

**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 №7

по дисциплине «Системное программирование»

«Процессы»

Вариант 9

Студент гр. БИБ201

Шадрунов Алексей

Дата выполнения: 26 февраля 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
2.1.2	Вспомогательный процесс	4
2.2	Компилятор gcc	5
2.3	Компилятор MSVC	8
2.4	Python	11
3	Выводы о проделанной работе	13
	Приложение А. Код linux/main.c	14
	Приложение Б. Код linux/calc.c	17
	Приложение В. Код win/main.c	19
	Приложение Г. Код win/calc.c	22
	Приложение Д. Код py/main.py	24
	Приложение Е. Код py/calc.py	26

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 дочерних процессов (N передаётся вторым аргументом командной строки) и передать каждому из них часть полученных чисел. Каждый из дочерних процессов должен рассчитать сумму квадратов переданных ему чисел и вернуть её родителю. Родительский процесс 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.

2 Ход работы

2.1 Описание алгоритма

2.1.1 Основной процесс

Работа основной программы (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. Читать 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 подпроцессов N больше $M / 2$, уменьшить N до $M / 2$.
8. Вычислить количество данных для каждого подпроцесса (M/N для всех, кроме последнего, и $n + M \% N$ для последнего), вывести на экран.
9. Для каждого подпроцесса в цикле:
 - Составить имя выходного файла (куда записывается часть входных данных) как `"_d_input.txt"`, где `%d` — номер итерации цикла.
 - Открыть файл на запись.
 - Записать нужное количество элементов входных данных в файл и закрыть

файл.

- Создать копию текущего процесса (fork).
- Заменить подпроцесс на программу calc, в качестве аргументов передать файл с частью входных данных и его размер.

10. Закрывать входной файл. Приостановить программу для принудительного ожидания подпроцессов (демонстрация процессов-зомби).

11. Для каждого подпроцесса в цикле:

- Ожидать завершения (wait).
- Составить имя файла с результатом работы процесса ("`_%d_result.txt`", где %d — PID завершённого процесса).
- Открыть файл на чтение, считать строку, увеличить сумму на число в строке, закрыть файл.

12. Распечатать результат.

2.1.2 Вспомогательный процесс

Работа подпрограммы (calc) также состоит из нескольких этапов:

1. Распечатать PID подпроцесса.
2. Проверить количество входных аргументов. Если их не два, вывести подсказку.
3. Сохранить аргументы в переменные `input_path` (путь к файлу) и `n` (размер входной последовательности).
4. Вывести полученные параметры в консоль (функция `printf`; для удобства к выводу подпроцессов добавляется PID).
5. Открыть файл на чтение. Если открыть не удалось (не существует файла), вывести сообщение об ошибке.
6. В цикле считывать числа из файла, возводить в квадрат и прибавлять к сумме.
7. Закрывать входной файл.
8. Составить имя файла с результатом работы процесса ("`_%d_result.txt`", где %d — PID завершённого процесса).
9. Открыть файл на запись, записать результат, закрыть файл.
10. Распечатать результат.
11. Приостановить процесс на 5 секунд, затем завершить с кодом `EXIT_SUCCESS`.

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

2.2 Компилятор gcc

При использовании компилятора gcc на Linux мы пользуемся системными вызовами из файла `unistd.h` и `wait.h`: `fork` и `wait`. Они используются для работы с подпроцессами. Исходный код программы для Linux приведён в приложении.

Для сборки основной программы используется команда: `gcc main.c -o main`. Для сборки вспомогательной программы используется команда: `gcc calc.c -o calc`. Процесс сборки показан на рисунке 1.

```
alex@alex-nb ~/D/y/h/7/linux (master)> gcc main.c -o main
alex@alex-nb ~/D/y/h/7/linux (master)> gcc calc.c -o calc
alex@alex-nb ~/D/y/h/7/linux (master)> █
```

Рисунок 1 – Сборка файлов

Далее продемонстрируем работу программы (рисунки 2-5).

```
alex@alex-nb ~/D/y/h/7/linux (master) [1]> ./main

Usage: main <path> <N>
      path – file to read
      N – fork number
```

Рисунок 2 – Неверные аргументы

```
alex@alex-nb ~/D/y/h/7/linux (master) [1]> ls
calc*  calc.c  main*  main.c
alex@alex-nb ~/D/y/h/7/linux (master)> ./main input.txt 3
Path to file: input.txt
Fork number: 3

Error: No such file or directory

Usage: main <path> <N>
      path – file to read
      N – fork number
```

Рисунок 3 – Входной файл не существует

```
alex@alex-nb ~/D/y/h/7/linux (master) [1]> touch input.txt
alex@alex-nb ~/D/y/h/7/linux (master)> ./main input.txt 3
Path to file: input.txt
Fork number: 3
Input size: 0
Too few numbers (M must be greater than 2)

Usage: main <path> <N>
      path – file to read
      N – fork number
alex@alex-nb ~/D/y/h/7/linux (master) [1]> █
```

Рисунок 4 – Входной файл пустой

```
alex@alex-nb ~/D/y/h/7/linux (master)> cat input.txt
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
alex@alex-nb ~/D/y/h/7/linux (master)> ./main input.txt 3
Path to file: input.txt
Fork number: 3
Input size: 20
Each proc gets: 6 numbers
Last proc gets: 8 numbers
Forked proc 106740 with output_path _0_input.txt
Forked proc 106741 with output_path _1_input.txt
Forked proc 106742 with output_path _2_input.txt

[Child 106740]   New proc
[Child 106740]   Path to file with numbers: _0_input.txt
[Child 106740]   Input length: 6
[Child 106740]   Current number: 1           Square: 1           Sum: 1
[Child 106740]   Current number: 2           Square: 4           Sum: 5
[Child 106740]   Current number: 3           Square: 9           Sum: 14
[Child 106740]   Current number: 4           Square: 16          Sum: 30
[Child 106740]   Current number: 5           Square: 25          Sum: 55
[Child 106740]   Current number: 6           Square: 36          Sum: 91
[Child 106740]   Result: 91

[Child 106741]   New proc
[Child 106741]   Path to file with numbers: _1_input.txt
[Child 106741]   Input length: 6
[Child 106741]   Current number: 7           Square: 49          Sum: 49
[Child 106741]   Current number: 8           Square: 64          Sum: 113
[Child 106741]   Current number: 9           Square: 81          Sum: 194
[Child 106741]   Current number: 10          Square: 100         Sum: 294
[Child 106741]   Current number: 11          Square: 121         Sum: 415
[Child 106741]   Current number: 12          Square: 144         Sum: 559
[Child 106741]   Result: 559

[Child 106742]   New proc
[Child 106742]   Path to file with numbers: _2_input.txt
[Child 106742]   Input length: 8
[Child 106742]   Current number: 13          Square: 169         Sum: 169
[Child 106742]   Current number: 14          Square: 196         Sum: 365
[Child 106742]   Current number: 15          Square: 225         Sum: 590
[Child 106742]   Current number: 16          Square: 256         Sum: 846
[Child 106742]   Current number: 17          Square: 289         Sum: 1135
[Child 106742]   Current number: 18          Square: 324         Sum: 1459
[Child 106742]   Current number: 19          Square: 361         Sum: 1820
[Child 106742]   Current number: 20          Square: 400         Sum: 2220
[Child 106742]   Result: 2220

Final Result: 2870
```

Рисунок 5 – Работа программы

Процессы отображаются в `htop` (рисунки 6-7). Зомби-процессы появляются, когда родительский процесс ещё не вызвал `wait`.

106351	alex	1021	20	0	555M	66304	50872	S	0.0	0.4	0:00.00	<pre> - /usr/lib/gnome-terminal-server - fish - ./main input.txt 3 - calc _0_input.txt 6 - calc _1_input.txt 6 - calc _2_input.txt 8 </pre>
106365	alex	106347	20	0	229M	8556	6140	S	0.7	0.1	0:00.40	
109167	alex	106365	20	0	2488	876	788	S	0.0	0.0	0:00.00	
109168	alex	109167	20	0	2488	808	720	S	0.0	0.0	0:00.00	
109169	alex	109167	20	0	2488	872	788	S	0.0	0.0	0:00.00	
109170	alex	109167	20	0	2488	880	788	S	0.0	0.0	0:00.00	

Рисунок 6 – Подпроцессы

106351	alex	1021	20	0	555M	66304	50872	S	0.0	0.4	0:00.00	<pre> - /usr/lib/gnome-terminal-server - fish - ./main input.txt 3 - calc _0_input.txt 6 - calc _1_input.txt 6 - calc _2_input.txt 8 </pre>
106365	alex	106347	20	0	229M	8556	6140	S	0.0	0.1	0:00.40	
109167	alex	106365	20	0	2488	876	788	S	0.0	0.0	0:00.00	
109168	alex	109167	20	0	0	0	0	Z	0.0	0.0	0:00.00	
109169	alex	109167	20	0	0	0	0	Z	0.0	0.0	0:00.00	
109170	alex	109167	20	0	0	0	0	Z	0.0	0.0	0:00.00	

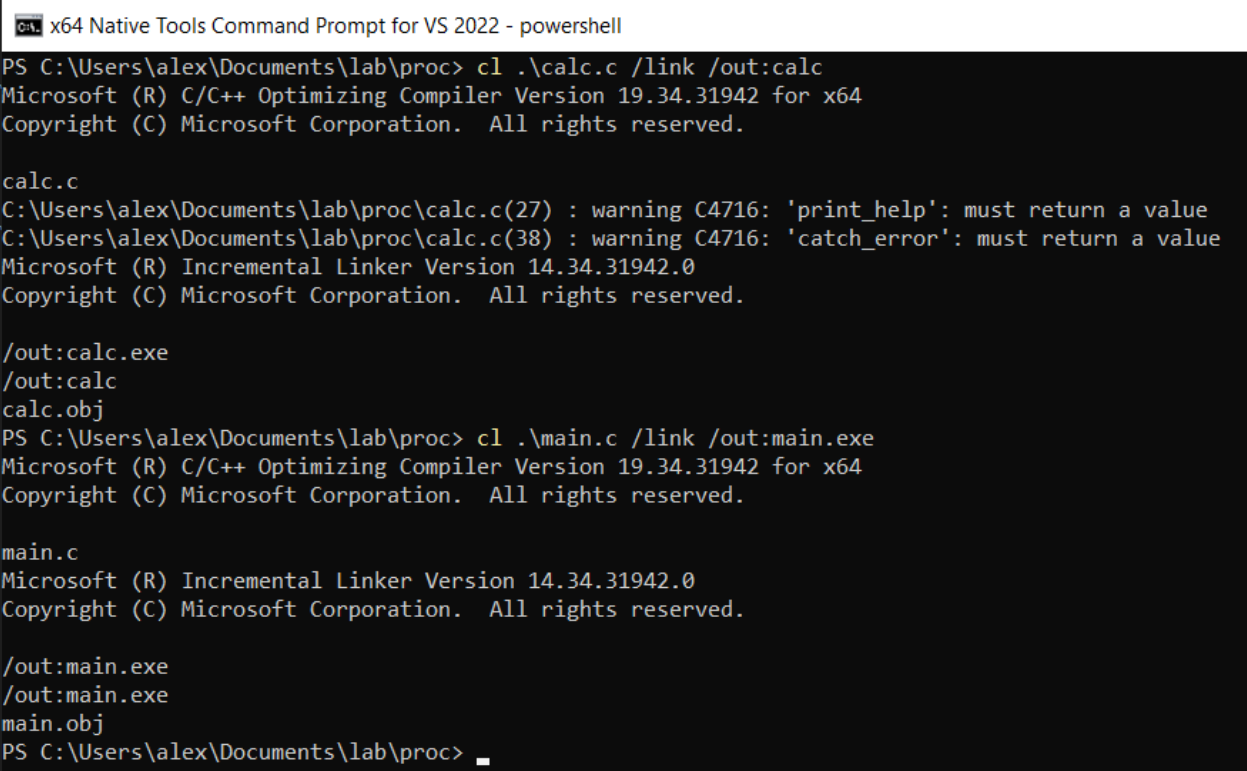
Рисунок 7 – Процессы-зомби

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 `GetStartupInfo`, `CreateProcess`, `WaitForSingleObject`. Исходный код программы для Linux приведён в приложении.

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

Процесс сборки показан на рисунке 8.



```
x64 Native Tools Command Prompt for VS 2022 - powershell
PS C:\Users\alex\Documents\lab\proc> cl .\calc.c /link /out:calc
Microsoft (R) C/C++ Optimizing Compiler Version 19.34.31942 for x64
Copyright (C) Microsoft Corporation. All rights reserved.

calc.c
C:\Users\alex\Documents\lab\proc\calc.c(27) : warning C4716: 'print_help': must return a value
C:\Users\alex\Documents\lab\proc\calc.c(38) : warning C4716: 'catch_error': must return a value
Microsoft (R) Incremental Linker Version 14.34.31942.0
Copyright (C) Microsoft Corporation. All rights reserved.

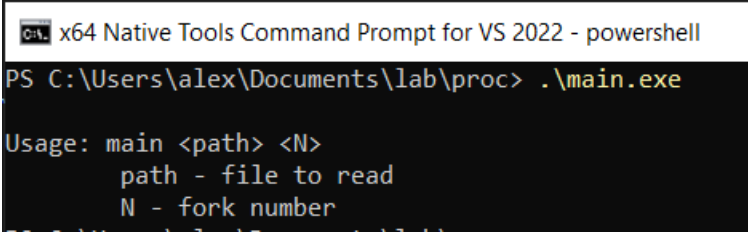
/out:calc.exe
/out:calc
calc.obj
PS C:\Users\alex\Documents\lab\proc> cl .\main.c /link /out:main.exe
Microsoft (R) C/C++ Optimizing Compiler Version 19.34.31942 for x64
Copyright (C) Microsoft Corporation. All rights reserved.

main.c
Microsoft (R) Incremental Linker Version 14.34.31942.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:main.exe
/out:main.exe
main.obj
PS C:\Users\alex\Documents\lab\proc> █
```

Рисунок 8 – Сборка файлов

Далее продемонстрируем работу программы (рисунки 9-11).



```
x64 Native Tools Command Prompt for VS 2022 - powershell
PS C:\Users\alex\Documents\lab\proc> .\main.exe
Usage: main <path> <N>
      path - file to read
      N - fork number
```

Рисунок 9 – Неверные аргументы


```
Git x64 Native Tools Command Prompt for VS 2022 - powershell
PS C:\Users\alex\Documents\lab\proc> .\main.exe input1.txt 3
Path to file: input1.txt
Fork number: 3
The system cannot find the file specified.

Usage: main <path> <N>
      path - file to read
      N - fork number
```

Рисунок 10 – Входной файл не существует

```
PS C:\Users\alex\Documents\lab\proc> .\main.exe input.txt 3
Path to file: input.txt
Fork number: 3
Input size: 20
Each proc gets: 6 numbers
Last proc gets: 8 numbers
Forked proc 1168 with output_path _0_input.txt
Forked proc 288 with output_path _1_input.txt

[Child 288]      New proc
[Child 288]      Path to file with numbers: _1_input.txt
[Child 288]      Input length: 6

[Child 1168]     New proc
[Child 1168]     Path to file with numbers: _0_input.txt
[Child 1168]     Input length: 6
[Child 288]     Current number: 7      Square: 49      Sum: 49
[Child 1168]     Current number: 1      Square: 1       Sum: 1
[Child 1168]     Current number: 2      Square: 4       Sum: 5
[Child 1168]     Current number: 3      Square: 9       Sum: 14
[Child 1168]     Current number: 4      Square: 16      Sum: 30
[Child 1168]     Current number: 5      Square: 25      Sum: 55
[Child 1168]     Current number: 6      Square: 36      Sum: 91
[Child 288]     Current number: 8      Square: 64      Sum: 113
[Child 1168]     Result: 91
[Child 288]     Current number: 9      Square: 81      Sum: 194
[Child 288]     Current number: 10     Square: 100     Sum: 294
[Child 288]     Current number: 11     Square: 121     Sum: 415
[Child 288]     Current number: 12     Square: 144     Sum: 559
[Child 288]     Result: 559
Forked proc 4476 with output_path _2_input.txt

[Child 4476]     New proc
[Child 4476]     Path to file with numbers: _2_input.txt
[Child 4476]     Input length: 8
[Child 4476]     Current number: 13     Square: 169     Sum: 169
[Child 4476]     Current number: 14     Square: 196     Sum: 365
[Child 4476]     Current number: 15     Square: 225     Sum: 590
[Child 4476]     Current number: 16     Square: 256     Sum: 846
[Child 4476]     Current number: 17     Square: 289     Sum: 1135
[Child 4476]     Current number: 18     Square: 324     Sum: 1459
[Child 4476]     Current number: 19     Square: 361     Sum: 1820
[Child 4476]     Current number: 20     Square: 400     Sum: 2220
[Child 4476]     Result: 2220

Final Result: 2870
```

Рисунок 11 – Работа программы

Процессы отображаются в Process Hacker (рисунки 12-13).

explorer.exe	4632	0.42	7.88 kB/s	52.85 MB	DESKTOP-OKMTD3E\ale	Windows Explorer
cmd.exe	5968			2.48 MB	DESKTOP-OKMTD3E\ale	Windows Command Processor
conhost.exe	5392		120 B/s	11.11 MB	DESKTOP-OKMTD3E\ale	Console Window Host
powershell.exe	5764			61.35 MB	DESKTOP-OKMTD3E\ale	Windows PowerShell
main.exe	1604			600 kB	DESKTOP-OKMTD3E\ale	
calc	6652			448 kB	DESKTOP-OKMTD3E\ale	
calc	4352			444 kB	DESKTOP-OKMTD3E\ale	
calc	1872			444 kB	DESKTOP-OKMTD3E\ale	

Рисунок 12 – Подпроцессы

explorer.exe	4632	0.27	7.88 kB/s	53.3 MB	DESKTOP-OKMTD3E\ale	Windows Explorer
cmd.exe	5968			2.48 MB	DESKTOP-OKMTD3E\ale	Windows Command Processor
conhost.exe	5392			11.11 MB	DESKTOP-OKMTD3E\ale	Console Window Host
powershell.exe	5764			61.28 MB	DESKTOP-OKMTD3E\ale	Windows PowerShell
main.exe	1604			600 kB	DESKTOP-OKMTD3E\ale	

Рисунок 13 – Основной процесс

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 14-5).

```
(.venv) alex@alex-nb ~/D/y/h/7/py (master) [0|SIGINT]> python main.py input.txt
Usage: main <path> <N>
      path - file to read
      N - fork number
```

Рисунок 14 – Неверные аргументы

```
(.venv) alex@alex-nb ~/D/y/h/7/py (master)> python main.py input1.txt 3
Path to file: input1.txt
Fork number: 3
Traceback (most recent call last):
  File "/home/alex/Documents/year-3-c/hw/7_proc/py/main.py", line 103, in <module>
    main()
  File "/home/alex/Documents/year-3-c/hw/7_proc/py/main.py", line 39, in main
    with open(input_path, "r") as f:
FileNotFoundError: [Errno 2] No such file or directory: 'input1.txt'
```

Рисунок 15 – Входной файл не существует

```
(.venv) alex@alex-nb ~/D/y/h/7/py (master) [0|1]> python main.py input.txt 4
Path to file: input.txt
Fork number: 4
Input size: 20
Each proc gets: 5 numbers
Last proc gets: 5 numbers
Forked proc 110150 with output_path _0_input.txt
Forked proc 110152 with output_path _1_input.txt
Forked proc 110153 with output_path _2_input.txt
Forked proc 110154 with output_path _3_input.txt

[Child 110154]    New proc
[Child 110154]    Path to file with numbers: _3_input.txt
[Child 110154]    Input length: 5
[Child 110154]    Result: 1630

[Child 110153]    New proc
[Child 110153]    Path to file with numbers: _2_input.txt
[Child 110153]    Input length: 5
[Child 110153]    Result: 855

[Child 110152]    New proc
[Child 110152]    Path to file with numbers: _1_input.txt
[Child 110152]    Input length: 5
[Child 110152]    Result: 330

[Child 110150]    New proc
[Child 110150]    Path to file with numbers: _0_input.txt
[Child 110150]    Input length: 5
[Child 110150]    Result: 55

Final Result: 2870
```

Рисунок 16 – Работа программы

Процессы отображаются в htop (рисунки 12-13).

106351	alex	1021	20	0	555M	65604	50608	S	0.0	0.4	0:00.00	/usr/lib/gnome-terminal-server
106365	alex	106347	20	0	229M	8724	6216	S	0.0	0.1	0:00.50	fish
109991	alex	106365	20	0	13184	8068	5172	S	0.7	0.1	0:00.01	└─ python main.py input.txt 3
109994	alex	109991	20	0	13188	7468	4832	S	0.7	0.0	0:00.01	└─ python calc.py _0_input.txt
109995	alex	109991	20	0	13188	7464	4828	S	0.7	0.0	0:00.01	└─ python calc.py _1_input.txt
109996	alex	109991	20	0	13188	7460	4824	S	0.0	0.0	0:00.00	└─ python calc.py _2_input.txt

Рисунок 17 – Подпроцессы

106351	alex	1021	20	0	555M	65604	50608	S	0.0	0.4	0:00.00	/usr/lib/gnome-terminal-server
106365	alex	106347	20	0	229M	8724	6216	S	0.0	0.1	0:00.50	fish
109991	alex	106365	20	0	13184	8068	5172	S	0.0	0.1	0:00.01	└─ python main.py input.txt 3
109994	alex	109991	20	0	0	0	0	Z	0.0	0.0	0:00.01	└─ python calc.py _0_input.txt
109995	alex	109991	20	0	0	0	0	Z	0.0	0.0	0:00.01	└─ python calc.py _1_input.txt
109996	alex	109991	20	0	0	0	0	Z	0.0	0.0	0:00.00	└─ python calc.py _2_input.txt

Рисунок 18 – Процессы-зомби

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 дочерних процессов (N передаётся вторым аргументом командной строки) и передать каждому из них часть полученных чисел. Каждый из дочерних процессов 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 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 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <fcntl.h>
5 #include <unistd.h>
6 #include <wait.h>
7
8 /*В
9 файле записан ряд целых чисел, разделённых пробелом. Программа должна
10 считать имя файла из первого аргумента
11 командной строки и рассчитать сумму квадратов записанных в файл чисел.
12 Для расчёта суммы квадратов программа
13 должна создать N дочерних процессов (N передаётся вторым аргументом
14 командной строки) и передать каждому
15 из них часть полученных чисел. Каждый из дочерних процессов должен
16 рассчитать сумму квадратов переданных
17 ему чисел и вернуть её родителю. Родительский процесс должен
18 просуммировать полученные от дочерних
19 числа и вывести на консоль итоговую сумму. Если исходный файл не
20 существует, или в нём записано менее
21 2 чисел, следует вывести соответствующее сообщение для пользователя и
22 завершить работу программы.
23 */
24
25 /**
26  * Prints help message to console
27  */
28 int print_help()
29 {
30     printf("\nUsage: main <path> <N> \n");
31     printf("\tpath - file to read \n");
32     printf("\tN - fork number \n");
33     return 1;
34 }
35
36 /**
37  * Prints error message and help message to console
38  * and closes the program
39  */
40 int catch_error()
41 {
42     perror("\nError");
43     print_help();
44 }
45
46 int main(int argc, char **argv)
47 {
48     // check number of arguments
49     if (argc != 3)
50         return print_help();
51
52     // get amount of numbers and file path
53     char *input_path = argv[1];
54     int N = atoi(argv[2]);
55
56     // print input values
57     printf("Path to file: %s \n", input_path);
58     printf("Fork number: %d \n", N);
59
60     // open file to read
61     FILE *input_file = fopen(input_path, "r");
62     if (!input_file)
63         return catch_error();
64
65     // get number of input digits M
66     char t[12];
67     int M = 0; // input size
68     while (fscanf(input_file, "%12s", t) == 1)
```

```

62         M++;
63
64     printf("Input size: %d \n", M);
65
66     // if too few numbers
67     if (M < 2)
68     {
69         printf("Too few numbers (M must be greater than 2) \n");
70         return print_help();
71     }
72
73     // if too many forks
74     if (N > M / 2)
75     {
76         N = M / 2;
77         printf("Too many forks. New fork number: %d \n", N);
78     }
79
80     // calculate division between processes
81     int n = M / N;
82     int n_last = n + M % N;
83
84     printf("Each proc gets: %d numbers \n", n);
85     printf("Last proc gets: %d numbers \n", n_last);
86
87     // move to file start
88     if (fseek(input_file, 0, SEEK_SET))
89         return catch_error();
90
91     // create files for each process and fork
92     for (int i = 0; i < N; i++)
93     {
94         // construct name of output file
95         char output_path[30];
96         if (sprintf(output_path, "_%d_input.txt", i) < 0)
97             return catch_error();
98
99         // open file to write
100        FILE *output_file = fopen(output_path, "w");
101        if (!output_file)
102            return catch_error();
103
104        // write numbers to output file
105        int n_effective = (i == N - 1) ? n_last : n;
106        for (int j = 0; j < n_effective; j++)
107        {
108            if (!fscanf(input_file, "%12s", t))
109                return catch_error();
110            if (fprintf(output_file, "%s ", t) < 0)
111                return catch_error();
112        }
113
114        // release file
115        if (fclose(output_file) != 0)
116            return catch_error();
117
118        // start process
119        pid_t res = fork();
120
121        switch (res)
122        {
123            case 0: // forked
124            {
125                if (sprintf(t, "%d", n_effective) < 0)
126                    return catch_error();
127                char *args[4] = {"calc", output_path, t, NULL};
128                printf("Forked proc %d with output_path %s \n", getpid(),
output_path);
129                execve("calc", args, NULL);

```

```

130     }
131     case -1: // error
132         return catch_error();
133     }
134 }
135
136 // release input file
137 if (fclose(input_file) != 0)
138     return catch_error();
139
140 // stop process to see zombies
141 getchar();
142
143 // calc results
144 int sum = 0;
145 for (int i = 0; i < N; i++)
146 {
147     int code = 0;
148     pid_t child = wait(&code);
149     if (child == -1)
150         return catch_error();
151
152     // construct name of output file
153     if (sprintf(t, "%d_result.txt", child) < 0)
154         return catch_error();
155
156     // open result file to read
157     FILE *result_file = fopen(t, "r");
158     if (!result_file)
159         return catch_error();
160
161     // read result from file
162     if (!fscanf(result_file, "%12s", t))
163         return catch_error();
164
165     // release input file
166     if (fclose(result_file) != 0)
167         return catch_error();
168
169     // calculate
170     sum += atoi(t);
171 }
172
173 printf("\nFinal Result: %d \n", sum);
174 return 0;
175 }

```


Приложение Б. Код linux/cal.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <fcntl.h>
5 #include <unistd.h>
6
7 /*В
8  файле записан ряд целых чисел, разделённых пробелом. Программа должна
9  считать имя файла из первого аргумента
10  командной строки и рассчитать сумму квадратов записанных в файл чисел.
11  Для расчёта суммы квадратов программа
12  должна создать N дочерних процессов (N передаётся вторым аргументом
13  командной строки) и передать каждому
14  из них часть полученных чисел. Каждый из дочерних процессов должен
15  рассчитать сумму квадратов переданных
16  ему чисел и вернуть её родителю. Родительский процесс должен
17  просуммировать полученные от дочерних
18  числа и вывести на консоль итоговую сумму. Если исходный файл не
19  существует, или в нём записано менее
20  2 чисел, следует вывести соответствующее сообщение для пользователя и
21  завершить работу программы.
22 */
23
24 /**
25  * Prints help message to console
26  */
27 int print_help()
28 {
29     printf("\nUsage: calc <path> <N> \n");
30     printf("\tpath - file to read \n");
31     printf("\tN - amount of numbers in input file \n");
32     exit(EXIT_FAILURE);
33 }
34
35 /**
36  * Prints error message and help message to console
37  * and closes the program
38  */
39 int catch_error()
40 {
41     perror("\nError");
42     print_help();
43 }
44
45 int main(int argc, char **argv)
46 {
47     // print PID
48     pid_t pid = getpid();
49     printf("\n[Child %d] \t New proc \n", pid);
50
51     // check number of arguments
52     if (argc != 3)
53         return print_help();
54
55     // get byte numbers and file path
56     char *input_path = argv[1];
57     int n = atoi(argv[2]);
58
59     // print input values
60     printf("[Child %d] \t Path to file with numbers: %s \n", pid,
61            input_path);
62     printf("[Child %d] \t Input length: %d \n", pid, n);
63
64     // open file to read
65     FILE *input_file = fopen(input_path, "r");
66     if (!input_file)
67         return catch_error();
68 }
```

```

61
62 // move to file start
63 if (fseek(input_file, 0, SEEK_SET))
64     return catch_error();
65
66 // perform calculations
67 char t[30];
68 int sum = 0;
69 for (int i = 0; i < n; i++)
70 {
71     if (!fscanf(input_file, "%12s", t))
72         return catch_error();
73
74     // calculate
75     int cur_number = atoi(t);
76     int square = cur_number * cur_number;
77     sum += square;
78     printf("[Child %d] \t Current number: %d \t Square: %d \t Sum: %d
79 \n", pid, cur_number, square, sum);
80 }
81
82 // release input file
83 if (fclose(input_file) != 0)
84     return catch_error();
85
86 printf("[Child %d] \t Result: %d \n", pid, sum);
87
88 // construct name of output file
89 if (sprintf(t, "%d_result.txt", pid) < 0)
90     return catch_error();
91
92 FILE *output_file = fopen(t, "w");
93 if (!output_file)
94     return catch_error();
95
96 if (fprintf(output_file, "%d", sum) < 0)
97     return catch_error();
98
99 // release input file
100 if (fclose(output_file) != 0)
101     return catch_error();
102
103 sleep(5);
104 exit(EXIT_SUCCESS);
105 }

```

Приложение В. Код win/main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <fcntl.h>
5 #include <Windows.h>
6
7 /*В
8  файле записан ряд целых чисел, разделённых пробелом. Программа должна
9  считать имя файла из первого аргумента
10  командной строки и рассчитать сумму квадратов записанных в файл чисел.
11  Для расчёта суммы квадратов программа
12  должна создать N дочерних процессов (N передаётся вторым аргументом
13  командной строки) и передать каждому
14  из них часть полученных чисел. Каждый из дочерних процессов должен
15  рассчитать сумму квадратов переданных
16  ему чисел и вернуть её родителю. Родительский процесс должен
17  просуммировать полученные от дочерних
18  числа и вывести на консоль итоговую сумму. Если исходный файл не
19  существует, или в нём записано менее
20  2 чисел, следует вывести соответствующее сообщение для пользователя и
21  завершить работу программы.
22 */
23
24 typedef int pid_t;
25
26 /**
27  * Prints help message to console
28  */
29 int print_help()
30 {
31     printf("\nUsage: main <path> <N> \n");
32     printf("\tpath - file to read \n");
33     printf("\tN - fork number \n");
34     return 1;
35 }
36
37 /**
38  * Prints error message and help message to console
39  * and closes the program
40  */
41 int catch_error()
42 {
43     LPSTR message;
44     DWORD dwMessageLen = FormatMessage(
45         FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_ALLOCATE_BUFFER,
46         NULL, GetLastError(), MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
47         (LPSTR)&message, 0, NULL);
48     printf(message);
49     print_help();
50     return 1;
51 }
52
53 int main(int argc, char **argv)
54 {
55     // check number of arguments
56     if (argc != 3)
57         return print_help();
58
59     // get amount of numbers and file path
60     char *input_path = argv[1];
61     int N = atoi(argv[2]);
62
63     // print input values
64     printf("Path to file: %s \n", input_path);
65     printf("Fork number: %d \n", N);
66
67     // open file to read
68     FILE *input_file = fopen(input_path, "r");
```

```

62     if (!input_file)
63         return catch_error();
64
65     // get number of input digits M
66     char t[100];
67     int M = 0; // input size
68     while (fscanf(input_file, " %12s", t) == 1)
69         M++;
70
71     printf("Input size: %d \n", M);
72
73     // if too few numbers
74     if (M < 2)
75     {
76         printf("Too few numbers (M must be greater than 2) \n");
77         return print_help();
78     }
79
80     // if too many forks
81     if (N > M / 2)
82     {
83         N = M / 2;
84         printf("Too many forks. New fork number: %d \n", N);
85     }
86
87     // calculate division between processes
88     int n = M / N;
89     int n_last = n + M % N;
90
91     printf("Each proc gets: %d numbers \n", n);
92     printf("Last proc gets: %d numbers \n", n_last);
93
94     // move to file start
95     if (fseek(input_file, 0, SEEK_SET))
96         return catch_error();
97
98     // create files for each process and fork
99     HANDLE *hProcesses = malloc(sizeof(HANDLE) * N);
100     DWORD *dwProcessIds = malloc(sizeof(DWORD) * N);
101     for (int i = 0; i < N; i++)
102     {
103         // construct name of output file
104         char output_path[30];
105         if (sprintf(output_path, "_%d_input.txt", i) < 0)
106             return catch_error();
107
108         // open file to write
109         FILE *output_file = fopen(output_path, "w");
110         if (!output_file)
111             return catch_error();
112
113         // write numbers to output file
114         int n_effective = (i == N - 1) ? n_last : n;
115         for (int j = 0; j < n_effective; j++)
116         {
117             if (!fscanf(input_file, " %12s", t))
118                 return catch_error();
119             if (fprintf(output_file, "%s ", t) < 0)
120                 return catch_error();
121         }
122
123         // release file
124         if (fclose(output_file) != 0)
125             return catch_error();
126
127         // start process
128         PROCESS_INFORMATION pi;
129         STARTUPINFO si;
130         GetStartupInfo(&si);

```

```

131         if (sprintf(t, "calc %s %d", output_path, n_effective) < 0)
132             return catch_error();
133         BOOL res = CreateProcess("calc", t, NULL, NULL, TRUE, 0, NULL,
134             NULL, &si, &pi);
135         if (!res)
136             return catch_error();
137
138         printf("Forked proc %d with output_path %s \n", pi.dwProcessId,
139             output_path);
140         hProcesses[i] = pi.hProcess;
141         dwProcessIds[i] = pi.dwProcessId;
142     }
143     // release input file
144     if (fclose(input_file) != 0)
145         return catch_error();
146
147     // stop process to see zombies
148     getchar();
149
150     // calc results
151     int sum = 0;
152     for (int i = 0; i < N; i++)
153     {
154         int code = 0;
155         DWORD dwRes = WaitForSingleObject(hProcesses[i], INFINITE);
156         if (dwRes == WAIT_FAILED)
157             return catch_error();
158
159         // construct name of output file
160         if (sprintf(t, "_%d_result.txt", dwProcessIds[i]) < 0)
161             return catch_error();
162
163         // open result file to read
164         FILE *result_file = fopen(t, "r");
165         if (!result_file)
166             return catch_error();
167
168         // read result from file
169         if (!fscanf(result_file, "%12s", t))
170             return catch_error();
171
172         // release input file
173         if (fclose(result_file) != 0)
174             return catch_error();
175
176         // calculate
177         sum += atoi(t);
178     }
179
180     free(hProcesses);
181     free(dwProcessIds);
182     printf("\nFinal Result: %d \n", sum);
183     return 0;
184 }

```

Приложение Г. Код win/calc.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <fcntl.h>
5
6 /*В
7  файл записан ряд целых чисел, разделённых пробелом. Программа должна
8  считать имя файла из первого аргумента
9  командной строки и рассчитать сумму квадратов записанных в файл чисел.
10  Для расчёта суммы квадратов программа
11  должна создать N дочерних процессов (N передаётся вторым аргументом
12  командной строки) и передать каждому
13  из них часть полученных чисел. Каждый из дочерних процессов должен
14  рассчитать сумму квадратов переданных
15  ему чисел и вернуть её родительскому. Родительский процесс должен
16  просуммировать полученные от дочерних
17  числа и вывести на консоль итоговую сумму. Если исходный файл не
18  существует, или в нём записано менее
19  2 чисел, следует вывести соответствующее сообщение для пользователя и
20  завершить работу программы.
21 */
22
23 typedef int pid_t;
24
25 /**
26  * Prints help message to console
27  */
28 int print_help()
29 {
30     printf("\nUsage: calc <path> <N> \n");
31     printf("\tpath - file to read \n");
32     printf("\tN - amount of numbers in input file \n");
33     ExitProcess(EXIT_FAILURE);
34 }
35
36 /**
37  * Prints error message and help message to console
38  * and closes the program
39  */
40 int catch_error()
41 {
42     perror("\nError");
43     print_help();
44     ExitProcess(EXIT_FAILURE);
45 }
46
47 int main(int argc, char **argv)
48 {
49     // print PID
50     pid_t pid = getpid();
51     printf("\n[Child %d] \t New proc \n", pid);
52
53     // check number of arguments
54     if (argc != 3)
55         return print_help();
56
57     // get byte numbers and file path
58     char *input_path = argv[1];
59     int n = atoi(argv[2]);
60
61     // print input values
62     printf("[Child %d] \t Path to file with numbers: %s \n", pid,
63           input_path);
64     printf("[Child %d] \t Input length: %d \n", pid, n);
65
66     // open file to read
67     FILE *input_file = fopen(input_path, "r");
```

```

61     if (!input_file)
62         return catch_error();
63
64     // move to file start
65     if (fseek(input_file, 0, SEEK_SET))
66         return catch_error();
67
68     // perform calculations
69     char t[30];
70     int sum = 0;
71     for (int i = 0; i < n; i++)
72     {
73         if (!fscanf(input_file, " %12s", t))
74             return catch_error();
75
76         // calculate
77         int cur_number = atoi(t);
78         int square = cur_number * cur_number;
79         sum += square;
80         printf("[Child %d] \t Current number: %d \t Square: %d \t Sum: %d
81 \n", pid, cur_number, square, sum);
82     }
83
84     // release input file
85     if (fclose(input_file) != 0)
86         return catch_error();
87
88     printf("[Child %d] \t Result: %d \n", pid, sum);
89
90     // construct name of output file
91     if (sprintf(t, "%d_result.txt", pid) < 0)
92         return catch_error();
93
94     FILE *output_file = fopen(t, "w");
95     if (!output_file)
96         return catch_error();
97
98     if (fprintf(output_file, "%d", sum) < 0)
99         return catch_error();
100
101     // release output file
102     if (fclose(output_file) != 0)
103         return catch_error();
104
105     Sleep(5000);
106     ExitProcess(EXIT_SUCCESS);
107 }

```

Приложение Д. 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 os
20 import sys
21
22 def print_help():
23     """Prints help message to console"""
24     print("\nUsage: main <path> <N>")
25     print("\tpath - file to read")
26     print("\tN - fork number")
27     return 1
28
29 def main():
30     # check number of arguments
31     if len(sys.argv) != 3:
32         return print_help()
33
34     # get amount of numbers and file path
35     input_path: str = sys.argv[1]
36     N: int = int(sys.argv[2])
37
38     # print input values
39     print("Path to file: %s" % input_path)
40     print("Fork number: %d" % N)
41
42     # open file to read
43     with open(input_path, "r") as f:
44         input_file = f.read().strip().split(" ")
45         input_file = list(map(int, input_file))
46
47     M = len(input_file)
48     print("Input size: %d" % M)
49
50     # if too few numbers
51     if M < 2:
52         print("Too few numbers (M must be greater than 2)")
53         return print_help()
54
55     # if too many forks
56     if N > M // 2:
57         N = M // 2
58         print("Too many forks. New fork number: %d" % N)
59
60     # calculate division between processes
61     n: int = M // N
62     n_last: int = n + M % N
63
64     print("Each proc gets: %d numbers" % n)
65     print("Last proc gets: %d numbers" % n_last)
```



```

62
63 # create files for each process and fork
64 for i in range(N):
65     # construct name of output file
66     output_path = "_%d_input.txt" % i
67
68     # open file to write
69     with open(output_path, "w") as output_file:
70         # write numbers to output file
71         n_effective: int = n_last if i == N - 1 else n
72         subset = input_file[n * i : n * i + n_effective]
73         strings = list(map(str, subset))
74         output_file.write(" ".join(strings))
75
76     # start process
77     pid = os.fork()
78
79     if not pid: # forked
80         print("Forked proc %d with output_path %s" % (os.getpid(),
81             output_path))
82         os.execvp("python", ["python", "calc.py", output_path,
83             str(n_effective)])
84
85     # stop process to see zombies
86     input()
87
88     # calc results
89     sum: int = 0
90     for i in range(N):
91         child, code = os.wait()
92
93         # construct name of output file
94         result_path = "_%d_result.txt" % child
95
96         # open result file to read
97         with open(result_path, "r") as result_file:
98             # read result from file
99             sum += int(result_file.read().strip())
100
101     print("\nFinal Result: %d" % sum)
102     return 0
103 main()

```

Приложение Е. Код `py/calc.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 родителю. 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 import os
19 import sys
20 import time
21
22 def print_help():
23     """Prints help message to console"""
24     print("\nUsage: calc <path> <N>")
25     print("\tpath - file to read")
26     print("\tN - amount of numbers in input file")
27     return 1
28
29 def main():
30     # print PID
31     pid: int = os.getpid()
32     print("\n[Child %d] \t New proc" % pid)
33
34     # get byte numbers and file path
35     input_path: str = sys.argv[1]
36     n: int = int(sys.argv[2].strip())
37
38     # print input values
39     print("[Child %d] \t Path to file with numbers: %s" % (pid,
40 input_path))
41     print("[Child %d] \t Input length: %d" % (pid, n))
42
43     # open file to read
44     with open(input_path, "r") as f:
45         input_file = f.read().strip().split(" ")
46         input_file = list(map(int, input_file))
47
48     # perform calculations
49     Sum: int = sum([i**2 for i in input_file])
50
51     print("[Child %d] \t Result: %d" % (pid, Sum))
52
53     # construct name of output file
54     output_file = "_%d_result.txt" % pid
55
56     with open(output_file, "w") as f:
57         f.write(str(Sum))
58
59     time.sleep(5)
60     return 0
61
62 if __name__ == '__main__':
63     main()
```