Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

**Лабораторная работа № 2 по курсу**

**«Операционные системы»**

Группа: М8О-214Б-23

Студент: Шестаков К. Р.

Преподаватель: Бахарев В.Д.

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

Дата: 15.11.24

Москва, 2024

**Постановка задачи**

**Вариант 8.**

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.

Есть К массивов одинаковой длины. Необходимо сложить эти массивы.

**Общий метод и алгоритм решения**

В данной лабораторной работе реализованы две версии программы для многопоточного суммирования массивов на языке Си с использованием WinAPI. Первая версия использует мьютекс для синхронизации доступа к общему ресурсу (результирующему массиву). Вторая версия применяет атомарные операции для инкремента значений в результирующем массиве, исключая необходимость явной блокировки. Основной алгоритм заключается в разделении задачи на подзадачи, каждая из которых обрабатывается отдельным потоком.

Использованные системные вызовы включают CreateThread для создания потоков, WaitForSingleObject и WaitForMultipleObjects для ожидания завершения потоков, CreateMutex для создания мьютекса, ReleaseMutex для освобождения мьютекса, и InterlockedExchangeAdd для атомарного сложения значений.

**Код программы**

**threads\_p.c**

#include <windows.h>

#include <stdlib.h>

#include <stdio.h>

typedef struct

{

    int \*data;

    int length;

} Array;

typedef struct

{

    Array \*arrays;

    int \*result;

    int start;

    int end;

    int array\_count;

    HANDLE mutex;

} ThreadData;

DWORD WINAPI ThreadFunction(LPVOID arg)

{

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

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

    {

        int sum = 0;

        for (int j = 0; j < data->array\_count; ++j)

        {

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

        }

        WaitForSingleObject(data->mutex, INFINITE);

        data->result[i] = sum;

        ReleaseMutex(data->mutex);

    }

    return 0;

}

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

{

    if (argc < 3)

    {

        MessageBoxW(NULL, L"Not enough arguments", L"Error", MB\_OK);

        return 1;

    }

    int max\_threads = atoi(argv[1]);

    int array\_count = atoi(argv[2]);

    int array\_length = 1000;

    Array \*arrays = (Array \*)malloc(array\_count \* sizeof(Array));

    for (int i = 0; i < array\_count; ++i)

    {

        arrays[i].data = (int \*)malloc(array\_length \* sizeof(int));

        arrays[i].length = array\_length;

        for (int j = 0; j < array\_length; ++j)

        {

            arrays[i].data[j] = rand() % 100;

        }

    }

    int \*result = (int \*)malloc(array\_length \* sizeof(int));

    HANDLE mutex = CreateMutex(NULL, FALSE, NULL);

    LARGE\_INTEGER frequency;

    LARGE\_INTEGER start, end;

    QueryPerformanceFrequency(&frequency);

    QueryPerformanceCounter(&start);

    HANDLE \*threads = (HANDLE \*)malloc(max\_threads \* sizeof(HANDLE));

    ThreadData \*threadData = (ThreadData \*)malloc(max\_threads \* sizeof(ThreadData));

    int chunk\_size = array\_length / max\_threads;

    for (int i = 0; i < max\_threads; ++i)

    {

        threadData[i].arrays = arrays;

        threadData[i].result = result;

        threadData[i].start = i \* chunk\_size;

        threadData[i].end = (i == max\_threads - 1) ? array\_length : (i + 1) \* chunk\_size;

        threadData[i].array\_count = array\_count;

        threadData[i].mutex = mutex;

        threads[i] = CreateThread(NULL, 0, ThreadFunction, &threadData[i], 0, NULL);

        if (threads[i] == NULL)

        {

            MessageBoxW(NULL, L"Could not create thread", L"Error", MB\_OK);

            free(threads);

            free(threadData);

            return 1;

        }

    }

    WaitForMultipleObjects(max\_threads, threads, TRUE, INFINITE);

    QueryPerformanceCounter(&end);

    double elapsed\_time = (double)(end.QuadPart - start.QuadPart) / frequency.QuadPart;

    wchar\_t message[256];

    swprintf\_s(message, 256, L"Elapsed time: %f seconds", elapsed\_time);

    MessageBoxW(NULL, message, L"Time", MB\_OK);

    for (int i = 0; i < array\_count; ++i)

    {

        free(arrays[i].data);

    }

    free(arrays);

    free(result);

    free(threads);

    free(threadData);

    CloseHandle(mutex);

    return 0;

}

**threads\_a.c**

#include <windows.h>

#include <stdlib.h>

#include <stdio.h>

typedef struct

{

    int \*data;

    int length;

} Array;

typedef struct

{

    Array \*arrays;

    volatile LONG \*result;

    int start;

    int end;

    int array\_count;

} ThreadData;

DWORD WINAPI ThreadFunction(LPVOID arg)

{

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

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

    {

        int sum = 0;

        for (int j = 0; j < data->array\_count; ++j)

        {

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

        }

        InterlockedExchangeAdd(data->result + i, sum);

    }

    return 0;

}

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

{

    if (argc < 3)

    {

        MessageBoxW(NULL, L"Not enough arguments", L"Error", MB\_OK);

        return 1;

    }

    int max\_threads = atoi(argv[1]);

    int array\_count = atoi(argv[2]);

    int array\_length = 1000;

    Array \*arrays = (Array \*)malloc(array\_count \* sizeof(Array));

    for (int i = 0; i < array\_count; ++i)

    {

        arrays[i].data = (int \*)malloc(array\_length \* sizeof(int));

        arrays[i].length = array\_length;

        for (int j = 0; j < array\_length; ++j)

        {

            arrays[i].data[j] = rand() % 100;

        }

    }

    volatile LONG \*result = (volatile LONG \*)malloc(array\_length \* sizeof(LONG));

    for (int i = 0; i < array\_length; ++i)

    {

        result[i] = 0;

    }

    LARGE\_INTEGER frequency;

    LARGE\_INTEGER start, end;

    QueryPerformanceFrequency(&frequency);

    QueryPerformanceCounter(&start);

    HANDLE \*threads = (HANDLE \*)malloc(max\_threads \* sizeof(HANDLE));

    ThreadData \*threadData = (ThreadData \*)malloc(max\_threads \* sizeof(ThreadData));

    int chunk\_size = array\_length / max\_threads;

    for (int i = 0; i < max\_threads; ++i)

    {

        threadData[i].arrays = arrays;

        threadData[i].result = result;

        threadData[i].start = i \* chunk\_size;

        threadData[i].end = (i == max\_threads - 1) ? array\_length : (i + 1) \* chunk\_size;

        threadData[i].array\_count = array\_count;

        threads[i] = CreateThread(NULL, 0, ThreadFunction, &threadData[i], 0, NULL);

        if (threads[i] == NULL)

        {

            MessageBoxW(NULL, L"Could not create thread", L"Error", MB\_OK);

            free(threads);

            free(threadData);

            return 1;

        }

    }

    WaitForMultipleObjects(max\_threads, threads, TRUE, INFINITE);

    QueryPerformanceCounter(&end);

    double elapsed\_time = (double)(end.QuadPart - start.QuadPart) / frequency.QuadPart;

    wchar\_t message[256];

    swprintf\_s(message, 256, L"Elapsed time: %f seconds", elapsed\_time);

    MessageBoxW(NULL, message, L"Time", MB\_OK);

    for (int i = 0; i < array\_count; ++i)

    {

        free(arrays[i].data);

    }

    free(arrays);

    free((void \*)result);

    free(threads);

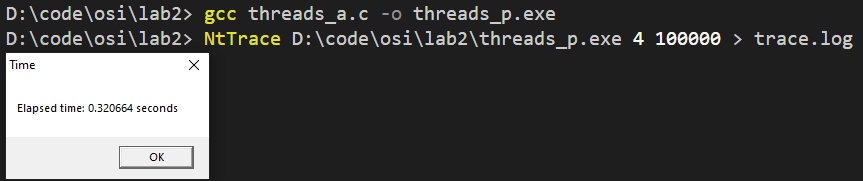
    free(threadData);

    return 0;

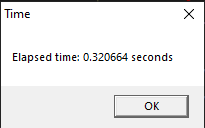
}

**Протокол работы программы**

**Тестирование:**



(Скриншот консоли программы)



(Вывод времени выполнения через MessageBoxW)

**nttrace:**

...

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x74137fec50 [0x000001e15b786000], ZeroBits=0, pSize=0x74137fecf8 [0x1000], flAllocationType=0x1000, flProtect=4) => 0

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x74137fec50 [0x000001e15b787000], ZeroBits=0, pSize=0x74137fecf8 [0x1000], flAllocationType=0x1000, flProtect=4) => 0

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x74137fec50 [0x000001e15b788000], ZeroBits=0, pSize=0x74137fecf8 [0x1000], flAllocationType=0x1000, flProtect=4) => 0

**NtCreateThreadEx**(ThreadHandle=0x74137feea8 [0xcc], DesiredAccess=DELETE|READ\_CONTROL|WRITE\_DAC|WRITE\_OWNER|SYNCHRONIZE|0xffff, ObjectAttributes=null, ProcessHandle=-1, StartRoutine=0x7ff77e6a1450, Argument=0x1e15a111df0, CreateFlags=0, ZeroBits=0, StackSize=0, MaximumStackSize=0, AttributeList=0x74137fefd0) => 0

Created thread: 4952 at 00007FF77E6A1450

NtCreateThreadEx(ThreadHandle=0x74137feea8 [0xd0], DesiredAccess=DELETE|READ\_CONTROL|WRITE\_DAC|WRITE\_OWNER|SYNCHRONIZE|0xffff, ObjectAttributes=null, ProcessHandle=-1, StartRoutine=0x7ff77e6a1450, Argument=0x1e15a111e10, CreateFlags=0, ZeroBits=0, StackSize=0, MaximumStackSize=0, AttributeList=0x74137fefd0) => 0

Created thread: 13784 at 00007FF77E6A1450

NtDeviceIoControlFile(FileHandle=0x50, Event=0, ApcRoutine=null, ApcContext=null, IoStatusBlock=0x74139fed80, IoControlCode=0x00500016, InputBuffer=0x74139fed90, InputBufferLength=0x30, OutputBuffer=null, OutputBufferLength=0) => 0xc00700bb [187 'Не найдено указанное имя системного семафора.']

Created thread: 10664 at 00007FF77E6A1450

NtCreateThreadEx(ThreadHandle=0x74137feea8 [0xb4], DesiredAccess=DELETE|READ\_CONTROL|WRITE\_DAC|WRITE\_OWNER|SYNCHRONIZE|0xffff, ObjectAttributes=null, ProcessHandle=-1, StartRoutine=0x7ff77e6a1450, Argument=0x1e15a111e30, CreateFlags=0, ZeroBits=0, StackSize=0, MaximumStackSize=0, AttributeList=0x74137fefd0) => 0

Created thread: 18024 at 00007FF77E6A1450

NtWaitForSingleObject(Handle=0x40, Alertable=false, Timeout=null) => 0

NtAllocateVirtualMemory(ProcessHandle=-1, lpAddress=0x74139fe990 [0x000001e15b789000], ZeroBits=0, pSize=0x74139fea38 [0x1000], flAllocationType=0x1000, flProtect=4) => 0

**NtCreateThreadEx**(ThreadHandle=0x74137feea8 [0xd4], DesiredAccess=DELETE|READ\_CONTROL|WRITE\_DAC|WRITE\_OWNER|SYNCHRONIZE|0xffff, ObjectAttributes=null, ProcessHandle=-1, StartRoutine=0x7ff77e6a1450, Argument=0x1e15a111e50, CreateFlags=0, ZeroBits=0, StackSize=0, MaximumStackSize=0, AttributeList=0x74137fefd0) => 0

NtSetEvent(EventHandle=0x40, PrevState=null) => 0

NtTestAlert() => 0

NtWaitForSingleObject(Handle=0x40, Alertable=false, Timeout=null) => 0

...

**Табличка график с тестированием по времениля 100000 массивов**

**Примитивы**

|  |  |  |  |
| --- | --- | --- | --- |
| Число потоков | Время исполнения (мс) | Ускорение | Эффективность |
| 1 | 990 | 1 | 1 |
| 2 | 520 | 1,9 | 0,95 |
| 3 | 390 | 2,54 | 0,847 |
| 4 | 340 | 2,91 | 0,728 |
| 5 | 310 | 3,19 | 0,638 |
| 6 | 300 | 3,3 | 0,55 |

**Атомики**

|  |  |  |  |
| --- | --- | --- | --- |
| Число потоков | Время исполнения (мс) | Ускорение | Эффективность |
| 1 | 980 | 1 | 1 |
| 2 | 515 | 1,9 | 0,95 |
| 3 | 385 | 2,55 | 0,85 |
| 4 | 335 | 2,93 | 0,733 |
| 5 | 305 | 3,21 | 0,642 |
| 6 | 295 | 3,32 | 0,553 |

Результаты показывают, что с увеличением числа потоков время выполнения программы уменьшается как для примитивов синхронизации, так и для атомиков. Эффективность снижается с ростом числа потоков, что говорит о насыщении ресурсов и увеличении конкуренции за них. Атомики показывают немного лучшую производительность по сравнению с примитивами синхронизации за счёт меньших накладных расходов.

**Вывод**

В данной лабораторной работе реализованы две версии программы для многопоточного суммирования массивов на языке Си с использованием WinAPI. При выполнении лабораторной работы основной проблемой была необходимость ручного управления памятью и синхронизацией потоков. Например, требовалось явно освобождать выделенную память для массивов и потоков, а также корректно инициализировать мьютекс и использовать атомарные операции. Также возникли сложности с преобразованием типов данных для корректного вывода информации через MessageBoxW.