Код первого упражнения:

#include <iostream>

#include <cstdlib>

#include <fstream>

#include <sstream>

#include <string>

#include <vector>

#include <algorithm>

#include <pthread.h>

using namespace std;

/\*Розробити програму, яка використовуючи потоки POSIX буде паралельно зчитувати дані із

заданого переліку джерел даних та обчислювати їх кількісні характеристики. Для цього

необхідно заздалегідь підготувати файли з текстовою та чисельною інформацію. Потоки,

що оброблюють текстові файли повинні обчислити статистику цих файлів (кількість символів,

слів, знаків пунктуації, абзаців тощо). Потоки, що оброблюють числову інформацію відповідні

статистичні дані для числових рядів (середнє, максимальне та мінімальне значення, кількість

додатних і від'ємних чисел тощо). Кожен потік після завершення роботи повинен опублікувати

на екрані повідомлення з обчисленною інформацією.\*/

#define NUM\_THREADS 2

struct result{

int count;

int punct;

int paragraph;

int symbol;

};

struct num\_result{

int mid;

int max;

int min;

int negative;

int positive;

};

static pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

void \*Printstrstat(void \*t) {

string line;

result res = {0,0,0,0};

int len = strlen((char\*)t);

char \*str = new char[len + 1];

strcpy(str, (char\*)t);

fstream fs(str, std::fstream::in);

if (!fs.is\_open())

{

cout << "unable to open file" << str;

exit(-1);

}

char c;

while (getline(fs,line))

{

ss(line);

if (line.empty())

{

res.paragraph++;

}

char n;

while (ss.get(n)){

if (isalnum(n))

{

res.symbol++;

}

if (isblank(n)){

res.count++;

}

if (ispunct(n)){

res.punct++;

}

}

}

pthread\_mutex\_lock(&mutex);

cout << "Total number of symbols in " << str << " is " << res.symbol << endl;

cout << "Average number of punctuation signs in"<<str<<" is " << res.punct << endl;

cout << "number of spaces is " << res.count << endl;

cout << "number of paragraphs is " << res.paragraph << endl;

fs.close();

pthread\_mutex\_unlock(&mutex);

pthread\_exit(NULL);

return NULL;

}

bool isNegative(int i){ return (i < 0); }

bool isPositive(int i){ return (i >= 0); }

void \*PrintNumStat(void \*t) {

string line;

int sum = 0;

vector<int>num\_vector;

num\_result resNum = { 0, 0, 0,0,0 };

int len = strlen((char\*)t);

char \*str = new char[len + 1];

strcpy(str, (char\*)t);

fstream fs(str, std::fstream::in);

if (!fs.is\_open())

{

cout << "unable to open file" << str;

exit(-1);

}

char c;

int number;

while (getline(fs, line))

{

stringstream ss(line);

while (ss>>number){

num\_vector.push\_back(number);

}

}

vector<int>::iterator itr= min\_element(num\_vector.begin(), num\_vector.end());

resNum.min = \*itr;

itr = max\_element(num\_vector.begin(), num\_vector.end());

resNum.max = \*itr;

resNum.negative = count\_if(num\_vector.begin(), num\_vector.end(), isNegative);

resNum.positive = count\_if(num\_vector.begin(), num\_vector.end(), isPositive);

for (auto&x : num\_vector)

{

sum += x;

}

/\*середнє, максимальне та мінімальне значення, кількість

додатних і від'ємних чисел тощо\*/

pthread\_mutex\_lock(&mutex);

cout << "Average number in "<<str<<" is " <<resNum.mid << endl;

cout << "Maximum number in "<<str<<" is " <<resNum.max << endl;

cout << "Number of positive integers in "<<str<<" is " << resNum.positive<< endl;

cout << "Number of negative integers in "<<str<<" is " << resNum.negative << endl;

cout << "The average number in "<<str<<" is " <<sum/num\_vector.size()<< endl;

fs.close();

pthread\_mutex\_unlock(&mutex);

pthread\_exit(NULL);

return NULL;

}

int main() {

pthread\_t tid1;

pthread\_t tid2;

pthread\_t did1;

pthread\_t did2;

int rc;

int rb;

int dc;

int db;

int i;

pthread\_attr\_t attr;

void \*status;

char\* file1 = "test.txt";

char\* file2 = "test2.txt";

char\* file3 = "number.txt";

char \*file4 = "number2.txt";

pthread\_attr\_init(&attr);

pthread\_attr\_setdetachstate(&attr, PTHREAD\_CREATE\_JOINABLE);

rc = pthread\_create(&tid1, &attr, Printstrstat, (void \*)file1);

if (rc) {

cout << "Error:unable to create thread," << rc << endl;

exit(-1);

}

rb = pthread\_create(&tid2, &attr, Printstrstat, (void \*)file2);

if (rb) {

cout << "Error:unable to create thread," << rb << endl;

exit(-1);

}

dc = pthread\_create(&did1, &attr, PrintNumStat, (void \*)file3);

if (rc) {

cout << "Error:unable to create thread," << rc << endl;

exit(-1);

}

db = pthread\_create(&did2, &attr, PrintNumStat, (void \*)file4);

if (rb) {

cout << "Error:unable to create thread," << rb << endl;

exit(-1);

}

pthread\_attr\_destroy(&attr);

pthread\_join(tid1, &status);

pthread\_join(tid2, &status);

pthread\_join(did1, &status);

pthread\_join(did2, &status);

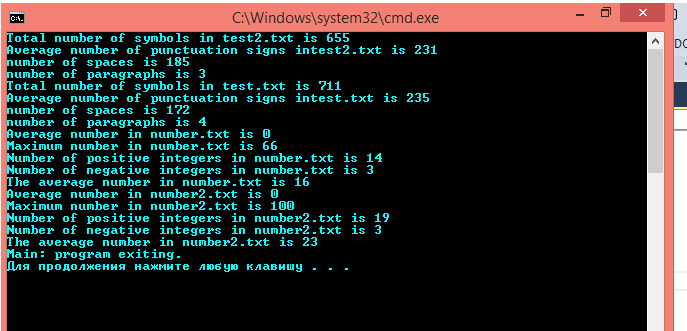
cout << "Main: program exiting. " << endl;

pthread\_mutex\_destroy(&mutex);

system("pause");

}

Скриншот результатов программы:



Описание программы:

Сначала, поключаем бибилиотеку #include <pthread.h>.

Согдаем глобальную структуру result в которой будет храниться информация по файлам, где текстовая информация.

Далее, создаем глобальную структуру num\_result,где будет храниться информация о числовом файле.

Далее, содается статическая переменная мьютекс mutex и статически инициализируется.

После этого, создаем функцию Printstrstat, которая будет аргументом в функции создания потока в главной функции. Для быборки информации из файла, создаем объект строку line. Инициализируем структуру result res нулями, чтобы выделить память под переменные. Создаем локальный массив char переменных str. Копируем в него агрумент функции, что является названием файла.

Инициализируем объект fstream fs, который будет работат с файлом, название которого храниться в символьном массиве str.

Проверяем на ошибку открытия файла, и в случае ошибки предусматриваем выходи из функции.

В цикле while происходит считывание построчно информации сначала в строку (объект String), а потом происходит инициализация объекта потока stringstream, благодаря котрому мы можем считывать информацию из строки файла посимвольно.

Во вложенном цикле while происходит анализ каждого символа на предмет:

1.Является ли элемент потока символом. Если да, то инкрементируется переменная symbol в локальной структуре.

2. Если элемент потока—знак, то соответственно инкрементируется переменная punct.

3. Если элемент потока—пробел, то инкрементируется переменная count.

4. Перед внутренним циклом делается проверка на пустую строку, т.е. на наличие абзаца.

Для того, чтобы процессы не конфликтовали друг с другом при доступе к ресурсу консоль, предусмотрено блокирование мьютекса (pthread\_mutex\_lock (&mutex)).После того, как один из прецессов выведет свою информацию на экран, мьютекс разблокируется, и другой процесс сможет получить доступ к консоли.

Далее, описываются две вспомогательные функции isNegative и isPositive, которые позволят использовать функцию из библиотеки STL algorithm.

Далее описывается функция PrintNumStat.

Первая часть функции такая же, как и у функции Printstrstat, но при обработке файловой информации, предусмотрено добавление информации из файла в вектор num\_vector. Далее идет поиск минимального и максимального элементов с помощью функцій min\_element и max\_element.

Используется итератор.

Далее, с помощью функций count\_if и вспомогательных функций isNegative и isPositive находится количество позитивных и негативных чисел. Далее, через цикл for происходит вычисление суммы всех чисел в файле, загруженных в вектор.

Далее, с помощью блокировки мьютекса, предоставляется консоль для одного потока, чтобы вывести на экран результаты анализа.

Закрывается файловый поток, разблокируется мьютекс и закрывается поток.

В главной функции создаются 4 потока, для работы с двумя файлами по два потока на каждый.

Инициализируются динамически атрибут для потоков, а позже присваиваются атрибуту характеристики, что поток будет JOINABLE.

Далее делается проверка при создании всех 4-х потоков на предмет ошибки. В случае ошибки предусмотрен выходи из программы.

После создания потоков, через pthread\_join предусматривается, что функция main будет ждать окончания работы потоков.

**Код упражнения номер 2:**

#include <iostream>

#include <cstdlib>

#include <fstream>

#include <sstream>

#include <string>

#include <vector>

#include <algorithm>

#include <pthread.h>

#define THREAD\_NUMBER 2

using namespace std;

fstream fs;

fstream ff;

/\*Розробити програму, яка використовуючи два паралельні потоки зчитає збережені

у файли дві матриці. Після завантаження матриць у пам'ять комп'ютера необхідно

знайти їх матричний добуток з використанням заданої користувачем кількості потоків.\*/

vector<vector<int> >first\_v;

vector<vector<int> >second\_v;

int \*array1 = new int[];

static pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

void \*GetMatrix(void \*t) {

string line;

int sum = 0;

int len = strlen((char\*)t);

char \*str = new char[len + 1];

strcpy(str, (char\*)t);

if (strcmp(str, "matrix1.txt") == 0){

if (!ff.is\_open()){

ff.open(str, std::fstream::in);

if (!ff.is\_open())

{

cout << "unable to open file" << str;

exit(-1);

}

char c;

int number;

pthread\_mutex\_lock(&mutex);

while (getline(ff, line))

{

stringstream ss(line);

vector<int>temp;

while (ss >> number){

temp.push\_back(number);

}

first\_v.push\_back(temp);

temp.erase(temp.begin(), temp.end());

}pthread\_mutex\_unlock(&mutex);

}

}

else {

if (!fs.is\_open()){

fs.open(str, std::fstream::in);

if (!fs.is\_open())

{

cout << "unable to open file" << str;

exit(-1);

}

char c;

int number;

pthread\_mutex\_lock(&mutex);

while (getline(fs, line))

{

stringstream ss(line);

vector<int>temp;

while (ss >> number){

temp.push\_back(number);

}

second\_v.push\_back(temp);

temp.erase(temp.begin(), temp.end());

}pthread\_mutex\_unlock(&mutex);

}

}

pthread\_exit(NULL);

return NULL;

}

int main() {

pthread\_t mythreads[THREAD\_NUMBER];

int rc;

pthread\_attr\_t attr;

void \*status;

char \* files[2] = { "matrix1.txt", "matrix2.txt" };

pthread\_attr\_init(&attr);

pthread\_attr\_setdetachstate(&attr, PTHREAD\_CREATE\_JOINABLE);

for (int i = 0,j=0; i < THREAD\_NUMBER; i++,j++)

{

rc = pthread\_create(&mythreads[i], &attr, GetMatrix, (void \*)files[j]);

if (rc) {

cout << "Error:unable to create thread," << rc << endl;

exit(-1);

} if (j == 1){ j = 0; }

}

pthread\_attr\_destroy(&attr);

for (int i = 0; i < THREAD\_NUMBER; i++)

{

pthread\_join(mythreads[i], &status);

}

ff.close();

if (fs.is\_open()){ fs.close(); }

pthread\_mutex\_destroy(&mutex);

cout << "Here is the first matrix: " << endl;

vector<vector<int> >::iterator itr;

vector<int>::iterator inside;

for (itr = first\_v.begin(); itr != first\_v.end(); itr++)

{

for (inside = itr->begin(); inside != itr->end(); inside++)

{

cout << \*inside << " ";

}

cout << endl;

}

cout << endl;

cout << "Below is the second matrix: " << endl;

for (itr = second\_v.begin(); itr != second\_v.end(); itr++)

{

for (inside = itr->begin(); inside != itr->end(); inside++)

{

cout << \*inside << " ";

}

cout << endl;

}

int row = first\_v.size();

int col = second\_v[0].size();

vector<vector<int> >multiple;

for (int i = 0; i < row; i++)

{ vector<int>temp;

for (int j = 0; j < col; j++)

{

temp.push\_back(0);

}multiple.push\_back(temp);

}

if (first\_v[0].size() == second\_v.size()){

for (int i = 0; i < row; i++)

{

for (int j = 0; j < col; j++)

{

for (int k = 0; k < second\_v.size(); k++)

{

multiple[i][j] += first\_v[i][k] \* second\_v[k][j];

}

}

}

}

else { cout << "Can't multiply matrices.";

exit(-1);

}

cout << "The product of two matrices is as follows: " << endl;

for (auto&x : multiple){

for (auto&y : x)

{

cout << y << " ";

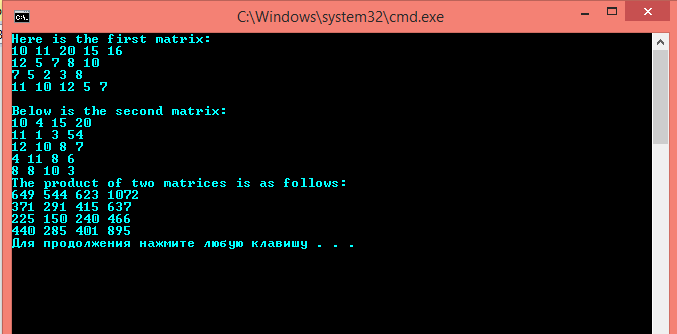
}cout << endl;

}

system("pause");

}

Ниже представлен результат на консоли:



**Описание программы:**

В начале .cpp файла подключаются файлы библиотек для препроцессора. Среди них файл <pthread.h> для работы с потоками.

Создается два файловых потока fs, ff глобально. Далее создается два вектора для загрузки данных из файлов. Размер векторов не предусмотрен, т.к. будет меняться динамически в зависимости от наполнения файла.

Далее создается мьютекс глобально и инициализируется статически.

После этого, объявляется и определяется функция GetMatrix, которая обрабатывает файлы с матрицами и сохраняет результат в глобальных векторах. В начале функции, через функции библиотеки string (strlen, strcpy) находится длина названия файла, что передается в аргумент, и создается символьный массив для локального хранения названия файла.

В условии if проверяется не открыт ли файл, т.к. несколько потоков будут постоянно открывать его и считывать в вектора, что нам не нужно.

Если файл закрыт, что происходит открытие файла с данными "matrix1.txt" и далее чтение его через метод getline класса fstream. Далее в потоке stringstream ss происходит анализ данных. Перегруженный оператор >> позволяет пропускать пробелы и считывать данные типа int. При чтении каждой строки создается вектор колонок и после цикла while вектор добавляется в глобальный вектор первой матрицы. Метод erase очищает временный вектор после окончания обработки каждой строки.

Далее, разблокируется мьютекс.

Перед обработкой первого файла, ставиться условие на аргумент функции. Функцией из библиотеки string.h strcmp(str, "matrix1.txt") == 0 делается проверка на то, что аргумент является первым файлом. Если же в аргументе второй файл, что блок кода в первом условии не исполняется, и поток в аргументе метода которого есть второй файл, обрабатывает второй блок кода.

Если файл открыт, что другие потоки, файл не открывают повторно. Далее исполняется код идентичный предыдущему блоку.

В главной функции создается массив потоков, где размер указывается пользователем через определение препроцессора THREAD\_NUMBER.

Создаются переменные для записи результата работы функций по созданию потоков. Создается переменная атрибутов потока. Создается указатель на статус для функции pthread\_join. Далее создается массив указателей на символьные массивы с названием файлов.

Инициализируется динамически аттрибут и устанавливается через pthread\_attr\_setdetachstate функцию того, чтобы потоки были JOINABLE.

Через массив создается потоки, а также делается проверка на создание потоков и в случае ошибки выходи из программы.

Удаляется атрибут и через массив указываются потоки, для которых главная функция будет остановлена.

Закрываются через метод fstream close() файлы открытые в потоках. Это нужно для того, чтобы делать проверку на открытие в функции потоков. Удаляется мьютекс.

После этого, идет блок кода для обработки двух векторов. В этом коде суммируется два вектора и результат вносится в двумерный вектор multiple.

Сначала создаются и инициализируются переменные для рядов и колонок матрицы int row, int col. По условию умножения векторов, количество рядов матрицы должно равняться рядам первого множителя, а количество колонок, колонкам второго множителя. Все элементы двумерного вектора multiple инициализируются нулями, т.к. в цикле происходит обращение к ним.

Далее используется трехуровневый вложенный цикл for для умножения векторов. Результат умножения выводиться на консоль.

Размер вектора можно менять с помощью константы THREAD\_NUMBER.