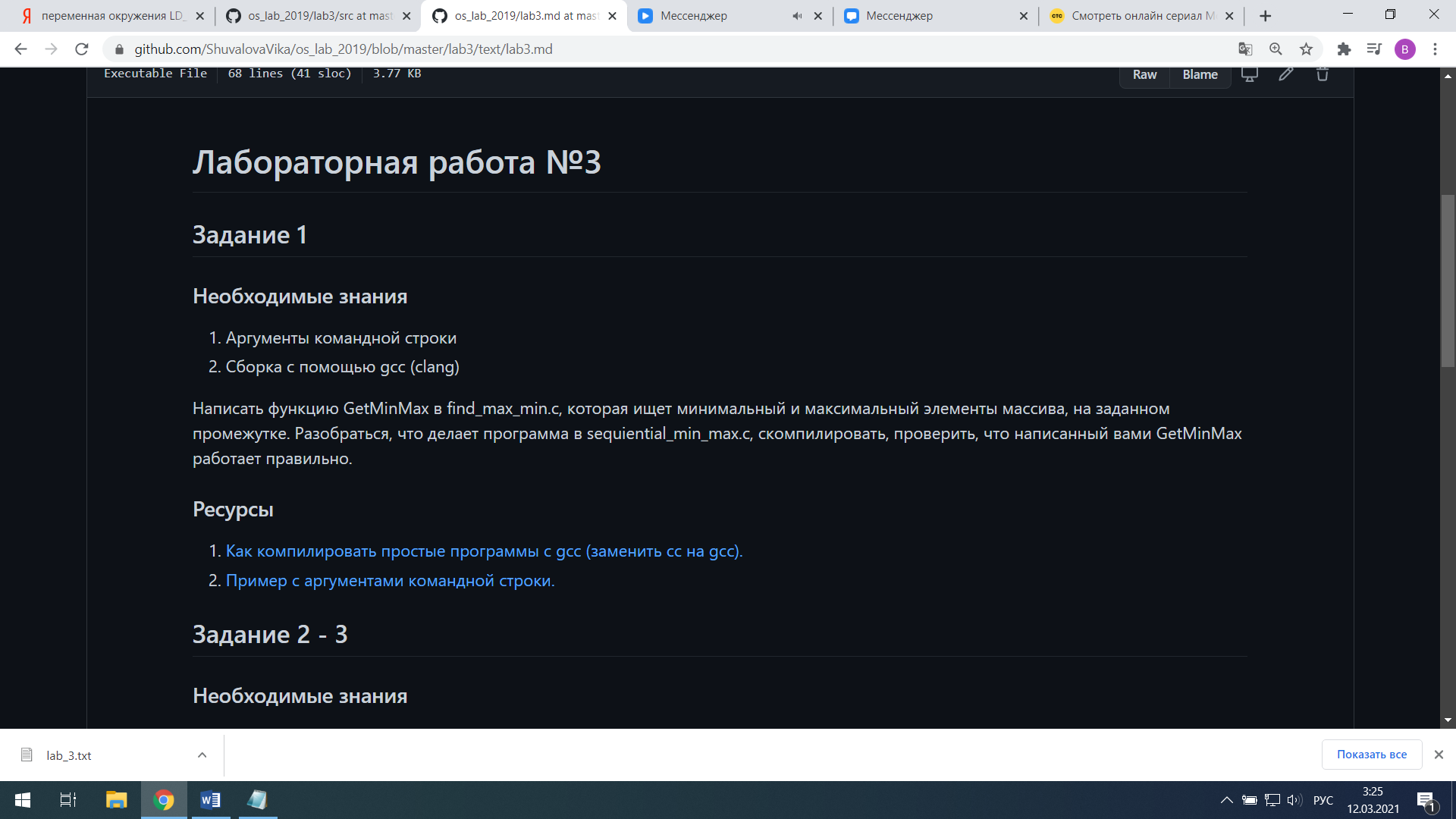
**Лабораторная работа № 3.**

Немецков Михаил, ПМ-31

**Задание 1.**

Написать функцию GetMinMax в find\_max\_min.c, которая ищет минимальный и максимальный элементы массива, на заданном промежутке. Разобраться, что делает программа в sequiential\_min\_max.c, скомпилировать, проверить, что написанный вами GetMinMax работает правильно.



1) **Аргументы командной строки** — это необязательные строковые аргументы, передаваемые операционной системой в программу при её запуске. Программа может их использовать в качестве входных данных, либо игнорировать. Подобно тому, как параметры одной функции предоставляют данные для параметров другой функции, так и аргументы командной строки предоставляют возможность людям или программам предоставлять входные данные для программы.

2)gcc- выполняет компиляцию

Функция GetMinMax

|  |
| --- |
| #include "find\_min\_max.h" |
|  |  |
|  | #include <limits.h> |
|  |  |
|  | struct MinMax GetMinMax(int \*array, unsigned int begin, unsigned int end) { |
|  | struct MinMax min\_max; |
|  | min\_max.min = INT\_MAX; |
|  | min\_max.max = INT\_MIN; |
|  | for(int i = begin; i < end; i++) { |
|  | if(array[i] < min\_max.min) |
|  | min\_max.min = array[i]; |
|  | if(array[i] > min\_max.max) |
|  | min\_max.max = array[i]; |
|  | } |
|  |  |
|  | return min\_max; |
|  | } |
|  |  |

**Компилируем**

~$ cd lab3/src/

~/lab3/src$ gcc sequential\_min\_max.c find\_min\_max.c utils.c -o sequential\_min\_max

~/lab3/src$ exec ./sequential\_min\_max 123 10

min: 95168426

max: 1888844001

Пояснения по коду

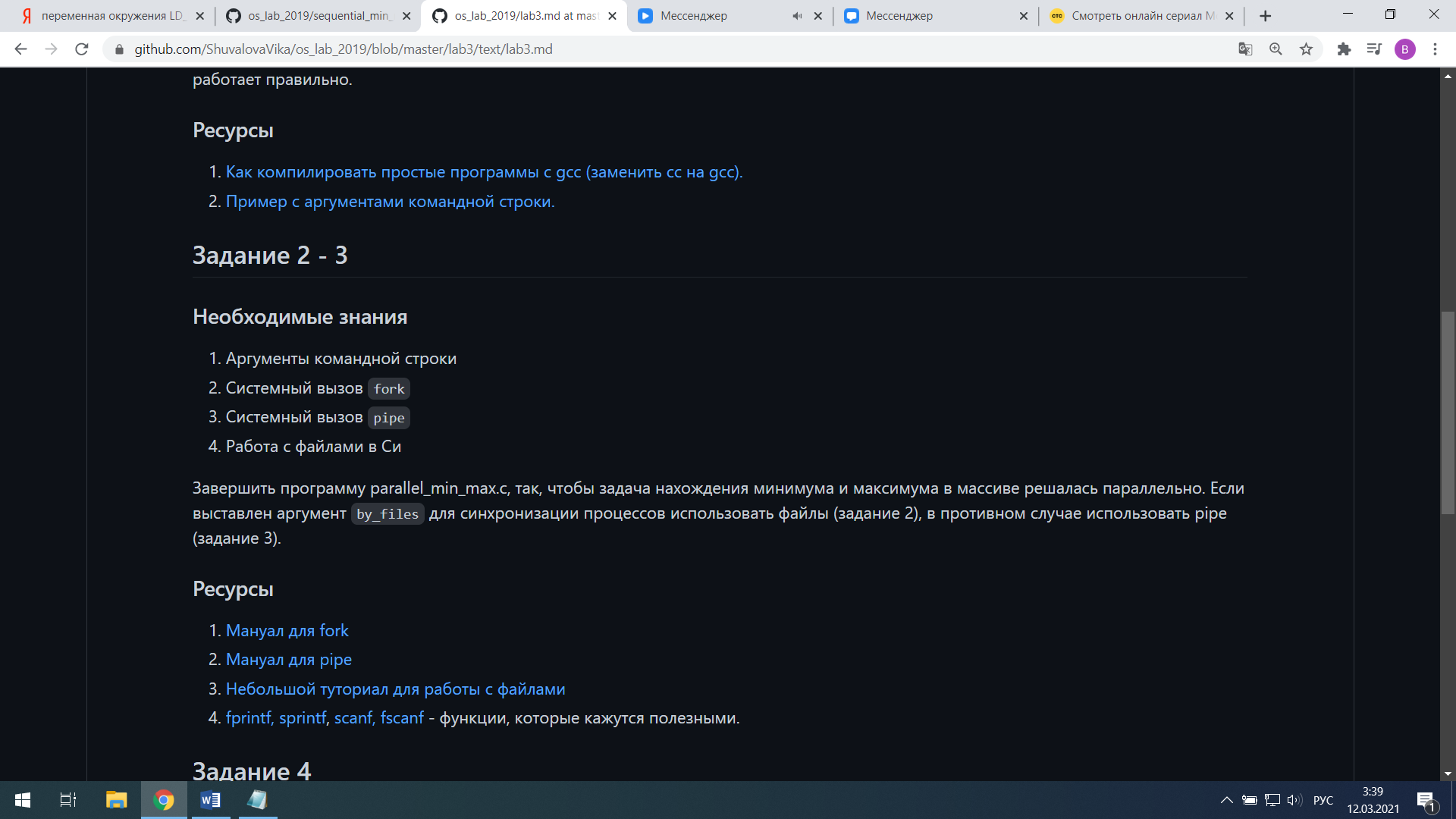
|  |
| --- |
| #include <stdio.h> |
|  | #include <stdlib.h> |
|  |  |
|  | #include "find\_min\_max.h" |
|  | #include "utils.h" |
|  |  |
|  | int main(int argc, char \*\*argv) { |
|  | if (argc != 3) { |
|  | printf("Usage: %s seed arraysize\n", argv[0]); |
|  | return 1; |
|  | } |
|  |  |
|  | int seed = atoi(argv[1]); |
|  | if (seed <= 0) { |
|  | printf("seed is a positive number\n"); |
|  | return 1; |
|  | } |
|  |  |
|  | int array\_size = atoi(argv[2]); |
|  | if (array\_size <= 0) { |
|  | printf("array\_size is a positive number\n"); |
|  | return 1; |
|  | } |
|  |  |
|  | int \*array = malloc(array\_size \* sizeof(int)); |
|  | GenerateArray(array, array\_size, seed); |
|  | struct MinMax min\_max = GetMinMax(array, 0, array\_size); |
|  | free(array); |
|  |  |
|  | printf("min: %d\n", min\_max.min); |
|  | printf("max: %d\n", min\_max.max); |
|  |  |
|  | return 0; |
|  | } |

(atoi-функция, которая превращает строку в интовую;

Seed-переменная для рандомизации)

**Задание 2-3.**

Завершить программу parallel\_min\_max.c, так, чтобы задача нахождения минимума и максимума в массиве решалась параллельно. Если выставлен аргумент by\_files для синхронизации процессов использовать файлы (задание 2), в противном случае использовать pipe (задание 3).



2)fork() — системный вызов, создающий новый процесс (потомок), который является практически полной копией процесса-родителя, выполняющего этот вызов.

3) Наиболее простым способом для передачи информации с помощью потоковой модели между различными процессами или даже внутри одного процесса в операционной системе Linux является pipe (канал, труба, конвейер).

Важное отличие pip’а от файла заключается в том, что прочитанная информация немедленно удаляется из него и не может быть прочитана повторно.

Pipe можно представить себе в виде трубы ограниченной емкости, расположенной внутри адресного пространства операционной системы, доступ к входному и выходному отверстию которой осуществляется с помощью системных вызовов. В действительности pipe представляет собой область памяти, недоступную пользовательским процессам напрямую, зачастую организованную в виде кольцевого буфера (хотя существуют и другие виды организации). По буферу при операциях чтения и записи перемещаются два указателя, соответствующие входному и выходному потокам. При этом выходной указатель никогда не может перегнать входной и наоборот. Для создания нового экземпляра такого кольцевого буфера внутри операционной системы используется системный вызов pipe().

|  |
| --- |
| Системный вызов pipe  Прототип системного вызова  #include <unistd.h>  int pipe(int \*fd);  Описание системного вызова  Системный вызов pipe предназначен для создания pip'а внутри операционной системы.  Параметр fd является указателем на массив из двух целых переменных. При нормальном завершении вызова в первый элемент массива – fd[0] – будет занесен файловый дескриптор, соответствующий выходному потоку данных pip’а и позволяющий выполнять только операцию чтения, а во второй элемент массива – fd[1] – будет занесен файловый дескриптор, соответствующий входному потоку данных и позволяющий выполнять только операцию записи. |

программа parallel\_min\_max.c,

(см.Гитхаб)

**Проверим(скомпилируем)**

~/lab3/src$ gcc parallel\_min\_max.c find\_min\_max.c utils.c -o parallel\_min\_max

1)~$ ./lab3/src/parallel\_min\_max --seed 1 --array\_size 100000 --pnum 2 // без использования файлов

Min: 3722

Max: 2147469841

Elapsed time: 0.513000ms

2)~$ ./lab3/src/parallel\_min\_max --seed 1 --array\_size 100000 --pnum 2 --by\_files // с файлом

Min: 3722

Max: 2147469841

Elapsed time: 1.026000ms

~$ open fake\_pipe.txt

**Задание 4.**

Используя makefile, собрать получившиеся решения. Добавьте target all, отвечающий за сборку всех программ.

(См.Гитхаб –makefile)

Makefile нужен для быстрой сборки программы

**Получаем**

~/lab3/src$ make

gcc -o utils.o -c utils.c -I.

gcc -o find\_min\_max.o -c find\_min\_max.c -I.

gcc -o sequential\_min\_max find\_min\_max.o utils.o sequential\_min\_max.c -I.

gcc -o parallel\_min\_max utils.o find\_min\_max.o parallel\_min\_max.c -I.

~/lab3/src$ ls .

find\_min\_max.c find\_min\_max.o parallel\_min\_max sequential\_min\_max utils.c utils.o

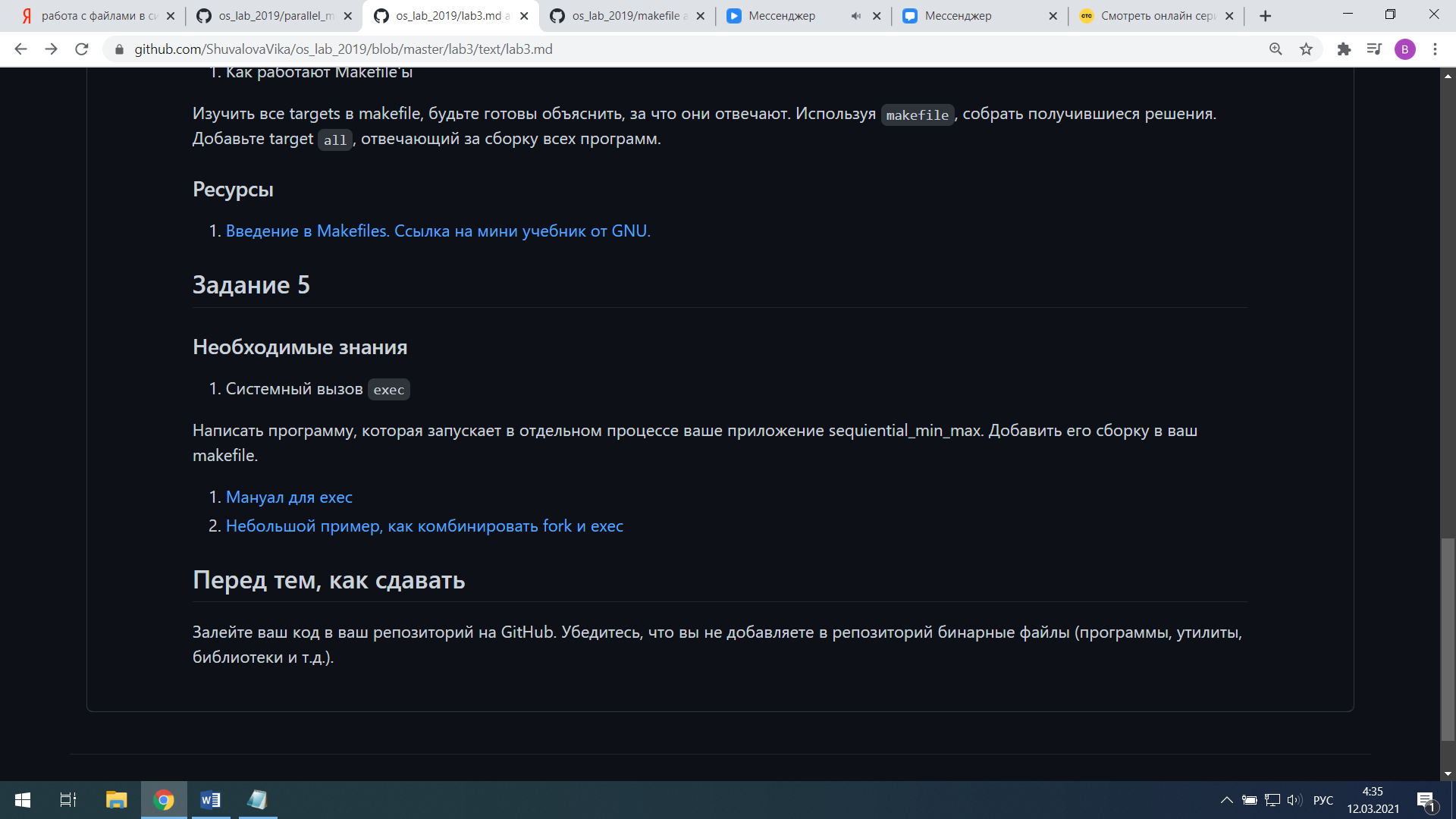
find\_min\_max.h makefile parallel\_min\_max.c sequential\_min\_max.c utils.h

~/lab3/src$ ./sequential\_min\_max 10 10

min: 54404747

max: 1753820418

**Задание 5.**



Команда **exec** в Linux используется для выполнения команды из самого bash. Эта команда не создает новый процесс, она просто заменяет bash командой, которая должна быть выполнена. Если команда exec успешна, она не возвращается к вызывающему процессу.

**Программа**

|  |
| --- |
| #include <unistd.h> |
|  |  |
|  | int main(int argc, char\* argv[]) { |
|  |  |
|  | execv("sequential\_min\_max", argv); |
|  | } |

Функции execv() предоставляет новой программе список аргументов в виде массива указателей на строки, заканчивающиеся null. Первый аргумент, по соглашению, должен указать на имя, ассоциированное с файлом, который необходимо запустить. Массив указателей должен заканчиваться указателем null.

**Добавили в makefile**

|  |
| --- |
| task\_5 : |
|  | $(CC) -o task\_5 task\_5.c $(CFLAGS) |
|  |  |

~/lab3/src$ touch task\_5.c

~/lab3/src$ open task\_5.c

~/lab3/src$ make -B

gcc -o utils.o -c utils.c -I.

gcc -o find\_min\_max.o -c find\_min\_max.c -I.

gcc -o sequential\_min\_max find\_min\_max.o utils.o sequential\_min\_max.c -I.

gcc -o parallel\_min\_max utils.o find\_min\_max.o parallel\_min\_max.c -I.

gcc -o task\_5 task\_5.c -I.

~/lab3/src$ ls .

find\_min\_max.c find\_min\_max.o parallel\_min\_max sequential\_min\_max task\_5 utils.c utils.o

find\_min\_max.h makefile parallel\_min\_max.c sequential\_min\_max.c task\_5.c utils.h

~/lab3/src$ ./task\_5 10 10

min: 54404747

max: 1753820418