Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

Студент: Уваров А. П.

Группа: М8О–206Б–19

Вариант: 8

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**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.

Найти сумму массивов.

**Общие сведения о программе**

Программа компилируется из файла 3.с. Применяется pthread.h. В программе используются следующие системные вызовы:

1. **pthread\_create** – создает поток.
2. **pthread\_join** – блокирует вызывающий поток, пока указанный поток не завершится.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы с потоками.
2. Написать программу, которая будет работать с заданным количеством потоков и сравнить ускорение и эффективность в зависимости от количества потоков.
3. Написать функции для параллельной работы функций.

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

Массив на 10000 чисел

**Потоки Время (с)**

**2 0.001516**

**10 0.003608**

**20 0.004859**

**50 0.007717**

**500 0.055879**

К сожалению, моя реализация оказалась неэффективной, поэтому в результате чем меньше потоков, тем быстрее работает моя программа.

Массив на 10 млн. чисел

**Потоки** **Время (с)**

**10 0.3154**

**50 0.3344**

**500 0.3547**

**5000 0.4943**

Результат на 10 млн. чисел оказался таким же

**Основные файлы программы**

**main.c:**

#include <stdlib.h>

#include <stdio.h>

#include <pthread.h>

#include <time.h>

typedef struct{

int \*\*matrix;

int size, K;

int ind;

} EMat;

int gcd(int a, int b){

return b?gcd(b, a%b) : a;

}

void\* Work(void \*matr){

EMat\* currentMat = (EMat\*) matr;

int \*\*currentMatr = currentMat->matrix;

int \*resultMatrix = malloc(sizeof(int\*) \* currentMat->size);

for(int i = 0; i < currentMat->size; ++i)

resultMatrix[i] = 0;

for(int i = 0; i < currentMat->K; ++i){

for(int j = 0; j < currentMat->size; ++j){

resultMatrix[j] += currentMatr[i][j];

}

}

return resultMatrix;

}

void\* Work2(void\* EM){

EMat\* currentEM = (EMat\*) EM;

int curInd = currentEM->ind;

int\*\* currentMat = currentEM->matrix;

int\* resultMat = malloc(sizeof(int\*) \* currentEM->size);

for(int i = 0; i < currentEM->size; ++i)

resultMat[i] = 0;

for(int i = 0; i < currentEM->K; ++i){

for(int j = 0; j < currentEM->size; ++j){

resultMat[j] += currentMat[i][j + curInd];

}

}

return resultMat;

}

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

if(argc != 2){

printf("USAGE: %s <count of threads>\n", argv[0]);

return 1;

}

srand(time(0));

int threadCount;

if(!(threadCount = strtol(argv[1], NULL, 10)) || threadCount < 1){

printf("Argument isn't int or less than 1!\n");

return 1;

}

int size, K;

printf("Write size and count: ");

scanf("%d%d",&size,&K);

printf("ThreadCount: %d\n",threadCount);

printf("Size: %d, K: %d\n",size,K);

int \*\*matrix = malloc(K \* sizeof(int\*));

for(int i = 0; i < K; ++i)

matrix[i] = malloc(size \* sizeof(int\*));

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

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

matrix[i][j] = rand() % 10 + 1;

}

}

printf("Your matrix:\n");

for(int i = 0; i < K && i < 10; ++i){

for(int j = 0; j < size && j < 10; ++j){

printf("%d ", matrix[i][j]);

}

printf("\n");

}

EMat\* test = malloc(sizeof(EMat\*));

test->matrix = matrix;

test->K = K;

test->size = size;

printf("\n[1] result: \n");

time\_t start = clock();

int\* res1 = Work(test);

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

if(i == 30){

printf("...");

break;

}

printf("%d ", res1[i]);

}

printf("\n");

printf("Time: %fs", (float)(clock()-start)/CLOCKS\_PER\_SEC);

printf("\n\n");

pthread\_t \*ptrArr;

if(threadCount < size){

int ptrCountDiv = size / threadCount, ptrCountMod = size % threadCount;

ptrArr = malloc(sizeof(pthread\_t) \* threadCount);

int currentIndexOfEM = 0;

start = clock();

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

EMat\* curEM = malloc(sizeof(EMat\*));

curEM->K = K;

curEM->size = ptrCountDiv;

curEM->matrix = matrix;

curEM->ind = currentIndexOfEM;

currentIndexOfEM += ptrCountDiv;

pthread\_create(&ptrArr[i], NULL, Work2, curEM);

}

int\* endMat = malloc(sizeof(int\*) \* ptrCountMod);

for(int i = 0; i < ptrCountMod; ++i)

endMat[i] = 0;

for(int i = size - ptrCountMod; i < size; ++i){

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

endMat[i - size + ptrCountMod] += matrix[j][i];

}

}

int\*\* resultMat = malloc(sizeof(int\*) \* (threadCount + 1));

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

void\* takeEM;

pthread\_join(ptrArr[i], &takeEM);

resultMat[i] = (int\*) takeEM;

}

printf("[2] result: \n");

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

if(i == 10){

printf("...");

break;

}

int\* curTake = resultMat[i];

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

if(j == 30){

printf("...");

break;

}

printf("%d ", curTake[j]);

}

}

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

printf("%d ", endMat[i]);

}

printf("\nTime: %fs\n", (float)(clock()-start)/CLOCKS\_PER\_SEC);

}

else{

ptrArr = malloc(sizeof(pthread\_t) \* size);

int curInd = 0;

start = clock();

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

EMat\* curEM = malloc(sizeof(EMat\*));

curEM->K = K;

curEM->size = 1;

curEM->matrix = matrix;

curEM->ind = curInd++;

pthread\_create(&ptrArr[i], NULL, Work2, curEM);

}

printf("[2] result: \n");

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

void\* curEM;

pthread\_join(ptrArr[i], &curEM);

int\* tmp = (int\*)curEM;

printf("%d ", tmp[0]);

}

printf("\nTime: %fs\n", (float)(clock()-start)/CLOCKS\_PER\_SEC);

}

printf("\n");

return 0;

}

**Вывод**

На СИ можно писать программы с применением потоков. К сожалению, в моем случае не получилось показать всю силу распараллеливания, однако, я научился использовать потоки в СИ.