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»

Выполнил: Д. И. Шнайдер
Группа: М8О-208БВ-24
Преподаватель: Е. С. Миронов

Москва, 2025

Условие

Разработать программу для решения системы линейных уравнений методом Гаусса с использованием многопоточности. Программа должна обеспечивать ограничение максимального количества потоков, работающих одновременно.

Цель работы: Приобретение практических навыков управления потоками в операционной системе Windows, а также организация параллельных вычислений с синхронизацией доступа к общим ресурсам.

Задание: Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке использовать стандартные средства создания потоков Windows API. Ограничение максимального количества потоков, работающих в один момент времени, должно задаваться параметром запуска программы.

Вариант: 10

Метод решения

Для решения системы линейных уравнений применен метод Гаусса с параллельной обработкой строк матрицы. Основная идея заключается в распараллеливании операций исключения элементов в прямом ходе метода Гаусса.

Основной алгоритм работы

1. **Инициализация:** Создание семафора для ограничения количества одновременно работающих потоков
2. **Прямой ход:** Приведение матрицы к треугольному виду:
    - Выбор главного элемента в текущем столбце
    - Параллельное исключение элементов в строках ниже текущей
    - Каждая строка обрабатывается в отдельном потоке
3. **Обратный ход:** Последовательное нахождение неизвестных из треугольной системы
4. **Завершение:** Освобождение ресурсов и уничтожение семафора

Особенности реализации

Для обеспечения многопоточности использованы средства Windows API. Реализовано ограничение максимального количества одновременно работающих потоков с помощью семафора.

Описание программы

Программа реализована в модульном стиле и состоит из трех основных компонентов.

Модуль main.c

Реализует основную логику программы:

- Обработка аргументов командной строки (-n, -t, -r, -f)
- Выделение памяти под матрицу и векторы
- Генерация случайной диагонально-доминируемой матрицы
- Чтение матрицы из файла (при указании параметра -f)
- Вызов функции решения системы уравнений
- Замер времени выполнения и вывод результатов

Модуль gauss.h/gauss.c

Содержит реализацию метода Гаусса с многопоточностью:

- `gauss_solve()` - основная функция решения СЛАУ
- Параллельное выполнение операций исключения для строк матрицы
- Выбор главного элемента для обеспечения устойчивости
- Управление потоками с использованием Windows API (CreateThread, WaitForMultipleObjects)
- Обработка ошибок вырожденной матрицы

Модуль threads_lim.h/threads_lim.c

Предоставляет механизм ограничения количества потоков:

- `threads_lim_init()` - инициализация семафора
- `threads_lim_acquire()` - захват семафора (уменьшение счетчика)
- `threads_lim_release()` - освобождение семафора (увеличение счетчика)
- `threads_lim_destroy()` - уничтожение семафора

Используемые системные вызовы Windows

- **Управление потоками:** CreateThread, WaitForMultipleObjects, CloseHandle
- **Синхронизация:** CreateSemaphore, WaitForSingleObject, ReleaseSemaphore
- **Таймер:** QueryPerformanceFrequency, QueryPerformanceCounter
- **Память:** malloc, free, calloc

Архитектура программы обеспечивает эффективное распараллеливание вычислительно сложных операций метода Гаусса при сохранении контроля над количеством используемых потоков.

Результаты экспериментов

Условия проведения экспериментов

Эксперименты проводились на компьютере со следующими характеристиками:

- **Процессор:** 12th Gen Intel(R) Core(TM) i3-1215U
- **Количество ядер:** 6
- **Количество потоков:** 8
- **Оперативная память:** 8 ГБ (2 × 4 ГБ)

Для обеспечения воспроизводимости результатов использовался фиксированный seed (-r 123), гарантирующий идентичные матрицы для всех тестов.

Зависимость времени выполнения от количества потоков

Количество потоков	Время выполнения, с	Ускорение	Размер матрицы
1	1.823	1.00	300×300
2	1.497	1.22	300×300
4	1.302	1.40	300×300
8	1.248	1.46	300×300

Таблица 1: Зависимость времени решения СЛАУ от количества потоков

Количество потоков	Время выполнения, с	Ускорение	Размер матрицы
1	5.372	1.00	500×500
2	4.131	1.30	500×500
4	3.682	1.46	500×500
8	3.612	1.49	500×500

Таблица 2: Зависимость времени решения СЛАУ от количества потоков

Комментарий: При увеличении количества потоков наблюдается ускорение вычислений для обоих размеров матриц. Для матрицы 300×300 максимальное ускорение составляет 1.46 раза при использовании 8 потоков, для матрицы 500×500 - 1.49 раза. Наибольший прирост производительности наблюдается при переходе от 1 к 2 потокам, что объясняется эффективным использованием двух физических ядер процессора.

Зависимость времени выполнения от размера матрицы

Комментарий: Время решения демонстрирует кубическую зависимость от размера матрицы, что соответствует теоретической сложности $O(n^3)$ метода Гаусса. При увеличении размера матрицы с 100×100 до 500×500 (в 5 раз по линейному размеру) время выполнения возрастает в 23.6 раза, что меньше ожидаемого увеличения в $5^3 = 125$ раз из-за оптимизаций процессора и эффектов кэширования.

Размер матрицы	Время выполнения, с	Относительное время	Количество потоков
100×100	0.156	1.0	4
200×200	0.603	3.9	4
300×300	1.302	8.3	4
500×500	3.682	23.6	4

Таблица 3: Зависимость времени решения СЛАУ от размера матрицы

Количество потоков	Время, с	Ускорение	Эффективность, %
1	1.823	1.00	100.0
2	1.497	1.22	61.0
4	1.302	1.40	35.0
8	1.248	1.46	18.3

Таблица 4: Эффективность параллелизации для матрицы 300×300

Эффективность параллелизации

Комментарий: Эффективность параллелизации рассчитывается по формуле:

$$\text{Эффективность} = \frac{\text{Ускорение}}{\text{Количество потоков}} \times 100\%$$

С увеличением количества потоков эффективность значительно снижается. Наибольшая эффективность (65%) достигается при использовании 2 потоков для матрицы 500×500. При 8 потоках эффективность падает до 18.6%, что объясняется накладными расходами на синхронизацию, создание потоков и конкуренцией за доступ к общей памяти, а также ограничениями 6-ядерного процессора при работе с 8 потоками.

Выводы по экспериментам

1. **Оптимальное количество потоков** для решения СЛАУ методом Гаусса составляет 2-4 потока, что соответствует физической архитектуре 6-ядерного процессора.

2. **Эффективность параллелизации** максимальна при малом количестве потоков и резко снижается при увеличении их числа, достигая 18% при использовании 8 потоков.

3. **Время выполнения** демонстрирует кубическую зависимость от размера матрицы, подтверждая теоретическую сложность $O(n^3)$ алгоритма Гаусса.

4. **Наибольшее ускорение** достигается при переходе от последовательной версии к использованию 2 потоков, дальнейшее увеличение количества потоков дает **затухающую отдачу**.

5. **Механизм ограничения потоков** успешно выполняет свою функцию, предотвращая избыточное использование ресурсов системы и обеспечивая предсказуемое поведение программы.

6. **Для больших матриц** (500×500) параллелизация эффективнее - ускорение достигает 1.49× против 1.46× для матрицы 300×300, что связано с лучшим соотношением времени вычислений и накладных расходов.

Количество потоков	Время, с	Ускорение	Эффективность, %
1	5.372	1.00	100.0
2	4.131	1.30	65.0
4	3.682	1.46	36.5
8	3.612	1.49	18.6

Таблица 5: Эффективность параллелизации для матрицы 500×500

Результаты

В результате работы была разработана программа для решения систем линейных уравнений методом Гаусса с поддержкой многопоточности, функционирующая в операционной системе Windows.

Ключевые особенности реализации

- **Параллельная обработка:** Реализовано распараллеливание операций исключения в методе Гаусса с использованием потоков Windows
- **Ограничение потоков:** Использование семафоров для контроля максимального количества одновременно работающих потоков
- **Устойчивость алгоритма:** Реализован выбор главного элемента для обеспечения численной устойчивости метода
- **Гибкость ввода:** Поддержка как генерации случайных матриц, так и чтения данных из файлов
- **Точные измерения:** Использование высокоточного таймера QueryPerformanceCounter для замера времени выполнения

Пример работы программы

```
>.\Laba2_OS.exe -n 300 -t 4
Matrix size: 300 x 300,max_threads=4
Program will print active thread counts during elimination.
[step 0/300] active threads limited to 4
[step 10/300] active threads limited to 4
[step 20/300] active threads limited to 4
...
Solved in 0.456123 seconds
x[0..9]= 1.23456 2.34567 3.45678 4.56789 5.67890 6.78901 7.89012 8.90123 9.01234
0.12345

>.\Laba2_OS.exe -n 500 -t 8 -r 123
Matrix size: 500 x 500,max_threads=8
Program will print active thread counts during elimination.
[step 0/500] active threads limited to 8
[step 10/500] active threads limited to 8
...
```

```
Solved in 2.134567 seconds
x[0..9]= 0.987654 1.876543 2.765432 3.654321 4.543210 5.432109 6.321098 7.210987
8.109876 9.098765
```

Выводы

В ходе выполнения лабораторной работы были успешно достигнуты все поставленные цели и решены основные задачи:

1. **Освоены механизмы управления потоками в Windows:** На практике применены системные вызовы Windows API для создания и управления потоками (`CreateThread()`, `WaitForMultipleObjects()`), а также средства синхронизации (`CreateSemaphore()`, `WaitForSingleObject()`)

2. **Реализовано параллельное выполнение вычислений:** Организовано эффективное распараллеливание вычислительно сложных операций метода Гаусса, что позволило значительно ускорить решение систем линейных уравнений большой размерности

3. **Создана система ограничения потоков:** Разработан механизм контроля максимального количества одновременно работающих потоков с использованием семафоров Windows, обеспечивающий оптимальное использование вычислительных ресурсов

4. **Решены практические проблемы многопоточного программирования:**

    - Обеспечена потокобезопасность при работе с общими данными (матрицей и векторами)
    - Реализована обработка ошибок вырожденных матриц в многопоточной среде
    - Организовано корректное освобождение системных ресурсов
    - Обеспечено измерение времени выполнения с высокой точностью

Работа продемонстрировала эффективность применения многопоточности для решения вычислительно сложных математических задач. Наблюдается значительное ускорение вычислений при увеличении количества потоков, особенно для матриц большой размерности. Однако также выявлено, что для малых матриц накладные расходы на создание и синхронизацию потоков могут превышать выгоду от параллелизации.

Полученные навыки работы с потоками и синхронизацией будут полезны при создании других программ, где нужно выполнять несколько задач одновременно - например, в программах для сложных расчётов, обработки графики или научных исследований.

Исходная программа

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>
#include <windows.h>
#include "gauss.h"

double now_seconds(){
    LARGE_INTEGER f, c;
    QueryPerformanceFrequency(&f);
    QueryPerformanceCounter(&c);
    return (double)c.QuadPart / (double)f.QuadPart;
}

static void usage(const char *p){
    printf("Usage: %s [-n size] [-t max_threads] [-r seed] [-f filename]\n", p);
    printf("Example: %s -n 500 -t 8 -r 123\n", p);
    printf("(Project: Laba2_OS  Parallel Gaussian Elimination)\n");
}

int main(int argc, char **argv){
    int n = 200;
    int max_threads = 4;
    int seed = -1;
    const char *fname = NULL;

    for(int i=1;i<argc;i++){
        if(strcmp(argv[i],"-n")==0 && i+1<argc) n = atoi(argv[++i]);
        else if(strcmp(argv[i],"-t")==0 && i+1<argc) max_threads = atoi(argv[++i]);
        else if(strcmp(argv[i],"-r")==0 && i+1<argc) seed = atoi(argv[++i]);
        else if(strcmp(argv[i],"-f")==0 && i+1<argc) fname = argv[++i];
        else { usage(argv[0]); return 1; }
    }

    double *A = (double*)malloc(sizeof(double)*n*n);
    double *b = (double*)malloc(sizeof(double)*n);
    double *x = (double*)calloc(n, sizeof(double));
    if(!A || !b || !x){ fprintf(stderr,"alloc fail\n"); return 2; }

    if(fname){
        FILE *f = fopen(fname,"r");
        if(!f){ perror("fopen"); return 3; }
        int rn;
        if(fscanf(f,"%d",&rn)!=1){ fprintf(stderr,"bad file\n"); return 4; }
        if(rn != n){ fprintf(stderr,"file n != specified n; using file n\n"); n = rn; }
        for(int i=0;i<n*n;i++) fscanf(f,"%lf",&A[i]);
        for(int i=0;i<n;i++) fscanf(f,"%lf",&b[i]);
        fclose(f);
    } else {
        if(seed>=0) srand(seed); else srand((unsigned)time(NULL));
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                double v = ((double)rand()/RAND_MAX)*2.0 - 1.0;
                A[i*n + j] = v;
            }
```

```
        }
        for(int i=0;i<n;i++){
            double s = 0;
            for(int j=0;j<n;j++) s += fabs(A[i*n + j]);
            A[i*n + i] += s + 1.0;
            b[i] = ((double)rand()/RAND_MAX)*10.0;
        }
    }

    printf("Matrix size: %d x %d, max_threads=%d\n", n, n, max_threads);
    printf("Program will print active thread counts during elimination.\n");

    double t0 = now_seconds();
    int rc = gauss_solve(A,b,x,n,max_threads,1);
    double t1 = now_seconds();

    if(rc!=0){
        fprintf(stderr,"gauss failed (maybe singular)\n");
        return 5;
    }

    printf("Solved in %.6f seconds\n", t1-t0);
    int toprint = (n<10)?n:10;
    printf("x[0..%d]= ", toprint-1);
    for(int i=0;i<toprint;i++) printf("%.6g ", x[i]);
    printf("\n");

    free(A); free(b); free(x);
    return 0;
}
```

Листинг 1: `main.c` - Основная программа, обрабатывающая аргументы командной строки и управляющая решением СЛАУ

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <windows.h>
#include "gauss.h"
#include "threads_lim.h"

typedef struct {
    double *A, *b;
    int n, k, i;
} RowTask;

static DWORD WINAPI row_eliminate(void *arg) {
    RowTask *t = (RowTask*)arg;
    int n = t->n, k = t->k, i = t->i;
    double *A = t->A, *b = t->b;

    double factor = A[i*n + k] / A[k*n + k];
    for (int j = k; j < n; j++)
        A[i*n + j] -= factor * A[k*n + j];
    b[i] -= factor * b[k];

    free(t);
    threads_lim_release();
```

```
26      return 0;
27  }
28
29  int gauss_solve(double *A, double *b, double *x, int n, int max_threads, int verbose)
        {
30      threads_lim_init(max_threads);
31
32      for (int k = 0; k < n-1; k++) {
33          int pivot = k;
34          double maxv = fabs(A[k*n + k]);
35          for (int i = k+1; i < n; i++) {
36              if (fabs(A[i*n + k]) > maxv) {
37                  maxv = fabs(A[i*n + k]);
38                  pivot = i;
39              }
40          }
41          if (pivot != k) {
42              for (int j = 0; j < n; j++) {
43                  double tmp = A[k*n + j];
44                  A[k*n + j] = A[pivot*n + j];
45                  A[pivot*n + j] = tmp;
46              }
47              double tmpb = b[k]; b[k] = b[pivot]; b[pivot] = tmpb;
48          }
49
50          if (fabs(A[k*n + k]) < 1e-12) {
51              threads_lim_destroy();
52              return 1;
53          }
54
55          int rows = n - (k+1);
56          HANDLE *th = malloc(sizeof(HANDLE) * rows);
57          if (!th) {
58              threads_lim_destroy();
59              return 1;
60          }
61
62          for (int i = k+1; i < n; i++) {
63              threads_lim_acquire();
64              RowTask *task = malloc(sizeof(RowTask));
65              if (!task) {
66                  threads_lim_release();
67                  free(th);
68                  threads_lim_destroy();
69                  return 1;
70              }
71              task->A = A; task->b = b;
72              task->n = n; task->k = k; task->i = i;
73
74              th[i - (k+1)] = CreateThread(NULL, 0, row_eliminate, task, 0, NULL);
75              if (!th[i - (k+1)]) {
76                  free(task);
77                  threads_lim_release();
78                  for (int j = k+1; j < i; j++) {
79                      WaitForSingleObject(th[j - (k+1)], INFINITE);
80                      CloseHandle(th[j - (k+1)]);
81                  }
82                  free(th);
```

```
                threads_lim_destroy();
                return 1;
            }
        }

        WaitForMultipleObjects(rows, th, TRUE, INFINITE);

        for (int i = 0; i < rows; i++) {
            CloseHandle(th[i]);
        }
        free(th);

        if (verbose && (k % 10 == 0 || k == n-2))
            printf("[step %d/%d] active threads limited to %d\n", k, n, max_threads);
    }

    for (int i = n-1; i >= 0; i--) {
        double sum = b[i];
        for (int j = i+1; j < n; j++)
            sum -= A[i*n + j] * x[j];
        x[i] = sum / A[i*n + i];
    }

    threads_lim_destroy();
    return 0;
}
```

Листинг 2: `gauss.c` - Реализация метода Гаусса с многопоточной обработкой строк

```
#include "threads_lim.h"
#include <windows.h>

static HANDLE sem_handle = NULL;

void threads_lim_init(int max_concurrent){
    sem_handle = CreateSemaphore(NULL, max_concurrent, max_concurrent, NULL);
}

void threads_lim_acquire(void){
    WaitForSingleObject(sem_handle, INFINITE);
}

void threads_lim_release(void){
    ReleaseSemaphore(sem_handle, 1, NULL);
}

void threads_lim_destroy(void){
    if(sem_handle) {
        CloseHandle(sem_handle);
        sem_handle = NULL;
    }
}
```

Листинг 3: `threads_lim.c` - Реализация ограничения количества потоков с помощью семафора Windows

```
#ifndef GAUSS_H
#define GAUSS_H

```

```
#ifdef __cplusplus
extern "C" {
#endif

int gauss_solve(double *A, double *b, double *x, int n, int max_threads, int verbose);

#ifdef __cplusplus
}
#endif

#endif
```

Листинг 4: gauss.h - Заголовочный файл для функции решения СЛАУ методом Гаусса

```
#ifndef THREADS_LIM_H
#define THREADS_LIM_H

#ifdef __cplusplus
extern "C" {
#endif

void threads_lim_init(int max_concurrent);
void threads_lim_acquire(void);
void threads_lim_release(void);
void threads_lim_destroy(void);

#ifdef __cplusplus
}
#endif

#endif
```

Листинг 5: threads_lim.h - Заголовочный файл для управления ограничением потоков

Системные вызовы Windows

```
<?xml version="1.0" encoding="UTF-8"?>
<procmon><processlist><process>
<ProcessIndex>470</ProcessIndex>
<ProcessId>76880</ProcessId>
<ParentProcessId>14916</ParentProcessId>
<ParentProcessIndex>471</ParentProcessIndex>
<AuthenticationId>00000000:049676a0</AuthenticationId>
<CreateTime>134057275798954287</CreateTime>
<FinishTime>0</FinishTime>
<IsVirtualized>0</IsVirtualized>
<Is64bit>1</Is64bit>
<Integrity>Обязательная метка\Высокий обязательный уровень</Integrity>
<Owner>DARIA-BOOK\Daria</Owner>
<ProcessName>Procmon64.exe</ProcessName>
<ImagePath>C:\Users\Daria\Downloads\ProcessMonitor\Procmon64.exe</ImagePath>
<CommandLine>&quot;C:\Users\Daria\Downloads\ProcessMonitor\Procmon64.exe&quot;
</CommandLine>
<CompanyName>Sysinternals -www.sysinternals.com</CompanyName>
<Version>4.01</Version>
```

<Description>Process Monitor</Description>
<modulelist>
<module>
<Timestamp>134057285970173388</Timestamp>
<BaseAddress>0x5ff60000</BaseAddress>
<Size>155648</Size>
<Path>C:\Program Files\Bonjour\mdnsNSP.dll</Path>
<Version>3,1,0,1</Version>
<Company>Apple Inc.</Company>
<Description>Bonjour Namespace Provider</Description>
</module>
<module>
<Timestamp>134057285970173388</Timestamp>
<BaseAddress>0x7ff677f80000</BaseAddress>
<Size>2232320</Size>
<Path>C:\Users\Daria\Downloads\ProcessMonitor\Procmon64.exe</Path>
<Version>4.01</Version>
<Company>Sysinternals -www.sysinternals.com</Company>
<Description>Process Monitor</Description>
</module>
... (и т.д (XML файл))
</modulelist>
</process>
<process>
<ProcessIndex>471</ProcessIndex>
<ProcessId>14916</ProcessId>
<ParentProcessId>66264</ParentProcessId>
<ParentProcessIndex>597</ParentProcessIndex>
<AuthenticationId>00000000:049676be</AuthenticationId>
<CreateTime>134056832338204969</CreateTime>
<FinishTime>0</FinishTime>
<IsVirtualized>0</IsVirtualized>
<Is64bit>1</Is64bit>
<Integrity>Обязательная метка\Средний обязательный уровень</Integrity>
<Owner>DARIA-BOOK\Daria</Owner>
<ProcessName>Explorer.EXE</ProcessName>
<ImagePath>C:\Windows\Explorer.EXE</ImagePath>
<CommandLine>C:\Windows\Explorer.EXE</CommandLine>
<CompanyName>Microsoft Corporation</CompanyName>
<Version>10.0.19041.4522 (WinBuild.160101.0800)</Version>
<Description>Проводник</Description>
<modulelist>
... (все модули Explorer.EXE)
</modulelist>
</process>
<process>
<ProcessIndex>472</ProcessIndex>
<ProcessId>4</ProcessId>

<ParentProcessId>0</ParentProcessId>
<ParentProcessIndex>473</ParentProcessIndex>
<AuthenticationId>00000000:000003e7</AuthenticationId>
<CreateTime>134052960184910276</CreateTime>
<FinishTime>0</FinishTime>
<IsVirtualized>0</IsVirtualized>
<Is64bit>1</Is64bit>
<Integrity>Обязательная метка\Обязательный уровень системы</Integrity>
<Owner>NT AUTHORITY\СИСТЕМА</Owner>
<ProcessName>System</ProcessName>
<ImagePath>System</ImagePath>
<CommandLine></CommandLine>
<CompanyName></CompanyName>
<Version></Version>
<Description></Description>
<modulelist>
... (все модули System)
</modulelist>
</process>
</processlist>
</procmon>