Московский Авиационный Институт
(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа №3
по курсу «Операционные системы»

Студент:	Марков А.Н.
Группа:	М80-208Б-18
Преподаватель:	Миронов Е.С.
Оценка:	
Дата:	

Москва
2019

Содержание

1. Постановка задачи.

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке использовать стандартные средства создания потоков операционной системы. При создании необходимо предусмотреть ключи, которые позволяли бы задать максимальное количество потоков, используемое программой. При возможности необходимо использовать максимальное количество возможных потоков. Ограничение потоков может быть задано или ключом запуска программы, или алгоритмом.

Вариант задания №16. Наложить K раз фильтры эрозии и наращивания на матрицу состоящую из вещественных чисел. На выходе получается 2 результирующие матрицы.

2. Общие сведения о программе.

Исходный код хранится в файле main.c. В данном файле используются заголовочные файлы stdio.h, stdlib.h, pthread.h. В программе используются следующие системные вызовы для работы с потоками из заголовочного файла pthread.h:

1. pthread_create — для создания нового потока.
2. pthread_join — для ожидания завершения потока, используется для синхронизации потоков.

Количество потоков, которые используются при работе программы, передается ключом запуска программы (максимальное количество потоков установлено в 2048). Программа считывает данные с stdin. Сначала считывается количество строк m и столбцов n в матрице, затем количество раз применения фильтров. Далее считываются m * n элементов матрицы.

3. Общий метод и алгоритм решения.

Для того чтобы наложить фильтр эрозии $B=(b_{ij})$ (матрица 3x3 заполненная единицами) на матрицу $A=(a_{ij})$ необходимо к каждому элементу матрицы A приложить центральный элемент фильтра и определить минимальный элемент среди элементов матрицы A, попавших в область шаблона. Элемент, к которому был приложен центральный элемент шаблона, заменяется на найденный минимальный. Аналогично с фильтром наращивания, только находится максимальный элемент в области.

Для обработки каждого элемента необходимо создавать отдельный потока. В случае, если количество потоков, которое можно создать одновременно меньше количества обрабатываемых элементов матрицы, необходимо ждать завершение потоков и создавать из снова с соответствующими элементами. Для синхронизации потоков используется вызов pthread_join.

4. Основные файлы программы.

main.c

```
#include <stdlib.h>

#include <stdio.h>

#include <pthread.h>

const int MAX_THREAD_NUM = 2047;
```

```
const int TEMPLATE_SIZE = 3;
typedef struct Matrix {
	double **arr;
} Matrix;
Matrix *MatrixCreate(int m, int n) {
	Matrix *mrx = (Matrix *) malloc(sizeof(Matrix));
	mrx->arr = (double **) malloc(sizeof(double *) * m);
	for (int i = 0; i < m; i++) {
		mrx->arr[i] = (double *) malloc(sizeof(double) * n);
	}
	return mrx;
}
void MatrixDestroy(Matrix *mrx, int m) {
	for (int i = 0; i < m; i++) {
		free(mrx->arr[i]);
		mrx->arr[i] = NULL;
	}
	free(mrx->arr);
	free(mrx);
}
typedef struct IOMatrix {
	Matrix *InMatrix;
	Matrix *OutMatrix;
	int inM;
	int outM;
	int inN;
	int outN;
} IOMatrix;
void ImageInput(Matrix *mrx, double *data, int m, int n) {
	for (int i = 1; i < m - 1; i++) {
		for (int j = 1; j < n - 1; j++) {
			mrx->arr[i][j] = data[(i - 1) * (n - 2) + (j - 1)];
		}
	}
	mrx->arr[0][0] = mrx->arr[1][1];
```

```
	mrx->arr[0][n - 1] = mrx->arr[1][n - 2];
	mrx->arr[n - 1][0] = mrx->arr[n - 2][1];
	mrx->arr[n - 1][n - 1] = mrx->arr[n - 2][n - 2];
	for (int i = 1; i < m - 1; i++) {
		mrx->arr[i][0] = mrx->arr[i][1];
		mrx->arr[i][n - 1] = mrx->arr[i][n - 2];
	}
	for (int j = 1; j < n - 1; j++) {
		mrx->arr[0][j] = mrx->arr[1][j];
		mrx->arr[n - 1][j] = mrx->arr[n - 2][j];
	}
}
IOMatrix *IOMatrixCreate(double *data, int m, int n) {
	IOMatrix *tempMatrix = (IOMatrix *) malloc(sizeof(IOMatrix));
	tempMatrix->InMatrix = MatrixCreate(m + 2, n + 2);
	tempMatrix->inM = m + 2;
	tempMatrix->inN = n + 2;
	tempMatrix->OutMatrix = MatrixCreate(m, n);
	tempMatrix->outM = m;
	tempMatrix->outN = n;
	ImageInput(tempMatrix->InMatrix, data, tempMatrix->inM, tempMatrix->inN);
	return tempMatrix;
}
void IOMatrixDestroy(IOMatrix *mrx) {
	MatrixDestroy(mrx->InMatrix, mrx->inM);
	MatrixDestroy(mrx->OutMatrix, mrx->outM);
	mrx->InMatrix = NULL;
	mrx->OutMatrix = NULL;
	free(mrx);
}
void IMatrixPrint(IOMatrix *mrx) {
	for (int i = 1; i < mrx->inM - 1; i++) {
		for (int j = 1; j < mrx->inN - 1; j++) {
			printf("%f ", mrx->InMatrix->arr[i][j]);
		}
```

```
            printf("\n");
        }
    }
    void OMatrixPrint(IOMatrix *mrx) {
        for (int i = 0; i < mrx->outM; i++) {
            for (int j = 0; j < mrx->outN; j++) {
                printf("%f ", mrx->OutMatrix->arr[i][j]);
            }
            printf("\n");
        }
    }
    void IOMatrixPrint(IOMatrix *mrx) {
        IMatrixPrint(mrx);
        printf("\n");
        OMatrixPrint(mrx);
    }
    double min(double a, double b) {
        return (a < b) ? a : b;
    }
    double max(double a, double b) {
        return (a > b) ? a : b;
    }
    typedef struct Args {
        IOMatrix *mrx;
        int i;
        int j;
    } Args;
    Args *ArgsCreate(IOMatrix *matrix, int m, int n) {
        Args *arr = (Args *) malloc(sizeof(Args) * m * n);
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                arr[i * n + j].mrx = matrix;
                arr[i * n + j].i = i + 1;
                arr[i * n + j].j = j + 1;
            }
```

```
    }
    return arr;
}
void ArgsDestroy(Args *arg) {
    free(arg);
}
void *Dilate(void *arg) {
    Args *targ = (Args *) arg;
    double minElem = targ->mrx->InMatrix->arr[targ->i - 1][targ->j - 1];
    for (int cord1 = 0; cord1 < TEMPLATE_SIZE; cord1++) {
        for (int cord2 = 0; cord2 < TEMPLATE_SIZE; cord2++) {
            minElem = min(targ->mrx->InMatrix->arr[targ->i - 1 + cord1][targ->j - 1 + cord2], minElem);
        }
    }
    targ->mrx->OutMatrix->arr[targ->i - 1][targ->j - 1] = minElem;
}
void *Erode(void *arg) {
    Args *targ = (Args *) arg;
    double maxElem = targ->mrx->InMatrix->arr[targ->i - 1][targ->j - 1];
    for (int cord1 = 0; cord1 < TEMPLATE_SIZE; cord1++) {
        for (int cord2 = 0; cord2 < TEMPLATE_SIZE; cord2++) {
            maxElem = max(targ->mrx->InMatrix->arr[targ->i - 1 + cord1][targ->j - 1 + cord2], maxElem);
        }
    }
    targ->mrx->OutMatrix->arr[targ->i - 1][targ->j - 1] = maxElem;
}
int main(int argc, char **argv) {
    if (argc != 2) {
        printf("Input the number of threads. ./name_of_program num_threads\n");
        return 0;
    }
    int numThreads = atoi(argv[1]);
    numThreads = min(max(numThreads, 1), MAX_THREAD_NUM);
    pthread_t *threads = (pthread_t *) malloc(sizeof(pthread_t) * numThreads);
    int m, n;
```

```
printf("Input size of matrix. Num_row Num_column: ");
scanf("%d %d", &m, &n);
if (m <= 0 || n <= 0) {
	printf("Invalid size of matrix\n");
	return 0;
}
double *startArr = (double *) malloc(sizeof(double) * m * n);
for (int i = 0; i < m * n; i++) {
	scanf("%lf", &startArr[i]);
}
IOMatrix *ioImage = IOMatrixCreate(startArr, m, n);
Args *argsArr = ArgsCreate(ioImage, m, n);
int k;
printf("Input the amount of filtering: ");
scanf("%d", &k);
if (k < 0) {
	printf("Invalid amount of filtering\n");
	return 0;
}
else if (k == 0) {
	printf("Dilate and Erode:\n");
	IMatrixPrint(ioImage);
	return 0;
}
double *helpArr = (double *) malloc(sizeof(double) * m * n);
for (int cnt = 0; cnt < k; cnt++) {
	int i = 0;
	while (i < m * n) {
		int t = min(m * n - i, numThreads);
		for (int j = 0; j < t; j++) {
			pthread_create(&threads[j], NULL, Dilate, &argsArr[i]);
			i++;
		}
		for (int j = 0; j < t; j++) {
			pthread_join(threads[j], NULL);
```

```
            }
        }
        for (int u = 0; u < m; u++) {
            for (int y = 0; y < n; y++) {
                helpArr[u * n + y] = ioImage->OutMatrix->arr[u][y];
            }
        }
        ImageInput(ioImage->InMatrix, helpArr, m + 2, n + 2);
    }
    printf("Dilate:\n");
    OMatrixPrint(ioImage);
    ImageInput(ioImage->InMatrix, startArr, m + 2, n + 2);
    for (int cnt = 0; cnt < k; cnt++) {
        int i = 0;
        while (i < m * n) {
            int t = min(m * n - i, numThreads);
            for (int j = 0; j < t; j++) {
                pthread_create(&threads[j], NULL, Erode, &argsArr[i]);
                i++;
            }
            for (int j = 0; j < t; j++) {
                pthread_join(threads[j], NULL);
            }
        }
        for (int u = 0; u < m; u++) {
            for (int y = 0; y < n; y++) {
                helpArr[u * n + y] = ioImage->OutMatrix->arr[u][y];
            }
        }
        ImageInput(ioImage->InMatrix, helpArr, m + 2, n + 2);
    }
    printf("Erode:\n");
    OMatrixPrint(ioImage);
    return 0;
}
```

5. Демонстрация работы программы.

```
oem@Alex-PC:~/Documents/OS/lab3/16varik$ ./main 25
Input size of matrix. Num_row Num_column: 5 5
1  2  3  4  5
6  7  8  9  10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
Input the amount of filtering: 1
Dilate:
1.000000 1.000000 2.000000 3.000000 4.000000
1.000000 1.000000 2.000000 3.000000 4.000000
6.000000 6.000000 7.000000 8.000000 9.000000
11.000000 11.000000 12.000000 13.000000 14.000000
16.000000 16.000000 17.000000 18.000000 19.000000
Erode:
7.000000 8.000000 9.000000 10.000000 10.000000
12.000000 13.000000 14.000000 15.000000 15.000000
17.000000 18.000000 19.000000 20.000000 20.000000
22.000000 23.000000 24.000000 25.000000 25.000000
22.000000 23.000000 24.000000 25.000000 25.000000
oem@Alex-PC:~/Documents/OS/lab3/16varik$ cat test_02
3 3
273   936   -520
35    -265  948
-259  831   -102
1
oem@Alex-PC:~/Documents/OS/lab3/16varik$ ./main 3 < test_02
Input size of matrix. Num_row Num_column: Input the amount of filtering: Dilate:
-265.000000 -520.000000 -520.000000
-265.000000 -520.000000 -520.000000
-265.000000 -265.000000 -265.000000
Erode:
936.000000 948.000000 948.000000
936.000000 948.000000 948.000000
831.000000 948.000000 948.000000
```

7. Вывод.

Потоки удобно применять для многозадачности и для ускорения некоторых алгоритмов. В отличие от процессов потоки имеют возможность совместно использовать одно адресное пространство. Также потоки быстрее выполняют операции создания и уничтожения, так как с потоком не связаны никакие ресурсы.