Московский авиационный институт
(Национальный исследовательский университет)

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

Лабораторная работа № 3

Тема: Управление потоками в ОС

Студент: Туманов Георгий
Группа: 80-201
Преподаватель: Соколов А.А.
Оценка:
Подпись:

Москва, 2019

1. Постановка задачи

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). 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 вашей программы, или алгоритмом.

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

Вариант 16: Наложить K раз фильтры эрозии и наращивания на матрицу состоящую из вещественных чисел. На выходе получается 2 результирующие матрицы.

2. Описание программы

Программа запускается с двумя ключами: количество цифр после запятой и максимальное число потоков. Если максимальное число потоков меньше или равно нулю, то программа не ограничивается в количестве потоков. По умолчанию количество цифр после запятой равно 3, а максимальное количество потоков равно 0.

При запуске программа просит ввести ширину и высоту исходной матрицы, элементы матрицы построчно, ширину и высоту окна и число K. После этого программа выводит две матрицы: результат повторного применения эрозии и результат повторного применения наращивания.

На обработку каждого элемента выходных матриц программа создает по потоку, ждёт их завершения и продолжает создавать потоки, пока не обработает матрицы. Программа повторяет этот процесс K раз.

Входными параметрами потоков являются указатель на входную и выходную матрицы, координаты обрабатываемого элемента, шириа и высота матрицы и области и режим обработки: эрозия или наращивание. В зависимости от режима, поток записывает в выходную матрицу либо минимальный элемент области, либо максимальный элемент области.

3. Набор testcases

№	Описание	Ввод
1	Примитивный тест демонстрации фильтров	3 3 0 1 2 0 1 2 1 0 3 1 1 1
2	Тот же тест, но для K = 2	3 3 0 1 2 0 1 2 1 0 3 1 1 2
3	Тот же тест, но с прямоугольной областью	3 3 0 1 2 0 1 2 1 0 3 1 0 1

4. Результаты выполнения тестов.

test 1:

Write size of matrix (width and height): 3 3
Width: 3
Height: 3
Write elements of matrix:
0 1 2
0 1 2
1 0 3
Inputed matrix:
0.000 1.000 2.000
0.000 1.000 2.000
1.000 0.000 3.000
Input half-width and half-height of filter matrix: 1 1
K = 1
Erosion matrix:
0.000 0.000 1.000
0.000 0.000 0.000
0.000 0.000 0.000
Increasing matrix:
1.000 2.000 2.000

1.000 3.000 3.000
1.000 3.000 3.000

test 2:

Write size of matrix (width and height): 3 3
Width: 3
Height: 3
Write elements of matrix:
0 1 2
0 1 2
1 0 3
Inputed matrix:
0.000 1.000 2.000
0.000 1.000 2.000
1.000 0.000 3.000
Input half-width and half-height of filter matrix: 1 1
K = 2
Erosion matrix:
0.000 0.000 0.000
0.000 0.000 0.000
0.000 0.000 0.000
Increasing matrix:
3.000 3.000 3.000
3.000 3.000 3.000
3.000 3.000 3.000

test 3:

Write size of matrix (width and height): 3 3
Width: 3
Height: 3
Write elements of matrix:
0 1 2
0 1 2
1 0 3
Inputed matrix:
0.000 1.000 2.000
0.000 1.000 2.000
1.000 0.000 3.000
Input half-width and half-height of filter matrix: 1 0
K = 1
Erosion matrix:
0.000 0.000 1.000
0.000 0.000 1.000
0.000 0.000 0.000
Increasing matrix:

1.000 2.000 2.000
1.000 2.000 2.000
1.000 3.000 3.000

5. Листинг программы

main.c:

```
#include "io.h"
#include <pthread.h>

#define min(a, b) (((a) < (b)) ? (a) : (b))
#define max(a, b) (((a) > (b)) ? (a) : (b))

enum Mod
{
	m_decrease,
	m_increase,
};
typedef struct { float *mIn, *mOut; int x, y, w, h, a, b; enum Mod mod; } ThreadData;

float getM(float *m, int w, int x, int y) { return m[x + y * w]; }
void setM(float *m, int w, int x, int y, float val)
{
	m[x + y * w] = val;
}

void* filter(void *arg)
{
	ThreadData td = *((ThreadData*)arg);
	float val = getM(td.mIn, td.w, td.x, td.y), t;

	for (int i = td.x - td.a; i <= td.x + td.a; i++)
	for (int j = td.y - td.b; j <= td.y + td.b; j++)
	{
	if (i < 0 || j < 0 || i >= td.w || j >= td.h) continue;
	t = getM(td.mIn, td.w, i, j);

	if (td.mod == m_decrease)
	{
	if (t < val) val = t;
	}
	else
	{
	if (t > val) val = t;
	}
	}

	setM(td.mOut, td.w, td.x, td.y, val);
	//writeString("Thread: "); writeInt(td.x + td.y * td.w); writeChar('\n');
}

void calculate(float *mD1, float *mI1, float *mD2, float *mI2, int w, int h, int a, int b, int k, int mt)
```

```
{
	int wh = w * h, sz = 2 * wh;
	if (mt <= 0 || mt > sz) mt = sz;
	int mtStart = mt;

	ThreadData td, tdd[sz];
	pthread_t threads[sz];

	td.a = a;
	td.b = b;
	td.w = w;
	td.h = h;

	for (int kk = 0; kk < k; kk++)
	{
	//init struct
	td.mIn = mD1;
	td.mOut = mD2;
	td.mod = m_decrease;

	mt = mtStart;

	//start threading
	int iter = 0;
	while (iter < sz)
	{
	mt = min(mt, sz - iter);
	for (int m = 0; m < mt; m++)
	{
	if (iter == wh)
	{
		td.mIn = mI1;
		td.mOut = mI2;
		td.mod = m_increase;
	}

	td.x = iter % wh % w; td.y = iter % wh / w;
	iter++;

	tdd[m] = td;
	if (pthread_create(threads + m, NULL, filter, (void*)(tdd + m)))
	{
		free(mD1); free(mD2); free(mI1); free(mI2);
		error("cannot create thread!", -10);
	}
	}
	//join
	for (int m = 0; m < mt; m++) pthread_join(threads[m], NULL);
	//writeInt(mt); writeString("!\n");
	}
	//copy result
	for (int i = 0; i < wh; i++)
```

```
		{
		mD1[i] = mD2[i];
		mI1[i] = mI2[i];
		}
		}
}

int main(int argc, char **argv)
{
		int maxThread = 0, afterPoint = 3;
		if (argc > 1)
		{
		afterPoint = atoi(argv[1]);
		if (argc > 2) maxThread = atoi(argv[2]);
		}

		//read matrix
		writeString("Write size of matrix (width and height): ");

		int w = readInt(), h = readInt(), wh = w * h; //size of matrix
		if (w <= 0) error("width must be positive!", -1);
		if (h <= 0) error("height must be positive!", -2);

		writeString("Width: "); writeInt(w); writeChar('\n');
		writeString("Height: "); writeInt(h); writeChar('\n');

		float *mDecrease, *mIncrease, *mDecrease2, *mIncrease2;
		mDecrease = malloc(wh * sizeof(float)); //create erosion matrix
		if (!mDecrease) error("cannot allocate memory for matrix!", -3);
		mIncrease = malloc(wh * sizeof(float)); //create increasing matrix
		if (!mIncrease) { free(mDecrease); error("cannot allocate memory for matrix!", -4); }
		mDecrease2 = malloc(wh * sizeof(float)); //create erosion matrix copy
		if (!mDecrease2) { free(mDecrease); free(mIncrease); error("cannot allocate memory for matrix!", -
5); }
		mIncrease2 = malloc(wh * sizeof(float)); //create increasing matrix copy
		if (!mIncrease2) { free(mDecrease); free(mIncrease); free(mDecrease2); error("cannot allocate memory
for matrix!", -6); }

		writeString("Write elements of matrix:\n");
		for (int i = 0; i < wh; i++) mDecrease[i] = mIncrease[i] = readFloat(); //read matrix

		//show matrix
		writeString("Inputed matrix:\n");
		for (int j = 0; j < h; j++)
		{
		for (int i = 0; i < w; i++) { writeFloat(getM(mDecrease, w, i, j), afterPoint); writeChar(' '); }
		writeChar('\n');
		}
		//filter setup
		writeString("Input half-width and half-height of filter matrix: ");
		int a = readInt(), b = readInt();
		if (a < 0) { free(mDecrease); free(mIncrease); free(mDecrease2); free(mIncrease2); error("width must
```

```
be non-negative!", -7); }
	if (b < 0) { free(mDecrease); free(mIncrease); free(mDecrease2); free(mIncrease2); error("height must be non-negative!", -8); }

	writeString("K = ");
	int k = readInt();
	if (k < 0) { free(mDecrease); free(mIncrease); free(mDecrease2); free(mIncrease2); error("K must be non-negative!", -9); }

	//calculate matrixes
	calculate(mDecrease, mIncrease, mDecrease2, mIncrease2,w, h, a, b, k, maxThread);

	//show matrixes
	writeString("Erosion matrix:\n");
	for (int j = 0; j < h; j++)
	{
	for (int i = 0; i < w; i++) { writeFloat(getM(mDecrease, w, i, j), afterPoint); writeChar(' '); }
	writeChar('\n');
	}
	writeString("Increasing matrix:\n");
	for (int j = 0; j < h; j++)
	{
	for (int i = 0; i < w; i++) { writeFloat(getM(mIncrease, w, i, j), afterPoint); writeChar(' '); }
	writeChar('\n');
	}

	free(mDecrease); //free memory
	free(mIncrease); //free memory
	free(mDecrease2); //free memory
	free(mIncrease2); //free memory
	return 0;
}
```

io.h:

```
#pragma once

#include <stdlib.h>
#include <unistd.h>

int readInt();
float readFloat();
void writeInt(int num);
void writeFloat(float real, int afterpoint);
void writeChar(char c);
void writeString(char *str);
void error(char *str, int code);
```

io.c:

```
#include "io.h"
#define BS 256

void fill(char *buf) //read single word from stream
{
	int i = 0; //index
```

```
	do { read(STDIN_FILENO, buf, sizeof(char)); } //read blanks
	while ((buf[i] == '\0' || buf[i] == ' ' || buf[i] == '\t' || buf[i] == '\n') && i < BS - 1);

	do { read(STDIN_FILENO, buf + ++i, sizeof(char)); } //read next char
	while (buf[i] != '\0' && buf[i] != ' ' && buf[i] != '\t' && buf[i] != '\n' && i < BS - 1); //stop if blank
symbol or overflow
	buf[i] = '\0'; //place zero as last char
}
int readInt() { char buf[BS]; fill(buf); return atoi(buf); }
float readFloat() { char buf[BS]; fill(buf); return atof(buf); }
void writeInt(int num)
{
	if (num < 0) { writeChar('-'); num = -num; }
	char buf[BS], t;
	int len = 0;
	do
	{
	buf[len++] = '0' + num % 10;
	num /= 10;
	}
	while (num > 0);

	for (int i = 0; i < len / 2; i++)
	{
	t = buf[i];
	buf[i] = buf[len - i - 1];
	buf[len - i - 1] = t;
	}
	write(STDOUT_FILENO, buf, len * sizeof(char));
}
void writeFloat(float real, int afterpoint)
{
	if (real < 0) { writeChar('-'); real = -real; }
	char buf[BS], t;
	int len = 0, num = real;
	do
	{
	buf[len++] = '0' + num % 10;
	num /= 10;
	}
	while (num > 0);

	for (int i = 0; i < len / 2; i++)
	{
	t = buf[i];
	buf[i] = buf[len - i - 1];
	buf[len - i - 1] = t;
	}

	buf[len++] = '.';
	for (int i = 0; i < afterpoint; i++) buf[len++] = '0' + (int)(real *= 10) % 10;
```

```
	write(STDOUT_FILENO, buf, len * sizeof(char));
}
void writeChar(char c) { write(STDOUT_FILENO, &c, sizeof(char)); }
void writeString(char *str)
{
	int len = 0;
	while (str[len++]);
	write(STDOUT_FILENO, str, len * sizeof(char));
}
void error(char *str, int code)
{
	int len = 0;
	while (str[len++]);
	write(STDERR_FILENO, "Error: ", 8 * sizeof(char));
	write(STDERR_FILENO, str, len * sizeof(char));
	writeChar('\n');
	exit(code);
}
```

6. Выводы:

Освоил работу с потоками в Linux для параллельной обработки данных, а также научился ограничивать максимальное число потоков.