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-209Б-24

Студент: Артонкин В.Н.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 10.11.25

Москва, 2025

Постановка задачи

Вариант 5.

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.

Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

В отчете привести исследование зависимости ускорения и эффективности алгоритма от входных данных и количества потоков. Получившиеся результаты необходимо объяснить.

Отсортировать массив целых чисел при помощи четно-нечетной сортировки Бетчера

Общий метод и алгоритм решения

Использованные системные вызовы:

- **int pthread_create**(**pthread_t *restrict** *thread*, **const pthread_attr_t *restrict** *attr*, **void *(***start_routine*)(void*), **void *restrict** *arg*) - создает поток, запускает функцию с переданными данными.
- **void pthread_exit(void *** *retval***) -** завершает поток и возвращает *retval*
- **int pthread_barrier_init**(pthread_barrier_t *restrict barrier, const pthread_barrierattr_t *restrict attr, unsigned count) - инициализирует барьер с указанными атрибутами
- **int pthread_barrier_wait**(pthread_barrier_t *barrier) - синхронизирует вовлеченные потоки на потоке barrier
- **int pthread_join**(thread_t tid, void **status) - блокирует ожидающий поток пока не завершится указанный
- **int pthread_barrier_destroy**(pthread_barrier_t *barrier) **-** уничтожает объявленный барьер

Сначала были созданы все необходимые переменные и структуры данных, с учетом тех, которые потом использовались для работы с потоками. Также были считаны: длина массива, количество потоков и способ задания массива. После происходит создание нужного клоичества потоков и передача в них функции, отвечающей за сортировку массива. Перед каждой сменой длины шагов происходит синхронизация работы потоков. По завершении сортировки все потоки завершаются, измеряется время работы программы.

Код программы

main.c

```
#include <stdio.h>

#include <stdlib.h>
```

```
#include <pthread.h>
#include <sys/time.h>
#include <time.h>

typedef struct {
    int *arr;
    int n;
    int num_threads;
    int thread_id;
    pthread_barrier_t *barrier;
} ThreadData;

// Сравнение и обмен
void compare_and_swap(int *arr, int i, int j, int ascending) {
    if (ascending == (arr[i] > arr[j])) {
        int tmp = arr[i];
        arr[i] = arr[j];
        arr[j] = tmp;
    }
}

// Основная функция, выполняемая каждым потоком
void *bitonic_thread(void *arg) {
    ThreadData *data = (ThreadData *)arg;
    int *arr = data->arr;
    int n = data->n;
    int tid = data->thread_id;
    int num_threads = data->num_threads;

    for (int k = 2; k <= n; k <<= 1) {
```

```
        for (int j = k >> 1; j > 0; j >>= 1) {

            // Каждый поток обрабатывает свои индексы
            for (int i = tid; i < n; i += num_threads) {
                int ixj = i ^ j; // XOR вычисляет пару для сравнения
                // swap происходит только в том случае, если ixj > i
                if (ixj > i) {
                    int ascending = ((i & k) == 0); // направление сортировки
                    compare_and_swap(arr, i, ixj, ascending);
                }
            }

            // Синхронизация между потоками
            pthread_barrier_wait(data->barrier);
        }
    }

    pthread_exit(NULL);
}

int is_power_of_two(int n) {
    return (n > 0) && ((n & (n - 1)) == 0);
}

void print_array(int *arr, int n) {
    for (int i = 0; i < n; i++) printf("%d ", arr[i]);
    printf("\n");
}

int main() {
```

```
int n, num_threads;
printf("Введите длину массива (степень двойки): ");
scanf("%d", &n);

if (!is_power_of_two(n)) {
    printf("Ошибка: длина массива должна быть степенью двойки!\n");
    return 1;
}

printf("Введите количество потоков (1–8): ");
scanf("%d", &num_threads);
if (num_threads < 1 || num_threads > 8) {
    printf("Ошибка: число потоков должно быть от 1 до 8.\n");
    return 1;
}

int *arr = malloc(n * sizeof(int));
if (!arr) {
    perror("malloc");
    return 1;
}

int mode;
printf("Режим ввода: 1 — вручную, 2 — случайно: ");
scanf("%d", &mode);

if (mode == 1) {
    printf("Введите %d элементов массива:\n", n);
    for (int i = 0; i < n; i++) scanf("%d", &arr[i]);
} else {
```

```
    srand(time(NULL));
    for (int i = 0; i < n; i++) arr[i] = rand() % 1000;
    printf("Сгенерированный массив:\n");
    print_array(arr, n);
}

pthread_t threads[num_threads];
ThreadData data[num_threads];
pthread_barrier_t barrier;
pthread_barrier_init(&barrier, NULL, num_threads);

struct timeval start, end;
gettimeofday(&start, NULL);

// Создание потоков
for (int t = 0; t < num_threads; t++) {
    data[t].arr = arr;
    data[t].n = n;
    data[t].num_threads = num_threads;
    data[t].thread_id = t;
    data[t].barrier = &barrier;
    pthread_create(&threads[t], NULL, bitonic_thread, &data[t]);
}

// Ожидание завершения
for (int t = 0; t < num_threads; t++) pthread_join(threads[t], NULL);

gettimeofday(&end, NULL);
pthread_barrier_destroy(&barrier);
```

```
    double elapsed = (end.tv_sec - start.tv_sec) * 1000.0 +
                     (end.tv_usec - start.tv_usec) / 1000.0;

    printf("\nОтсортированный массив:\n");
    print_array(arr, n);
    printf("\nВремя сортировки: %.3f мс (%d потоков)\n", elapsed, num_threads);

    free(arr);
    return 0;
}
```

Протокол работы программы

```
$ gcc main.c -o sort -lpthread -O2
vscode ➜ /workspaces/OS/lab2/src (main) $ strace -o trace.log ./sort
Введите длину массива (степень двойки): 16
Введите количество потоков (1–8): 4
Режим ввода: 1 – вручную, 2 – случайно: 2
Сгенерированный массив:
402 867 249 118 568 674 188 678 78 208 968 904 96 597 901 993

Отсортированный массив:
78 96 118 188 208 249 402 568 597 674 678 867 901 904 968 993

Время сортировки: 16.072 мс (4 потоков)
vscode ➜ /workspaces/OS/lab2/src (main) $

execve("./sort", ["./sort"], 0x7ffc45ddcd50 /* 32 vars */) = 0
brk(NULL)                               = 0x5570a88a4000
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=16648, ...}) = 0
mmap(NULL, 16648, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f926e217000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0 l\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=149520, ...}) = 0
```

```
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f926e215000

mmap(NULL, 136304, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f926e1f3000

mmap(0x7f926e1f9000, 65536, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x6000) = 0x7f926e1f9000

mmap(0x7f926e209000, 24576, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x16000)
= 0x7f926e209000

mmap(0x7f926e20f000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1b000) = 0x7f926e20f000

mmap(0x7f926e211000, 13424, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x7f926e211000

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\260>\2\0\0\0\0\0"..., 832) =
832

fstat(3, {st_mode=S_IFREG|0755, st_size=1901536, ...}) = 0

mmap(NULL, 1914496, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f926e01f000

mmap(0x7f926e041000, 1413120, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x22000) = 0x7f926e041000

mmap(0x7f926e19a000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x17b000) = 0x7f926e19a000

mmap(0x7f926e1e9000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1c9000) = 0x7f926e1e9000

mmap(0x7f926e1ef000, 13952, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x7f926e1ef000

close(3)                                = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f926e01c000

arch_prctl(ARCH_SET_FS, 0x7f926e01c740) = 0

mprotect(0x7f926e1e9000, 16384, PROT_READ) = 0

mprotect(0x7f926e20f000, 4096, PROT_READ) = 0

mprotect(0x5570a2221000, 4096, PROT_READ) = 0

mprotect(0x7f926e246000, 4096, PROT_READ) = 0

munmap(0x7f926e217000, 16648)           = 0

set_tid_address(0x7f926e01ca10)         = 14882

set_robust_list(0x7f926e01ca20, 24)     = 0

rt_sigaction(SIGRTMIN, {sa_handler=0x7f926e1f9690, sa_mask=[],
sa_flags=SA_RESTORER|SA_SIGINFO, sa_restorer=0x7f926e206140}, NULL, 8) = 0

rt_sigaction(SIGRT_1, {sa_handler=0x7f926e1f9730, sa_mask=[],
sa_flags=SA_RESTORER|SA_RESTART|SA_SIGINFO, sa_restorer=0x7f926e206140}, NULL, 8) = 0

rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
```

```
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0

brk(NULL)                               = 0x5570a88a4000

brk(0x5570a88c5000)                     = 0x5570a88c5000

fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265
\320\264\320\273\320\270\320\275\321\203 \320\274\320\260\321\201"..., 72) = 72

read(0, "16\n", 1024)                   = 3

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265
\320\272\320\276\320\273\320\270\321\207\320\265\321\201\321\202\320"..., 60) = 60

read(0, "4\n", 1024)                    = 2

write(1, "\320\240\320\265\320\266\320\270\320\274
\320\262\320\262\320\276\320\264\320\260: 1 \342\200\224 \320\262\321"..., 69) = 69

read(0, "2\n", 1024)                    = 2

write(1,
"\320\241\320\263\320\265\320\275\320\265\321\200\320\270\321\200\320\276\320\262\320\2
60\320\275\320\275\321\213\320\271 \320"..., 45) = 45

write(1, "402 867 249 118 568 674 188 678 "..., 63) = 63

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f926d81b000

mprotect(0x7f926d81c000, 8388608, PROT_READ|PROT_WRITE) = 0

clone(child_stack=0x7f926e01afb0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETT
LS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tid=[14957], tls=0x7f926e01b700,
child_tidptr=0x7f926e01b9d0) = 14957

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f926d01a000

mprotect(0x7f926d01b000, 8388608, PROT_READ|PROT_WRITE) = 0

clone(child_stack=0x7f926d819fb0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETT
LS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tid=[14958], tls=0x7f926d81a700,
child_tidptr=0x7f926d81a9d0) = 14958

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f926c819000

mprotect(0x7f926c81a000, 8388608, PROT_READ|PROT_WRITE) = 0

clone(child_stack=0x7f926d018fb0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETT
LS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tid=[14959], tls=0x7f926d019700,
child_tidptr=0x7f926d0199d0) = 14959

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f926c018000

mprotect(0x7f926c019000, 8388608, PROT_READ|PROT_WRITE) = 0

clone(child_stack=0x7f926c817fb0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETT
LS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tid=[14960], tls=0x7f926c818700,
child_tidptr=0x7f926c8189d0) = 14960
```

```
futex(0x7f926e01b9d0, FUTEX_WAIT, 14957, NULL) = 0

futex(0x7f926d0199d0, FUTEX_WAIT, 14959, NULL) = 0

write(1, "\n", 1)                       = 1

write(1,
"\320\236\321\202\321\201\320\276\321\200\321\202\320\270\321\200\320\276\320\262\320\2
60\320\275\320\275\321\213\320\271 \320"..., 45) = 45

write(1, "78 96 118 188 208 249 402 568 59"..., 63) = 63

write(1, "\n", 1)                       = 1

write(1, "\320\222\321\200\320\265\320\274\321\217
\321\201\320\276\321\200\321\202\320\270\321\200\320\276\320\262\320\272\320\270:"...,
64) = 64

lseek(0, -1, SEEK_CUR)                  = -1 ESPIPE (Illegal seek)

exit_group(0)                           = ?

+++ exited with 0 +++
```

Число потоков	Время исполнения (мс)	Ускорение	Эффективность
1	453.137	1	1
2	354.387	1.27	0,635
4	226.778	1.99	0,4975
8	183.844	2.46	0,3075

Ускорение и, как следствие, эффективность уменьшаются с увеличением количества потоков. Имеет место эффект локтя. Также стоит обратить внимание на то, что данные измерения были проведены при сортировке массива длиной 2^20 элементов. При работе с массивами, чьи длины в несколько порядков ниже, многопоточный код выполняется дольше из-за системных вызовов.

Вывод

Я научился создавать многопоточный код. Понял, как взаимодействуют между собой процессы, как обеспечить их корректную взаимосвязанную работу. Было неочевидно, как реализовать сортировку итеративно, чтобы можно было писать многопоточный код. Многопоточный код бывает избыточен при выполнении недолгих операций. Также эффективность каждого потока снижается при их увеличении.