****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 и синхронизация**

Группа: М80 – 206Б-18

Студент: Черненко И.Д.

Преподаватель: Соколов А.А.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2019.

**Содержание**

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

**Постановка задачи**

Составить программу, 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 вашей программы, или алгоритмом.

**Общие сведения о программе**

Программа компилируется из файла main.cpp. Также используется заголовочные файлы: iostream, string, pthread.h. В программе используются следующие системные вызовы:

1. **pthread\_create** – создает новый поток
2. **pthread\_join**– ожидает завершения переданного потока, получает его выходное значение

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Используя системный вызов pipe создать канал, по которому будут обмениваться данными два процесса.
2. Используя системный вызов fork создать дочерний процесс.
3. В родительском процессе считывать данные cо стандартного потока и правильно распарсить подаваемую команду.
4. Как в родительском процессе данные считались, необходимо записать их в канал с помощью системного вызова write.
5. Как только родительский процесс записал данные в канал дочерний процесс считывает их, производит вычисления с деревом общего вида.
6. Дочерний процесс выводит результат используя write.

**Основные файлы программы**

**main.cpp:**

#include <iostream>

#include <pthread.h>

#include <string>

#include <cmath>

using namespace std;

int Length;

int NumberOfThreads;

pthread\_mutex\_t mutex;

struct arguments {

string\* orig;

int left;

int right;

string\* mod;

};

int char\_to\_int(char c) {

if (c >= '0' && c <= '9') {

return (c-'0');

} else {

c -= 'W';

if (c > 0) {

return (c);

} else {

return 0;

}

}

}

bool a\_lower\_or\_eq\_b(string a, string b) {

int v1 = 0;

int v2 = 0;

int i = a.size();

int k = 1;

while (i > 0) {

i--;

v1 += char\_to\_int(a[i])\*k;

k\*=10;

}

i = b.size();

k = 1;

while (i > 0) {

i--;

v2 += char\_to\_int(b[i])\*k;

k\*=10;

}

if (v1 <= v2) {

return true;

}

return false;

}

void merge(string \*orig, int left, int mid, int right, string \*modif) {

int l = left;

int r = mid;

for (int i = left; i < right; i++)

if (l < mid && (r >= right || a\_lower\_or\_eq\_b(orig[l], orig[r])))

modif[i] = orig[l++];

else

modif[i] = orig[r++];

for (int i = left; i < right; i++)

orig[i] = modif[i];

}

void\* split(void\* param) {

struct arguments\* temp\_args = new arguments;

temp\_args = (arguments\*)param;

if (temp\_args->right - temp\_args->left < 2) return NULL;

int tmp\_right = temp\_args->right;

int tmp\_left = temp\_args->left;

temp\_args->right = (tmp\_left + tmp\_right) / 2;

split((void\*)temp\_args);

temp\_args->right = tmp\_right;

temp\_args->left = (tmp\_left + tmp\_right) / 2;

split((void\*)temp\_args);

temp\_args->left = tmp\_left;

merge(temp\_args->orig, temp\_args->left, (temp\_args->left + temp\_args->right) / 2, temp\_args->right, temp\_args->mod);

}

void merge\_sort(string \*in) {

struct arguments\* a = new arguments;

string temp[Length];

pthread\_t threads[NumberOfThreads];

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

int new\_left = i \* Length / NumberOfThreads;

int new\_right = (i + 1) \* Length / NumberOfThreads;

a->mod = temp;

a->orig = in;

a->left = new\_left;

a->right = new\_right;

pthread\_mutex\_init(&mutex, NULL);

pthread\_create(&threads[i], NULL, split, (void\*)a);

pthread\_join(threads[i], NULL);

pthread\_mutex\_destroy(&mutex);

}

for (int i = NumberOfThreads/2; i > 0; i = i >> 1) //divide by 2

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

int left = (j)\*Length / i;

int right = (j + 1)\*Length / i;

merge(in, left, (left + right) / 2, right, temp);

}

}

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

cin >> Length;

string array[Length];

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

cin >> array[i];

NumberOfThreads = atoi(argv[1]);

int power = 0;

while (NumberOfThreads > 0) {

NumberOfThreads = NumberOfThreads >> 1;

power++;

}

//power--;

NumberOfThreads = (int)pow(2.0, (double)(power));

merge\_sort(array);

cout << "Sorted array:\n";

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

cout << array[i] << "\n";

return 0;

}

**Вывод**

Обрел навыки работы с системным вызовом fork(). Ознакомился с принципами работы pipe(), и применил этот системный вызов для передачи данных дочернему процессу. Изучил принципы работы буфера потоков, а также ознакомился с устройством взаимодействия процессов в ОС Linux.