Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**Тема работы**

**“Потоки”**

Студент: Слободин Никита Алексеевич

Группа: М8О-203Б-23

Вариант: 8

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

**Постановка задачи**

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение максимального количества потоков, работающих в один момент времени, должно быть задано ключом запуска вашей программы.  
8. Есть К массивов одинаковой длины. Необходимо сложить эти массивы. Необходимо предусмотреть стратегию, адаптирующуюся под количество массивов и их длину (по количеству операций)

**Общие сведения о программе**

Программа суммирует массивы, при этом адаптируясь под их длину и количество.

Программа демонстрирует использование многопоточности для ускорения вычислений, что полезно при проведении большого количества симуляций (экспериментов).  
Исходный код представлен в приложении.

**Вывод.**

Работая над данной лабораторной работой, я понял, как устроена многопоточность в ОС, какие есть преимущества и недостатки в работе с потоками. Научился использовать функции pthread\_create и pthread\_join для управления потоками. Так же выяснилось, что при излишне большом количестве потоков, программа выполняется дольше, чем при минимально необходимом количестве.

Приложение

summing.cpp

#include "summing.h"

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <vector>

int\*\* create\_arrays(int K, int N, int value){

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

if(arrays == nullptr){

return nullptr;

}

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

arrays[i] = (int\*)malloc(N \* sizeof(int));

if(arrays[i] == nullptr){

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

free(arrays[j]);

}

free(arrays);

return nullptr;

}

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

arrays[i][j] = value;

}

}

return arrays;

}

void free\_arrays(int\*\* arrays, int K){

if(arrays == nullptr){

return;

}

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

free(arrays[i]);

}

free(arrays);

}

int\* create\_result(int N){

int\* result = (int\*)malloc(N \* sizeof(int));

if(result == nullptr){

return nullptr;

}

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

result[i] = 0;

}

return result;

}

void free\_result(int\* result){

free(result);

}

struct ThreadData { // Для стратегии sum\_chunk

int start; // Начальный индекс для этого потока

int end; // Конечный индекс (не включительно) для этого потока

int K; // Количество массивов

int \*\*arrays; // Указатель на массивы

int \*result; // Указатель на результирующий массив

pthread\_mutex\_t \*mutex; // Указатель на мьютекс

pthread\_cond\_t \*cond; // Указатель на условную переменную

int \*active\_threads; // Указатель на количество активных потоков

int \*max\_active\_threads; // Указатель на максимальное количество активных потоков

};

struct ArrayThreadData { // Для стратегии sum\_array

int start\_array; // Начальный индекс массива для этого потока

int end\_array; // Конечный индекс массива (не включительно) для этого потока

int N; // Длина массивов

int \*\*arrays; // Указатель на массивы

int \*partial\_result; // Указатель на частичный результирующий массив

pthread\_mutex\_t \*mutex; // Указатель на мьютекс

pthread\_cond\_t \*cond; // Указатель на условную переменную

int \*active\_threads; // Указатель на количество активных потоков

int \*max\_active\_threads; // Указатель на максимальное количество активных потоков

};

void\* sum\_chunk(void\* arg) { // Стратегия sum\_chunk

ThreadData\* data = (ThreadData\*)arg;

// Сумируем чанки поэлементно

for(int i = data->start; i < data->end; ++i){

int sum = 0;

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

sum += data->arrays[j][i];

}

data->result[i] = sum;

}

// Отслеживаем максимум используемых потоков

pthread\_mutex\_lock(data->mutex);

(\*(data->active\_threads))--;

pthread\_cond\_signal(data->cond); // Завершение потока

pthread\_mutex\_unlock(data->mutex);

free(arg);

return NULL;

}

void\* sum\_array(void\* arg) { // Стратегия sum\_array

ArrayThreadData\* data = (ArrayThreadData\*)arg;

// Суммируем назначенные массивы и записываем в частичный результат

for(int i = data->start\_array; i < data->end\_array; ++i){

for(int j = 0; j < data->N; ++j){

data->partial\_result[j] += data->arrays[i][j];

}

}

// Отслеживаем максимум используемых потоков

pthread\_mutex\_lock(data->mutex);

(\*(data->active\_threads))--;

pthread\_cond\_signal(data->cond); // Завершение потока

pthread\_mutex\_unlock(data->mutex);

free(arg);

return NULL;

}

bool sum\_arrays(int K, int N, int\*\* arrays, int\* result, int max\_threads, int& max\_active\_threads){

if(K <= 0 || N <= 0 || max\_threads <= 0 || arrays == nullptr || result == nullptr){

fprintf(stderr, "Некорректные входные данные: K=%d, N=%d\n", K, N);

return false;

}

pthread\_mutex\_t mutex;

if(pthread\_mutex\_init(&mutex, NULL) != 0){

perror("Не удалось инициализировать мьютекс");

return false;

}

pthread\_cond\_t cond;

if(pthread\_cond\_init(&cond, NULL) != 0){

perror("Не удалось инициализировать условную переменную");

pthread\_mutex\_destroy(&mutex);

return false;

}

int active\_threads = 0;

max\_active\_threads = 0;

std::vector<pthread\_t> threads;

if((2 \* K) > N){ // Используем стратегию sum\_array

int num\_threads = std::min(max\_threads, K); // Не создавать больше потоков, чем массивов

int arrays\_per\_thread = K / num\_threads;

int remaining\_arrays = K % num\_threads;

// Для избежания Race Condition даем каждому потоку складывать часть результата в своем частичном результирующем массиве

std::vector<int\*> partial\_results(num\_threads, nullptr);

for(int t = 0; t < num\_threads; ++t){

partial\_results[t] = (int\*)calloc(N, sizeof(int));

if(partial\_results[t] == nullptr){

perror("Не удалось выделить память для частичного результирующего массива");

// Очистка и завершение

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

free(partial\_results[s]);

}

pthread\_mutex\_destroy(&mutex);

pthread\_cond\_destroy(&cond);

return false;

}

}

int current\_array = 0;

for(int t = 0; t < num\_threads; ++t){

int start\_array = current\_array;

int end\_array = start\_array + arrays\_per\_thread + (t < remaining\_arrays ? 1 : 0);

current\_array = end\_array;

// Выделение и настройка данных для потока

ArrayThreadData\* data = (ArrayThreadData\*)malloc(sizeof(ArrayThreadData));

if(data == nullptr){

perror("Не удалось выделить память для ArrayThreadData");

// Очистка и завершение

for(auto &thread : threads){

pthread\_join(thread, NULL);

}

for(int s = 0; s < num\_threads; ++s){

free(partial\_results[s]);

}

pthread\_mutex\_destroy(&mutex);

pthread\_cond\_destroy(&cond);

return false;

}

data->start\_array = start\_array;

data->end\_array = end\_array;

data->N = N;

data->arrays = arrays;

data->partial\_result = partial\_results[t];

data->mutex = &mutex;

data->cond = &cond;

data->active\_threads = &active\_threads;

data->max\_active\_threads = &max\_active\_threads;

// Ограничение количества активных потоков

pthread\_mutex\_lock(&mutex);

while(active\_threads >= max\_threads){

pthread\_cond\_wait(&cond, &mutex);

}

active\_threads++;

if(active\_threads > max\_active\_threads){

max\_active\_threads = active\_threads;

}

pthread\_mutex\_unlock(&mutex);

// Создание рабочего потока

pthread\_t thread;

if(pthread\_create(&thread, NULL, sum\_array, data) != 0){

perror("Не удалось создать поток");

// Очистка

pthread\_mutex\_lock(&mutex);

active\_threads--;

pthread\_cond\_signal(&cond);

pthread\_mutex\_unlock(&mutex);

free(data);

continue;

}

// Добавление потока в вектор

threads.push\_back(thread);

}

// Ожидание завершения всех рабочих потоков

for(auto &thread : threads){

pthread\_join(thread, NULL);

}

// Объединение частичных результатов в окончательный результирующий массив

for(int t = 0; t < num\_threads; ++t){

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

result[j] += partial\_results[t][j];

}

free(partial\_results[t]);

}

}

else{ // Используем стратегию sum\_chunk

int num\_chunks = std::min(N, max\_threads \* 4);

int chunk\_size = N / num\_chunks;

if(chunk\_size == 0){

chunk\_size = 1;

num\_chunks = N;

}

for(int c = 0; c < num\_chunks; ++c){

int start = c \* chunk\_size;

int end = (c == num\_chunks - 1) ? N : start + chunk\_size;

// Выделение и настройка данных для потока

ThreadData\* data = (ThreadData\*)malloc(sizeof(ThreadData));

if(data == nullptr){

perror("Не удалось выделить память для ThreadData");

// Очистка и завершение

for(auto &thread : threads){

pthread\_join(thread, NULL);

}

pthread\_mutex\_destroy(&mutex);

pthread\_cond\_destroy(&cond);

return false;

}

data->start = start;

data->end = end;

data->K = K;

data->arrays = arrays;

data->result = result;

data->mutex = &mutex;

data->cond = &cond;

data->active\_threads = &active\_threads;

data->max\_active\_threads = &max\_active\_threads;

// Ограничение количества активных потоков

pthread\_mutex\_lock(&mutex);

while(active\_threads >= max\_threads){

pthread\_cond\_wait(&cond, &mutex);

}

active\_threads++;

if(active\_threads > max\_active\_threads){

max\_active\_threads = active\_threads;

}

pthread\_mutex\_unlock(&mutex);

// Создание рабочего потока

pthread\_t thread;

if(pthread\_create(&thread, NULL, sum\_chunk, data) != 0){

perror("Не удалось создать поток");

// Очистка

pthread\_mutex\_lock(&mutex);

active\_threads--;

pthread\_cond\_signal(&cond);

pthread\_mutex\_unlock(&mutex);

free(data);

continue;

}

// Добавление потока в вектор

threads.push\_back(thread);

}

// Ожидание завершения всех рабочих потоков

for(auto &thread : threads){

pthread\_join(thread, NULL);

}

}

pthread\_mutex\_destroy(&mutex);

pthread\_cond\_destroy(&cond);

return true;

}

Пример вывода:

root@c34508d80232:/workspaces/OS\_MAI\_Slobodin/build# ./lab2/multithread

Введите количество массивов (K): 1000

Введите длину каждого массива (N): 1000

Общая сумма всех элементов результирующего массива: 1000000

Суммирование завершено.

Максимальное количество одновременно работающих потоков: 4

Время суммирования: 2.00417 мс

root@c34508d80232:/workspaces/OS\_MAI\_Slobodin/build#