

**Федеральное государственное автономное образовательное учреждение
высшего образования**

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Московский институт электроники и математики им. Тихонова

Департамент электронной инженерии

ОТЧЕТ

О ПРАКТИЧЕСКОЙ РАБОТЕ №8

по дисциплине «Системное программирование»

«Потоки и синхронизация»

Вариант 9

Студент гр. БИБ201

Шадрунов Алексей

Дата выполнения: 22 марта 2023 г.

Преподаватель:

Морозов В. И.

«___» _____ 2023 г.

Москва, 2023

Содержание

1	Цель работы	3
2	Ход работы	3
2.1	Описание алгоритма	3
2.1.1	Основной процесс	3
2.2	Компилятор gcc	4
2.3	Компилятор MSVC	6
2.4	Python	8
3	Выводы о проделанной работе	9
	Приложение А. Код linux/main.c	10
	Приложение Б. Код win/main.c	13
	Приложение Д. Код ru/main.py	16

1 Цель работы

В файле записан ряд целых чисел, разделённых пробелом. Программа должна считать имя файла из первого аргумента командной строки и рассчитать сумму квадратов записанных в файл чисел. Для расчёта суммы квадратов программа должна создать N потоков (N передаётся вторым аргументом командной строки) и передать каждому из них часть полученных чисел. Каждый из потоков должен рассчитать сумму квадратов переданных ему чисел и вернуть её родительскому. Родительский процесс должен просуммировать полученные от дочерних числа и вывести на консоль итоговую сумму. Если исходный файл не существует, или в нём записано менее двух чисел, следует вывести соответствующее сообщение для пользователя и завершить работу программы.

2 Ход работы

2.1 Описание алгоритма

2.1.1 Основной процесс

Работа программ (main) состоит из нескольких этапов:

1. Проверить количество входных аргументов. Если их не два, вывести подсказку.
2. Сохранить аргументы в переменные `input_path` (путь к файлу) и N (число байт).
3. Вывести полученные параметры в консоль (функция `printf`).
4. Открыть файл на чтение. Если открыть не удалось (не существует файла), вывести сообщение об ошибке.
5. Считать файл слово за словом, считая количество входных чисел M (для чтения используем небольшой буфер и функцию `fscanf`). Полученное количество вывести на экран.
6. Если количество меньше 2, вывести сообщение об ошибке.
7. Если количество потоков N больше $M / 2$, уменьшить N до $M / 2$.
8. Вычислить количество данных для каждого потока (M/N для всех, кроме последнего, и $n + M \% N$ для последнего), вывести на экран.
9. Для каждого потока в цикле:
 - Записать в массив номер потока, количество чисел и числа
 - Запустить новый поток, передать массив в качестве аргумента.
 - Поток рассчитывает сумму квадратов, блокирует мьютекс и обновляет глобальную переменную.

10. Закрывать входной файл.
11. Распечатать результат — глобальную переменную.

Программа использует C Standard Library для работы с файлами и OS API для работы с потоками, поэтому эта часть отличается для Linux и Windows.

2.2 Компилятор gcc

При использовании компилятора gcc на Linux мы пользуемся библиотекой `pthread.h`. Исходный код программы для Linux приведён в приложении.

Для сборки основной программы используется команда: `gcc main.c -o main`. Продемонстрируем работу программы (рисунки 1-4).

```
alex@alex-nb ~/D/y/h/9/linux (master)> ./main input.txt

Usage: main <path> <N>
      path - file to read
      N - thread number
alex@alex-nb ~/D/y/h/9/linux (master) [1]> █
```

Рисунок 1 – Неверные аргументы

```
alex@alex-nb ~/D/y/h/9/linux (master) [1]> ./main input2.txt 3
Path to file: input2.txt
Thread number: 3

Error: No such file or directory

Usage: main <path> <N>
      path - file to read
      N - thread number
alex@alex-nb ~/D/y/h/9/linux (master) [1]> █
```

Рисунок 2 – Входной файл не существует

```
alex@alex-nb ~/D/y/h/9/linux (master) [1]> ./main empty 3
Path to file: empty
Thread number: 3
Input size: 0
Too few numbers (M must be greater than 2)

Usage: main <path> <N>
      path - file to read
      N - thread number
alex@alex-nb ~/D/y/h/9/linux (master) [1]> █
```

Рисунок 3 – Входной файл пустой

```

alex@alex-nb ~/D/y/h/9/linux (master) [1]> ./main input.txt 3
Path to file: input.txt
Thread number: 3
Input size: 20
Each thread gets: 6 numbers
Last thread gets: 8 numbers
[Thread 0] started
[Thread 0] numberCount: 6
[Thread 0] calculated result: 91
[Thread 0] new globalResult: 91
[Thread 0] finished
[Thread 1] started
[Thread 1] numberCount: 6
[Thread 1] calculated result: 559
[Thread 1] new globalResult: 650
[Thread 1] finished
[Thread 2] started
[Thread 2] numberCount: 8
[Thread 2] calculated result: 2220
[Thread 2] new globalResult: 2870
[Thread 2] finished

Final Result: 2870
alex@alex-nb ~/D/y/h/9/linux (master)>

```

Рисунок 4 – Работа программы

Потоки отображаются в htop (рисунок 5).

156272	alex	133830	20	0	229M	8624	6144	S	0.0	0.1	0:00.64					/usr/bin/fish
156919	alex	156272	20	0	27076	860	772	S	0.0	0.0	0:00.00					└─ ./main input.txt 3
156920	alex	156272	20	0	27076	860	772	S	0.0	0.0	0:00.00					└─ ./main input.txt 3
156921	alex	156272	20	0	27076	860	772	S	0.0	0.0	0:00.00					└─ ./main input.txt 3
156922	alex	156272	20	0	27076	860	772	S	0.0	0.0	0:00.00					└─ ./main input.txt 3
156273	alex	133777	20	0	37.3G	34612	17072	S	0.0	0.2	0:00.00					└─ /opt/visual-studio-code/cod

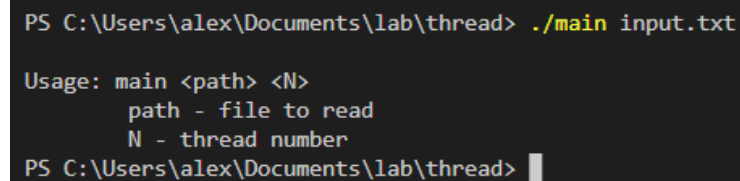
Рисунок 5 – Потоки

2.3 Компилятор MSVC

Чтобы запустить эту программу на Windows, нужно заменить системные вызовы на WinAPI. Для этого подключаем файл `Windows.h` и используем функции `CreateThread`, `ResumeThread`, `WaitForSingleObject`. Исходный код программы для Windows приведён в приложении.

Для компиляции и сборки программы используем Developer Command Prompt и команду: `cl main.c /link /out:main.exe`

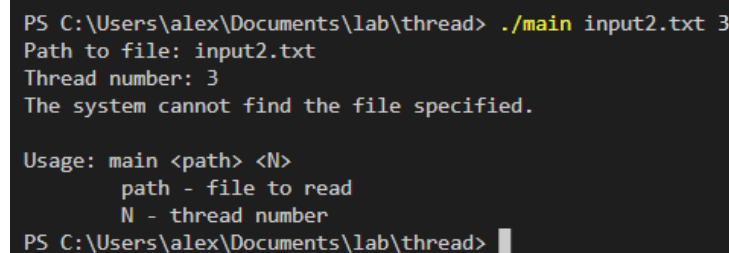
Далее продемонстрируем работу программы (рисунки 6-9).



```
PS C:\Users\alex\Documents\lab\thread> ./main input.txt

Usage: main <path> <N>
      path - file to read
      N - thread number
PS C:\Users\alex\Documents\lab\thread> 
```

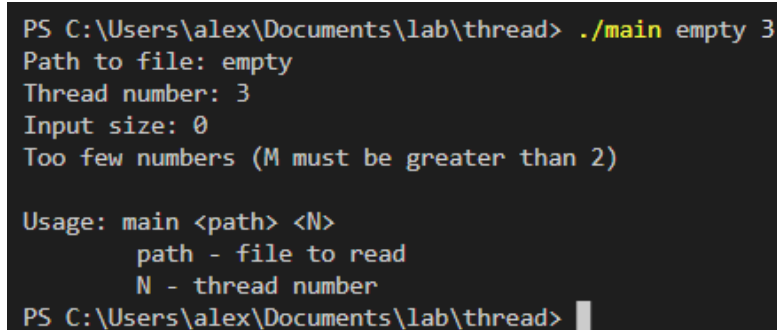
Рисунок 6 – Неверные аргументы



```
PS C:\Users\alex\Documents\lab\thread> ./main input2.txt 3
Path to file: input2.txt
Thread number: 3
The system cannot find the file specified.

Usage: main <path> <N>
      path - file to read
      N - thread number
PS C:\Users\alex\Documents\lab\thread> 
```

Рисунок 7 – Входной файл не существует



```
PS C:\Users\alex\Documents\lab\thread> ./main empty 3
Path to file: empty
Thread number: 3
Input size: 0
Too few numbers (M must be greater than 2)

Usage: main <path> <N>
      path - file to read
      N - thread number
PS C:\Users\alex\Documents\lab\thread> 
```

Рисунок 8 – Входной файл пустой

```
PS C:\Users\alex\Documents\lab\thread> ./main input.txt 3
Path to file: input.txt
Thread number: 3
Input size: 20
Each thread gets: 6 numbers
Last thread gets: 8 numbers
[Thread 1] started
[Thread 1] numberCount: 6
[Thread 1] calculated result: 559
[Thread 1] new globalResult: 559
[Thread 1] finished
[Thread 0] started
[Thread 0] numberCount: 6
[Thread 0] calculated result: 91
[Thread 0] new globalResult: 650
[Thread 0] finished
[Thread 2] started
[Thread 2] numberCount: 8
[Thread 2] calculated result: 2220
[Thread 2] new globalResult: 2870
[Thread 2] finished

Final Result: 2870
PS C:\Users\alex\Documents\lab\thread> 
```

Рисунок 9 – Работа программы

Потоки отображаются в Process Hacker (рисунок 10).

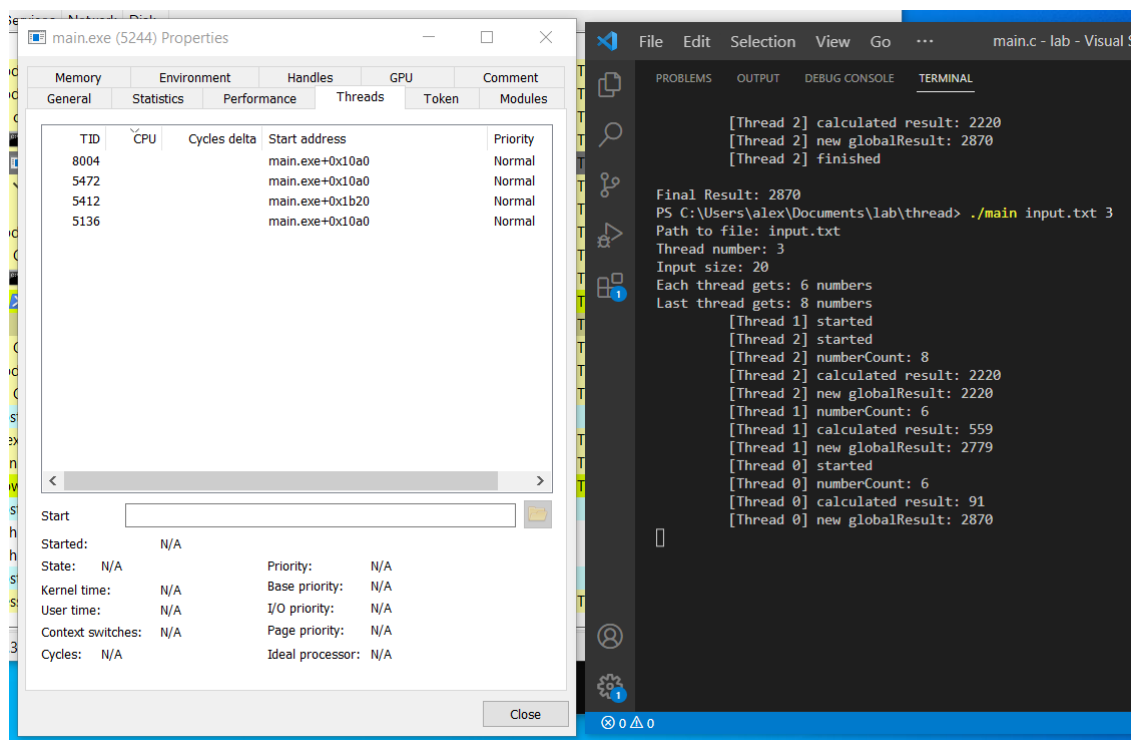


Рисунок 10 – Потоки

2.4 Python

Аналогичный функционал присутствует в языке Python. Код программы на Python приведён в приложении.

Компиляция не предусмотрена и запуск осуществляется командой: `python main.py input.txt 3`.

Продemonстрируем работу программы (рисунки 11-13).

```
(.venv) alex@alex-nb ~/D/y/h/9/py (master)> python main.py input.txt

Usage: main <path> <N>
      path - file to read
      N - fork number
(.venv) alex@alex-nb ~/D/y/h/9/py (master)> █
```

Рисунок 11 – Неверные аргументы

```
(.venv) alex@alex-nb ~/D/y/h/9/py (master)> python main.py input2.txt 3
Path to file: input2.txt
Thread number: 3
Traceback (most recent call last):
  File "/home/alex/Documents/year-3-c/hw/9_thread/py/main.py", line 108, in <module>
    main()
  File "/home/alex/Documents/year-3-c/hw/9_thread/py/main.py", line 65, in main
    with open(input_path, "r") as f:
FileNotFoundError: [Errno 2] No such file or directory: 'input2.txt'
(.venv) alex@alex-nb ~/D/y/h/9/py (master) [0|1]> █
```

Рисунок 12 – Входной файл не существует

```
(.venv) alex@alex-nb ~/D/y/h/9/py (master) [0|1]> python main.py input.txt 3
Path to file: input.txt
Thread number: 3
Input size: 20
Each thread gets: 6 numbers
Last thread gets: 8 numbers
[Thread 0] started
[Thread 0] numberCount: 6
[Thread 1] started
[Thread 0] calculated result: 91
[Thread 1] numberCount: 6
[Thread 0] new globalResult: 91
[Thread 2] started
[Thread 2] numberCount: 8
[Thread 2] calculated result: 2220
[Thread 0] finished
[Thread 2] new globalResult: 2311
[Thread 2] finished
[Thread 1] calculated result: 559
[Thread 1] new globalResult: 2870
[Thread 1] finished

Final Result: 2870
(.venv) alex@alex-nb ~/D/y/h/9/py (master)> █
```

Рисунок 13 – Работа программы

Потоки отображаются в htop (рисунки 12-13).

133908	alex	133830	20	0	234M	9976	3932	S	0.0	0.1	0:01.13				/usr/bin/fish
158355	alex	133908	20	0	229M	8348	4896	S	0.0	0.1	0:00.00				python main.py input.txt 3
158356	alex	133908	20	0	229M	8348	4896	S	0.0	0.1	0:00.00				python main.py input.txt 3
158357	alex	133908	20	0	229M	8348	4896	S	0.0	0.1	0:00.00				python main.py input.txt 3
158358	alex	133908	20	0	229M	8348	4896	S	0.0	0.1	0:00.00				python main.py input.txt 3
133909	alex	133777	20	0	37.3G	36464	17136	S	0.0	0.2	0:00.00				/opt/visual-studio-code/code --ms
156272	alex	133830	20	0	229M	8596	6068	S	0.0	0.1	0:00.68				/usr/bin/fish

Рисунок 14 – Потоки

3 Выводы о проделанной работе

В рамках данной работы я написал программу, которая считывает имя файла из первого аргумента командной строки и рассчитывает сумму квадратов записанных в файл чисел. Для расчёта суммы квадратов программа должна создать N потоков (N передаётся вторым аргументом командной строки) и передать каждому из них часть полученных чисел. Каждый из потоков должен рассчитать сумму квадратов переданных ему чисел и вернуть её родительскому. Родительский поток должен просуммировать полученные от дочерних числа и вывести на консоль итоговую сумму. Если исходный файл не существует, или в нём записано менее 2 чисел, следует вывести соответствующее сообщение для пользователя и завершить работу программы. Скомпилировал программу с помощью компиляторов gcc и MSVC, а также реализовал аналогичный функционал на языке Python.

Приложение А. Код linux/main.c

```
1  /*В
2  файле записан ряд целых чисел, разделённых пробелом. Программа должна
   считать имя файла из первого аргумента
3  командной строки и рассчитать сумму квадратов записанных в файл чисел.
   Для расчёта суммы квадратов программа
4  должна создать N дочерних потоков (N передаётся вторым аргументом
   командной строки) и передать каждому
5  из них часть полученных чисел. Каждый из дочерних потоков должен
   рассчитать сумму квадратов переданных
6  ему чисел и вернуть её родительскому. Родительский поток должен
   просуммировать полученные от дочерних
7  числа и вывести на консоль итоговую сумму. Если исходный файл не
   существует, или в нём записано менее
8  2 чисел, следует вывести соответствующее сообщение для пользователя и
   завершить работу программы.
9  */
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <string.h>
14 #include <fcntl.h>
15 #include <unistd.h>
16 #include <wait.h>
17 #include <pthread.h>
18
19 static pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
20 int globalResult = 0;
21
22 /**
23  * Prints help message to console
24  */
25 int print_help()
26 {
27     printf("\nUsage: main <path> <N> \n");
28     printf("\tpath - file to read \n");
29     printf("\tN - thread number \n");
30     return 1;
31 }
32
33 /**
34  * Prints error message and help message to console
35  * and closes the program
36  */
37 int catch_error()
38 {
39     perror("\nError");
40     print_help();
41 }
42
43 void *calc(void *arg)
44 {
45     int total = 0;
46     int *args = (int *)arg;
47     int threadNumber = args[0];
48     int numberCount = args[1];
49     printf("\t [Thread %d] started \n", threadNumber);
50     printf("\t [Thread %d] numberCount: %d \n", threadNumber, numberCount);
51
52     for (int i = 2; i < numberCount + 2; i++)
53     {
54         // calc sum of squares
55         int number = args[i];
56         total += number * number;
57     }
58
59     printf("\t [Thread %d] calculated result: %d \n", threadNumber, total);
60
61     // update global result
```

```

62     pthread_mutex_lock(&mutex);
63     globalResult += total;
64     printf("\t [Thread %d] new globalResult: %d \n", threadNumber,
globalResult);
65     pthread_mutex_unlock(&mutex);
66
67     // sleep(500);
68     printf("\t [Thread %d] finished \n", threadNumber);
69     pthread_exit(0);
70 }
71
72 int main(int argc, char **argv)
73 {
74     // check number of arguments
75     if (argc != 3)
76         return print_help();
77
78     // get amount of numbers and file path
79     char *input_path = argv[1];
80     int N = atoi(argv[2]);
81
82     // print input values
83     printf("Path to file: %s \n", input_path);
84     printf("Thread number: %d \n", N);
85
86     // open file to read
87     FILE *input_file = fopen(input_path, "r");
88     if (!input_file)
89         return catch_error();
90
91     // get number of input digits M
92     char t[12];
93     int M = 0; // input size
94     while (fscanf(input_file, " %12s", t) == 1)
95         M++;
96
97     printf("Input size: %d \n", M);
98
99     // if too few numbers
100    if (M < 2)
101    {
102        printf("Too few numbers (M must be greater than 2) \n");
103        return print_help();
104    }
105
106    // if too many threads
107    if (N > M / 2)
108    {
109        N = M / 2;
110        printf("Too many threads. New thread number: %d \n", N);
111    }
112
113    // calculate division between threads
114    int n = M / N;
115    int n_last = n + M % N;
116
117    printf("Each thread gets: %d numbers \n", n);
118    printf("Last thread gets: %d numbers \n", n_last);
119
120    // move to file start
121    if (fseek(input_file, 0, SEEK_SET))
122        return catch_error();
123
124    // init thread arrays
125    pthread_t *threadArray = malloc(sizeof(pthread_t) * N);
126    int *argArray[N];
127    for (int i = 0; i < N - 1; i++)
128        argArray[i] = (int *)malloc(sizeof(int) * (n + 2));
129    argArray[N - 1] = (int *)malloc(sizeof(int) * (n_last + 2));

```

```

130
131 // create array for each process and thread
132 for (int i = 0; i < N; i++)
133 {
134     int n_effective = (i == N - 1) ? n_last : n; // calc length of
output array
135     argArray[i][0] = i; // save thread number
136     argArray[i][1] = n_effective; // save n_effective
137     for (int j = 2; j < n_effective + 2; j++)
138     {
139         if (!fscanf(input_file, "%12s", t))
140             return catch_error();
141         argArray[i][j] = atoi(t); // write number to array
142     }
143
144     // create thread
145     pthread_t thread;
146     pthread_create(&thread, NULL, calc, argArray[i]);
147     threadArray[i] = thread;
148 }
149
150 for (int i = 0; i < N; i++)
151     pthread_join(threadArray[i], NULL);
152
153 // release input file
154 if (fclose(input_file) != 0)
155     return catch_error();
156
157 // release memory
158 free(threadArray);
159 for (int i = 0; i < N; i++)
160     free(argArray[i]);
161
162 printf("\nFinal Result: %d \n", globalResult);
163 return 0;
164 }

```

Приложение Б. Код win/main.c

```
1  /*В
2  файле записан ряд целых чисел, разделённых пробелом. Программа должна
   считать имя файла из первого аргумента
3  командной строки и рассчитать сумму квадратов записанных в файл чисел.
   Для расчёта суммы квадратов программа
4  должна создать N дочерних потоков (N передаётся вторым аргументом
   командной строки) и передать каждому
5  из них часть полученных чисел. Каждый из дочерних потоков должен
   рассчитать сумму квадратов переданных
6  ему чисел и вернуть её родителскому. Родительский поток должен
   просуммировать полученные от дочерних
7  числа и вывести на консоль итоговую сумму. Если исходный файл не
   существует, или в нём записано менее
8  2 чисел, следует вывести соответствующее сообщение для пользователя и
   завершить работу программы.
9  */
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <string.h>
14 #include <fcntl.h>
15 #include <Windows.h>
16
17 CRITICAL_SECTION cs;
18 int globalResult = 0;
19
20 /**
21  * Prints help message to console
22  */
23 int print_help()
24 {
25     printf("\nUsage: main <path> <N> \n");
26     printf("\tpath - file to read \n");
27     printf("\tN - thread number \n");
28     return 1;
29 }
30
31 /**
32  * Prints error message and help message to console
33  * and closes the program
34  */
35 int catch_error()
36 {
37     LPSTR message;
38     DWORD dwMessageLen = FormatMessage(
39         FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_ALLOCATE_BUFFER,
40         NULL, GetLastError(), MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
41         (LPSTR)&message, 0, NULL);
42     printf(message);
43     print_help();
44     return 1;
45 }
46
47 DWORD WINAPI calc(void *arg)
48 {
49     int total = 0;
50     int *args = (int *)arg;
51     int threadNumber = args[0];
52     int numberCount = args[1];
53     printf("\t [Thread %d] started \n", threadNumber);
54     printf("\t [Thread %d] numberCount: %d \n", threadNumber, numberCount);
55
56     for (int i = 2; i < numberCount + 2; i++)
57     {
58         // calc sum of squares
59         int number = args[i];
60         total += number * number;
61     }
```

```

62
63     printf("\t [Thread %d] calculated result: %d \n", threadNumber, total);
64
65     // update global result
66     EnterCriticalSection(&cs);
67     globalResult += total;
68     printf("\t [Thread %d] new globalResult: %d \n", threadNumber,
globalResult);
69     LeaveCriticalSection(&cs);
70
71     // Sleep(50000);
72     printf("\t [Thread %d] finished \n", threadNumber);
73     ExitThread(0);
74 }
75
76 int main(int argc, char **argv)
77 {
78     // check number of arguments
79     if (argc != 3)
80         return print_help();
81
82     // get amount of numbers and file path
83     char *input_path = argv[1];
84     int N = atoi(argv[2]);
85
86     // print input values
87     printf("Path to file: %s \n", input_path);
88     printf("Thread number: %d \n", N);
89
90     // open file to read
91     FILE *input_file = fopen(input_path, "r");
92     if (!input_file)
93         return catch_error();
94
95     // get number of input digits M
96     char t[12];
97     int M = 0; // input size
98     while (fscanf(input_file, " %12s", t) == 1)
99         M++;
100
101     printf("Input size: %d \n", M);
102
103     // if too few numbers
104     if (M < 2)
105     {
106         printf("Too few numbers (M must be greater than 2) \n");
107         return print_help();
108     }
109
110     // if too many threads
111     if (N > M / 2)
112     {
113         N = M / 2;
114         printf("Too many threads. New thread number: %d \n", N);
115     }
116
117     // calculate division between threads
118     int n = M / N;
119     int n_last = n + M % N;
120
121     printf("Each thread gets: %d numbers \n", n);
122     printf("Last thread gets: %d numbers \n", n_last);
123
124     // move to file start
125     if (fseek(input_file, 0, SEEK_SET))
126         return catch_error();
127
128     // init thread arrays
129     InitializeCriticalSection(&cs);

```

```

130 HANDLE * threadArray = malloc(sizeof(HANDLE) * N);
131 DWORD *threadIds = malloc(sizeof(HANDLE) * N);
132 int** argArray = (int**)malloc(N * sizeof(int*));
133 // int *argArray[N];
134 for (int i = 0; i < N - 1; i++)
135     argArray[i] = (int *)malloc(sizeof(int) * (n + 2));
136 argArray[N - 1] = (int *)malloc(sizeof(int) * (n_last + 2));
137
138 // create array for each process and thread
139 for (int i = 0; i < N; i++)
140 {
141     int n_effective = (i == N - 1) ? n_last : n; // calc length of
output array
142     argArray[i][0] = i; // save thread number
143     argArray[i][1] = n_effective; // save n_effective
144     for (int j = 2; j < n_effective + 2; j++)
145     {
146         if (!fscanf(input_file, " %12s", t))
147             return catch_error();
148         argArray[i][j] = atoi(t); // write number to array
149     }
150
151     // create thread
152     HANDLE thread = CreateThread(NULL, 0, calc, argArray[i],
CREATE_SUSPENDED, &threadIds[i]);
153     if (thread == NULL)
154     {
155         printf("Error: %d\n", GetLastError());
156         exit(5);
157     }
158     if (ResumeThread(thread) == -1)
159     {
160         printf("Error: %d\n", GetLastError());
161         exit(5);
162     }
163     threadArray[i] = thread;
164 }
165
166 for (int i = 0; i < N; i++)
167     WaitForSingleObject(threadArray[i], INFINITE);
168
169 // release input file
170 if (fclose(input_file) != 0)
171     return catch_error();
172
173 // release memory
174 free(threadArray);
175 for (int i = 0; i < N; i++)
176     free(argArray[i]);
177
178 printf("\nFinal Result: %d \n", globalResult);
179 return 0;
180 }

```

Приложение В. Код ru/main.py

```
1  """В
2  файле записан ряд целых чисел, разделённых пробелом. Программа должна
3  считать имя файла из первого аргумента
4  командной строки и рассчитать сумму квадратов записанных в файл чисел.
5  Для расчёта суммы квадратов программа
6  должна создать N дочерних процессов (N передаётся вторым аргументом
7  командной строки) и передать каждому
8  из них часть полученных чисел. Каждый из дочерних процессов должен
9  рассчитать сумму квадратов переданных
10 ему чисел и вернуть её родительскому. Родительский процесс должен
11 просуммировать полученные от дочерних
12 числа и вывести на консоль итоговую сумму. Если исходный файл не
13 существует, или в нём записано менее
14 2 чисел, следует вывести соответствующее сообщение для пользователя и
15 завершить работу программы.
16
17 """
18
19 import sys
20 import time
21 import threading
22
23 mutex = threading.Lock()
24 globalResult = 0
25
26 def print_help():
27     """Prints help message to console"""
28     print("\nUsage: main <path> <N>")
29     print("\tpath - file to read")
30     print("\tN - fork number")
31     return 1
32
33 def calc(args):
34     global globalResult
35     total = 0
36     threadNumber = args[0]
37     numberCount = args[1]
38     print(f"\t [Thread {threadNumber}] started")
39     print(f"\t [Thread {threadNumber}] numberCount: {numberCount}")
40
41     for i in range(2, numberCount + 2):
42         # calc sum of squares
43         total += args[i] ** 2
44
45     print(f"\t [Thread {threadNumber}] calculated result: {total}")
46
47     # update global result
48     mutex.acquire()
49     globalResult += total
50     print(f"\t [Thread {threadNumber}] new globalResult: {globalResult}")
51     mutex.release()
52     # time.sleep(500)
53     print(f"\t [Thread {threadNumber}] finished")
54
55 def main():
56     # check number of arguments
57     if len(sys.argv) != 3:
58         return print_help()
59
60     # get amount of numbers and file path
61     input_path: str = sys.argv[1]
62     N: int = int(sys.argv[2])
63
64     # print input values
```



```

62     print("Path to file: %s" % input_path)
63     print("Thread number: %d" % N)
64
65     # open file to read
66     with open(input_path, "r") as f:
67         input_file = f.read().strip().split(" ")
68         input_file = list(map(int, input_file))
69
70     M = len(input_file)
71     print("Input size: %d" % M)
72
73     # if too few numbers
74     if M < 2:
75         print("Too few numbers (M must be greater than 2)")
76         return print_help()
77
78     # if too many forks
79     if N > M // 2:
80         N = M // 2
81         print("Too many threads. New fork number: %d" % N)
82
83     # calculate division between processes
84     n: int = M // N
85     n_last: int = n + M % N
86
87     print("Each thread gets: %d numbers" % n)
88     print("Last thread gets: %d numbers" % n_last)
89
90     # create files for each process and fork
91     threadArray = []
92     for i in range(N):
93         # write numbers to output file
94         n_effective: int = n_last if i == N - 1 else n
95         subset = input_file[n * i : n * i + n_effective]
96         subset = [i, len(subset)] + subset
97
98         # start thread
99         threadArray.append(threading.Thread(target=calc, args=(subset,)))
100        threadArray[i].start()
101
102    for i in range(N):
103        threadArray[i].join()
104
105    print("\nFinal Result: %d" % globalResult)
106    return 0
107
108
109 main()

```