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-210Б-23

Студент: Коростин Н.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 21.12..24

Москва, 2024

Постановка задачи

Вариант 9.

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.

Рассчитать детерминант матрицы (используя определение детерминанта).

Общий метод и алгоритм решения

Использованные системные вызовы:

- read – чтение данных из файла (потока)
- write – запись данных в файл (поток)
- clock – количество “тиков” процессора с момента запуска программы
- sem_init – инициализирует семафор, задает начальное значение
- sem_wait – декрементирует значение семафора; если оно 0, то вызвавший функцию поток блокируется
- sem_post – инкрементирует значение семафора и освобождает заблокированные потоки, если такие есть
- pthread_create – создание нового потока
- pthread_join – ожидание завершения потока

Для расчета детерминанта по определению нужно разложить его, например, по первой строке. Тогда детерминант всей матрицы будет равен каждому числу строки, умноженному на (-1)^(colomn % 2), умноженному на определитель соответствующего минора. При разложении матрицы по первой строке создаются потоки, в каждом из которых выполняется функция calculate_minor. Определитель миноров вычисляется с помощью рекурсивной функции recursive_determinant.

Распараллеливать алгоритм вычисления будем по первой строке матрицы. Соответственно имеет смысл задавать количество потоков не большее, чем сам размер матрицы. Число потоков и размер матрицы задаются аргументами командной строки.

Код программы

main.c

```
#include "../include/io.h"

#include "../include/errors.h"

#include "../include/utils_lab2.h"

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

    if (argc != 3) {

        log_errors(WRONG_NUMBER_OF_PARAMS);
```

```
        return 1;
    }
    size = atoi(argv[1]);
    int maxThreads = atoi(argv[2]);
    if (size <= 0 || size > MAX_SIZE) {
        write_to_file(STDERR_FILENO, "Invalid matrix size.\n");
        return 1;
    }
    sem_init(&semaphore, 0, maxThreads);
    read_matrix(matrix, size);
    clock_t start_time = clock();
    int determinant = calculate_determinant();
    clock_t end_time = clock();
    double time_taken = ((double)(end_time - start_time)) /
CLOCKS_PER_SEC;
    print_to_stdout("Determinant: ");
    char detString[20];
    int_to_string(determinant, detString, sizeof(detString));
    print_to_stdout(detString);
    print_to_stdout(".\n");
    char timeString[50];
    snprintf(timeString, sizeof(timeString), "Time taken: %.6f
seconds.\n", time_taken);
    print_to_stdout(timeString);
    sem_destroy(&semaphore);
    pthread_mutex_destroy(&mutex);
    return 0;
}
```

util_lab2.c

```
#include "../include/utils_lab2.h"
#include <time.h>
```

```
#include <unistd.h>

int matrix[MAX_SIZE][MAX_SIZE];
int size;
sem_t semaphore;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

char* int_to_string(int number, char* str, int bufferSize) {
    int isNegative = 0;
    if (number < 0) {
        isNegative = 1;
        number = -number;
    }

    int index = 0;

    do {
        str[index++] = number % 10 + '0';
        number /= 10;
    } while (number > 0 && index < bufferSize - 1);

    if (isNegative) {
        str[index++] = '-';
    }

    str[index] = '\0';

    int start = 0;
    int end = index - 1;
    while (start < end) {
        char temp = str[start];
        str[start] = str[end];
        str[end] = temp;
        start++;
        end--;
    }
    return str;
}

void read_matrix(int matrix[MAX_SIZE][MAX_SIZE], int size) {
    print_to_stdout("Enter the matrix elements (row by row):\n");
    for (int i = 0; i < size; i++) {
        char buffer[BUFF_SIZE];
        ssize_t bytesRead = read_from_stdin(buffer, sizeof(buffer));
        if (bytesRead <= 0) {
            write_to_file(STDERR_FILENO, "Error reading matrix row.\n");
            exit(EXIT_FAILURE);
        }
```

```
        char* token = strtok(buffer, " \t\n");
        for (int j = 0; j < size; j++) {
            if (token == NULL) {
                write_to_file(STDERR_FILENO, "Not enough elements in the row.\n");
                exit(EXIT_FAILURE);
            }
            matrix[i][j] = atoi(token);
            token = strtok(NULL, " \t\n");
        }

        if (token != NULL) {
            write_to_file(STDERR_FILENO, "Too many elems in a row.\n");
            exit(EXIT_FAILURE);
        }
    }
}

int recursive_determinant(int** tempMatrix, int n) {

    if (tempMatrix == NULL || n <= 0) {
        fprintf(stderr, "Invalid matrix or size in recursive_determinant.\n");
        return 0;
    }

    if (n == 1) {
        return tempMatrix[0][0];
    }
    if (n == 2) {
        return tempMatrix[0][0] * tempMatrix[1][1] - tempMatrix[0][1] *
tempMatrix[1][0];
    }

    //int minorMatrix[MAX_SIZE][MAX_SIZE];

    int **minorMatrix = (int**)malloc((n - 1) * sizeof(int *));
    if (minorMatrix == NULL) {
        log_errors(MEM_ALLOC_FAILED);
        return 0;
    }
    for (int i = 0; i < n - 1; i++) {
        minorMatrix[i] = (int*)malloc((n - 1) * sizeof(int));
        if (minorMatrix[i] == NULL) {
            log_errors(MEM_ALLOC_FAILED);
            for (int j = 0; j < i; j++) {
                free(minorMatrix[j]);
            }
            free(minorMatrix);
            return 0;
        }
```

```
    }

    int determinant = 0;
    for (int j = 0; j < n; ++j) {
        for (int i = 1, x = 0; i < n; ++i, ++x) {
            for (int k = 0, y = 0; k < n; ++k) {
                if (k == j) continue;
                minorMatrix[x][y++] = tempMatrix[i][k];
            }
        }

        int minorDet = recursive_determinant(minorMatrix, n - 1);
        //printf("minor of size %d = %d\n", n, minorDet);
        determinant += ((j % 2 == 0) ? 1 : -1) * tempMatrix[0][j] * minorDet;
    }

    for (int i = 0; i < n - 1; i++) {
        free(minorMatrix[i]);
    }
    free(minorMatrix);
    return determinant;
}

void *calculate_minor(void *arg) {
    MinorArgs *args = (MinorArgs *)arg;

    //int tempMatrix[MAX_SIZE][MAX_SIZE];
    int **tempMatrix = (int**)malloc((MAX_SIZE - 1) * sizeof(int *));
    if (tempMatrix == NULL) {
        log_errors(MEM_ALLOC_FAILED);
        return 0;
    }
    for (int i = 0; i < MAX_SIZE - 1; i++) {
        tempMatrix[i] = (int*)malloc((MAX_SIZE - 1) * sizeof(int));
        if (tempMatrix[i] == NULL) {
            log_errors(MEM_ALLOC_FAILED);
            for (int j = 0; j < i; j++) {
                free(tempMatrix[j]);
            }
            free(tempMatrix);
            return 0;
        }
    }
    int minorSize = size - 1;

    for (int i = 0, x = 0; i < size; ++i) {
        if (i == args->row) continue;
        for (int j = 0, y = 0; j < size; ++j) {
            if (j == args->col) continue;
```

```
            tempMatrix[x][y++] = matrix[i][j];
        }
        ++x;
    }

    int determinant = recursive_determinant(tempMatrix, minorSize);

    pthread_mutex_lock(&mutex);
    args->result = determinant * ((args->col % 2 == 0) ? 1 : -1) *
matrix[args->row][args->col];
    pthread_mutex_unlock(&mutex);

    sem_post(&semaphore);
    return NULL;
}

int calculate_determinant() {
    pthread_t threads[MAX_SIZE];
    MinorArgs args[MAX_SIZE];
    int determinant = 0;

    for (int j = 0; j < size; ++j) {
        sem_wait(&semaphore);
        args[j].row = 0;
        args[j].col = j;

        if (pthread_create(&threads[j], NULL, calculate_minor, &args[j]) != 0) {
            log_errors(THREAD_CREATION_FAILED);
            exit(EXIT_FAILURE);
        }
    }
    for (int j = 0; j < size; ++j) {
        pthread_join(threads[j], NULL);
        determinant += args[j].result;
    }
    return determinant;
}
```

errors.c

```
#include "../include/errors.h"
void log_errors(Error err) {
    switch(err) {
        case THREAD_CREATION_FAILED:
            write_to_file(STDERR_FILENO, "THREAD_CREATION_FAILED\n");
            break;
        case MEM_ALLOC_FAILED:
            write_to_file(STDERR_FILENO, "MEM_ALLOC_FAILED\n");
```

```
            break;
        case WRONG_NUMBER_OF_PARAMS:
            write_to_file(STDERR_FILENO, "WRONG_NUMBER_OF_PARAMS\n");
            break;
        default:
            write_to_file(STDERR_FILENO, "UNKNOWN_ERROR\n");
    }
}
```

io.cpp

```
#include "../include/io.h"

ssize_t read_input(int input_file, char *message, size_t message_size) {
    char buffer[BUFF_SIZE];
    ssize_t bytesRead;
    size_t totalBytes = 0;

    if (message == NULL || message_size == 0) {
        return -1;
    }

    while ((bytesRead = read(input_file, buffer, BUFF_SIZE)) > 0) {
        if (totalBytes + bytesRead > message_size - 1) {
            return -1;
        }

        memcpy(message + totalBytes, buffer, bytesRead);
        totalBytes += bytesRead;

        if (message[totalBytes - 1] == '\n') {
            message[totalBytes - 1] = '\0';
            break;
        }
    }

    if (bytesRead == -1) {
        return -1;
    }

    message[totalBytes] = '\0';
    return (ssize_t)totalBytes;
}

ssize_t read_from_stdin(char *message, size_t message_size) {
    return read_input(STDIN_FILENO, message, message_size);
}

ssize_t write_to_file(int file_output, const char *message) {
```

```
    size_t len = strlen(message);
    ssize_t bytesWritten = write(file_output, message, len);
    return bytesWritten;
}

void print_to_stdout(const char *message) {
    write_to_file(STDOUT_FILENO, message);
}
```

Протокол работы программы

```
execve("./main_exec", ["./main_exec", "3", "3"], 0x7ffdd9b51360 /* 56 vars */) = 0
brk(NULL)                               = 0x60abcfb5a000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x785a67e2b000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=70647, ...}) = 0
mmap(NULL, 70647, PROT_READ, MAP_PRIVATE, 3, 0) = 0x785a67e19000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) =
832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64)
= 784
fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64)
= 784
mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x785a67c00000
mmap(0x785a67c28000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x785a67c28000
mmap(0x785a67db0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x785a67db0000
mmap(0x785a67dff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1fe000) = 0x785a67dff000
mmap(0x785a67e05000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x785a67e05000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x785a67e16000
arch_prctl(ARCH_SET_FS, 0x785a67e16740) = 0
set_tid_address(0x785a67e16a10)         = 9721
set_robust_list(0x785a67e16a20, 24)     = 0
rseq(0x785a67e17060, 0x20, 0, 0x53053053) = 0
mprotect(0x785a67dff000, 16384, PROT_READ) = 0
mprotect(0x60abce5c2000, 4096, PROT_READ) = 0
mprotect(0x785a67e63000, 8192, PROT_READ) = 0
```

```
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x785a67e19000, 70647)           = 0
write(1, "Enter the matrix elements (row b"..., 40Enter the matrix elements (row by
row):
) = 40
read(0, 1 2 3
"1 2 3\n", 1024)                = 6
read(0, 2 3 4
"2 3 4\n", 1024)                = 6
read(0, 3 4 19
"3 4 19\n", 1024)               = 7
rt_sigaction(SIGRT_1, {sa_handler=0x785a67c99520, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x785a67c45320},
NULL, 8) = 0
rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x785a67200000
mprotect(0x785a67201000, 8388608, PROT_READ|PROT_WRITE) = 0
getrandom("\xa6\x18\xb5\x24\xac\x2b\x95\x9b", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x60abcfb5a000
brk(0x60abcfb7b000)                     = 0x60abcfb7b000
rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x785a67a00990,
parent_tid=0x785a67a00990, exit_signal=0, stack=0x785a67200000, stack_size=0x7fff80,
tls=0x785a67a006c0} => {parent_tid=[9725]}, 88) = 9725
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x785a5e800000
mprotect(0x785a5e801000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x785a5f000990,
parent_tid=0x785a5f000990, exit_signal=0, stack=0x785a5e800000, stack_size=0x7fff80,
tls=0x785a5f0006c0} => {parent_tid=[9726]}, 88) = 9726
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x785a66800000
mprotect(0x785a66801000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x785a67000990,
parent_tid=0x785a67000990, exit_signal=0, stack=0x785a66800000, stack_size=0x7fff80,
tls=0x785a670006c0} => {parent_tid=[9727]}, 88) = 9727
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
futex(0x785a67000990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 9727, NULL,
FUTEX_BITSET_MATCH_ANY) = -1 EAGAIN (Resource temporarily unavailable)
write(1, "Determinant: ", 13Determinant: )          = 13
write(1, "-14", 3-14)                       = 3
```

```
write(1, ".\n", 2.
)                       = 2
write(1, "Time taken: 0.002301 seconds.\n", 30Time taken: 0.002301 seconds.
) = 30
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

Распараллеливание алгоритма расчета детерминанта оказалось довольно очевидным, исходя из определения детерминанта матрицы, и легко реализуемым. Сложность состояла в том, чтобы дебажить программу в нескольких потоках. В результате сначала дебаг производился при работе в один поток, а после уже производился тестинг на нескольких потоках. Для генерации больших матриц использовался сайт https://planetcalc.ru.

Число потоков	Время исполнения(с)	Ускорение	Эффективность
1	10.496	1	1
2	5.364	1.957	0.978
4	2.791	3.761	0.940
8	1.985	5.288	0.661