# 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.

## Необходимые знания

1. Аргументы командной строки
2. Сборка с помощью gcc (clang)

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

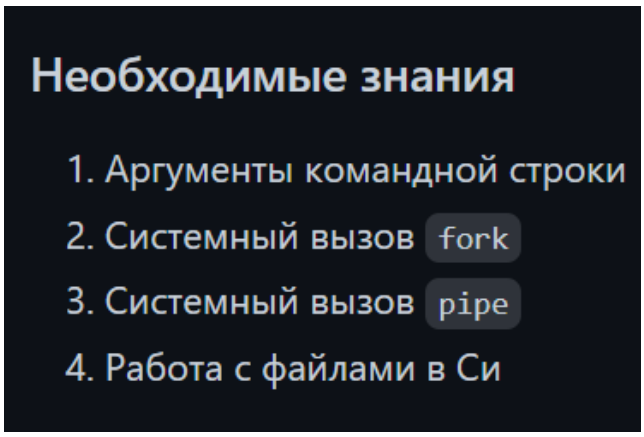
```
#include
<stdio.h>
            #include <stdlib.h>

            #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’a 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'a 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’a 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.**

**Необходимые знания**

1. Системный вызов `exec`

Написать программу, которая запускает в отдельном процессе ваше приложение sequiential_min_max. Добавить его сборку в ваш makefile.

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