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(c)** | **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 |

