Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

Студент: Юсупов Артем Маратович

Группа: М8О-209Б-23

Вариант: 11

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в:

1. Управление процессами в ОС
2. Обеспечение обмена данных между процессами посредством каналов

**Задание**

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение максимального количества потоков, работающих в один момент времени, должно быть задано ключом запуска вашей программы.

Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

В отчете привести исследование зависимости ускорения и эффективности алгоритма от входных данных и количества потоков. Получившиеся результаты необходимо объяснить.

11. Наложить K раз медианный фильтр на матрицу, состоящую из целых чисел. Размер окна задается пользователем

**Общие сведения о программе**

Программа представляет собой однопроцессное приложение, реализованное в файле program.c, и предназначена для многопоточной обработки матрицы целых чисел с применением медианного фильтра. Она работает исключительно в среде Windows и использует потоки для параллельной обработки данных. Максимальное количество одновременно работающих потоков задается пользователем через аргумент командной строки при запуске программы. Результаты обработки выводятся на экран, а количество активных потоков можно проверить с помощью стандартных средств Windows (Диспетчер задач).

Программа использует следующие заголовочные файлы:

* windows.h — для работы с потоками и системными функциями Windows.
* stdio.h — для ввода-вывода данных (scanf, printf).
* stdlib.h — для выделения памяти (malloc, free) и преобразования строк (atoi).
* string.h — для работы с массивами данных (memcpy).

В программе используются следующие системные вызовы и функции:

1. **CreateThread** — создает новый поток для параллельной обработки части матрицы.
2. **WaitForMultipleObjects** — ожидает завершения всех созданных потоков.
3. **CloseHandle** — закрывает дескрипторы потоков после их завершения.
4. **malloc** — выделяет динамическую память для матриц и временных массивов.
5. **free** — освобождает выделенную память.
6. **scanf** — считывает ввод пользователя (размеры матрицы, элементы и параметры).
7. **printf** — выводит результаты обработки на экран.
8. **atoi** — преобразует строковый аргумент командной строки (количество потоков) в целое число.

Программа не использует межпроцессное взаимодействие или файлы для хранения данных, вся обработка происходит в памяти, а результат отображается в консоли. Пользователь задает размеры матрицы, размер окна фильтра, количество итераций фильтрации и исходные элементы матрицы, после чего программа выполняет обработку и выводит итоговую матрицу.

**Общий метод и алгоритм решения**

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы многопоточной обработки в Windows с использованием функций **CreateThread, WaitForMultipleObjects** и **CloseHandle**, а также основные операции выделения памяти и ввода-вывода (**malloc, free, scanf, printf**).
2. Написать одну программу program.exe, которая:
   * Получает от пользователя максимальное количество потоков через аргумент командной строки.
   * Запрашивает у пользователя размеры матрицы, размер окна медианного фильтра, количество итераций фильтрации и элементы матрицы.
   * Выполняет многопоточную обработку матрицы с применением медианного фильтра и выводит результат.

**Исходный код**

main.c

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <windows.h>**

**typedef struct {**

**int\*\* matrix;**

**int\*\* result;**

**int rows;**

**int cols;**

**int window\_size;**

**int start\_row;**

**int end\_row;**

**} ThreadData;**

**int getMedian(int\* window, int size) {**

**int\* temp = (int\*)malloc(size \* sizeof(int));**

**memcpy(temp, window, size \* sizeof(int));**

**for (int i = 0; i < size - 1; i++) {**

**for (int j = 0; j < size - i - 1; j++) {**

**if (temp[j] > temp[j + 1]) {**

**int t = temp[j];**

**temp[j] = temp[j + 1];**

**temp[j + 1] = t;**

**}**

**}**

**}**

**int median = temp[size / 2];**

**free(temp);**

**return median;**

**}**

**DWORD WINAPI processSection(LPVOID param) {**

**ThreadData\* data = (ThreadData\*)param;**

**int half\_window = data->window\_size / 2;**

**for (int i = data->start\_row; i < data->end\_row; i++) {**

**for (int j = 0; j < data->cols; j++) {**

**int window\_size = data->window\_size \* data->window\_size;**

**int\* window = (int\*)malloc(window\_size \* sizeof(int));**

**int k = 0;**

**for (int di = -half\_window; di <= half\_window; di++) {**

**for (int dj = -half\_window; dj <= half\_window; dj++) {**

**int ni = i + di;**

**int nj = j + dj;**

**if (ni >= 0 && ni < data->rows && nj >= 0 && nj < data->cols) {**

**window[k++] = data->matrix[ni][nj];**

**} else {**

**window[k++] = 0;**

**}**

**}**

**}**

**data->result[i][j] = getMedian(window, window\_size);**

**free(window);**

**}**

**}**

**return 0;**

**}**

**int main(int argc, char\* argv[]) {**

**if (argc != 2) {**

**printf("Usage: %s <max\_threads>\n", argv[0]);**

**return 1;**

**}**

**int max\_threads = atoi(argv[1]);**

**int rows, cols, window\_size, k;**

**printf("Enter matrix rows: ");**

**scanf("%d", &rows);**

**printf("Enter matrix columns: ");**

**scanf("%d", &cols);**

**printf("Enter window size (odd number): ");**

**scanf("%d", &window\_size);**

**printf("Enter number of filter applications: ");**

**scanf("%d", &k);**

**int\*\* matrix = (int\*\*)malloc(rows \* sizeof(int\*));**

**int\*\* temp = (int\*\*)malloc(rows \* sizeof(int\*));**

**int\*\* result = (int\*\*)malloc(rows \* sizeof(int\*));**

**for (int i = 0; i < rows; i++) {**

**matrix[i] = (int\*)malloc(cols \* sizeof(int));**

**temp[i] = (int\*)malloc(cols \* sizeof(int));**

**result[i] = (int\*)malloc(cols \* sizeof(int));**

**}**

**printf("Enter matrix elements:\n");**

**for (int i = 0; i < rows; i++) {**

**for (int j = 0; j < cols; j++) {**

**scanf("%d", &matrix[i][j]);**

**temp[i][j] = matrix[i][j];**

**}**

**}**

**for (int iteration = 0; iteration < k; iteration++) {**

**int rows\_per\_thread = rows / max\_threads;**

**ThreadData\* thread\_data = (ThreadData\*)malloc(max\_threads \* sizeof(ThreadData));**

**HANDLE\* threads = (HANDLE\*)malloc(max\_threads \* sizeof(HANDLE));**

**for (int t = 0; t < max\_threads; t++) {**

**thread\_data[t].matrix = temp;**

**thread\_data[t].result = result;**

**thread\_data[t].rows = rows;**

**thread\_data[t].cols = cols;**

**thread\_data[t].window\_size = window\_size;**

**thread\_data[t].start\_row = t \* rows\_per\_thread;**

**thread\_data[t].end\_row = (t == max\_threads - 1) ? rows : (t + 1) \* rows\_per\_thread;**

**threads[t] = CreateThread(NULL, 0, processSection, &thread\_data[t], 0, NULL);**

**}**

**WaitForMultipleObjects(max\_threads, threads, TRUE, INFINITE);**

**for (int t = 0; t < max\_threads; t++) {**

**CloseHandle(threads[t]);**

**}**

**for (int i = 0; i < rows; i++) {**

**for (int j = 0; j < cols; j++) {**

**temp[i][j] = result[i][j];**

**}**

**}**

**free(thread\_data);**

**free(threads);**

**}**

**printf("\nResult after %d iterations:\n", k);**

**for (int i = 0; i < rows; i++) {**

**for (int j = 0; j < cols; j++) {**

**printf("%d ", result[i][j]);**

**}**

**printf("\n");**

**}**

**for (int i = 0; i < rows; i++) {**

**free(matrix[i]);**

**free(temp[i]);**

**free(result[i]);**

**}**

**free(matrix);**

**free(temp);**

**free(result);**

**return 0;**

**}**

**Демонстрация работы программы**

**PS C:\Development\c++\os\lab2\build\Debug> .\main.exe 4**

**Enter matrix rows: 4**

**Enter matrix columns: 4**

**Enter window size (odd number): 3**

**Enter number of filter applications: 2**

**Enter matrix elements:**

**1 2 3 4**

**5 6 7 8**

**9 10 11 12**

**1 1 1 1**

**Result after 2 iterations:**

**0 2 2 0**

**1 3 4 1**

**1 2 4 1**

**0 1 1 0**

**Выводы**

Поток — это единица выполнения внутри процесса, которая имеет собственный стек и состояние выполнения, но разделяет общие ресурсы процесса, такие как память и дескрипторы. В данной программе создание потоков осуществляется с помощью системного вызова CreateThread(), который запускает новый поток внутри текущего процесса, передавая ему функцию обработки и параметры, такие как участок матрицы для вычислений. Таким образом, после запуска программа разделяет обработку данных на несколько параллельных ветвей — потоков, которые работают одновременно над общей памятью матрицы.

Многопоточная обработка реализована следующим образом:

* Основной поток запрашивает у пользователя входные данные: размеры матрицы, размер окна фильтра, количество итераций и элементы матрицы.
* Затем он создает заданное пользователем количество рабочих потоков (через аргумент командной строки), распределяя строки матрицы между ними.
* Каждый поток применяет медианный фильтр к своей части матрицы, используя общую память для исходных и результирующих данных.

Медианный фильтр реализован с учетом скользящего окна:

* Для каждого элемента матрицы собираются значения в окне заданного размера.
* При выходе окна за пределы матрицы недостающие элементы либо заполняются нулями (в текущей версии), либо игнорируются (в исправленной версии).
* Значения сортируются, и медиана записывается в результирующую матрицу.

Синхронизация потоков осуществляется с помощью WaitForMultipleObjects(), который обеспечивает ожидание завершения всех рабочих потоков перед переходом к следующей итерации или выводу результата. После завершения обработки все дескрипторы потоков закрываются с помощью CloseHandle(), а выделенная память освобождается.

Таким образом, в ходе выполнения данной задачи были получены знания и практические навыки управления потоками в Windows, организации параллельной обработки данных и реализации медианного фильтра в многопоточном режиме. Программа демонстрирует влияние подхода к обработке краев матрицы на итоговый результат, что подчеркивает важность выбора стратегии в задачах цифровой обработки сигналов.