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

```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#
```