**Задача 1. Робота з декількома джерелами даних**

Розробити програму, яка використовуючи потоки C++11 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.

Код:

#include <iostream>

#include <cstdlib>

#include <fstream>

#include <sstream>

#include <string>

#include <vector>

#include <algorithm>

#include <thread>

#include <mutex>

using namespace std;

std::mutex g\_lock;

struct result{

int count;

int punct;

int paragraph;

int symbol;

};

struct num\_result{

int mid;

int max;

int min;

int negative;

int positive;

};

void Printstrstat(char t[]) {

string line;

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

int len = strlen(t);

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

strcpy(str, 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))

{

stringstream 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++;

}

}

}

g\_lock.lock();

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;

g\_lock.unlock();

fs.close();

}

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

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

void PrintNumStat(char t[]) {

string line;

int sum = 0;

vector<int>num\_vector;

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

int len = strlen(t);

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

strcpy(str, 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\*/

g\_lock.lock();

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;

g\_lock.unlock();

fs.close();

}

int main() {

char\* file1 = "test.txt";

char\* file2 = "test2.txt";

char\* file3 = "number.txt";

char \*file4 = "number2.txt";

thread \*tid1;

thread \*tid2;

thread \*did1;

thread \*did2;

tid1 = new thread(Printstrstat, file1);

tid2 = new thread(Printstrstat, file2);

did1 = new thread(PrintNumStat, file3);

did2 = new thread(PrintNumStat, file4);

tid1->join(); tid2->join(); did1->join(); did2->join();

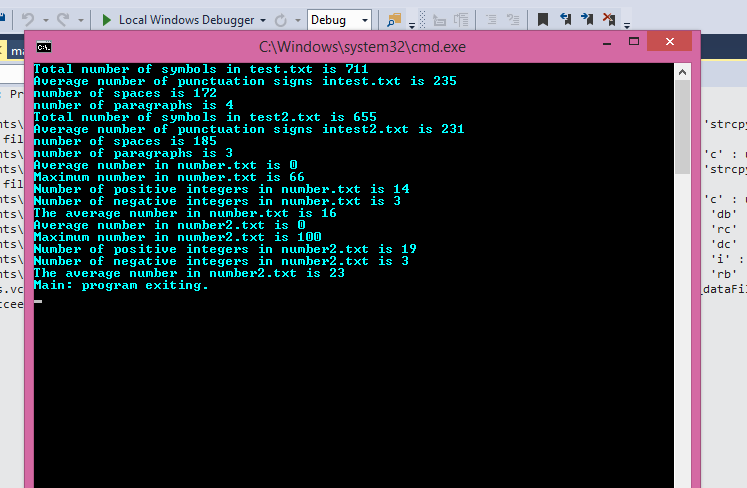
cout << "Main: program exiting. " << endl;

getchar();

system("pause");

return 0;

}



**Пояснения:**

В данной программе происходит распараллеливание работы по считыванию информации из 4 фалов. В первых двух содержится текстовая информация, а во вторых двух содержится информация о данных.

В программе в директивах препроцессора используются библиотеки по работе с потоками: <thread> и <mutex>.

Создается две структуры: в первой поля будут хранить информацию о текстовом файле, а во второй—о файле с данными. Функция Printstrstat() обрабатывает файл, который будет передан в качестве аргумента(char t).

Вначале функции создается массив char, в который копируется информация из аргументе(т.е. название файла). Далее, предполагаемый поток, открывает файл, через объект класса потока fstream. Название файла передается в конструктор потока и определяется в флаге (fstream::in), что информация для чтения данных. Далее, идет проверка на ошибки при открытии. После этого, идет чтение файла с помощью метода getline() потока fstream. Для анализа строк информации, создается объект stringstream ss класса stringstream, в конструкторе которого используется считанная строка. Идет посимвольный анализ символов через условный оператор if. Сначала анализируется информация о символах, далее о пробелах, символах табуляции, и далее о знаках препинания. Вся информация записывается потоком в локальную структуру, т.к. задача потока состоит в том, чтобы вывести информацию на экран. Поток будет обрабатывать файл, пока метод getline() потока fstream возвращается не Null.

После анализа кода и инициализации всех переменных структуры res происходит вывод информации в консоль. Поскольку консоль является общим ресурсом, без блокировки мьютекса, информация будет выводится произвольно. Поэтому перед как использовать ресурс cout мьютекс g\_lock(глобально созданный) блокируется. После вывода информации на экран, мьютекс освобождается.

Для работы с файлами цифровых данный используется функция PrintNumStat(). В данную функцию передается ссылка на массив с названием файла.

Так же как и с предыдущей функцией, создается локальная структура для хранения анализа информации. Отрывается файл. Для этого создается локальный поток fstream fs. Идет проверка на правильность отработки функции открытия файлового потока. Далее в цикле while (getline(fs, line)) считывание информации и запись ее в вектор<int> num\_vector для анализа.

После выхода из цикла обработки файла, с использованием библиотеки algorithm STL идет анализ информации вектора. Ищется максимальный элемент max\_element(), минимальный элемент min\_element(), с помощью функции count\_if(), а также вспомогательных функций isNegative() и isPositive() находиться количество положительных и отрицательных чисел. Для доступа к общему ресурсу cout блокируется мьютекс g\_lock.lock(), а по окончании вывода информации в консоль, он освобождается.

В функции main создаются массивы с названием файлов. Далее создаются указатели на названия потоков. Далее инициализируются указатели и создаются потоки, в которые передаются функции и названия файлов.

Далее вызываются для каждого потока методы join(), чтобы убедиться, что функция main не закончиться раньше, чем закончатся потоки.

Результат программы показан на скриншоте.

**Задача 2. Робота зі спільною областю пам'яті**

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.

Код:

#include <iostream>

#include <cstdlib>

#include <fstream>

#include <sstream>

#include <string>

#include <vector>

#include <algorithm>

#include <thread>

#include <mutex>

#include <condition\_variable>

#define THREAD\_NUMBER 2

using namespace std;

condition\_variable cv;

fstream fs;

fstream ff;

vector<vector<int> >first\_v;

vector<vector<int> >second\_v;

bool DataLoaded\_one = false;

bool DataLoaded\_two = false;

int \*array1 = new int[];

std::mutex g\_lock;

std::mutex m;

bool isDataLoaded()

{

return DataLoaded\_one&&DataLoaded\_two;

}

void GetMatrix(char t[]) {

string line;

int sum = 0;

int len = strlen(t);

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

strcpy(str, 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;

g\_lock.lock();

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());

}g\_lock.unlock();

DataLoaded\_one = true;

}

}

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;

g\_lock.lock();

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());

}g\_lock.unlock();

}

DataLoaded\_two = true;

}

}

void mainfunct();

int main() {

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

vector<std::thread\*> threadList;

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

{

threadList.push\_back( new thread(GetMatrix, files[j]));

}

thread\*third = new thread(mainfunct);

if (third->joinable())

third->join();

for\_each(threadList.begin(), threadList.end(), [](thread\*thread){if (thread->joinable()){ thread->join(); } });

ff.close();

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

getchar();

return 0;

}

void mainfunct(){

unique\_lock<std::mutex>lk(m);

cv.wait(lk, []{return isDataLoaded(); });

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 << " ";

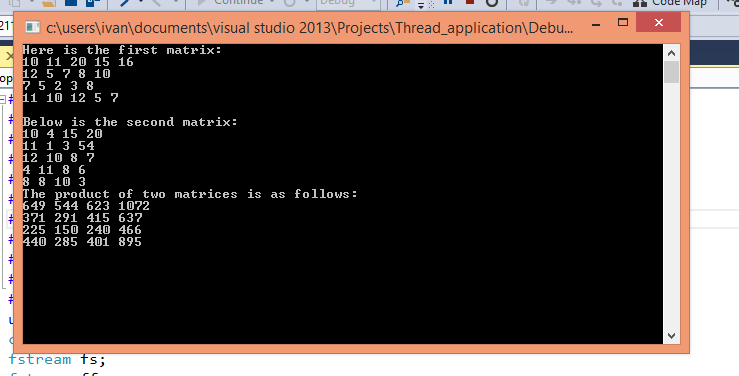
}cout << endl;

}

lk.unlock();

cv.notify\_all();

}



Пояснения:

В данной программе используются библиотеки <thread>, <mutex> и <condition\_variable>.

Данные библиотеки нужны для создания потоков, для блокировки доступа к общему ресурсу, а также для синхронизации работы потоков с помощью условных переменных.

В глобальной области создаются: условная переменная condition\_variable cv, потоки (для исключения одновременного открытия файлов несколькими потоками одновременно), создаются два двумерных вектора first\_v, second\_v. Также создается два флага для проверки условия окончания работы потоков с датой. Далее создается два мьютекса

std::mutex g\_lock, std::mutex m для блокировки потоков. Для проверки флагов используется функция isDataLoaded(), которая передается позже в функцию wait();

Основная работа идет в функции GetMatrix(char t[]). Функция разбита условными операторами if и else на две части, в зависимости от того, с каким из файлов потоки работают.

В функцию передается название файлы. В операторе if (strcmp(str, "matrix1.txt") == 0) идет сравнение названия файла с "matrix1.txt", и если сравнение дает 0(т.е. массивы равны), то поток обрабатывает кусок кода в блоке оператора. В противном случае, идет код для анализа информации по второму файлу.

Далее анализируется открыт ли поток, который объявлен глобально. Если да, то он уже не открывается. Это исключает анализ кода повторно другими потоками.

В условном цикле while идет считывание информации сначала в буфер line, а позже буфер line (String класс ) переходит в поток stringstream для записи чисел в временный вектор temp, который по выходу из второго цикла while (ss >> number) добавляется в вектор first\_v. После завершения обработки информации, поток устанавливает переменную DataLoaded\_one в true. Это делается для того, чтобы передать «отпустить» поток, который обрабатывает результирующие данные.

Во второй части условного оператора, идет анализ информации из второго файла. Запись производиться в двумерный вектор second\_v. По завершению работы с вектором, поток устанавливает DataLoaded\_two в true.

Для работы с глобальными ресурсами в функции GetMatrix используется блокировка мьютекса g\_lock.

Работа по анализу информации в двух файлах передается потоками в два двумерных вектора first\_v, second\_v. Работу с результирующими данными осуществляет третий поток, который обрабатывает функциию mainfunct().

В данной функции происходит блокировка потока unique\_lock<std::mutex>lk(m). Далее устанавливается ожидание потоком выполнения функцией wait. Условием выхода из ожидания ставиться возврат функцией isDataLoaded() значения true, что возможно только после завершения анализа файлов и записи данных в вектора first\_v, second\_v.

Функция mainfunct() выводит в консоль содержание двух матриц из файлов. Далее происходит умножение этих матриц в трех вложенных циклах for (). Результат записывается в вектор multiple. Также делается проверка на соответствие матриц условию умножения.

В функции main происходит создание потоков, указатели на которые помещаются в вектор threadList. Далее функцией join() обеспечивается завершение работы потоков до завершения main.

Количество потоков можно менять с помощью статической директивы THREAD\_NUMBER . Оптимальным является три потока.

Результаты работы программы выведены на скриншоте.