Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №2 по курсу

«Операционные системы»

Группа: М8О-214БВ-23

Студент: Богданов М.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 15.10.25

Москва, 2025

Постановка задачи

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.

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

Общий метод и алгоритм решения

Использованные системные вызовы:

1) sysconf — получение информации о системе (количество процессоров)
2) pthread_create — создание нового потока
3) pthread_join — ожидание завершения потока

Для применения медианного фильтра необходимо вычислить медианное значение для каждого элемента матрицы. Медианное значение – серединное значение из всех значений из окна вокруг элемента. Очевидно, что при больших матрицах нахождение медианы для каждого элемента будет занимать время. Поэтому была реализована много поточность – теперь один поток не обрабатывает всю матрицу, а обрабатывает n – ое количество элементов матрицы, где n = кол-во элементов/ кол-во потоков. При этом гарантируется что гонки данных не будет так как каждый поток работает только со своими индексами.

Код программы

consistent_task.c (1 поток)

```c
#include <stdint.h>
#include <stdbool.h>
#include <limits.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>


#define MAX_LENGTH 256



int cmp(const void *a, const void *b) {
  int x = *(const int*)a, y = *(const int*)b;
  return (x > y) - (x < y);
}


int main(int argc, char ** argv){
  int max_threads = 0;
  char input_file[MAX_LENGTH];
  char output_file[MAX_LENGTH];
  for (int i = 1; i < argc; ++i) {
    if (strcmp(argv[i], "--max-threads") == 0 && i+1 < argc) {
      max_threads = atoi(argv[i+1]);

    }
  }
  int x,y;
  fprintf(stdout, "Введите размер марицы\nx: ");
  scanf("%d", &x);
  fprintf(stdout, "y: ");
  scanf("%d", &y);
  int matrix[x*y];
  fprintf(stdout, "Введите имя файла с матрицей: ");
  scanf("%s", input_file);
  fprintf(stdout, "Введите имя файла для вывода: ");
  scanf("%s", output_file);
  FILE *f_in = fopen(input_file, "r");
  if (!f_in){
    fprintf(stderr, "error: cannot open file %s\n", input_file);
    return -1;
  }
  for (int i = 0; i < x * y; i++){
    if (fscanf(f_in, "%d", &matrix[i]) != 1){
      fprintf(stderr, "error: failed to read element %d\n", i);
      fclose(f_in);
```

```
        return -1;
    }
}
fclose(f_in);
fprintf(stdout, "Матрица успешно прочитана\n");
int K, window_x, window_y;
fprintf(stdout, "\nВведите размер окна фильтра\nx: ");
scanf("%d", &window_x);
fprintf(stdout, "y: ");
scanf("%d", &window_y);
fprintf(stdout, "\nКол-во обходов матрицы (K): ");
scanf("%d", &K);



struct timespec start, end;
clock_gettime(CLOCK_MONOTONIC, &start);



for (int iteration = 0; iteration < K; iteration++) {
    int result[x * y];
    for (int k = 0; k < x * y ; k++) {
        int i = k / y;
        int j = k % y;
        int window[window_x * window_y];
        int index = 0;
        int half_wx = window_x / 2, half_wy = window_y / 2;
        for (int di = -half_wx; di <= half_wx; di++) {
            for (int dj = -half_wy; dj <= half_wy; dj++) {
                int ni = i + di, nj = j + dj;
                if (ni >= 0 && ni < x && nj >= 0 && nj < y) {
                    window[index++] = matrix[ni * y + nj];
                }
            }
        }
        qsort(window, index, sizeof(int), cmp);
        result[i * y + j] = window[index / 2];
    }
    memcpy(matrix, result, sizeof(matrix));
}
FILE *f_out = fopen(output_file, "w");
for (int k = 0; k<x*y; k++){
    if (k > 0 && k % y == 0){
        fprintf(f_out, "\n");
    }
    fprintf(f_out, "%d ", matrix[k]);
}
fclose(f_out);



clock_gettime(CLOCK_MONOTONIC, &end);
double time_taken;
time_taken = (end.tv_sec - start.tv_sec) * 1e9;
```

```
    time_taken = (time_taken + (end.tv_nsec - start.tv_nsec)) * 1e-9;
    printf("Время выполнения: %.9f секунд\n", time_taken);
    return 0;
}
```

parallel_task.c

```
#include <linux/limits.h>
#include <stddef.h>
#include <stdint.h>
#include <stdbool.h>
#include <limits.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <time.h>


#define MAX_LENGTH 256


typedef struct {
    int * input;
    int * output;
    size_t i_start;
    size_t i_end;
    size_t window_x;
    size_t window_y;
    size_t x;
    size_t y;
} ThreadArgs;



int cmp(const void *a, const void *b) {
    int x = *(const int*)a, y = *(const int*)b;
    return (x > y) - (x < y);
}



static void *task(void *_args)
{
    ThreadArgs *args = (ThreadArgs *)_args;
    int *window = malloc( sizeof(int) * (args->window_x * args->window_y));
    if (!window) return NULL;
    for (size_t k = args->i_start; k <= args->i_end; k++) {
        int i = k / args->y;
```

```
    int j = k % args->y;
    int index = 0;
    int half_wx = args->window_x / 2, half_wy = args->window_y / 2;
    for (int di = -half_wx; di <= half_wx; di++) {
      for (int dj = -half_wy; dj <= half_wy; dj++) {
        int ni = i + di;
        int nj = j + dj;
        if (ni >= 0 && ni < (int)args->x &&
          nj >= 0 && nj < (int)args->y) {
          window[index++] = args->input[ni * args->y + nj];
        }
      }
    }
    qsort(window, index, sizeof(int), cmp);
    args->output[i * args->y + j] = window[index / 2];
  }
  free(window);
  return NULL;
}



int main(int argc, char ** argv){
  int max_threads = 0;
  int flag = 0;
  char input_file[MAX_LENGTH];
  char output_file[MAX_LENGTH];
  for (int i = 1; i < argc; ++i) {
    if (strcmp(argv[i], "--max-threads") == 0 && i + 1 < argc) {
      max_threads = atoi(argv[i+1]);
      flag = 1;
    }
    if (strcmp(argv[i], "--force-threads") == 0 && i + 1 < argc) {
      max_threads = atoi(argv[i+1]);
      flag = 2;
    }
  }
  if (flag != 0 && max_threads < 1){
    fprintf(stderr, "error: need at least 1 thread\n");
    return -1;
  }
  size_t n_threads;
  if (flag == 2){
    n_threads = max_threads;
  } else{
    n_threads = sysconf(_SC_NPROCESSORS_ONLN);
    if (flag != 0 && n_threads > max_threads) n_threads = max_threads;
  }
  if (n_threads == 0){
    fprintf(stderr, "error: cant realize multithreading with 0 threads\n");
    return -1;
  }
  size_t x, y;
  fprintf(stdout, "Введите размер матрицы\nx: ");
```

```
scanf("%zu", &x);
fprintf(stdout, "y: ");
scanf("%zu", &y);
int *matrix = (int *)malloc(sizeof(int) * (x * y));
if (!matrix) return -1;
int *result = (int *)malloc(sizeof(int) * (x * y));
if (!result) {
    free(matrix);
    return -1;
}
fprintf(stdout, "Введите имя файла с матрицей: ");
scanf("%s", input_file);
fprintf(stdout, "Введите имя файла для вывода: ");
scanf("%s", output_file);
FILE *f_in = fopen(input_file, "r");
if (!f_in){
    fprintf(stderr, "error: cannot open file %s\n", input_file);
    free(matrix);
    free(result);
    return -1;
}
for (int i = 0; i < x * y; i++){
    if (fscanf(f_in, "%d", &matrix[i]) != 1){
        fprintf(stderr, "error: failed to read element %d\n", i);
        fclose(f_in);
        free(matrix);
        free(result);
        return -1;
    }
}
fclose(f_in);
fprintf(stdout, "Матрица успешно прочитана\n");
size_t K, window_x, window_y;
fprintf(stdout, "\nВведите размер окна фильтра\nx: ");
scanf("%zu", &window_x);
fprintf(stdout, "y: ");
scanf("%zu", &window_y);
fprintf(stdout, "\nКол-во обходов матрицы (K): ");
scanf("%zu", &K);




struct timespec start, end;
clock_gettime(CLOCK_MONOTONIC, &start);




pthread_t *threads = malloc(n_threads * sizeof(pthread_t));
if (!threads) {
    free(matrix);
    free(result);
    return -1;
}
```

```
ThreadArgs *thread_args = malloc(n_threads * sizeof(ThreadArgs));
if (!thread_args)
{
    free(matrix);
    free(result);
    free(threads);
    return -1;
}
int max_len = (x * y) / n_threads;
for (int iteration = 0; iteration < K; iteration++) {
    for (int k = 0; k< x * y; k++){
        result[k] = matrix[k];
    }
    for (size_t i = 0; i < n_threads; ++i) {
        thread_args[i] = (ThreadArgs){
            .i_start = max_len * i,
            .input = matrix,
            .output = result,
            .window_x = window_x,
            .window_y = window_y,
            .x = x,
            .y = y
        };
        if (i == n_threads -1){
            thread_args[i].i_end = x * y - 1;
        } else {
            thread_args[i].i_end = max_len * (i+1) - 1;
        }
        pthread_create(&threads[i], NULL, task, &thread_args[i]);
    }
    for (size_t i = 0; i < n_threads; ++i) {
        pthread_join(threads[i], NULL);
    }
    memcpy(matrix, result, x * y * sizeof(int));
}
free(thread_args);
free(threads);
free(result);
FILE *f_out = fopen(output_file, "w");
if (!f_out){
    fprintf(stderr, "error: cannot create output file %s\n", output_file);
    free(matrix);
    return -1;
}
for (int k = 0; k<x*y; k++){
    if (k > 0 && k % y == 0){
        fprintf(f_out, "\n");
    }
    fprintf(f_out, "%d ", matrix[k]);
}
fclose(f_out);
fprintf(stdout, "Результат записан в файл %s\n", output_file);
free(matrix);
```

```
  clock_gettime(CLOCK_MONOTONIC, &end);
  double time_taken;
  time_taken = (end.tv_sec - start.tv_sec) * 1e9;
  time_taken = (time_taken + (end.tv_nsec - start.tv_nsec)) * 1e-9;
  printf("Время выполнения: %.9f секунд\n", time_taken);
  return 0;
}
```

Протокол работы программы

Тестирование:

#1

bogdanoff@arch ~/s/O/l/src (main)> strace -o trace.log ./consistent_task

Введите размер марицы

x: 1000

y: 1000

Введите имя файла с матрицей: in.txt

Введите имя файла для вывода: out.txt

Матрица успешно прочитана

Введите размер окна фильтра

x: 5

y: 5

Кол-во обходов матрицы (K): 5

Время выполнения: 3.372802650 секунд

#2

bogdanoff@arch ~/s/O/l/src (main)> strace -o trace2.log ./parallel_task --max-threads 2

Введите размер матрицы

x: 1000

y: 1000

Введите имя файла с матрицей: in.txt

Введите имя файла для вывода: out.txt

Матрица успешно прочитана

Введите размер окна фильтра

x: 5

y: 5

Кол-во обходов матрицы (K): 5

Результат записан в файл out.txt

Время выполнения: 1.810367129 секунд

#3

bogdanoff@arch ~/s/O/l/src (main)> strace -o trace3.log ./parallel_task --max-threads 6

Введите размер матрицы

x: 1000

y: 1000

Введите имя файла с матрицей: in.txt

Введите имя файла для вывода: out.txt

Матрица успешно прочитана

Введите размер окна фильтра

x: 5

y: 5

Кол-во обходов матрицы (K): 5

Результат записан в файл out.txt

Время выполнения: 0.731811337 секунд

#4

bogdanoff@arch ~/s/O/l/src (main)> strace -o trace4.log ./parallel_task --max-threads 12

Введите размер матрицы

x: 1000

y: 1000

Введите имя файла с матрицей: in.txt

Введите имя файла для вывода: out.txt

Матрица успешно прочитана

Введите размер окна фильтра

x: 5

y: 5

Кол-во обходов матрицы (K): 5

Результат записан в файл out.txt

Время выполнения: 0.624952805 секунд

#5

bogdanoff@arch ~/s/O/l/src (main)> strace -o trace5.log ./parallel_task

Введите размер матрицы

x: 1000

y: 1000

Введите имя файла с матрицей: in.txt

Введите имя файла для вывода: out.txt

Матрица успешно прочитана

Введите размер окна фильтра

x: 5

y: 5

Кол-во обходов матрицы (K): 5

Результат записан в файл out.txt

Время выполнения: 0.589226921 секунд

#6

bogdanoff@arch ~/s/O/l/src (main)> strace -o trace6.log ./parallel_task --force-threads 64

Введите размер матрицы

x: 1000

y: 1000

Введите имя файла с матрицей: in.txt

Введите имя файла для вывода: out.txt

Матрица успешно прочитана

Введите размер окна фильтра

x: 5

y: 5

Кол-во обходов матрицы (K): 5

Результат записан в файл out.txt

Время выполнения: 0.569464868 секунд

#7

bogdanoff@arch ~/s/O/l/src (main)> strace -o trace7.log ./parallel_task --force-threads 128

Введите размер матрицы

x: 1000

y: 1000

Введите имя файла с матрицей: in.txt

Введите имя файла для вывода: out.txt

Матрица успешно прочитана

Введите размер окна фильтра

x: 5

y: 5

Кол-во обходов матрицы (K): 5

Результат записан в файл out.txt

Время выполнения: 0.611234421 секунд

#8

bogdanoff@arch ~/s/O/l/src (main)> strace -o trace8.log ./parallel_task --force-threads 1024

Введите размер матрицы

x: 1000

y: 1000

Введите имя файла с матрицей: in.txt

Введите имя файла для вывода: out.txt

Матрица успешно прочитана

Введите размер окна фильтра

x: 5

y: 5

Кол-во обходов матрицы (K): 5

Результат записан в файл out.txt

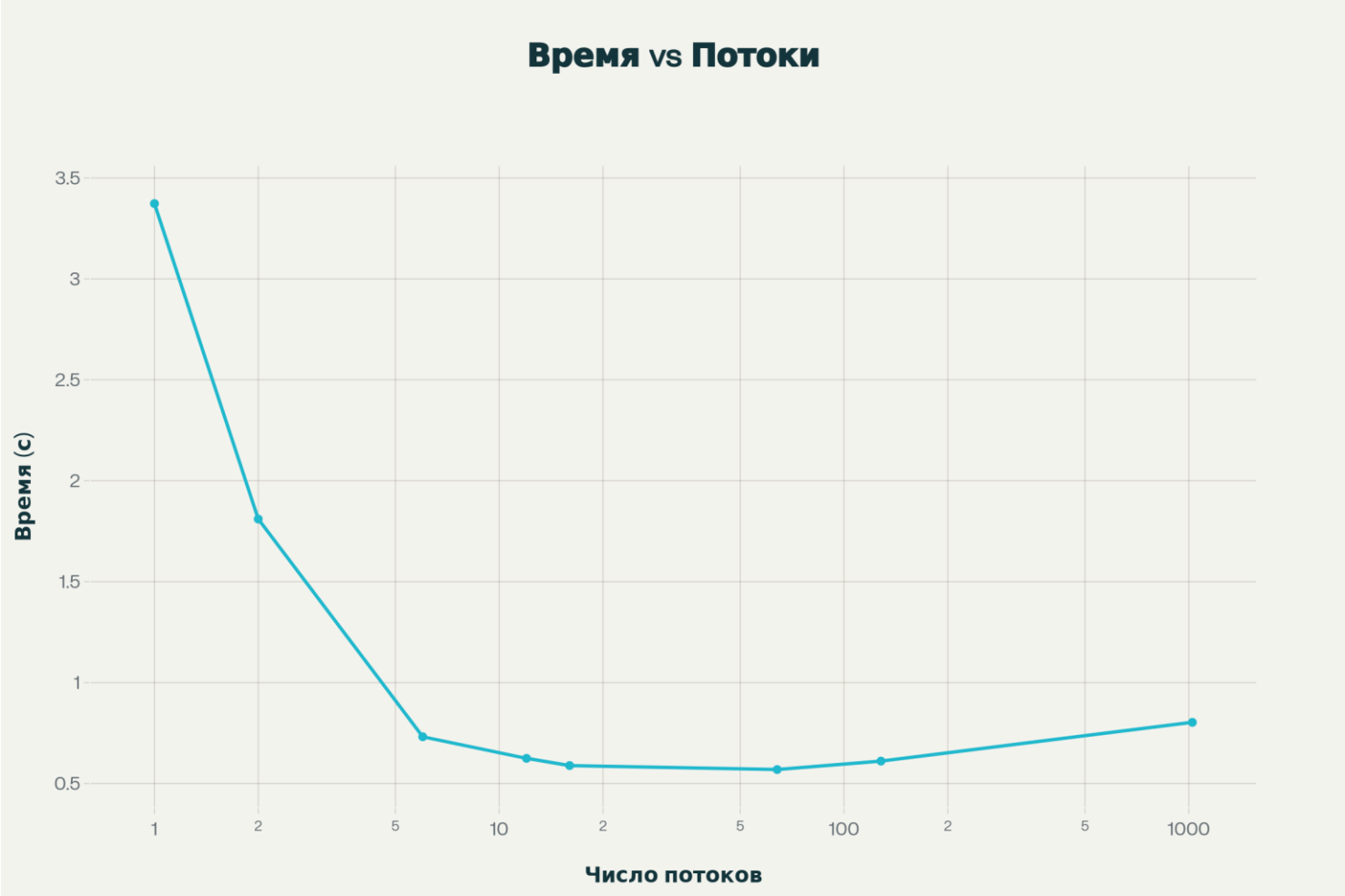
Время выполнения: 0.803244475 секунд

bogdanoff@arch ~/s/O/l/src (main)>

Тестирование strace не поместится здесь изза огромного размера, посмотреть можно в папке /strace

Вывод

Число потоков	Время исполнения (с)	Ускорение	Эффективность
1	3.372802650	1	1
2	1.810367129	1,86	0,93
6	0.731811337	4,60	0.76
12	0.624952805	5,39	0.45
16	0.589226921	5,72	0,36
64	0.569464868	5,92	0,09
128	0.611234421	5,51	0,04
1024	0.803244475	4,19	0,004



На основе экспериментальных данных многопоточной реализации медианного фильтра для матрицы 1000×1000 можно сделать следующие выводы:

1)Оптимальная конфигурация 12 -64 потока – при это максимальное ускорение достигается при 64 потоках = 0.57 сек.

2)Кратное увеличение кол-ва потоков не всегда равно ускорению работ программы

3)Многопоточная реализация даёт почти 6-кратное ускорение, но требует тщательного выбора количества потоков.