Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа № 3 по курсу

«Операционные системы»

Процессы и потоки

Студент: Пирогов М.Д.
Группа: М8О-207Б-21
Вариант: 10
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/pirogovmark/OS-Labs

Постановка задачи

Цель работы

Целью является приобретение практических навыков в:

- Управление потоками в ОС
- Обеспечение синхронизации между потоками

Задание

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение потоков 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 входящих данных и количества потоков. Получившиеся результаты необходимо объяснить.

Вариант 10) Наложить K раз медианный фильтр на матрицу, состоящую из целых чисел. Размер окна задается.

Общие сведения о программе

Программа компилируется из файла main.cpp. Также используется заголовочные файлы: unistd.h, ...

Общий метод и алгоритм решения

Обработка матрицы будет распределяться по строкам для потоков. Каждую матрицу будем дополнять элементами, которые не будут влиять на подсчет “среднего” значения в области. Алгоритм медианного фильтрования: для каждого элемента в области NxN (N - odd) добавляем элементы из этой области в список, сортируем быстрой сортировкой и берем серединный элемент, который уже и записываем в новую матрицу.

Исходный код

```
#include <iostream>
#include <unistd.h>
#include <time.h>
#include <stdlib.h>
#include <pthread.h>


int const N = 20;
int matrix[20][20], new_matrix[20][20];
int window, frame;


typedef struct threadArguments{
   int numberOfThreads;
   int currentThread;
} threadArguments;


void quickSort(int *array, int low, int high) {
   int i = low;
   int j = high;
   int pivot = array[(i + j) / 2];
   int temp;

   while (i <= j) {
      while (array[i] < pivot) i++;
      while (array[j] > pivot) j--;
      if (i <= j) {
         temp = array[i];
         array[i] = array[j];
         array[j] = temp;
         ++i;
         --j;
      }
   }
```

```
    if (j > low) quickSort(array, low, j);
    if (i < high) quickSort(array, i, high);
}


void printAugMatrix(int matrix[N][N]) {
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            std::cout << matrix[i][j] << ' ';
        }
        std::cout << '\n';
    }
}


void printMatrix(int matrix[N][N]) {
    for (int i = frame; i < N - frame - 1; ++i) {
        for (int j = frame; j < N - frame - 1; ++j) {
            std::cout << matrix[i][j] << ' ';
        }
        std::cout << '\n';
    }
}


void fillMatrix() {
    srand(time(NULL));
    // Заполнение матрицы внутри
    for (int i = frame; i < N - frame; ++i) {
        for (int j = frame; j < N - frame; ++j) {
            matrix[i][j] = rand() % 9 + 1;
        }
    }
    // Дополнение матрицы для крайних элементов
    for (int i = 1; i < N - 1; ++i) {
        for (int j = 0; j < frame; ++j) {
```

```
            matrix[i][j] = matrix[i][frame];
            matrix[i][N - j - 1] = matrix[i][N - frame - 1];
        }
    }
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < frame; ++j) {
            matrix[j][i] = matrix[frame][i];
            matrix[N - j - 1][i] = matrix[N - frame - 1][i];
        }
    }
}


void resultToMatrix() {
    for (int i = frame; i < N - frame; ++i) {
        for (int j = frame; j < N - frame; ++j) {
            matrix[i][j] = new_matrix[i][j];
        }
    }
    for (int i = 1; i < N - 1; ++i) {
        for (int j = 0; j < frame; ++j) {
            matrix[i][j] = matrix[i][frame];
            matrix[i][N - j - 1] = matrix[i][N - frame - 1];
        }
    }
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < frame; ++j) {
            matrix[j][i] = matrix[frame][i];
            matrix[N - j - 1][i] = matrix[N - frame - 1][i];
        }
    }
}


int medianForElem(int i, int j) {
    int frame = window / 2;
```

```
    int sortSize = window * window - 1;
    int numbers[sortSize];
    int counter = 0;

    for (int l = i - frame; l <= i + frame; ++l) {
        for (int k = j - frame; k <= j + frame; ++k) {
            numbers[counter] = matrix[l][k];
            ++counter;
        }
    }
    quickSort(numbers, 0, sortSize - 1);

    return numbers[sortSize / 2];
}


void filterString(int stringNumber) {
    for (int j = frame; j < N - frame; ++j) {
        new_matrix[stringNumber][j] = medianForElem(stringNumber, j);
    }
}


void* threadFilter(void* arg) {
    threadArguments data = *((threadArguments*) arg);
    for (int i = data.currentThread; i < N; i += data.numberOfThreads) {
        filterString(i);
    }
}


int main() {
    int numberOfThreads;
    std::cout << "Enter the number of threads: ";
    std::cin >> numberOfThreads;
```

```
std::cout << "Enter window size: ";
std::cin >> window;
frame = window / 2;

int overlays;
std::cout << "Enter the number of overlays: ";
std::cin >> overlays;

fillMatrix();
printMatrix(matrix);
std::cout << '\n';

double start = clock();

for (int k = 0; k < overlays; ++k) {
    pthread_t threads[numberOfThreads];
    threadArguments* data = (threadArguments*)malloc(sizeof(threadArguments) * numberOfThreads);
    for (int i = 0; i < numberOfThreads; ++i) {
        data[i].currentThread = i;
        data[i].numberOfThreads = numberOfThreads;
    }
    for (int i = 0; i < numberOfThreads; ++i) {
        if (pthread_create(&threads[i], NULL, &threadFilter, &data[i]) != 0) {
            std::cout << "Failed to create thread\n";
            return 1;
        }
    }
    for (int i = 0; i < numberOfThreads; ++i) {
        if (pthread_join(threads[i], NULL) != 0) {
            std::cout << "Failed to join thread\n";
            return 1;
        }
    }
    resultToMatrix();
}
```

```
    std::cout << "\nNew matrix:\n";
    printMatrix(matrix);

    std::cout << "\nThe program ran for " << (clock() - start) / (CLOCKS_PER_SEC) << " seconds\n";

    return 0;
}
```

Демонстрация работы программы

```
markp@Air-Mark src % g++ main.cpp
main.cpp:136:1: warning: non-void function does not return a
 value [-Wreturn-type]
}
^
1 warning generated.
markp@Air-Mark src % ./a.out
Enter the number of threads: 5
Enter window size: 5
Enter the number of overlays: 5
9 2 4 5 9 4 5 9 9 5 1 2 4 4 2
4 9 2 6 5 3 7 2 8 7 9 9 3 3 2
4 9 5 8 9 6 3 3 3 8 3 7 9 9 3
6 8 5 7 8 6 2 7 4 3 1 7 2 1 5
7 9 7 4 9 8 3 8 3 9 9 2 5 6 8
8 1 9 9 3 6 6 9 1 9 4 3 1 7 7
3 8 7 7 2 9 6 9 3 9 1 2 2 8 4
2 3 4 5 5 7 5 9 1 5 2 2 6 7 7
1 7 7 3 7 7 5 6 2 2 8 8 4 3 4
8 8 4 9 1 7 7 9 5 7 5 4 6 3 5
4 9 3 1 2 9 1 3 8 9 1 6 4 3 2
9 8 1 3 7 5 1 9 4 9 7 7 4 9 7
7 4 6 8 2 7 5 8 4 1 1 2 3 6 5
5 6 4 2 9 9 5 9 3 5 8 3 8 1 4
2 9 3 1 4 2 5 4 4 1 1 8 3 1 8


New matrix:
5 5 5 5 5 5 5 5 5 5 5 5 5 4 4
6 6 6 6 5 5 5 5 5 5 5 5 5 4 4
6 6 6 6 6 6 5 5 5 5 5 5 5 5 4
7 7 6 6 6 6 6 6 5 5 5 5 5 5 5
7 7 7 7 6 6 6 6 5 5 5 5 5 5 5
7 7 7 7 7 6 6 6 6 5 5 5 5 5 5
7 7 7 7 7 6 6 6 6 5 5 5 5 5 5
7 7 7 7 7 6 6 6 6 5 5 5 5 5 5
7 7 7 7 6 6 6 6 6 5 5 5 5 5 5
7 7 7 6 6 6 6 6 6 5 5 5 5 5 5
5 6 6 6 6 6 6 6 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5 5 5 4 4 5

The program ran for 0.005714 seconds
```

Выводы

В результате выполнения этой лабораторной работы я познакомился с потоками в ОС, управлять ими и создавать многопоточные программы.