**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Образовательная программа бакалавриата «Программная инженерия»

|  |
| --- |
|  |

|  |  |  |
| --- | --- | --- |
| **Домашнее задание №4**  **Пояснительная записка** | | |
|  |  | |
| Исполнитель  студентка группы БПИ193  В.В.Степанова  28.11.2020 | |
|  | | |
|  | |  |

**Москва 2020**

**Формулировка задачи**

Вариант 20.

Задача о производстве булавок. В цехе по заточке булавок все необходимые операции осуществляются тремя рабочими. Первый из них берет булавку и проверяет ее на предмет кривизны. Если булавка не кривая, то рабочий передает ее своему напарнику. Напарник осуществляет собственно заточку и передает заточенную булавку третьему рабочему, который осуществляет контроль качества операции. Требуется создать многопоточное приложение, моделирующее работу цеха. При решении использовать парадигму «производитель-потребитель».

**Тестирование программы**

Программа считывает исходные данные с файла “in.txt”.

В файле через пробел записаны три числа: all, curve, rate.

all - общее количество булавок;

curve - количество кривых булавок;

rate – вероятность ошибки второго рабочего (в процентах).

Программа выводит данные в консоль.

Первой строкой программа выводит информацию о входных данных в формате

all: {all}, curve: {curve}, rate: {rate}.

Второй строкой программа выводит количество верно изготовленных булавок в формате

result: {result}

Булавка считается верно изготовленной, если она не кривая и второй рабочий не совершил ошибку при заточке.

Пример 1.

Входные данные (рис.1):

all = 100 – общее количество булавок;

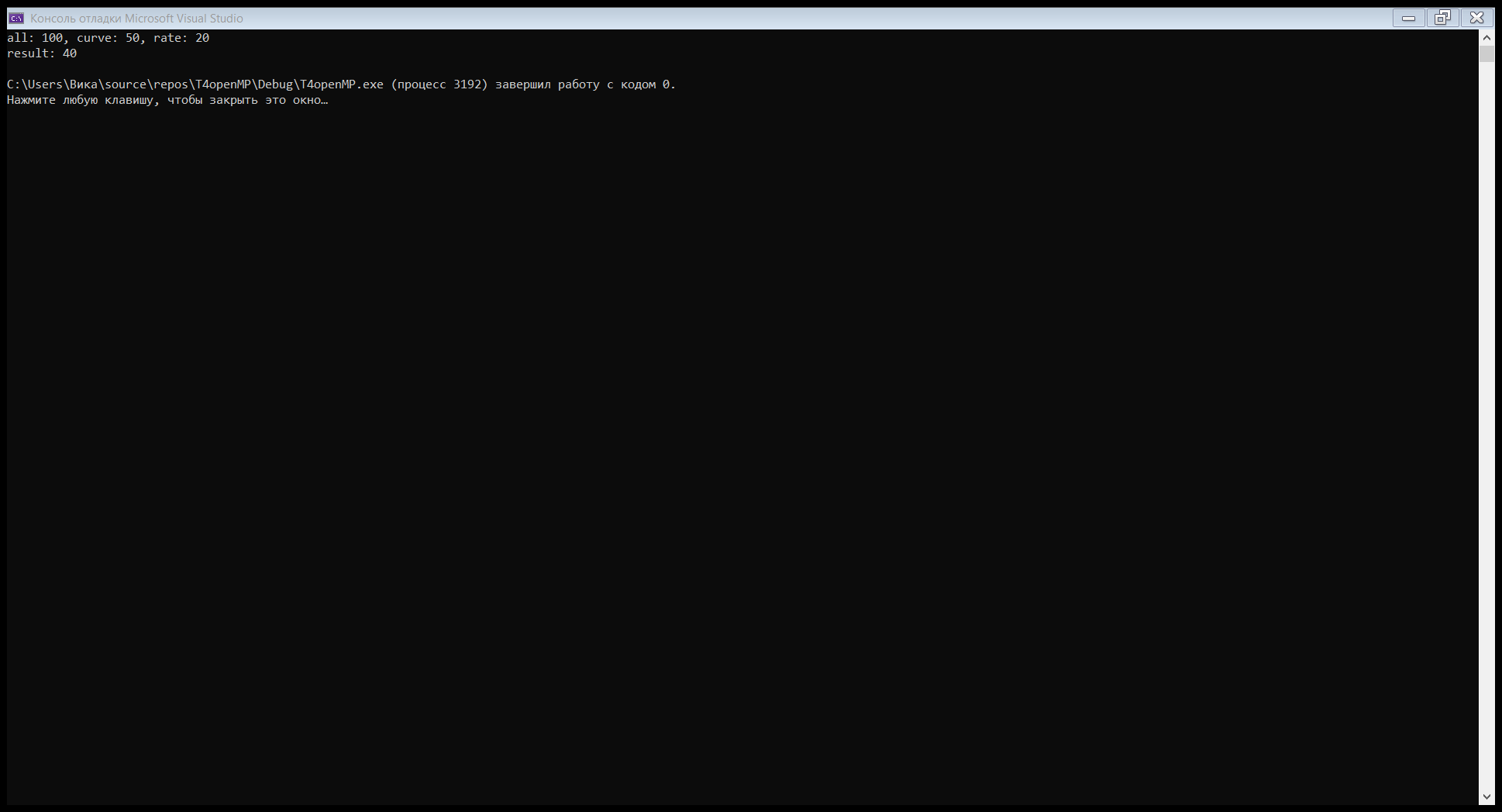
curve = 50 – количество кривых булавок;

rate = 20 – вероятность ошибки второго рабочего (в процентах).



*Рисунок 1.*

Выходные данные (рис.2):



*Рисунок 2.*

Пример 2.

Входные данные (рис.3):

all = 1000 – общее количество булавок;

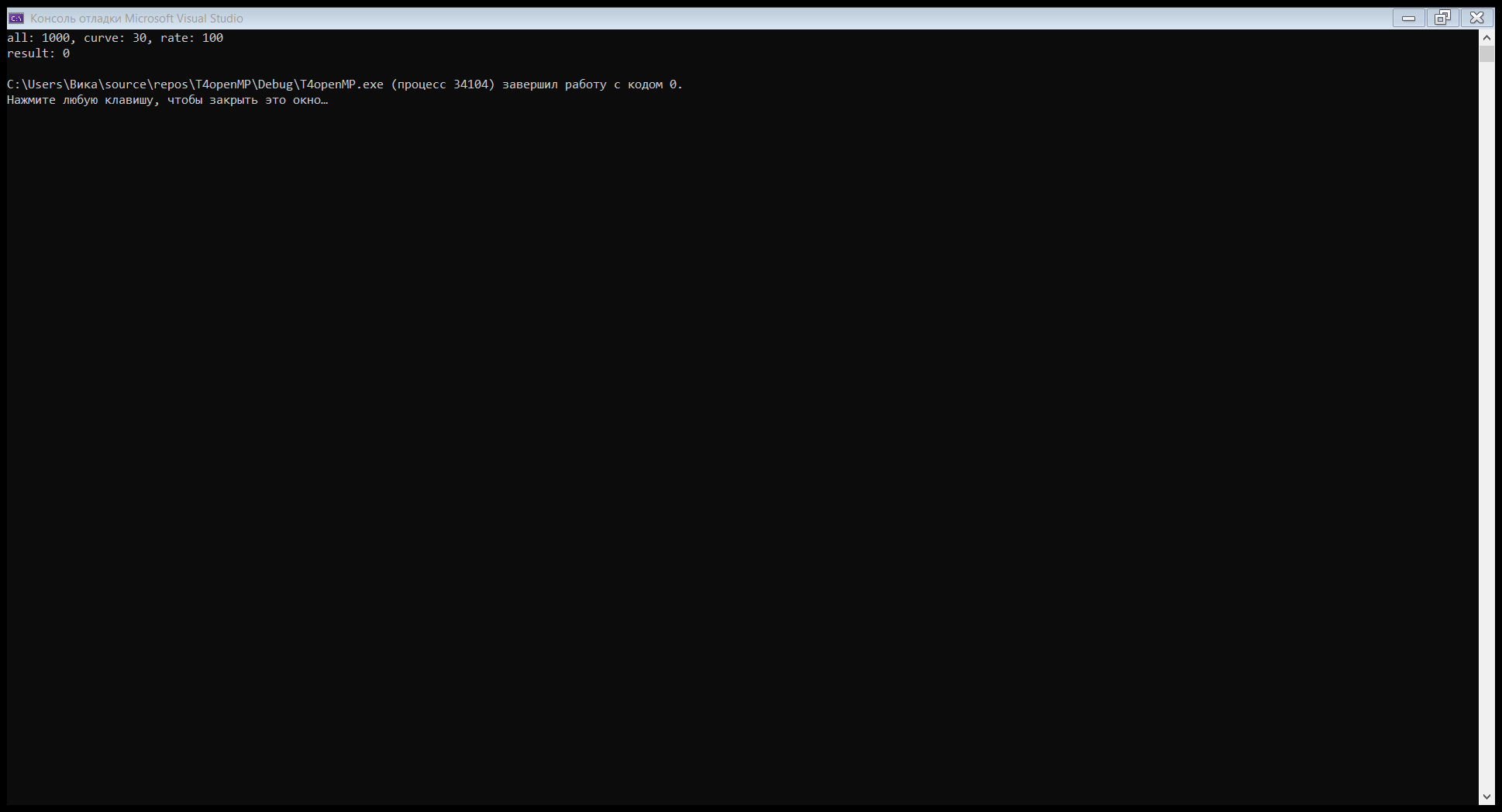
curve = 30 – количество кривых булавок;

rate = 100 – вероятность ошибки второго рабочего (в процентах).



*Рисунок 3.*

Выходные данные (рис.4):



*Рисунок 4.*

Пример 3.

Входные данные (рис.3):

all = 1000 – общее количество булавок;

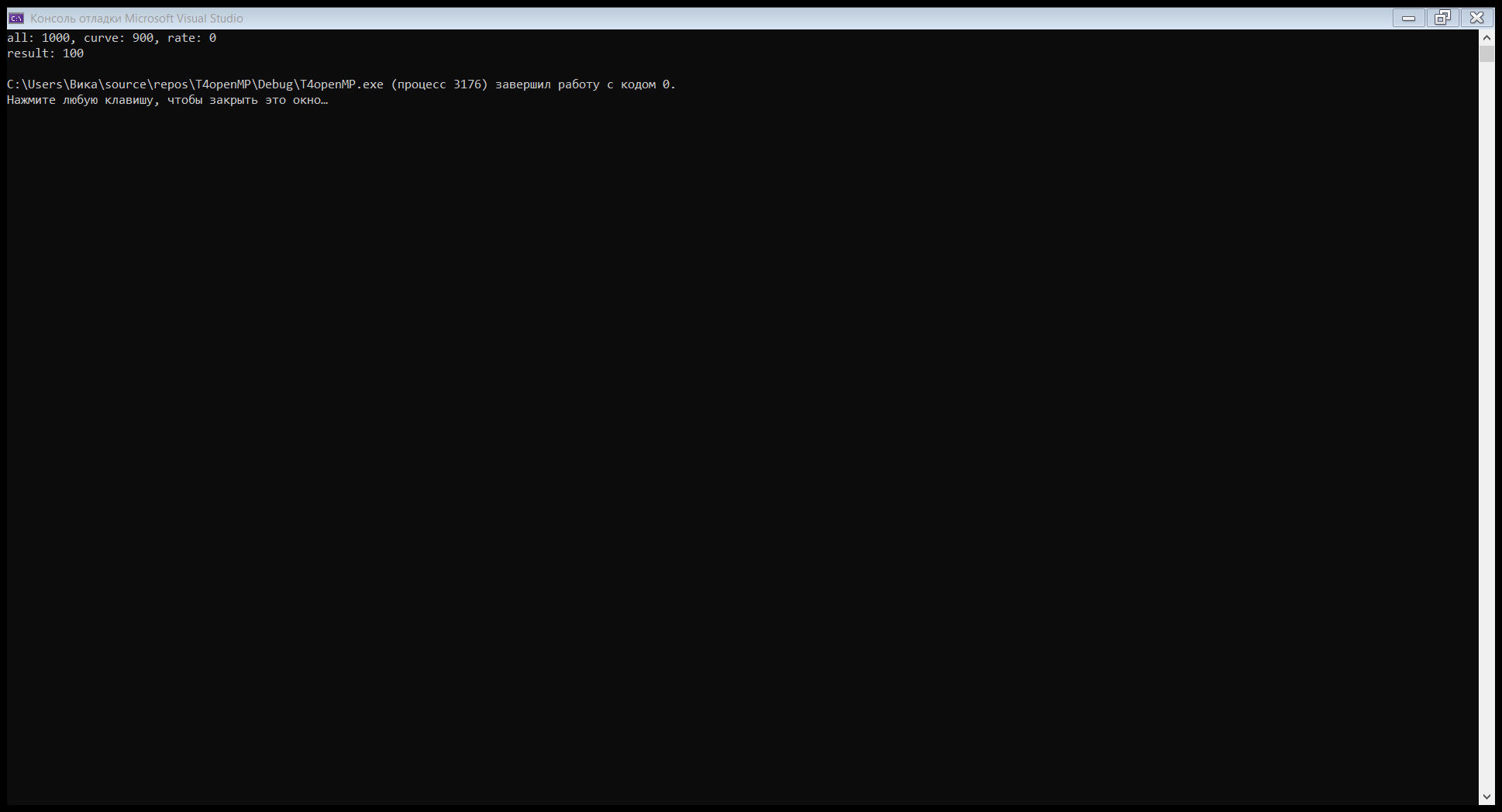
curve = 900 – количество кривых булавок;

rate = 0 – вероятность ошибки второго рабочего (в процентах).



*Рисунок 5.*

Выходные данные (рис.6):

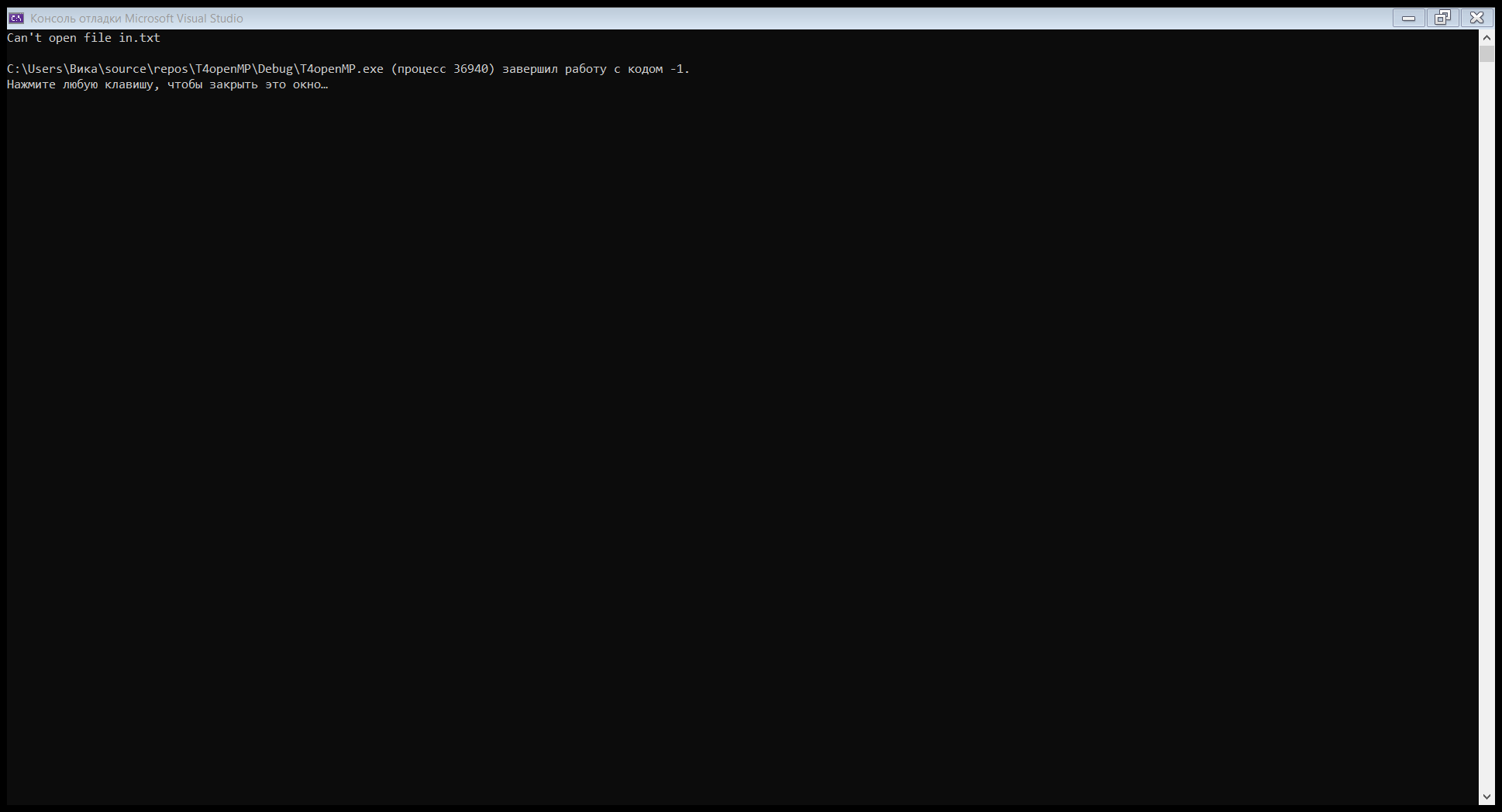


*Рисунок 6.*

Если файл “in.txt” не найден, программа выводит в консоль соответствующий текст:

Пример 4:

Выходные данные (рис. 7)



*Рисунок 7.*

**Список используемой литературы**

1. Уильямс Энтони. C++. Практика многопоточного программирования. - СПб.: Питер, 2020.-640с.
2. Грегори Р.Эндрюс. Основы многопоточного, параллельного и распределенного программирования. -М.: Издательский дом “Вильямс”, 2003.
3. [Антонов А.С. Технологии параллельного программирования MPI и OpenMP: Учеб. пособие. Предисл.: В.А.Садовничий. — М.: Издательство Московского университета, 2012.-344 с.-(Серия «Суперкомпьютерное образование»).](https://pro-prof.com/books-flp)
4. OpenMP [Электронный ресурс] / Wikipedia. Режим доступа: <https://ru.wikipedia.org/wiki/OpenMP> , свободный. (дата обращения: 25.11.2020)
5. Параллельное программирование на OpenMP [Электронный ресурс]. Режим доступа:

<http://ccfit.nsu.ru/arom/data/openmp.pdf> , свободный. (дата обращения 27.11.2020)

**Приложение. Текст программы.**

#include <omp.h>

#include <stdlib.h>

#include <time.h>

#include <stdio.h>

#include <errno.h>

#include <iostream>

#include <fstream>

#include <string>

typedef struct Data {

int all;

int curve;

int rate;

} Data;

// Считывает исходные данные из файла

// В файле записаны через пробел три числа:

// all, curve, rate

// где:

// all - общее количество булавок

// curve - количество кривых булавок

// rate – процент успеха выполнения заточки булавки вторым рабочим

int read\_data\_from\_file(const char\* file\_name, Data\* data) {

std::string line;

std::string allStr;

std::string curveStr;

std::string rateStr;

std::ifstream in(file\_name); // окрываем файл для чтения

if (in.is\_open())

{

in >> allStr;

in >> curveStr;

in >> rateStr;

}

else {

std::cout << "Can't open file " << file\_name << std::endl;

return errno;

}

in.close();

try {

data->all = stoi(allStr);

data->curve = stoi(curveStr);

data->rate = stoi(rateStr);

}

catch (std::invalid\_argument){

return -1;

}

catch (std::out\_of\_range){

return -1;

}

return 0;

}

// Первый рабочий проверяет булавку на годность (1 - ок, 0 - не ок)

int worker1(int material) {

return material;

}

// Второй берет хорошую булавку, и пытается ее заточить (с вероятностью rate ошибается)

int worker2(int material, int rate) {

if (material == 0) return 0;

return ((rand() % 100) < rate) ? 0 : 1;

}

// Третий рабочий выполняет контроль и считает количество выполненной работы

void worker3(int result, int\* sum) {

\*sum += result;

}

int main() {

srand(time(NULL));

Data data; // Переменная, в которую залетают исходные данные

if (read\_data\_from\_file("in.txt", &data) != 0) {

return -1;

}

fprintf(stdout, "all: %d, curve: %d, rate: %d\n", data.all, data.curve, data.rate);

// Массив с булавками, 1 - нормальная, 0 - кривая

int\* material = (int\*)malloc(sizeof(int) \* data.all);

int curves = data.curve;

// Генерируем случайные данные для булавок

// Сначала все нормальные (1)

for (size\_t i = 0; i < data.all; ++i) {

material[i] = 1;

}

// Затем случайным образом распределяем data.curve кривых булавок в массиве (0)

for (int i = 0; i < curves; ++i) {

size\_t index = rand() % data.all;

while (material[index] == 0) {

index = rand() % data.all;

}

material[index] = 0;

}

// Теперь из data.all булавок кривые data.curve булавок (0), остальные нормальные (1)

int b; // Булавка

int sum = 0; // Переменная для подсчета результатов третьим процессом

omp\_set\_dynamic(0);

omp\_set\_num\_threads(3);

#pragma omp parallel

{

for (size\_t i = 0; i < data.all; ++i) {

b = material[i];

if (omp\_get\_thread\_num() == 0) {

b = worker1(b);

}

#pragma omp barrier

if (omp\_get\_thread\_num() == 1) {

b = worker2(b, data.rate);

}

#pragma omp barrier

if (omp\_get\_thread\_num() == 2) {

worker3(b, &sum);

}

#pragma omp barrier

}

}

std::cout << "result: " << sum << std::endl;

return 0;

}