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-24

Студент: Василянская А.Н.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 01.12.25

Москва, 2024

Постановка задачи

Вариант 1.

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

1. Отсортировать массив целых чисел при помощи битонической сортировки

Общий метод и алгоритм решения

Битоническая сортировка распараллеливается рекурсивно: каждый поток обрабатывает свою часть массива, а глубина рекурсии ограничена заданным числом потоков. Алгоритм создаёт потоки через pthread, пока не достигнет предела, после чего работает последовательно. Ускорение измеряется как отношение времени последовательной и параллельной версий. Эффективность падает с ростом числа потоков из-за накладных расходов на синхронизацию.

Использованные системные вызовы:

- ssize_t write(int fd, void *buf, size_t count)– записывает файлы из буфера в файловый дескриптор.
- void exit (int status) – завершения выполнения процесса и возвращение статуса.
- int clock_gettime(clockid_t clock_id, struct timespec *tp) – получает текущее время выбранных часов.
- int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg) – создаёт новый поток выполнения.
- int pthread_join(pthread_t thread, void **retval) – ожидает завершения указанного потока.

Анализ метрик ускорения и эффективности

Число потоков	Время исполнения (мс)	Ускорение	Эффективность
1	7.058	1	1
2	4.961	1.42	0.71
4	4.215	1.67	0.42
8	6.669	1.06	0.13
12	10.077	0.7	0.06
16	10.996	0.64	0.04
128	64.615	0.11	0.0009
1024	242.17	0.03	0.00003

Ускорение: $S = T_1 / T_n$. T_1 — время выполнения 1 потока, T_n — время выполнения на потоках n = max_threads. Ускорение показывает, во сколько раз параллельная программа выполняется быстрее последовательной версии.

Эффективность: X = S / max_threads. S — ускорение, max_threads — число потоков. Эффективность показывает, насколько эффективно используются ресурсы.

Код программы

main.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <time.h>
#include <math.h>



typedef struct {
    int *array;
    int low;
    int cnt;
    int dir;
    int depth_limit;
} ThreadArgs;

static void swap_elems(int *x, int *y) {
    int tmp = *x;
    *x = *y;
    *y = tmp;
}

static void bitonic_merge(int *data, int low, int cnt, int dir) {
    if (cnt > 1) {
        int k = cnt / 2;
        for (int i = low; i < low + k; i++) {
            if (dir == (data[i] > data[i + k])) {
                swap_elems(&data[i], &data[i + k]);
            }
        }
```

```
        bitonic_merge(data, low, k, dir);
        bitonic_merge(data, low + k, k, dir);
    }
}

static void bitonic_sort_sequential(int *data, int low, int cnt, int dir) {
    if (cnt > 1) {
        int k = cnt / 2;
        bitonic_sort_sequential(data, low, k, 1);
        bitonic_sort_sequential(data, low + k, k, 0);
        bitonic_merge(data, low, cnt, dir);
    }
}

static void *work(void *_args) {
    ThreadArgs *args = (ThreadArgs*)_args;
    int low = args->low;
    int cnt = args->cnt;
    int dir = args->dir;
    int depth = args->depth_limit;

    if (cnt <= 1) return NULL;

    int k = cnt / 2;

    if (depth > 0) {
        pthread_t thread1, thread2;
        ThreadArgs args1, args2;

        args1.array = args->array;
        args1.low = low;
        args1.cnt = k;
        args1.dir = 1;
        args1.depth_limit = depth - 1;
```

```
        args2.array = args->array;
        args2.low = low + k;
        args2.cnt = k;
        args2.dir = 0;
        args2.depth_limit = depth - 1;

        pthread_create(&thread1, NULL, work, &args1);
        pthread_create(&thread2, NULL, work, &args2);

        pthread_join(thread1, NULL);
        pthread_join(thread2, NULL);

    } else {
        bitonic_sort_sequential(args->array, low, k, 1);
        bitonic_sort_sequential(args->array, low + k, k, 0);
    }

    bitonic_merge(args->array, low, cnt, dir);

    return NULL;
}

static double get_time_diff(struct timespec start, struct timespec end) {
    return (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1e9;
}

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

int main(int argc, char** argv) {
    if (argc != 3) {
        const char msg[] = "./program array_size max_threads\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
```

```
}

int n = atoi(argv[1]);
int max_threads = atoi(argv[2]);

if (n <= 0 || max_threads <= 0) {
    const char msg[] = "error: invalid input (must be > 0)\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}

if (!is_power_of_two(n)) {
    const char msg[] = "error: array size must be a power of 2 \n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}

int *array = malloc(n * sizeof(int));
int *seq_array = malloc(n * sizeof(int));

if (!array || !seq_array) {
    const char msg[] = "error: memory allocation failed\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    exit(EXIT_FAILURE);
}

srand(time(NULL));
for (int i = 0; i < n; ++i) {
    array[i] = rand() % 10000;
    seq_array[i] = array[i];
}

char buf[256];

struct timespec start_paral, end_paral;
```

```
    if (clock_gettime(CLOCK_MONOTONIC, &start_paral) != 0) {
        const char msg[] = "error: cant get start time\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    int depth_limit = (int)(log2(max_threads));

    pthread_t root_thread;
    ThreadArgs args;
    args.array = array;
    args.low = 0;
    args.cnt = n;
    args.dir = 1;
    args.depth_limit = depth_limit;

    pthread_create(&root_thread, NULL, work, &args);
    pthread_join(root_thread, NULL);

    if (clock_gettime(CLOCK_MONOTONIC, &end_paral) != 0) {
        const char msg[] = "error: cant get end time\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    double total_paral_time = get_time_diff(start_paral, end_paral);

    if (snprintf(buf, sizeof(buf), "\nparallel\nsort completed in %.6f sec\nmax threads param: %d (depth: %d), size: %d\n",
                 total_paral_time, max_threads, depth_limit, n) >= 0) {
        write(STDOUT_FILENO, buf, strlen(buf));
    }

    free(array);
    return 0;
```

}

Протокол работы программы

Тестирование:

```
aressui@DESKTOP-O6E9TGD:/mnt/c/Users/Alyona/2kurs/os/lab2$ gcc src/main.c -o sol -lm
aressui@DESKTOP-O6E9TGD:/mnt/c/Users/Alyona/2kurs/os/lab2$ ./sol 8192 1

parallel
sort completed in 0.007058 sec
max threads param: 1 (depth: 0), size: 8192
aressui@DESKTOP-O6E9TGD:/mnt/c/Users/Alyona/2kurs/os/lab2$ ./sol 8192 2

parallel
sort completed in 0.004961 sec
max threads param: 2 (depth: 1), size: 8192
aressui@DESKTOP-O6E9TGD:/mnt/c/Users/Alyona/2kurs/os/lab2$ ./sol 8192 4

parallel
sort completed in 0.004215 sec
max threads param: 4 (depth: 2), size: 8192
aressui@DESKTOP-O6E9TGD:/mnt/c/Users/Alyona/2kurs/os/lab2$ ./sol 8192 8

parallel
sort completed in 0.006669 sec
max threads param: 8 (depth: 3), size: 8192
aressui@DESKTOP-O6E9TGD:/mnt/c/Users/Alyona/2kurs/os/lab2$ ./sol 8192 12

parallel
sort completed in 0.010077 sec
max threads param: 12 (depth: 3), size: 8192
aressui@DESKTOP-O6E9TGD:/mnt/c/Users/Alyona/2kurs/os/lab2$ ./sol 8192 16

parallel
sort completed in 0.010996 sec
max threads param: 16 (depth: 4), size: 8192
aressui@DESKTOP-O6E9TGD:/mnt/c/Users/Alyona/2kurs/os/lab2$ ./sol 8192 128

parallel
sort completed in 0.064615 sec
max threads param: 128 (depth: 7), size: 8192
aressui@DESKTOP-O6E9TGD:/mnt/c/Users/Alyona/2kurs/os/lab2$ ./sol 8192 1024

parallel
sort completed in 0.242170 sec
max threads param: 1024 (depth: 10), size: 8192
```

Strace:

aressui@DESKTOP-O6E9TGD:/mnt/c/Users/Alyona/2kurs/os/lab2$ strace ./sol 8192 8

execve("./sol", ["./sol", "8192", "8"], 0x7fff7b5aacd0 /* 27 vars */) = 0

brk(NULL) = 0x654728ebd000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ebb079a3000

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=20243, ...}) = 0

mmap(NULL, 20243, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ebb0799e000

close(3)   = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libm.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\0\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st_mode=S_IFREG|0644, st_size=952616, ...}) = 0

mmap(NULL, 950296, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ebb078b5000

mmap(0x7ebb078c5000, 520192, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x10000) = 0x7ebb078c5000

mmap(0x7ebb07944000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x8f000) = 0x7ebb07944000

mmap(0x7ebb0799c000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0xe7000) = 0x7ebb0799c000

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\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\0\0"..., 784, 64) = 784

mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ebb07600000

mmap(0x7ebb07628000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x28000) = 0x7ebb07628000

mmap(0x7ebb077b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x7ebb077b0000

mmap(0x7ebb077ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x1fe000) = 0x7ebb077ff000

mmap(0x7ebb07805000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0x7ebb07805000

close(3)   = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7ebb078b2000

arch_prctl(ARCH_SET_FS, 0x7ebb078b2740) = 0

set_tid_address(0x7ebb078b2a10)   = 22318

set_robust_list(0x7ebb078b2a20, 24)   = 0

rseq(0x7ebb078b3060, 0x20, 0, 0x53053053) = 0

mprotect(0x7ebb077ff000, 16384, PROT_READ) = 0

mprotect(0x7ebb0799c000, 4096, PROT_READ) = 0

mprotect(0x654707146000, 4096, PROT_READ) = 0

mprotect(0x7ebb079db000, 8192, PROT_READ) = 0

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

munmap(0x7ebb0799e000, 20243) = 0

getrandom("\x07\x1d\x12\xbd\xaf\xab\xe3\x56", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x654728ebd000

brk(0x654728ede000) = 0x654728ede000

rt_sigaction(SIGRT_1, {sa_handler=0x7ebb07699530, sa_mask=[], sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x7ebb07645330}, 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) = 0x7ebb06dff000

mprotect(0x7ebb06e00000, 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|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7ebb075ff990, parent_tid=0x7ebb075ff990, exit_signal=0, stack=0x7ebb06dff000, stack_size=0x7fff80, tls=0x7ebb075ff6c0} => {parent_tid=[22319]}, 88) = 22319

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

futex(0x7ebb075ff990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 22319, NULL, FUTEX_BITSET_MATCH_ANY) = 0

munmap(0x7ebb065fe000, 8392704) = 0

write(1, "\nparallel\nsort completed in 0.00"..., 85

parallel

sort completed in 0.006796 sec

max threads param: 8 (depth: 3), size: 8192

) = 85

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В лабораторной работе была реализована параллельная битоническая сортировка с использованием библиотеки pthread. Программа создаёт рекурсивную иерархию потоков, где каждый сортирует свою часть массива, а их количество ограничивается параметром глубины рекурсии. Исследование показало, что ускорение эффективно

на больших массивах, но снижается при малых размерах данных из-за накладных расходов на создание потоков.Алгоритм демонстрирует классическую для многопоточных приложений зависимость с убывающей отдачей при чрезмерном увеличении параллелизма.