Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**Управление потоками**

Студент: Камеш Михаил Алексеевич

Группа: М8О–207Б–18

Вариант: 28

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Содержание**

1. Постановка задачи
2. Общие сведения о программе
3. Общий метод и алгоритм решения
4. Основные файлы программы
5. Примеры работы
6. Вывод

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). 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.

Вариант задания.

Реализовать битоническую сортировку, реализовать для неё многопоточность.

Общие сведения о программе

Программа компилируется из одного файла main.c. В программе используются следующие системные вызовы:

1. omp\_set\_dynamic - включает или отключает динамическую настройку количества потоков, доступных для выполнения последующих параллельных областей, путем установки значения ICV dyn-var.

2. omp\_get\_num\_procs - возвращает количество процессоров, доступных устройству.

3. omp\_set\_num\_threads - влияет на количество потоков, которые будут использоваться для последующих параллельных областей, которые не определяют предложение num\_threads, путем установки значения первого элемента nthreads-var ICV текущей задачи.

4. pthread\_create - функция используется для создания нового потока с атрибутами, указанными attr, в процессе

5. pthread\_join - функция должна приостанавливать выполнение вызывающего потока до завершения целевого потока, если целевой поток уже не завершился.

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Создать битоническую сортировку.

2. Использовать omp для создания нескольких потоков.

3. Создание дополнительных потоков с помощью pthread\_create.

4. При рекурсивных вызовах сортировки сделать так, чтобы каждая рекурсия параллельно использовалась в отдельном потоке.

Файлы программы

#include<stdio.h>

#include <omp.h>

#include <stdlib.h>

#include <pthread.h>

#define ERROR\_CREATE\_THREAD -11

#define SUCCESS 0

void ascendingSwap(int index1, int index2, int \*ar) {

if(ar[index2] < ar[index1])

{

int temp = ar[index2];

ar[index2] = ar[index1];

ar[index1] = temp;

}

}

void decendingSwap(int index1, int index2, int \*ar)

{

if(ar[index1] < ar[index2])

{

int temp = ar[index2];

ar[index2] = ar[index1];

ar[index1] = temp;

}

}

void\* bitonicSortFromBitonicSequence(void \*argp)

{

struct thread\_args

{

int a;

int b;

int c;

int \*d;

};

struct thread\_args \*args = argp;

int startIndex = args->a;

int lastIndex = args->b;

int dir = args->c;

int \*ar = args->d;

if(dir == 1)

{

int counter = 0;

int noOfElements = lastIndex - startIndex + 1;

for(int j = noOfElements/2; j>0; j = j/2)

{

counter =0;

for(int i = startIndex ; i +j <= lastIndex ; i++)

{

if(counter < j)

{

ascendingSwap(i,i+j,ar);

counter++;

}

else

{

counter =0;

i = i+ j-1;

}

}

}

}

else

{

int counter = 0;

int noOfElements = lastIndex - startIndex + 1;

for(int j = noOfElements/2; j>0; j = j/2)

{

counter =0;

for(int i = startIndex ; i <= (lastIndex-j) ; i++)

{

if(counter < j)

{

decendingSwap(i,i+j,ar);

counter++;

}

else

{

counter =0;

i = i+ j-1;

}

}

}

}

}

void bitonicSequenceGenerator(int startIndex, int lastIndex, int \*ar)

{

int noOfElements = lastIndex - startIndex +1;

for(int j = 2; j<=noOfElements; j = j\*2)

{

pthread\_t thread;

int status;

struct thread\_args

{

int a;

int b;

int c;

int \*d;

};

void \*bitonicSortFromBitonicSequence(void \*);

struct thread\_args \*args = malloc(sizeof \*args);

for(int i=0; i<noOfElements; i=i+j)

{

if(((i/j)%2) ==0)

{

if (args != NULL)

{

args->a = i;

args->b = i+j-1;

args->c = 1;

args->d = ar;

status = pthread\_create(&thread, NULL, &bitonicSortFromBitonicSequence, args);

}

if (status != 0)

{

printf("main error: can't create thread, status = %d\n", status);

exit(ERROR\_CREATE\_THREAD);

}

pthread\_join(thread, NULL);

}

else

{

if (args != NULL)

{

args->a = i;

args->b = i+j-1;

args->c = 0;

args->d = ar;

status = pthread\_create(&thread, NULL, &bitonicSortFromBitonicSequence, args);

}

if (status != 0)

{

printf("main error: can't create thread, status = %d\n", status);

exit(ERROR\_CREATE\_THREAD);

}

pthread\_join(thread, NULL);

}

}

}

}

int main()

{

omp\_set\_dynamic(0);

int maxNumberOfThreads = omp\_get\_num\_procs();

omp\_set\_num\_threads(maxNumberOfThreads);

int n;

printf("enter the number of elements to be sorted (number should be in the order of 2^n\n");

scanf("%d", &n);

int \*ar = (int\*)malloc(sizeof(int)\*n);

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

{

scanf("%d", &ar[i]);

}

bitonicSequenceGenerator(0,n-1,ar);

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

{

printf("%d \n", ar[i] );

}

}

Демонстрация работы программы.

Enter number of elements to be sorted :

8

-1 5 6 2 7 2 3 1

-1 1 2 2 3 5 6 7

Выводы:

В процессе выполнения работы были изучены способы управления потоками в ОС, обеспечение синхронизации между потоками и принципы работы битонической сортировки.

Количество потоков было задано максимально возможное с помощью функций библиотеки omp. Возникали проблемы с функцией создания потоков, из-за того, что как аргумент

должна передаваться одна переменная, а переменных внутри сортировки было много. Поэтому эти переменные были перенесены в структуру, а передавалась ссылка на эту

структуру. Битоническая сортировка сама по себе не эффективна, однако мне распараллеливании её на много потоков она работает со скоростью, сравнимой с быстрой сортировкой.