Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа**

**по курсу «Операционные системы»**

**III Семестр**

**Задание 3**

**Вариант 2**

|  |  |
| --- | --- |
| Студент: | Анисимов В.А. |
| Группа: | М80-208Б-18 |
| Преподаватель: | Миронов Е.С |
|  |  |
| Оценка: |  |
| Дата: |  |

Москва 2019

**1. Описание задания**

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 2:

Отсортировать массив строк при помощи параллельного алгоритма быстрой сортировки.

**2. Код программы**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <string.h>

#define MAX 4893

int THREAD\_MAX;

int k;

struct Args {

char \*\*arr;

int l;

int r;

};

typedef struct Args Args;

int max(int a, int b) {

return (a > b) ? a : b;

}

int min(int a, int b) {

return (a < b) ? a : b;

}

void swap(char \*\*o1, char \*\*o2) {

char \*tmp = \*o1;

\*o1 = \*o2;

\*o2 = tmp;

}

int Partion(char \*\*arr, int l, int r) {

char \*x = arr[r];

int i = l - 1;

for (int j = l; j <= r - 1; j++) {

if (strcmp(arr[j], x) <= 0) {

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[r]);

return i + 1;

}

void \*quick\_sort(void \*arg){

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

if(args.l < args.r){

int q = Partion(args.arr, args.l, args.r);

Args left;

left.arr = args.arr;

left.l = args.l;

left.r = q - 1;

Args right;

right.arr = args.arr;

right.l = q + 1;

right.r = args.r;

if(THREAD\_MAX != 0){

pthread\_t left\_thread;

THREAD\_MAX --;

k++;

pthread\_create(&left\_thread, NULL, &quick\_sort, &left);

quick\_sort(&right);

pthread\_join(left\_thread, NULL);

}

else{

quick\_sort(&left);

quick\_sort(&right);

}

}

return NULL;

}

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

if (argc < 2) {

printf("Error. Input number of threads.\n");

return 0;

}

int s;

int arg = atoi(argv[1]);

k = 0;

THREAD\_MAX = min(max(arg, 1), MAX) - 1;

printf("Input array size\n");

scanf("%d", &s);

char \*\*arr = (char \*\*) malloc(sizeof(char \*) \* s);

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

arr[i] = (char \*) malloc(sizeof(char) \* 30);

scanf("%s", arr[i]);

}

Args args;

args.arr = arr;

args.l = 0;

args.r = s - 1;

pthread\_t first\_thread;

pthread\_create(&first\_thread, NULL, &quick\_sort, &args);

k++;

pthread\_join(first\_thread, NULL);

printf("\nResult:\n");

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

printf("%s\n", arr[i]);

printf("Number of threads: %d\n", k);

return 0;

}

**3. Протокол работы программы**

walien@PC-name:~/2kurs/OS/lab3$ ./OS\_lab3 2

Input array size

10

vsd

ebrv

cxv

sdv

br

xvbr

eg

svsv

ebfd

sdf

Result:

br

cxv

ebfd

ebrv

eg

sdf

sdv

svsv

vsd

xvbr

Number of threads: 2

**5. Объяснение работы программы**

Поток выбирает ведущий элемент и сортирует остальные элементы относительно него. Далее элементы меньше ведущего передаются для сортировки другому потоку, а элементы больше ведущего сортирует первый поток. Когда все потоки получат свою часть ускорение будет максимальным.

Если большее количество потоков недоступно – оставшаяся часть сортируется последовательным алгоритмом линейной сортировки.

**Вывод**

Потоки применяются для многозадачности и ускорения работы программ, однако, при неправильном использовании только усложняют программу и её отладку, не давая прироста производительности.

Потоки эффективны, когда создание отдельного процесса слишком расточительно. Даже учитывая оптимизацию при создании процесса создаются новые копии страниц памяти, а потоки используют единое пространство имён. В целом создать поток на порядок быстрее, чем новый процесс.