

Федеральное агентство связи
Сибирский государственный университет телекоммуникаций и
информатики

Лабораторная работа №8

Выполнил: студент группы ИП-211
Оганесян Альберт
Лацук Андрей
Проверил:
Профессор кафедры ПМиК
Малков Е. А.

Новосибирск 2024

Задание: программно реализуйте вычисление суммы последовательности чисел на основе последовательного кода, интерфейсов Pthreads и C++11 <thread>. Сравните время вычислений.

Цель: знакомство с программными интерфейсами управления потоками в Linux.

Ход работы:

1. Напишем код для вычисления суммы последовательности, без использования многопоточности. Для измерения времени будем использовать **“ctime”**:

```
#include <iostream>
#include <vector>
#include <ctime>

long long sum_sequence(const std::vector<int>& numbers) {
    long long sum = 0;
    for (int num : numbers) {
        sum += num;
    }
    return sum;
}

int main() {
    const int N = 100000000;
    std::vector<int> numbers(N, 1); // создаем массив из N чисел,
    равных 1

    clock_t start = clock();
    long long sum = sum_sequence(numbers);
    clock_t end = clock();

    double elapsed_time = double(end - start) / CLOCKS_PER_SEC;
    std::cout << "Сумма равна: " << sum << std::endl;
    std::cout << "Затраченное время: " << elapsed_time << "
секунд" << std::endl;

    return 0;
}
```

2. Напишем такой же код, но с использованием интерфейсов Pthreads.
Для сборки используем `gcc pthread.c -lpthread -o pthread`.

Код программы:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>

#define ARRAY_SIZE 1000000
#define NUM_THREADS 8

typedef struct
{
    int start;
    int end;
    int *array;
    long long partial_sum;
} ThreadData;

void *calculate_partial_sum(void *arg)
{
    ThreadData *data = (ThreadData *)arg;
    data->partial_sum = 0;
    for (int i = data->start; i < data->end; i++)
    {
        data->partial_sum += data->array[i];
    }
    return NULL;
}

int main()
{
    int array[ARRAY_SIZE];
    pthread_t threads[NUM_THREADS];
    ThreadData thread_data[NUM_THREADS];
    long long total_sum = 0;
    clock_t start_time, end_time;

    for (int i = 0; i < ARRAY_SIZE; i++)
    {
        array[i] = 1;
    }
}
```

```

start_time = clock();

int segment_size = ARRAY_SIZE / NUM_THREADS;
for (int i = 0; i < NUM_THREADS; i++)
{
    thread_data[i].start = i * segment_size;
    thread_data[i].end = (i == NUM_THREADS - 1) ? ARRAY_SIZE : (i +
1) * segment_size;
    thread_data[i].array = array;

    pthread_create(&threads[i], NULL, calculate_partial_sum,
&thread_data[i]);
}

for (int i = 0; i < NUM_THREADS; i++)
{
    pthread_join(threads[i], NULL);
    total_sum += thread_data[i].partial_sum;
}

end_time = clock();

printf("Сумма элементов массива: %lld\n", total_sum);
printf("Время выполнения: %.6f секунд\n", (double)(end_time -
start_time) / CLOCKS_PER_SEC);

return 0;
}

```

3. Теперь напомним код, использующий интерфейсы C++11 <thread>:

```

#include <iostream>
#include <vector>
#include <thread>
#include <ctime>

#define ARRAY_SIZE 1000000
#define NUM_THREADS 8

void calculate_partial_sum(const std::vector<int> &array, int start,
int end, long long &partial_sum)
{
    partial_sum = 0;
    for (int i = start; i < end; i++)
    {

```

```

        partial_sum += array[i];
    }
}

int main()
{
    std::vector<int> array(ARRAY_SIZE);
    long long total_sum = 0;
    std::vector<std::thread> threads(NUM_THREADS);
    std::vector<long long> partial_sums(NUM_THREADS);

    for (int i = 0; i < ARRAY_SIZE; i++)
    {
        array[i] = 1;
    }

    clock_t start_time = clock();

    int segment_size = ARRAY_SIZE / NUM_THREADS;
    for (int i = 0; i < NUM_THREADS; i++)
    {
        int start = i * segment_size;
        int end = (i == NUM_THREADS - 1) ? ARRAY_SIZE : (i + 1) *
segment_size;

        threads[i] = std::thread(calculate_partial_sum, std::cref(array),
start, end, std::ref(partial_sums[i]));
    }

    for (int i = 0; i < NUM_THREADS; i++)
    {
        threads[i].join();
    }

    for (const auto &partial_sum : partial_sums)
    {
        total_sum += partial_sum;
    }

    clock_t end_time = clock();
    double duration = double(end_time - start_time) / CLOCKS_PER_SEC;

    std::cout << "Сумма элементов массива: " << total_sum <<
std::endl;

```

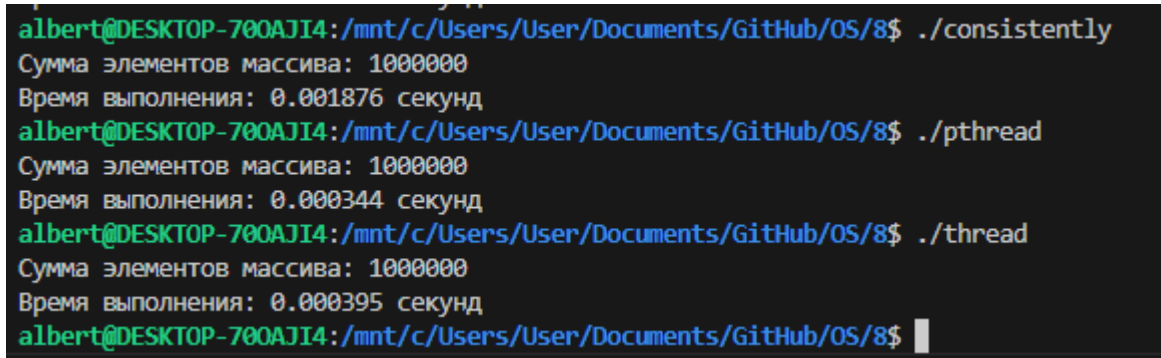
```

std::cout << "Время выполнения: " << duration << " секунд" <<
std::endl;

return 0;
}

```

4. Запустим каждую программу и сравним время (Рис. 4.1). Программы с использованием многопоточности справились с задачей примерно в 5 раз быстрее.



```

albert@DESKTOP-700AJI4:/mnt/c/Users/User/Documents/GitHub/OS/8$ ./consistently
Сумма элементов массива: 1000000
Время выполнения: 0.001876 секунд
albert@DESKTOP-700AJI4:/mnt/c/Users/User/Documents/GitHub/OS/8$ ./pthread
Сумма элементов массива: 1000000
Время выполнения: 0.000344 секунд
albert@DESKTOP-700AJI4:/mnt/c/Users/User/Documents/GitHub/OS/8$ ./thread
Сумма элементов массива: 1000000
Время выполнения: 0.000395 секунд
albert@DESKTOP-700AJI4:/mnt/c/Users/User/Documents/GitHub/OS/8$

```

Рис. 4.1. вывод программ

Вывод: мы познакомились с программными интерфейсами управления потоками в Linux и научились использовать их для улучшения производительности программ.