Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**Управление потоками в ОС. Обеспечение синхронизации между потоками.**

Студент: Ивенкова Л.В.

Группа: М80 – 208Б-19

Вариант: 10

Преподаватель: Миронов Е. С.

Дата: \_\_\_\_\_\_\_\_\_\_\_\_

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

1. **Постановка задачи**

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение потоков должно быть задано ключом запуска программы.

Необходимо уметь продемонстрировать количество потоков, используемых программой, с помощью стандартных средств операционной системы.

Привести исследование зависимости ускорения и эффективности алгоритма от входящих данных и количества потоков. Объяснить получившиеся результаты.

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

1. **Общие сведения о программе**

Программа написана на языке Си в UNIX-подобной операционной системе (Ubuntu). Для компиляции программы требуется указать ключ –pthread. Для запуска программы в качестве аргумента командной строки необходимо указать количество потоков, которые могут быть использованы программой.

Программа содержит две структуры: квадратную матрицу и аргументы для каждого потока.

Программа включает в себя потоковую функцию void\* thread\_func(void \*Args), которая накладывает медианный фильтр на строку исходной матрицу, причём у каждого потока своя определенная строка. Так как все потоки программы работают в одном и том же пространстве памяти, аргументы для передачи потоковой функции хранятся по разным адресам (в массиве, размер которого равен количеству потоков).

В программе предусмотрена проверка на системные ошибки – ошибки выделения памяти, ошибки запуска.

1. **Общий метод и алгоритм решения**

При запуске программы пользователю предлагается ввести размер матрицы, элементы матрицы, размер окна медианного фильтра и количество его наложений на введённую матрицу.

Основная идея алгоритма заключается в том, что каждый поток работает с определенной строкой матрицы, причём необходимо определить количество строк для каждого потока, найдя отношение порядка исходной матрицы и количества потоков, округленное в большую сторону.

Затем пробегаем по циклу threads\_amount раз, определяя левую и правую позиции, которые мы передаём каждому потоку для того, что он понимал, до какой границы необходимо накладывать медианный фильтр. Далее, в массив структур аргументов поток записываем найденные границы, исходную матрицу и новую результирующую матрицу. Создаём потоки и ждём окончания выполнения работы.

Сам алгоритм наложения матрицы заключается в следующем: накладываем матрицу размером m, на матрицу размером n, причём m < n. Те элементы второй матрицы, которые попали в накладываемое окно, сортируем в порядке возрастания, и выводим медиану массива — среднее из всех чисел. Делаем так столько раз, сколько указал пользователь при запуске программы.

**Исследование ускорения и эффективности от количества потоков.**

Si – ускорение, где i = количество потоков.

Si = T1/Ti

Ei – эффективность, где i = количество потоков.

Ei = Si / i

Ti – время работы алгоритма, где i = количество потоков

1. **Основные файлы программы**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

typedef struct {

int \*\*mat;

int order;

}matrix;

typedef struct{

matrix \*m;

matrix \*new\_m;

int begin;

int end;

int window\_size;

}Args;

int max(int a, int b){

return (a > b ? a : b);

}

int min(int a, int b){

return (a < b ? a : b);

}

void sort(int \*arr, int arr\_size){

for (int i = 0; i < arr\_size; i++){

for (int j = arr\_size - 1; j > 1; j--){

if (arr[j - 1] > arr[j]) {

int tmp = arr[j - 1];

arr[j - 1] = arr[j];

arr[j] = tmp;

}

}

}

}

void \*Median\_Filtr(void \*token) {

Args \*args = (Args\*)token;

int l = args->begin;

int r = args->end;

int ws = args->window\_size;

matrix \*M = args->m;

matrix \*new\_M = args->new\_m;

int \*tmp = (int\*) malloc(sizeof(int) \* (2 \* ws + 1) \* (2 \* ws + 1));

for (int i = 0; i < M->order; ++i) {

for (int j = 0; j < M->order; ++j) {

int tmp\_index = 0;

for (int a = max(0, i - ws); a < min(M->order, i + ws); ++a) {

for (int b = max(0, j - ws); b < min(M->order, j + ws); ++b) {

tmp[tmp\_index] = M->mat[a][b];

++tmp\_index;

}

}

sort(tmp, tmp\_index);

new\_M->mat[i][j] = tmp[(tmp\_index) / 2];

}

}

free(tmp);

return NULL;

}

void matrix\_init(matrix \*m, int n){

m->order = n;

m->mat = (int\*\*)malloc(n \* sizeof(int\*));

if (m->mat == NULL){

printf("Can't allocate memory!\n");

exit(2);

}

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

m->mat[i] = (int\*)malloc(n \* sizeof(int));

if (m->mat[i] == NULL){

printf("Can't allocate memory!\n");

exit(2);

}

}

}

void matrix\_input(matrix \*m){

for (int i = 0; i < m->order; ++i){

for (int j = 0; j < m->order; ++j){

scanf("%d", &m->mat[i][j]);

}

}

}

void matrix\_delete(matrix \*m){

for (int i = 0; i < m->order; ++i){

free(m->mat[i]);

}

free(m->mat);

}

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

if (argc != 2){

printf("No argument for main function!\n");

exit(1);

}

int thread\_amount = atoi(argv[1]);

if (thread\_amount < 0){

printf("Number of threads must be > 0");

}

int n;

matrix M, new\_M;

printf("Enter size of matrix: ");

scanf("%d", &n);

matrix\_init(&M, n);

matrix\_input(&M);

matrix\_init(&new\_M, n);

int ws;

printf("Enter window size: ");

scanf("%d", &ws);

int K;

printf("Enter amount of median filters: ");

scanf("%d", &K);

if (thread\_amount > M.order){

thread\_amount = M.order;

}

Args arr[thread\_amount];

pthread\_t threads[thread\_amount];

int strings\_per\_thread = (M.order + 1)/ thread\_amount;

for (int i = 0; i < thread\_amount; ++i) {

int l = strings\_per\_thread \* i;

int r = min(M.order, strings\_per\_thread \* (i + 1));

arr[i].m = &M;

arr[i].new\_m = &new\_M;

arr[i].begin = l;

arr[i].end = r;

arr[i].window\_size = ws;

pthread\_create(&threads[i],NULL,\*Median\_Filtr, (void\*)&arr[i]);

}

for (int i = 0; i < thread\_amount; ++i) {

pthread\_join(threads[i],NULL);

}

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

for (int j = 0; j < n; ++j){

printf("%d ", new\_M.mat[i][j]);

}

printf("\n");

}

matrix\_delete(&M);

matrix\_delete(&new\_M);

return 0;

}

1. **Демонстрация работы программы**

**parsifal@DESKTOP-3G70RV4:~/OS/Lab3$ cat t.txt**

**5**

**1 2 3 4 5**

**2 3 4 6 7**

**7 8 9 0 1**

**1 2 4 7 8**

**3 4 5 8 9**

**3**

**1parsifal@DESKTOP-3G70RV4:~/OS/Lab3$ strace -f -e trace="%process,write" -o strace\_log.txt ./main 4 < t.txt > /dev/null**

**parsifal@DESKTOP-3G70RV4:~/OS/Lab3$ cat strace\_log.txt**

**3805 execve("./main", ["./main", "4"], 0x7fffccd22c10 /\* 27 vars \*/) = 0**

**3805 arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7fffc2d85470) = -1 EINVAL (Invalid argument)**

**3805 arch\_prctl(ARCH\_SET\_FS, 0x7f1e7aff0740) = 0**

**3805 clone(child\_stack=0x7f1e7afdffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[3806], tls=0x7f1e7afe0700, child\_tidptr=0x7f1e7afe09d0) = 3806**

**3805 clone(child\_stack=0x7f1e7a7cffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[3807], tls=0x7f1e7a7d0700, child\_tidptr=0x7f1e7a7d09d0) = 3807**

**3805 clone(child\_stack=0x7f1e79fbffb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID <unfinished ...>**

**3806 exit(0 <unfinished ...>**

**3805 <... clone resumed>, parent\_tid=[3808], tls=0x7f1e79fc0700, child\_tidptr=0x7f1e79fc09d0) = 3808**

**3806 <... exit resumed>) = ?**

**3806 +++ exited with 0 +++**

**3807 exit(0 <unfinished ...>**

**3805 clone(child\_stack=0x7f1e797affb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID <unfinished ...>**

**3807 <... exit resumed>) = ?**

**3807 +++ exited with 0 +++**

**3805 <... clone resumed>, parent\_tid=[3809], tls=0x7f1e797b0700, child\_tidptr=0x7f1e797b09d0) = 3809**

**3808 exit(0) = ?**

**3808 +++ exited with 0 +++**

**3809 exit(0) = ?**

**3809 +++ exited with 0 +++**

**3805 write(1, "Enter size of matrix: Enter wind"..., 128) = 128**

**3805 exit\_group(0) = ?**

**3805 +++ exited with 0 +++**

1. **Исследование ускорения и эффективности**

Исследование ускорения и эффективности производились на следующих входных данных: размер матрицы — 50x50, размер окна 35x35, количество наложений — 1 раз.

Время работы программы будет замеряться при помощи утилиты time. Нужно учитывать, что время работы может варьироваться в небольших пределах из-за постоянной работы фоновых процессов.

|  |  |  |  |
| --- | --- | --- | --- |
| Количество потоков (n) | Время работы программы (Tn), сек | Ускорение  (Sn = T1 / Tn) | Эффективность (Xn = Sn / n) |
| 1 | 3,568 | 1 | 1 |
| 2 | 1,839 | 1,940184883 | 0,9700924415 |
| 3 | 1,644 | 2,170316302 | 0,7234387672 |
| 4 | 1,191 | 2,995801847 | 0,7489504618 |
| 5 | 1,124 | 3,174377224 | 0,6348754448 |
| 6 | 0,999 | 3,571571572 | 0,5952619286 |
| 7 | 0,97 | 3,678350515 | 0,5254786451 |
| 8 | 0,962 | 3,708939709 | 0,4636174636 |
| 9 | 0,911 | 3,916575192 | 0,4351750213 |
| 10 | 0,967 | 3,689762151 | 0,3689762151 |
| 11 | 0,983 | 3,629704985 | 0,3299731804 |
| 12 | 0,999 | 3,571571572 | 0,2976309643 |

Можно заметить, что ускорение стремится к 4 (а число ядер в моём процессоре как раз 4), достигает там своего пика и потом снова начинает падать.

1. **Выводы**

Данная лабораторная работа направлена на изучение потоков в языке СИ. Для работы с потоками необходимо подключить библиотеку pthread.h.

Преимущества потоков перед процессами:

1. Создать поток быстрее, чем создать процесс.

2. Потоки используют одну область памяти, следовательно, с помощью них можно ускорить вычисление каких-либо данных, отводя каждому потоку какое-то определенное действие.

Вышеупомянутая библиотека pthread.h имеют весьма достаточный функционал для работы с потоками. Приведу в пример функции создания потока, ожидания завершения потока и другие функции.

Помимо изучения потоков в Си, я познакомилась с медианным фильтром, который предназначен для сглаживания изображения или нахождения среднего значения какой-либо характеристики.