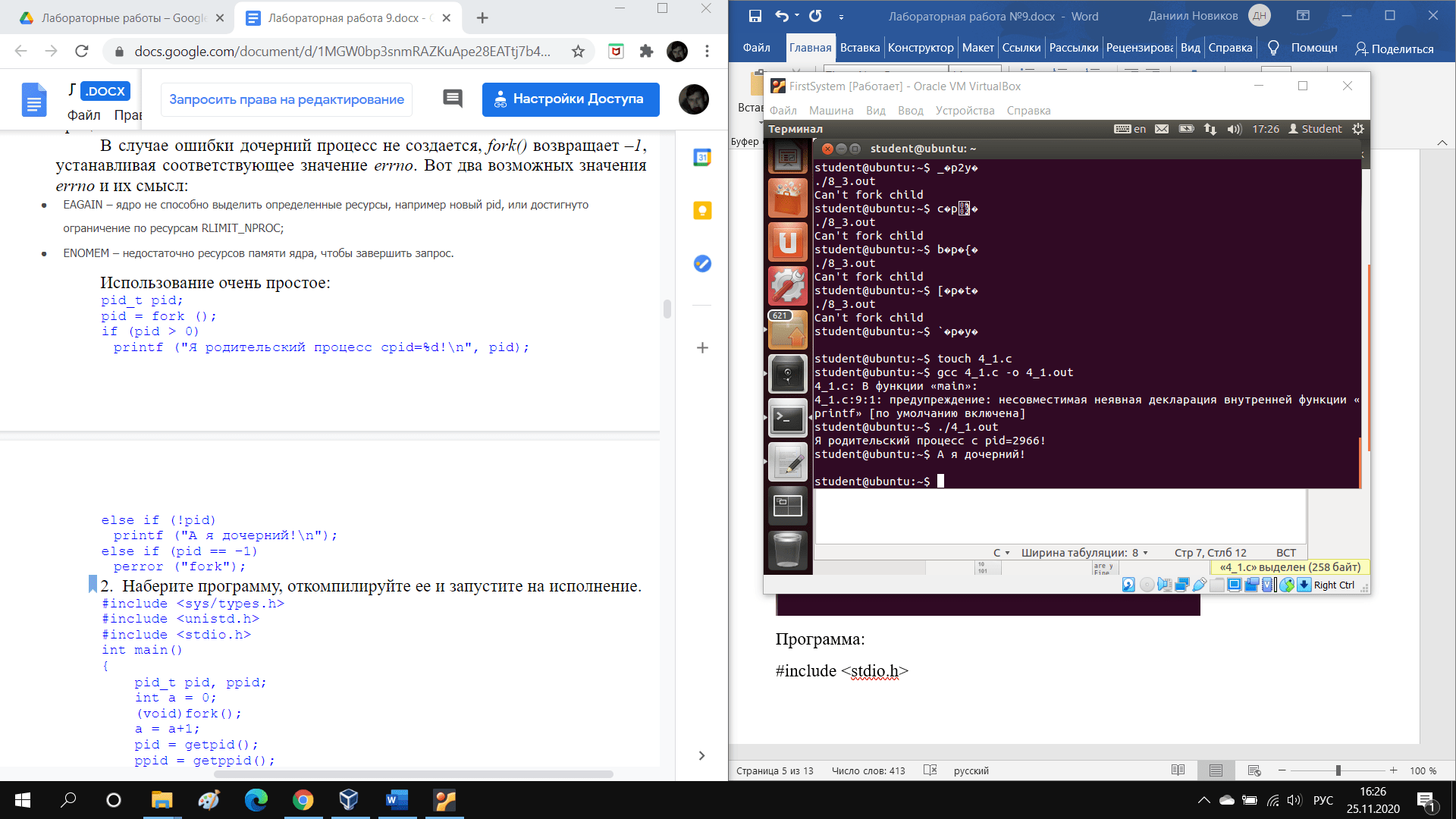
Новые процессы создаются вызовом int pid=fork(). PID назначаются последовательно. При создании нового процесса вызовом fork ищется PID, больший по значению, чем тот, который был возвращён предыдущим вызовом fork.

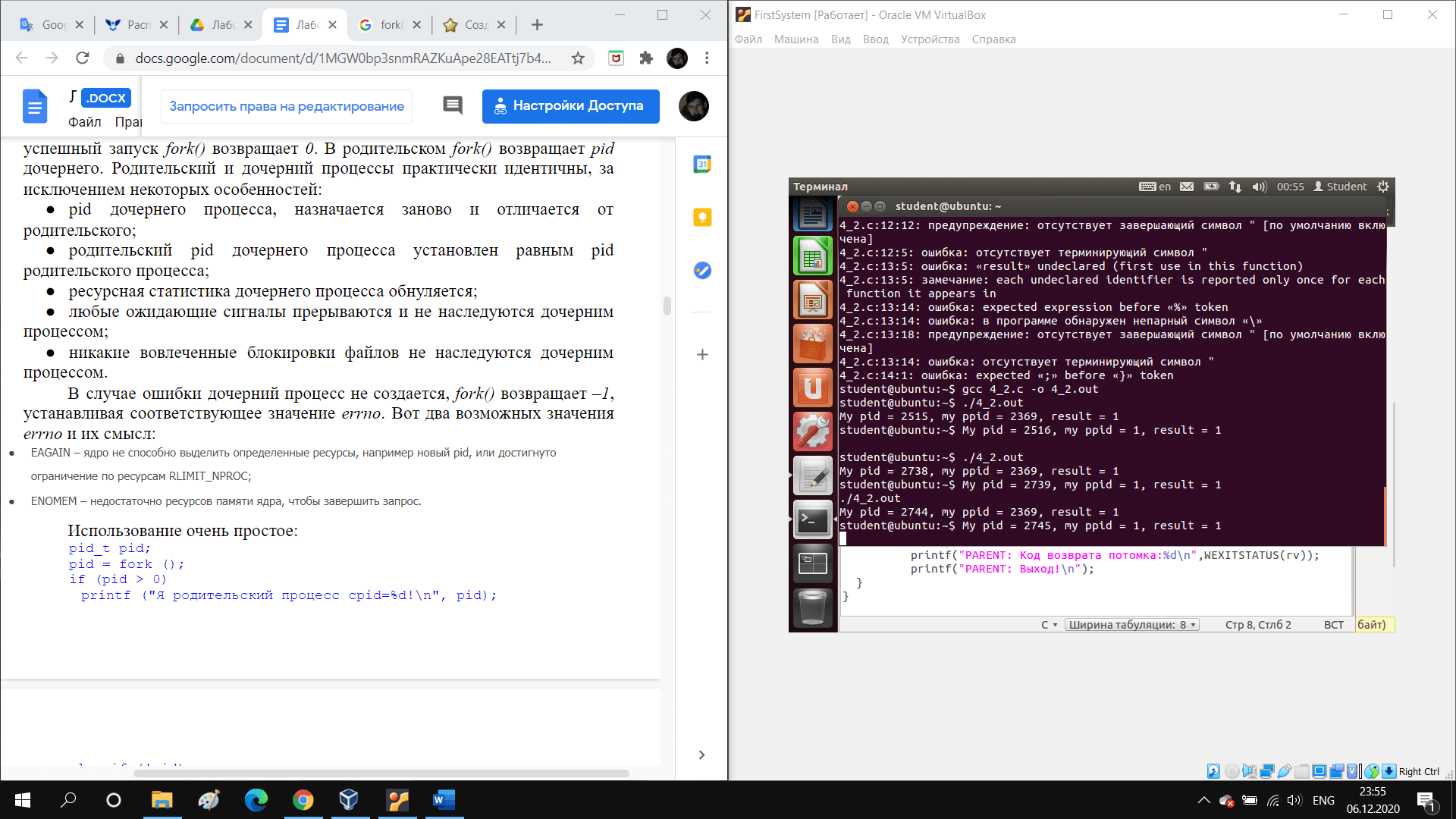
5. Дополним программу так, чтобы она печатала идентификатор пользователя, запустившего программу, и идентификатор его группы.





**Задание 4.**

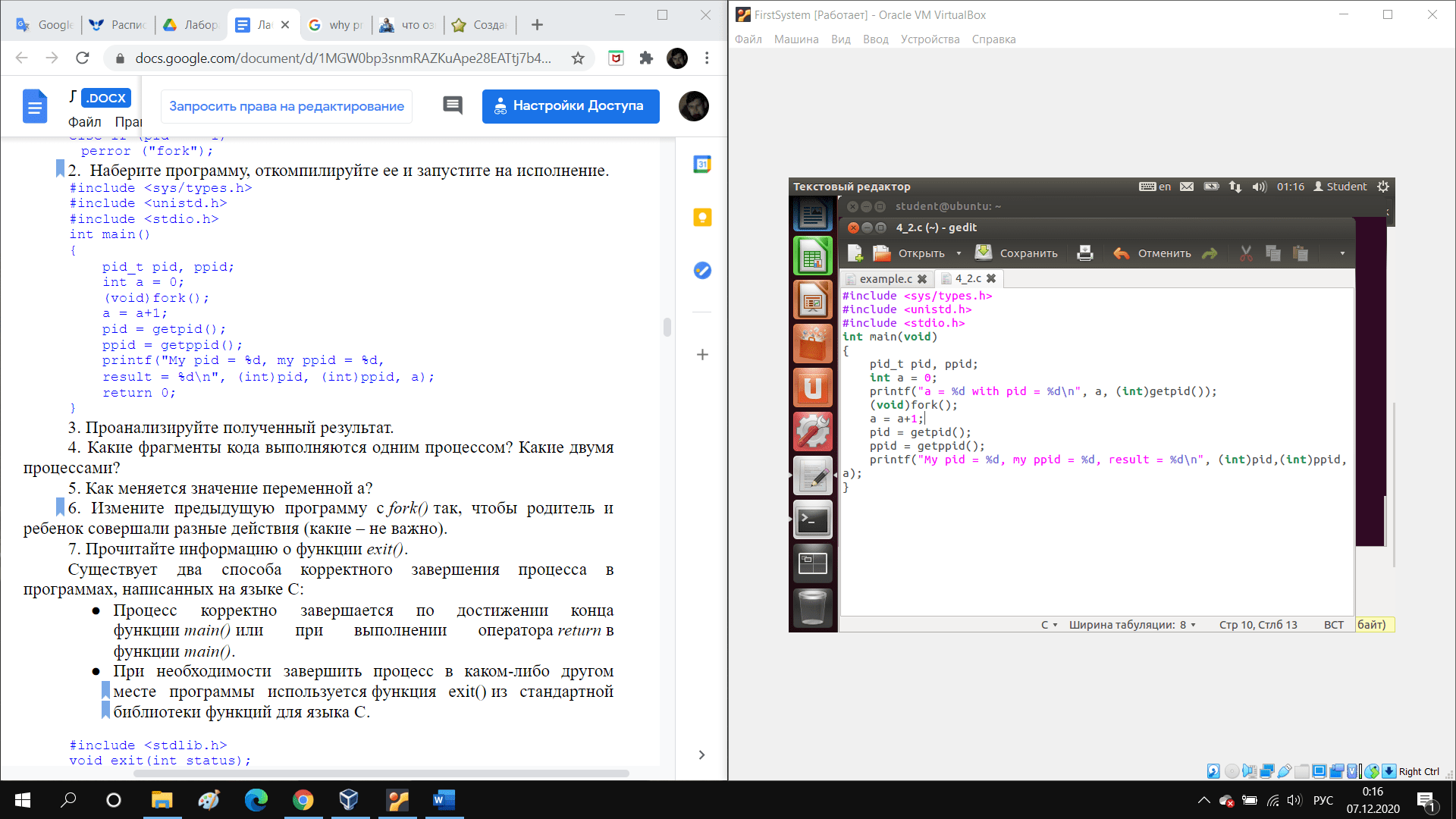


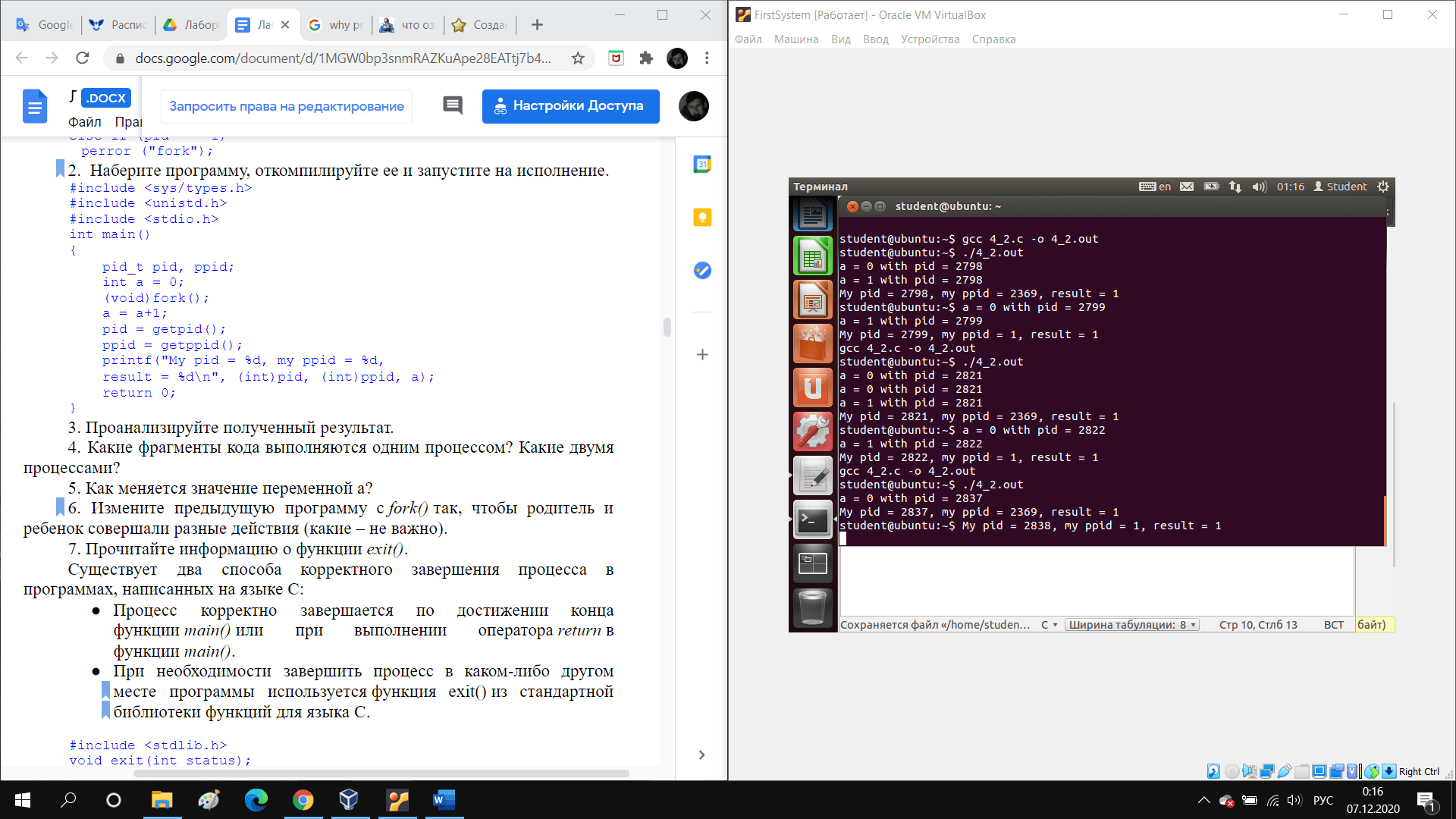


Сначала выполнился процесс-родитель, а затем процесс-потомок. Потомок получает следующий pid после pid родителя. В итоге процесс-родитель завершил работу и процесса-потомка родителем стал процесс init, с pid=1.

4.

Допишем код:

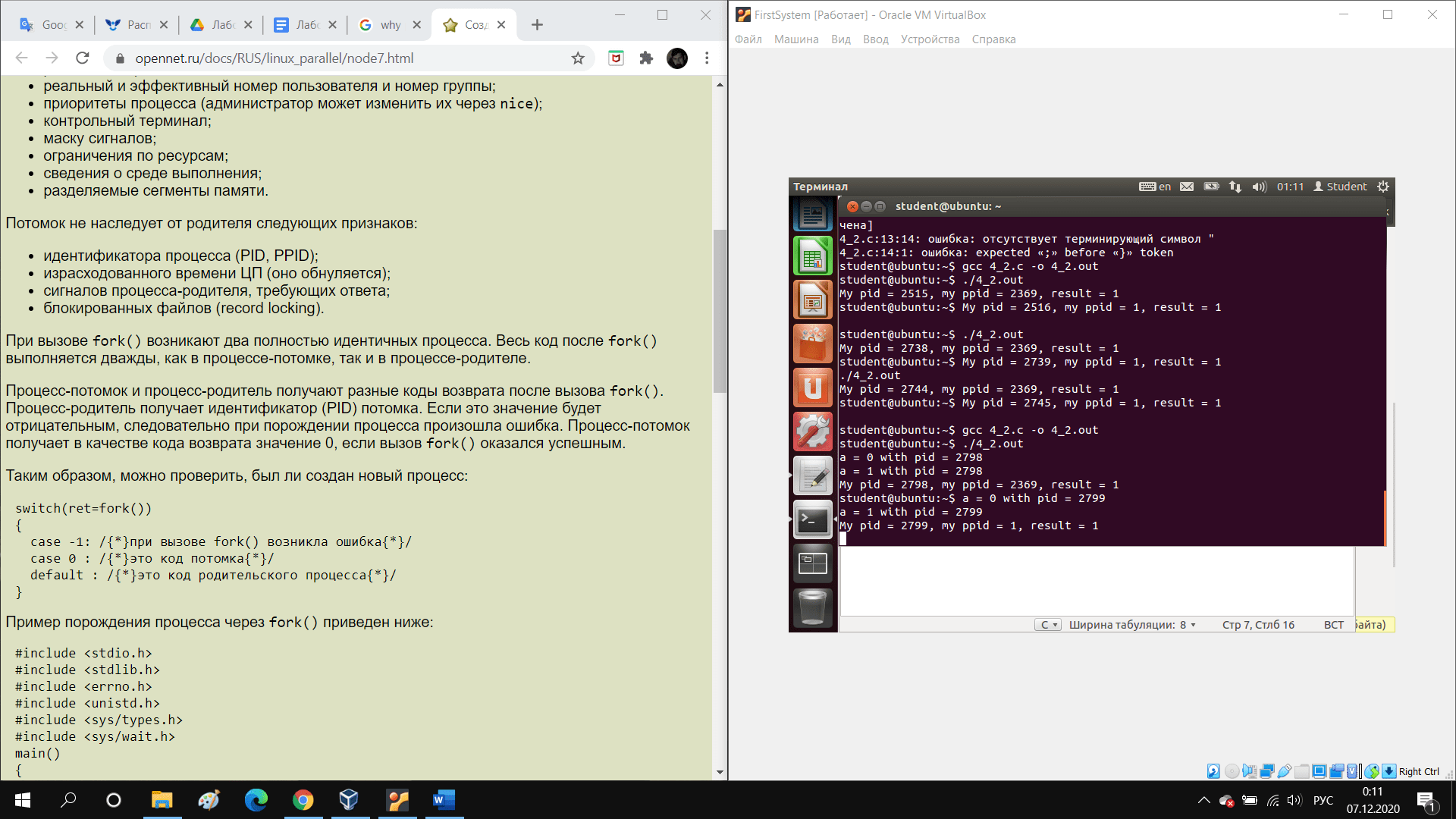




Таким образом, код после вызова fokr() вызывается двумя процессами, а после – одним родительским.

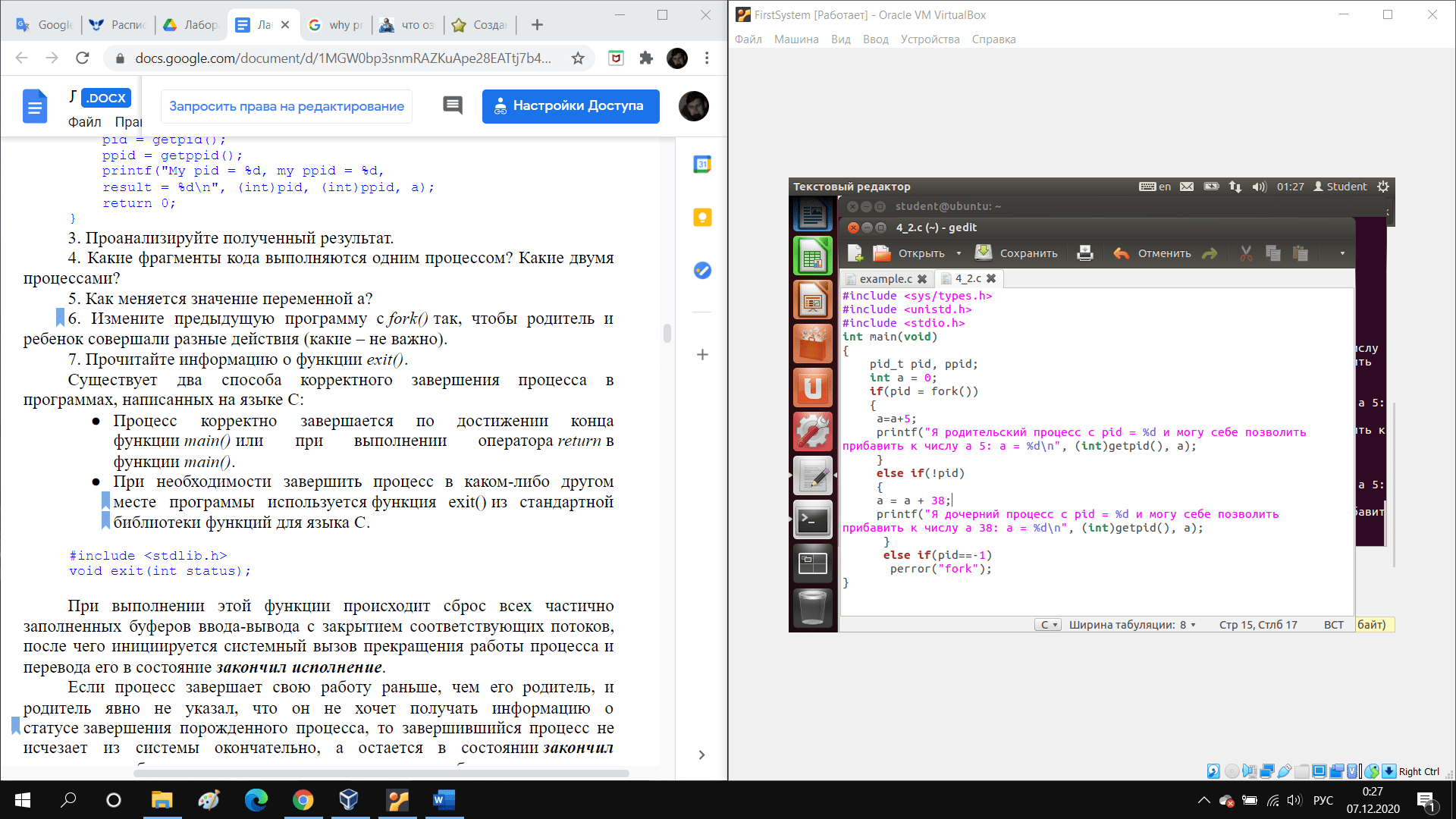
5. Проанализируем, почему значение переменной a одинаковое в 2-х процессах. Весь код после вызова fork() вызывается дважды, как в процессе-родителе, как и в процесса потомке. Дополнил код:

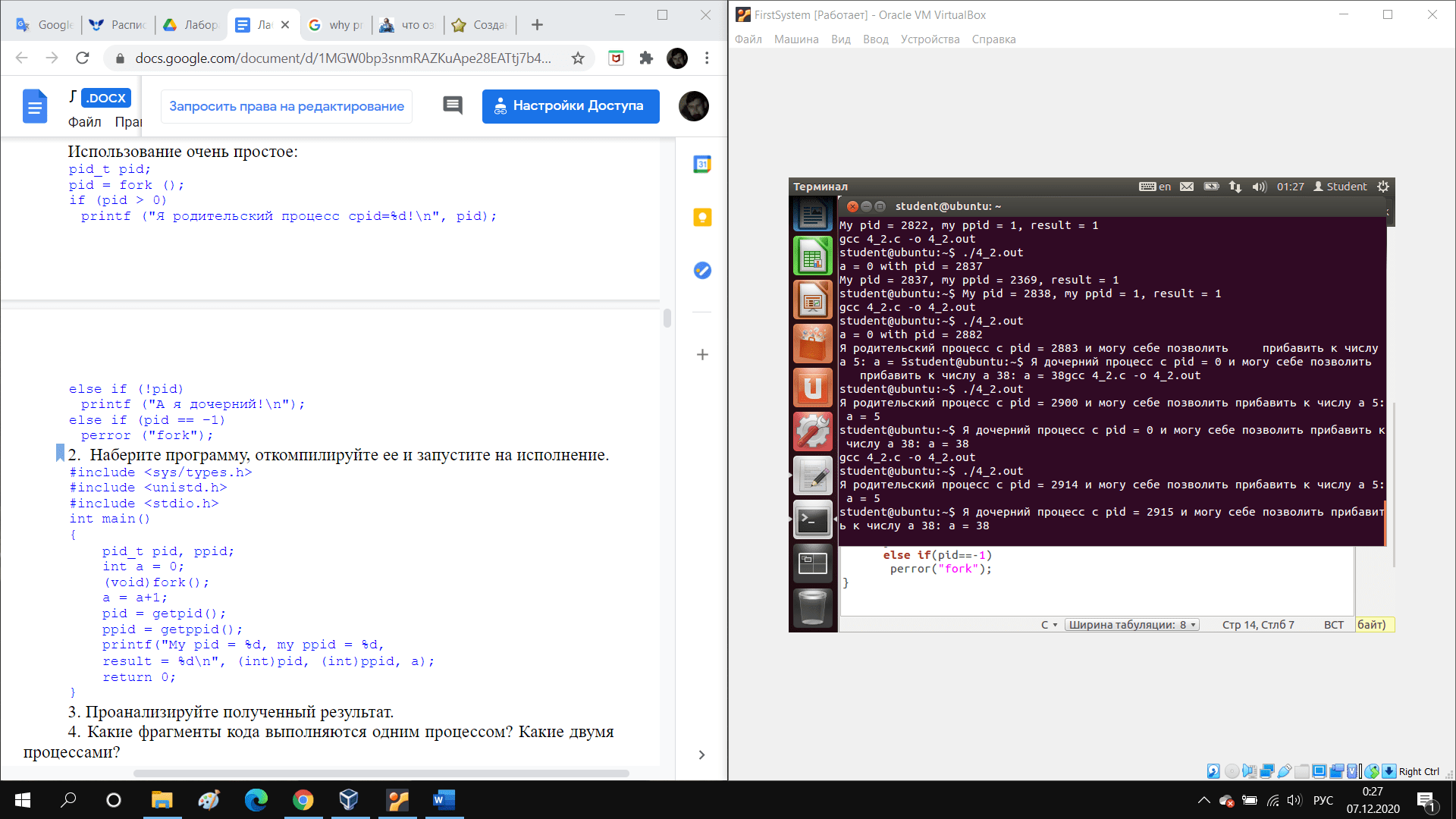




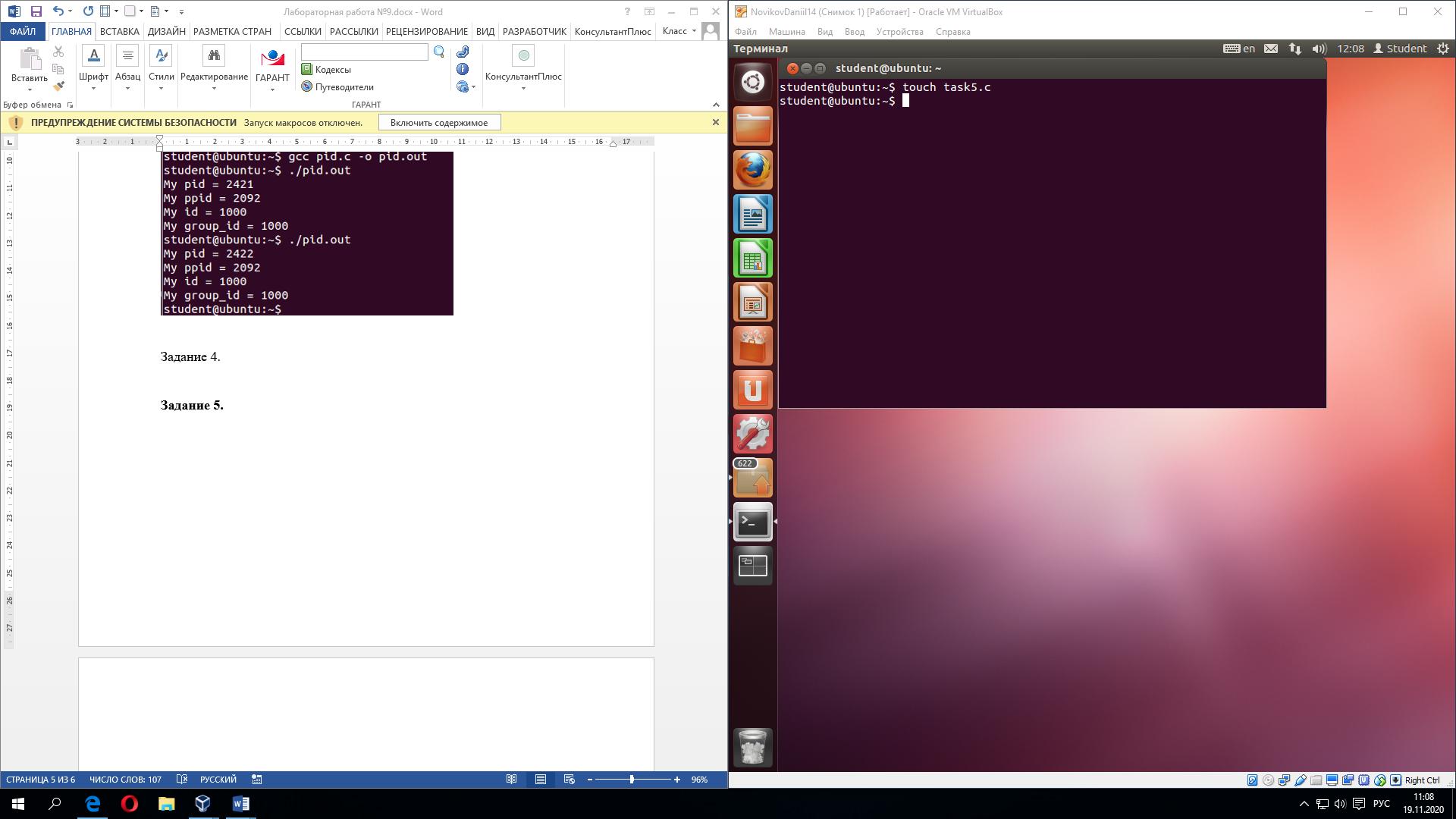
Каждый раз при начале работы значение a=0.

6.





**Задание 5.**



Программа:

#include <stdio.h>

int main(int argc, char \*argv[], char \*envp[])

{

printf("Значения аргументов командной строки:\n");

int i;

for(i=1;i<argc;i++)

{

printf("%s\n",argv[i]);

}

printf("Значения параметров окружающей среды:\n");

int index=0;

char \*parameter=envp[index];

while(parameter!=NULL)

{

printf("%s\n",parameter);

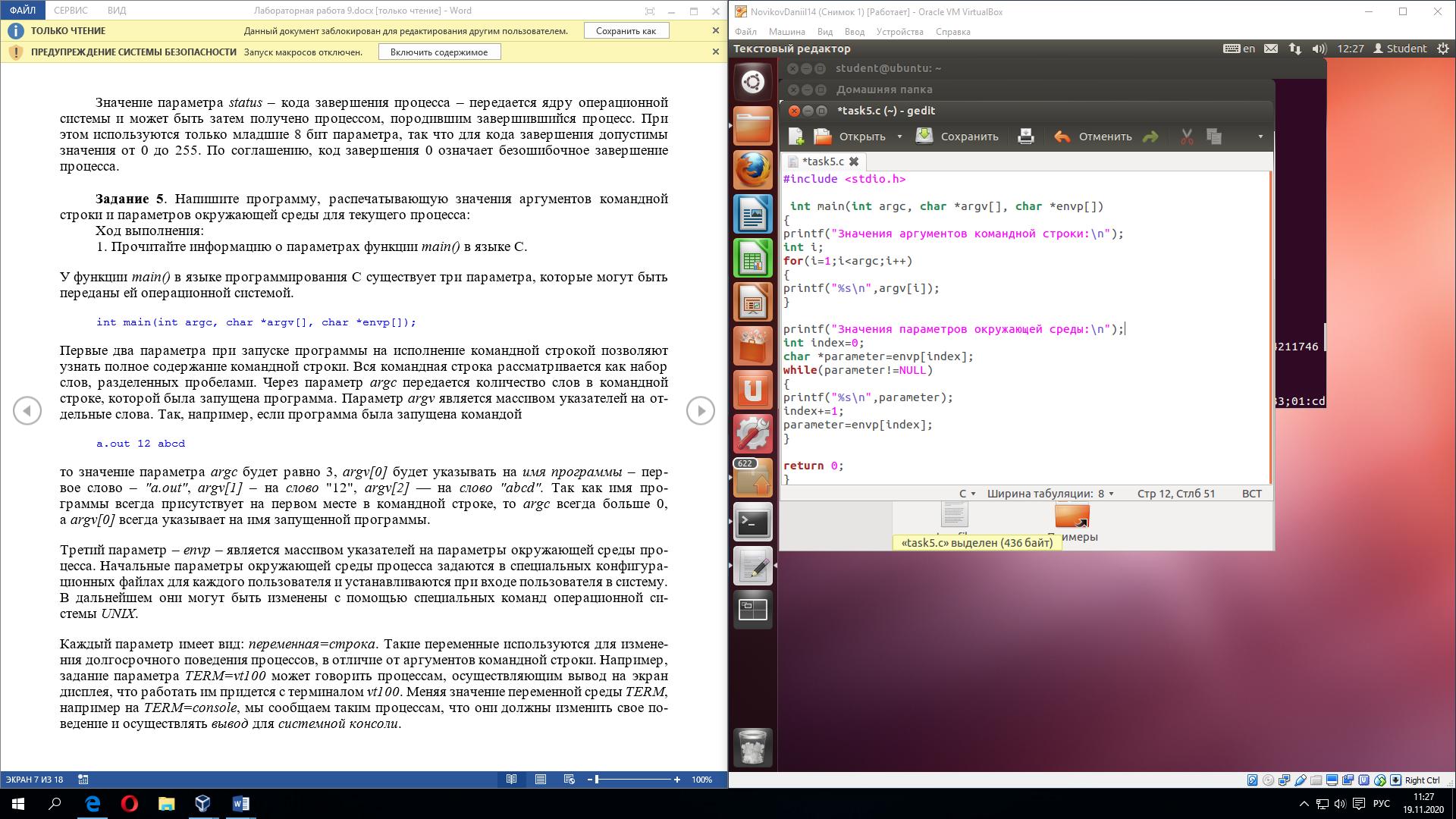
index+=1;

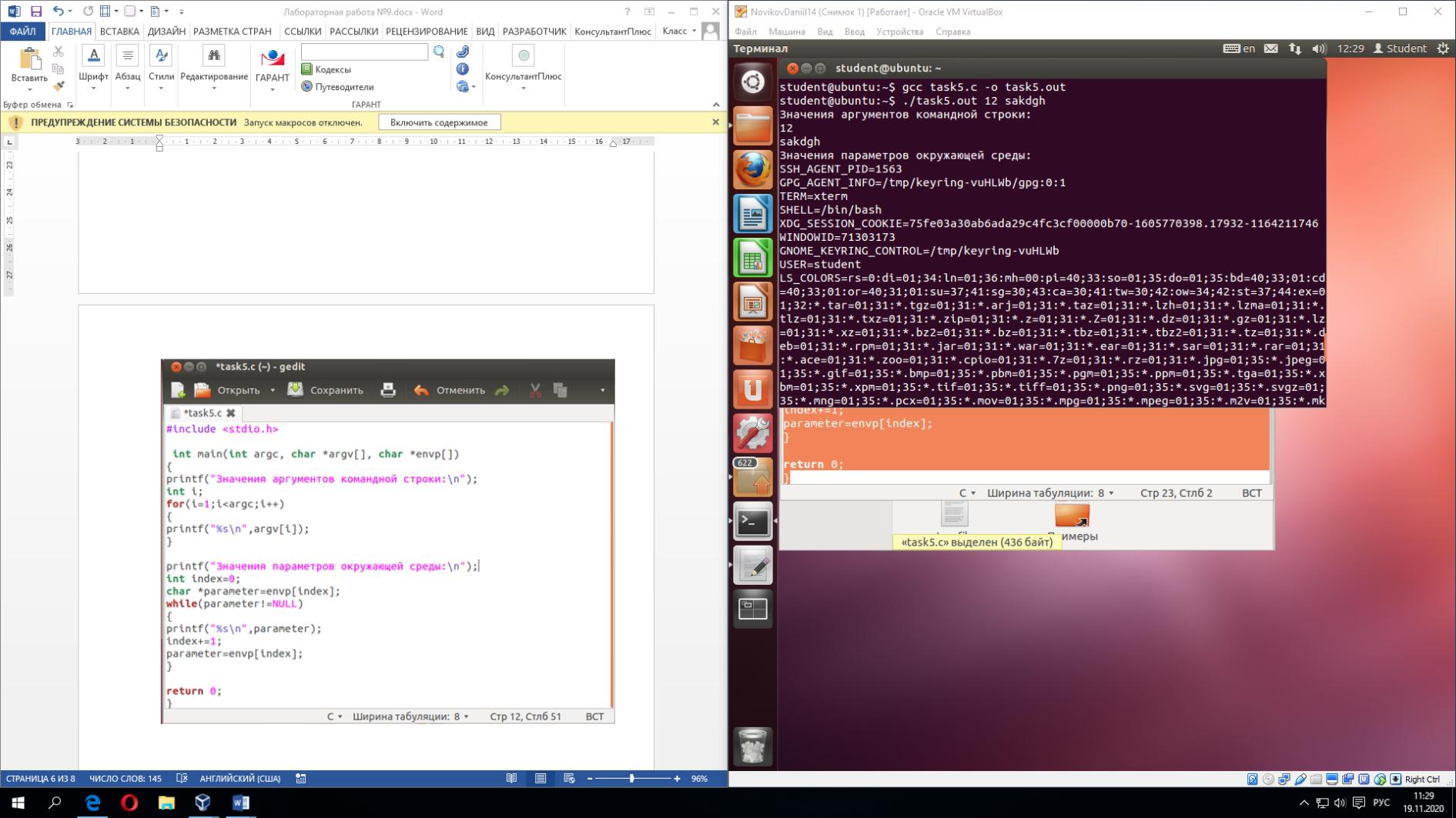
parameter=envp[index];

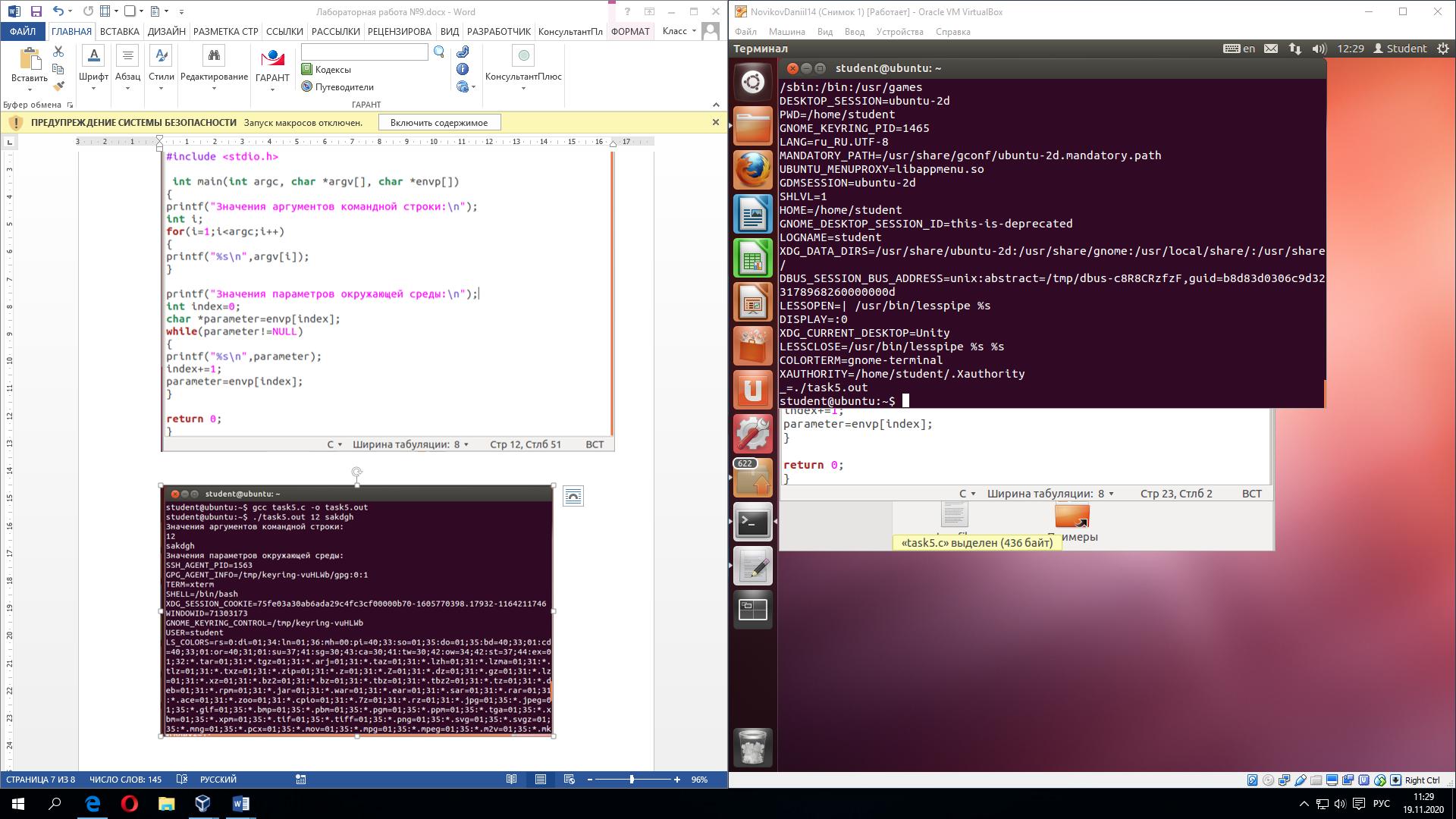
}

return 0;

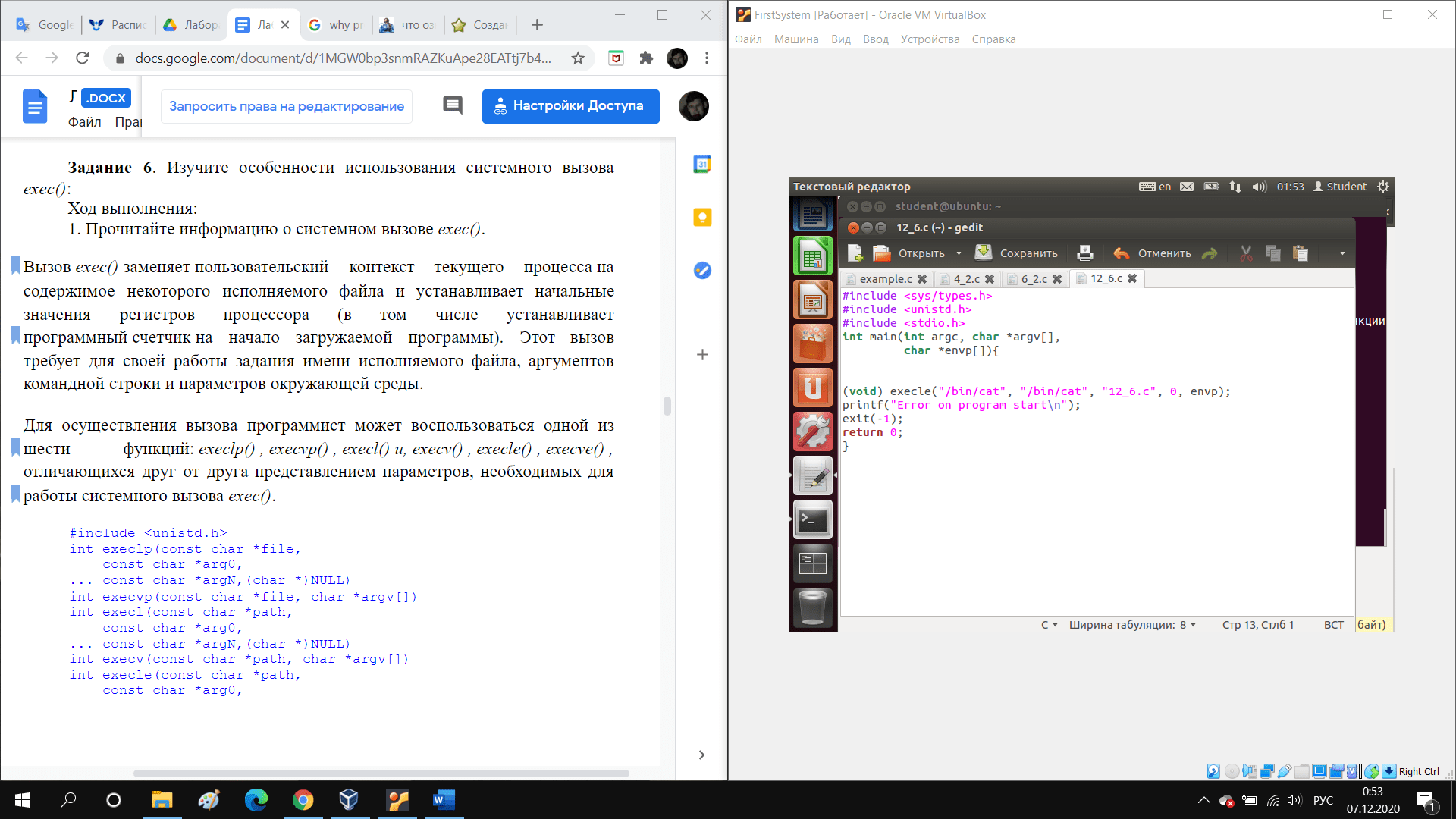
}

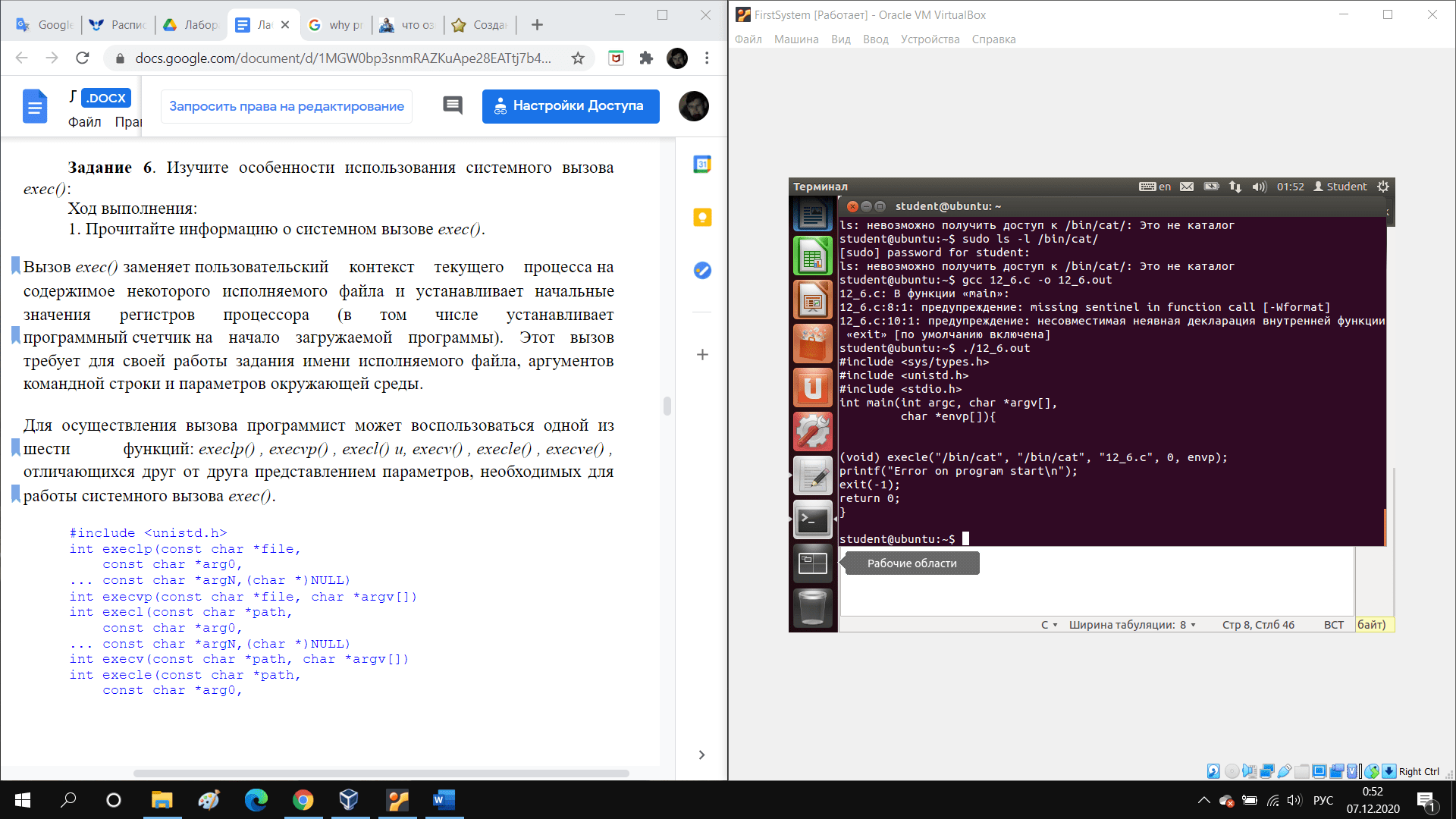






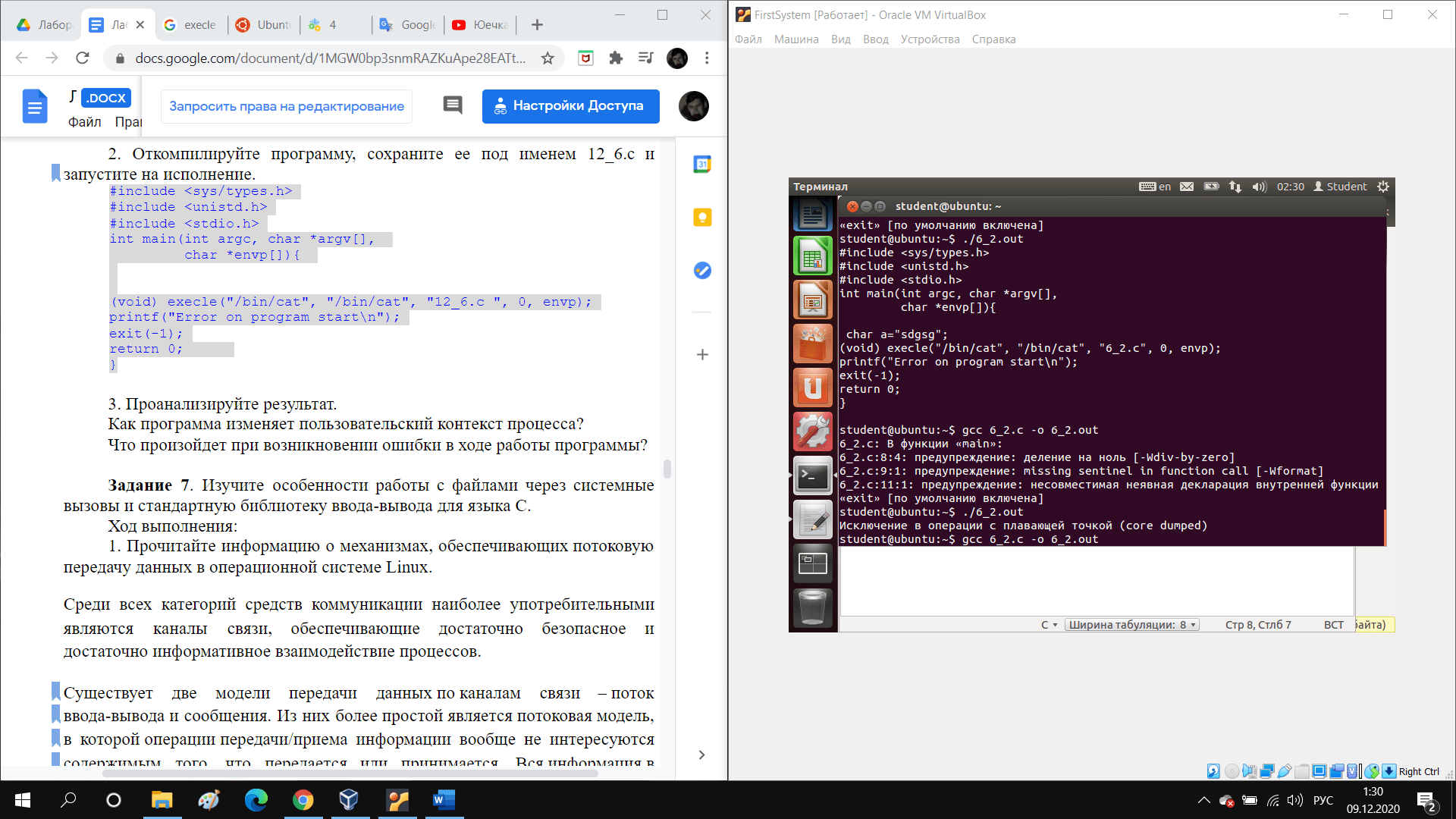
**Задание 6.**





Напишем в коде ошибку, чтобы посмотреть на результат.

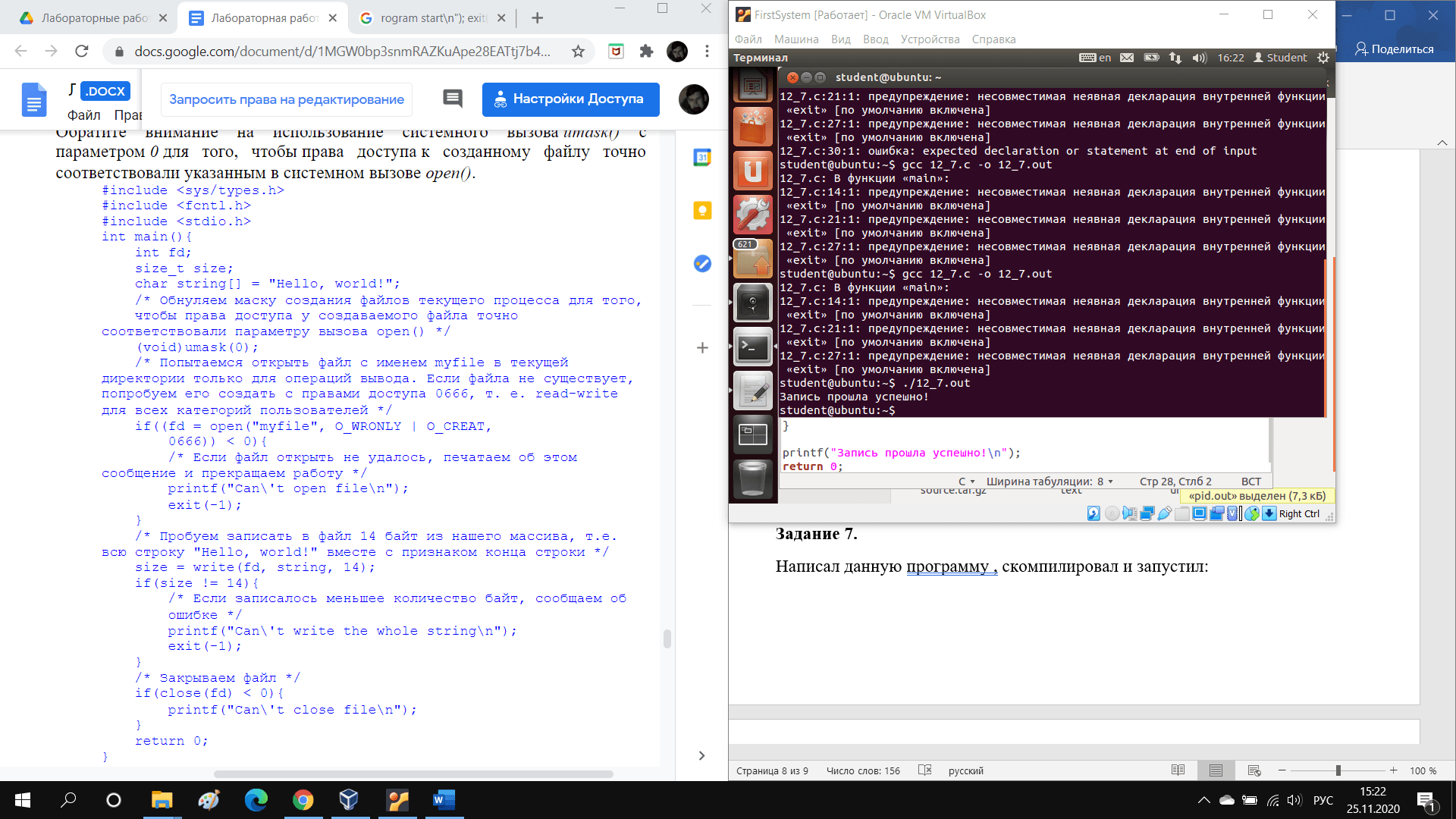




Вызов execle() заменяет пользовательский контекст текущего процесса на содержимое некоторого исполняемого файла.

**Задание 7.**

Написал данную программу, скомпилировал и запустил:



Добавил в программу операцию считывания содержимого файла:

Программа получилась следующей:

#include <sys/types.h>

#include <fcntl.h>

#include <stdio.h>

int main**()**

**{**

int fd**;**

size\_t size**;**

char string**[]=**"Hello, world!"**;**

**(**void**)**umask**(**0**);**

**if((**fd **=** open**(**"myfile444"**,** O\_WRONLY **|** O\_CREAT**,** 0666**))<**0**)**

**{**

printf**(**"Can't open file\n"**);**

exit**(-**1**);**

**}**

size**=**write**(**fd**,**string**,**14**);**

**if(**size**!=**14**)**

**{**

printf**(**"Can't write whole string\n"**);**

exit**(-**1**);**

**}**

**if(**close**(**fd**)<**0**)**

**{**

printf**(**"Can't close file\n"**);**

exit**(-**1**);**

**}**

printf**(**"Запись прошла успешно!\n"**);**

printf**(**"Начинается процесс чтения файла\n"**);**

**if((**fd **=** open**(**"myfile"**,** O\_RDONLY**))<**0**)**

**{**

printf**(**"Can't open file\n"**);**

exit**(-**1**);**

**}**

char ch**;**

ssize\_t ret**;**

printf**(**"Считанная строка: "**);**

**while((**ret**=**read**(**fd**,&**ch**,**1**))>**0**)**

**{**

putchar**(**ch**);**

**}**

**if(**ret**<**0**)**

**{**

printf**(**"Can't read file\n"**);**

exit**(-**1**);**

**}**

**if(**close**(**fd**)<**0**)**

**{**

printf**(**"Can't close file\n"**);**

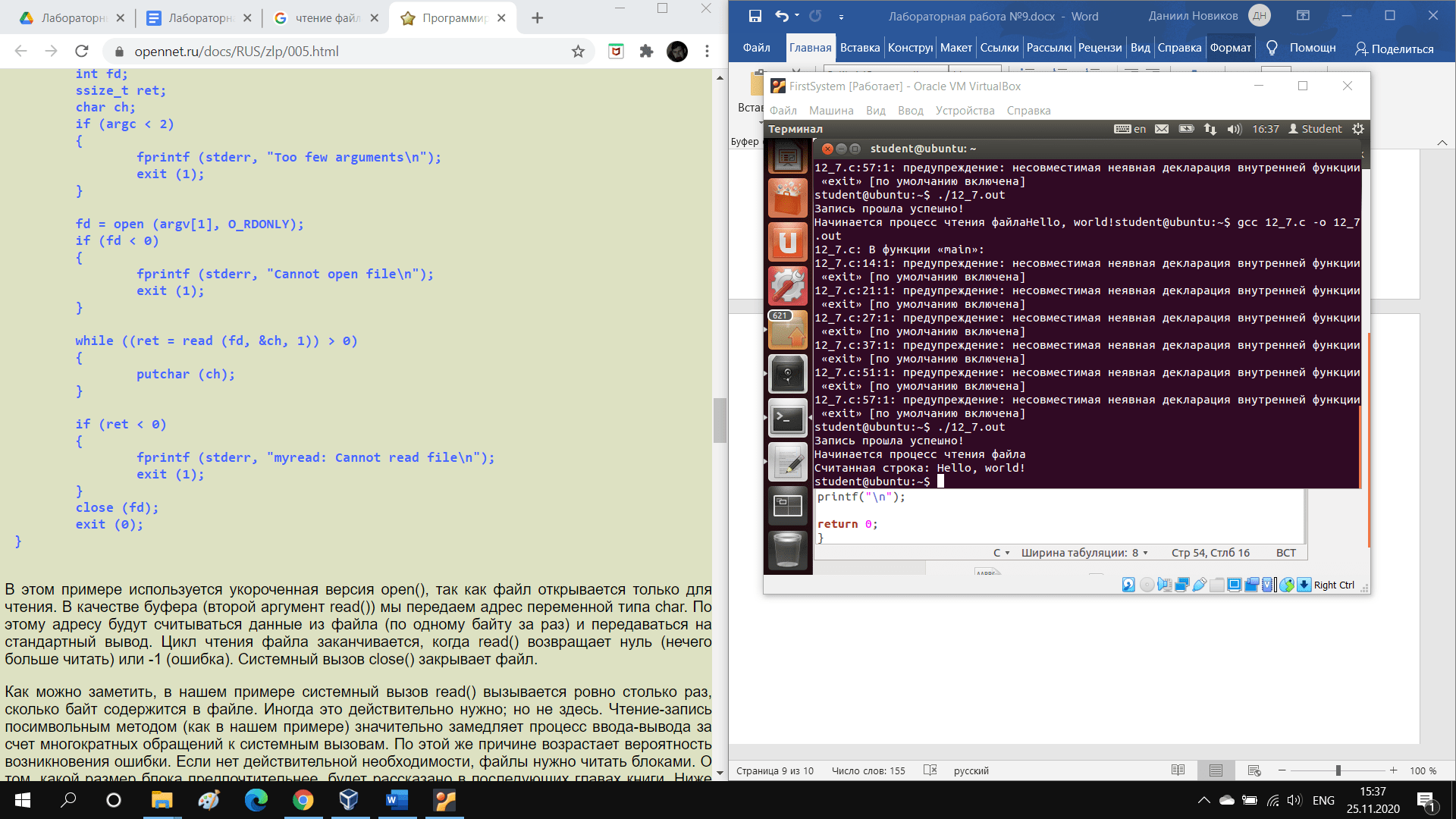
exit**(-**1**);**

**}**

printf**(**"\n"**);**

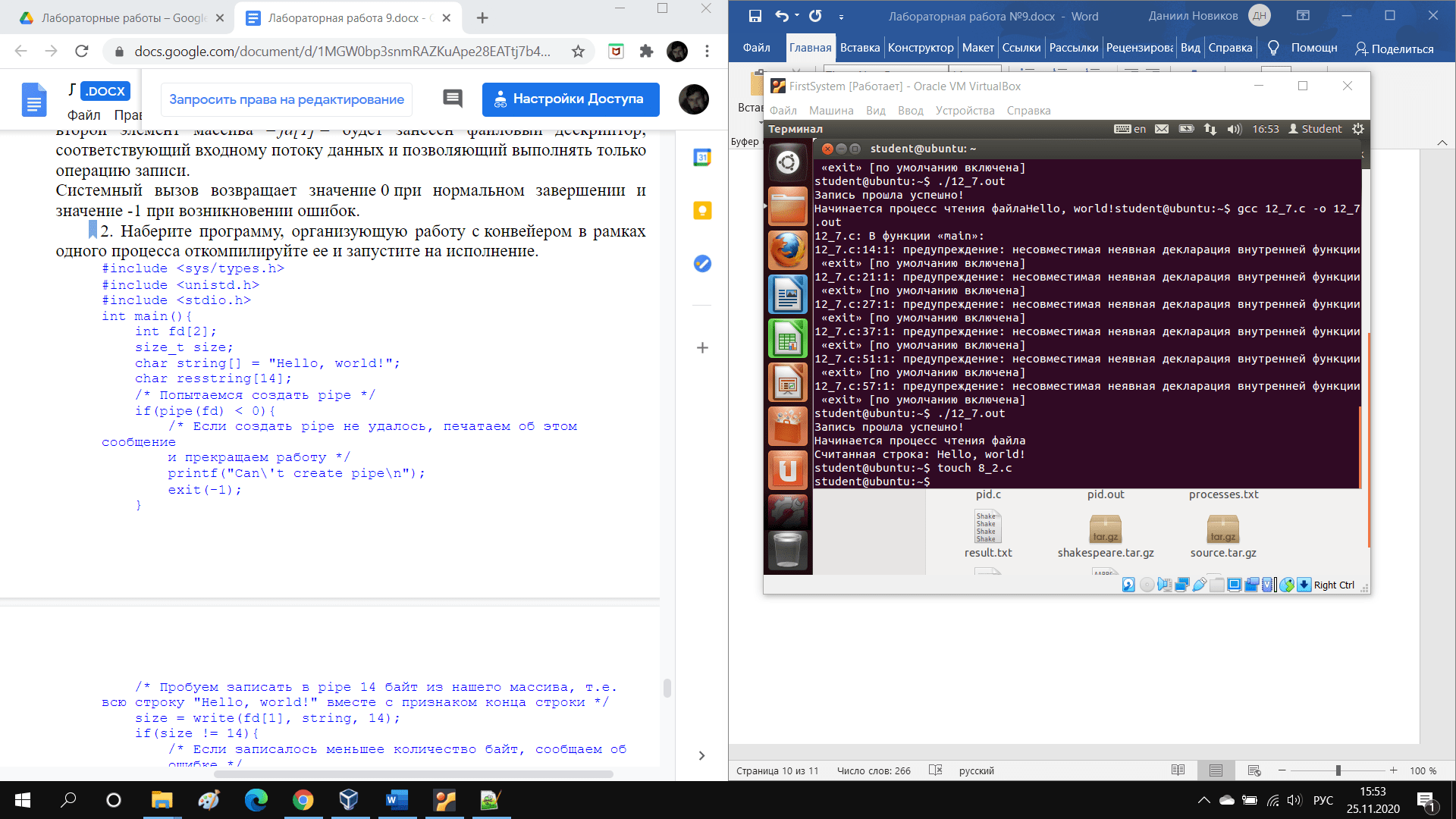
**return** 0**;**

**}**



**Задание 8.**

2. Набрал данную программу и запустил ее:



#include <sys/types.h>

#include <unistd.h>

#include <stdio.h>

int main**()**

**{**

int fd**[**2**];**

size\_t size**;**

char string**[]=**"Hello, world!"**;**

char resstring**[**14**];**

**if(**pipe**(**fd**)<**0**)**

**{**

printf**(**"Can't create pipe\n"**);**

exit**(-**1**);**

**}**

size**=**write**(**fd**[**1**],**string**,**14**);**

**if(**size**!=**14**)**

**{**

printf**(**"Can't write all string\n"**);**

exit**(-**1**);**

**}**

size**=**read**(**fd**[**0**],**resstring**,**14**);**

**if(**size**<**0**)**

**{**

printf**(**"Can't read string\n"**);**

exit**(-**1**);**

**}**

printf**(**"%s\n"**,**resstring**);**

**if(**close**(**fd**[**0**])<**0**)**

**{**

printf**(**"Can't close input stream\n"**);**

**}**

**if(**close**(**fd**[**1**])<**0**)**

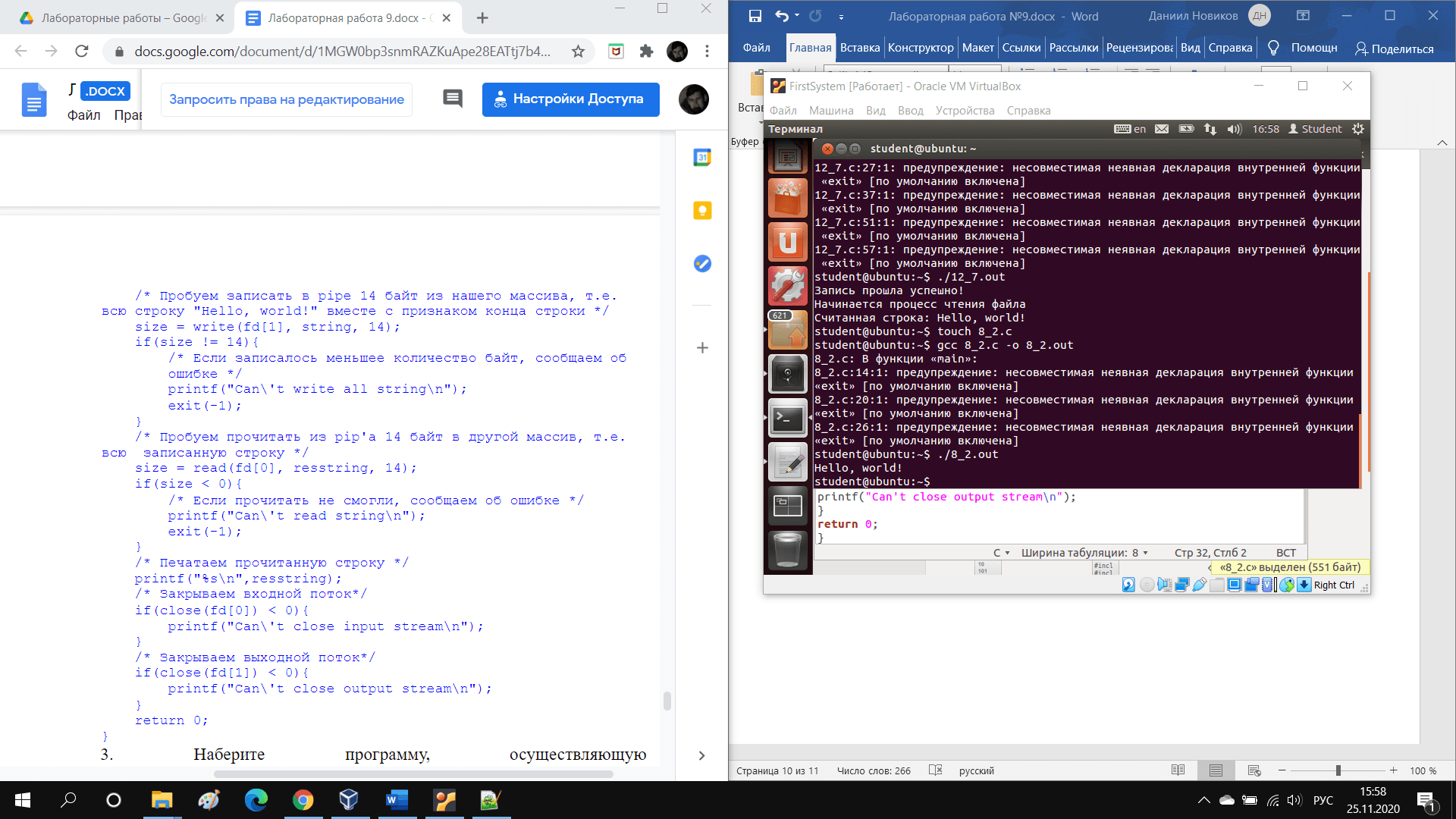
**{**

printf**(**"Can't close output stream\n"**);**

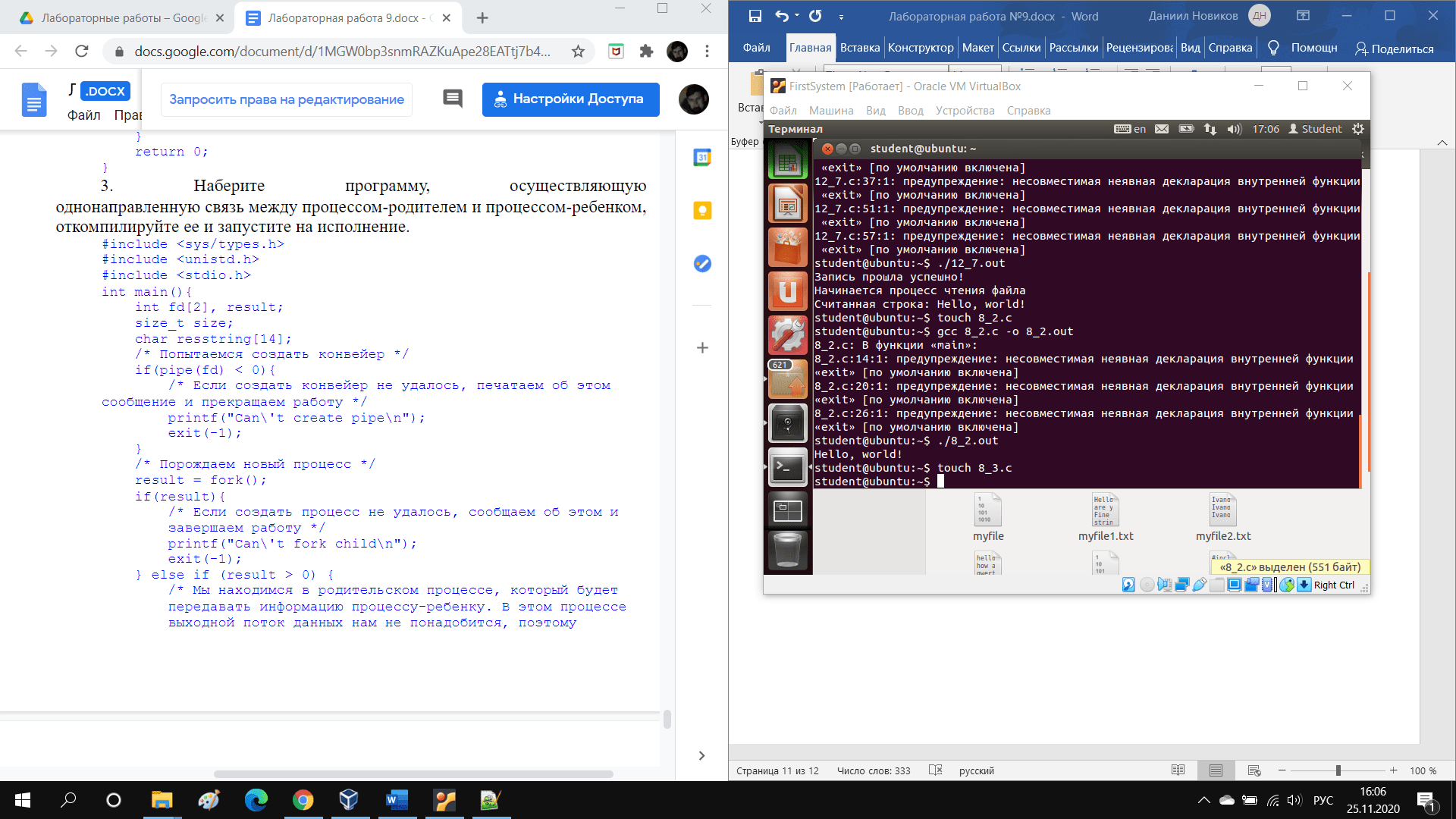
**}**

**return** 0**;**

**}**



3. Набрал программу и запустил на исполнение:



#include <sys/types.h>

#include <unistd.h>

#include <stdio.h>

int main**(){**

int fd**[**2**],** result**;**

size\_t size**;**

char resstring**[**14**];**

/\* Попытаемся создать конвейер \*/

**if(**pipe**(**fd**)** **<** 0**){**

/\* Если создать конвейер не удалось, печатаем об этом сообщение и прекращаем работу \*/

printf**(**"Can\'t create pipe\n"**);**

exit**(-**1**);**

**}**

/\* Порождаем новый процесс \*/

result **=** fork**();**

**if(**result**<**0**){**

/\* Если создать процесс не удалось, сообщаем об этом и

завершаем работу \*/

printf**(**"Can\'t fork child\n"**);**

exit**(-**1**);**

**}** **else** **if** **(**result **>** 0**)** **{**

/\* Мы находимся в родительском процессе, который будет

передавать информацию процессу-ребенку. В этом процессе

выходной поток данных нам не понадобится, поэтому

закрываем его.\*/

close**(**fd**[**0**]);**

/\* Пробуем записать в pipe 14 байт, т.е. всю строку

"Hello, world!" вместе с признаком конца строки \*/

size **=** write**(**fd**[**1**],** "Hello, world!"**,** 14**);**

**if(**size **!=** 14**){**

/\* Если записалось меньшее количество байт, сообщаем

об ошибке и завершаем работу \*/

printf**(**"Can\'t write all string\n"**);**

exit**(-**1**);**

**}**

/\* Закрываем входной поток данных, на этом

родитель прекращает работу \*/

close**(**fd**[**1**]);**

printf**(**"Parent exit\n"**);**

**}** **else** **{**

/\* Мы находимся в порожденном процессе, который будет

получать информацию от процесса-родителя. Он унаследовал

от родителя таблицу открытых файлов.

В этом процессе входной поток данных нам не

понадобится, поэтому закрываем его.\*/

close**(**fd**[**1**]);**

/\* Пробуем прочитать из конвейера 14 байт в массив, т.е. всю записанную строку \*/

size **=** read**(**fd**[**0**],** resstring**,** 14**);**

**if(**size **<** 0**){**

/\* Если прочитать не смогли, сообщаем об ошибке и

завершаем работу \*/

printf**(**"Can\'t read string\n"**);**

exit**(-**1**);**

**}**

/\* Печатаем прочитанную строку \*/

printf**(**"%s\n"**,**resstring**);**

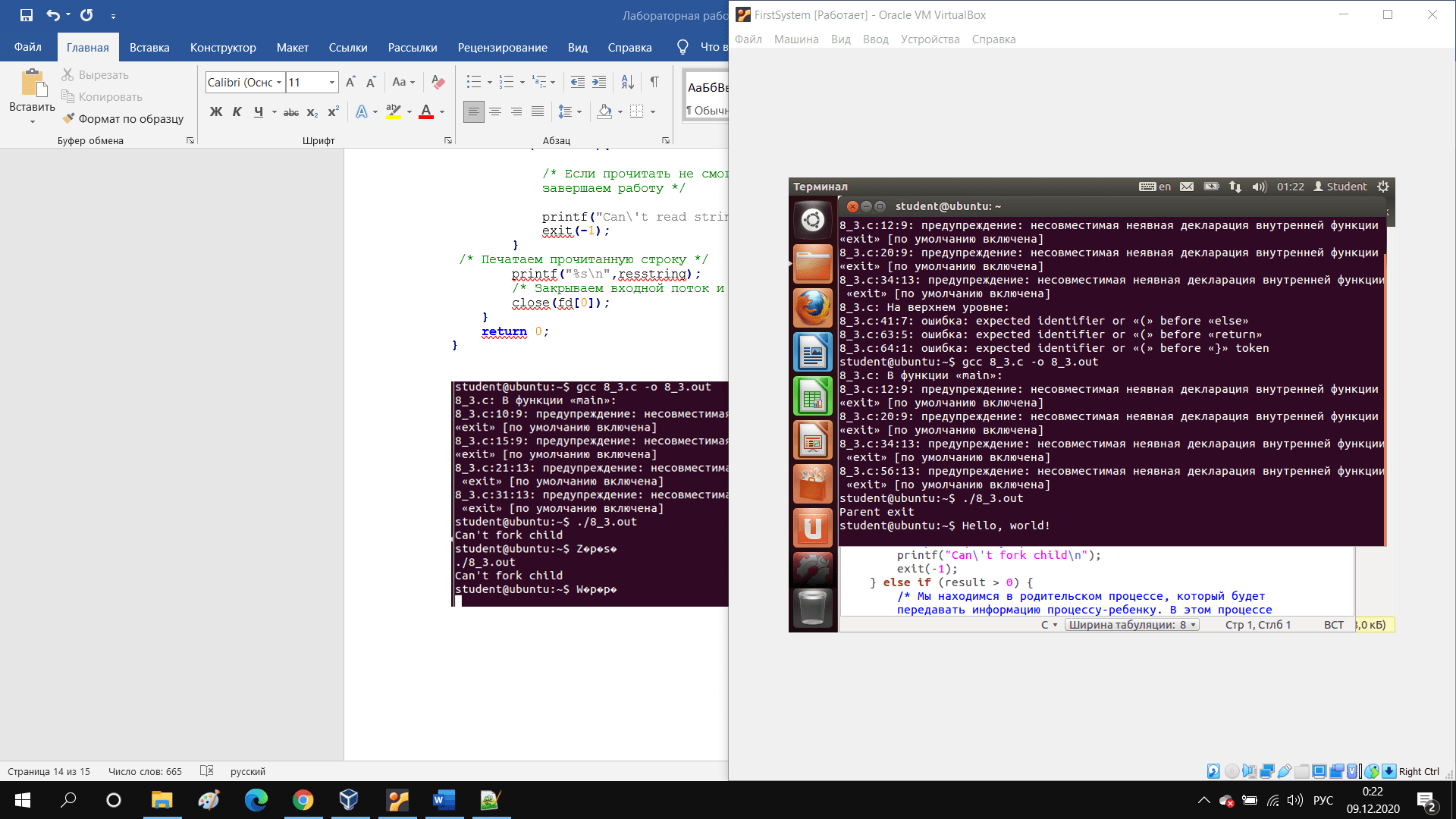
/\* Закрываем входной поток и завершаем работу \*/

close**(**fd**[**0**]);**

**}**

**return** 0**;**

**}**



4.

Для данной задачи создадим два конвейера – дли родителя и для ребенка.

Код программы:

#include <sys/types.h>

#include <unistd.h>

#include <stdio.h>

#include <sys/types.h>

#include <sys/wait.h>

int main**(){**

int result**;**

int fd\_p**[**2**];**//для родителя

int fd\_c**[**2**];**//для потомка

size\_t size**;**

char parent\_string**[**17**];**//строка от отца

char child\_string**[**15**];**//строка, которую будет принимать отец от ребенка

**if(**pipe**(**fd\_p**)** **<** 0**){**

printf**(**"Can\'t create pipe\n"**);**

exit**(-**1**);**

**}**

**if(**pipe**(**fd\_c**)** **<** 0**){**

printf**(**"Can\'t create pipe\n"**);**

exit**(-**1**);**

**}**

result **=** fork**();**

**if(**result**<**0**){**

printf**(**"Can\'t fork child\n"**);**

exit**(-**1**);**

**}** **else** **if** **(**result **>** 0**)** **{**

size **=** write**(**fd\_p**[**1**],** "Hello, my child!"**,** 17**);**

**if(**size **!=** 17**){**

printf**(**"Can\'t write all string\n"**);**

exit**(-**1**);**

**}**

close**(**fd\_p**[**1**]);**

wait**(NULL);**//ждем пока не завершится дочерний процесс

size **=** read**(**fd\_c**[**0**],** child\_string**,** 15**);**

printf**(**"А ребенок мне послал это: %s\n"**,**child\_string**);**

printf**(**"Parent exit\n"**);**

close**(**fd\_p**[**0**]);**

**}** **else** **{**

size **=** read**(**fd\_p**[**0**],** parent\_string**,** 17**);**

**if(**size **<** 0**){**

printf**(**"Can\'t read string\n"**);**

exit**(-**1**);**

**}**

printf**(**"Отец послал мне вот это: %s\n"**,**parent\_string**);**

size**=**write**(**fd\_c**[**1**],** "Hi, my parent!"**,** 15**);**//записываем строку для родителя

close**(**fd\_c**[**1**]);**

**if(**size **!=** 15**){**

printf**(**"Can\'t write all string\n"**);**

exit**(-**1**);**

**}**

close**(**fd\_c**[**0**]);**

**}**

**return** 0**;**

**}**

