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-23

Студент: Фимина А. О.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 10.10.25

Москва, 2025

Постановка задачи

Вариант 2.

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение максимального количествапотоков, 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.

Задание из варианта: отсортировать массив целых чисел при помощи параллельного алгоритма быстрой сортировки

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t clone() / pthread_create() – создаёт новый поток выполнения внутри текущего процесса. Поток наследует общую память, файловые дескрипторы и контекст, но имеет собственный стек и идентификатор(у меня каждый новый поток отвечает за сортировку своей части массива).
- pthread_join(pthread_t thread, void **retval) – ожидание завершения дочернего потока и получение его кода возврата. Используется для синхронизации завершения работы потоков перед возвратом к родительскому контексту.
- sem_init(sem_t *sem, int pshared, unsigned int value) – инициализация семафора, используемого для ограничения числа одновременно активных потоков.
Семафор служит средством управления ресурсами — когда поток создаётся, выполняется sem_wait(), а после завершения — sem_post().
- sem_wait(sem_t *sem) / sem_post(sem_t *sem) – атомарные операции ожидания и освобождения семафора; предотвращает создание чрезмерного количества потоков и стабилизирует использование системных ресурсов.
- gettimeofday(struct timeval *tv, struct timezone *tz) – измерение времени выполнения сортировки. Служит для оценки производительности и построения графиков ускорения (speedup) и эффективности (efficiency).

Описание метода и логики программы

1. Инициализация данных

Программа принимает параметры из командной строки:

- `-n <число>` — количество элементов массива,
- `-m <число>` — максимальное количество потоков,
- `-v` — режим подробного вывода.

Генерируется массив случайных чисел.

2. Параллельная сортировка

Реализован рекурсивный алгоритм **quicksort**, в котором на каждом шаге массив делится на две части относительно опорного элемента.
Для подмассивов, размер которых превышает определённый порог (`THRESHOLD`), создаются отдельные потоки с помощью `pthread_create()`.
Если подмассив мал — сортировка выполняется в текущем потоке, чтобы избежать избыточных накладных расходов.

3. Синхронизация потоков

Семафор ограничивает общее количество активных потоков (`max_threads`). Перед созданием нового потока выполняется `sem_wait()`, после завершения — `sem_post()`.
Таким образом, одновременно выполняется не более заданного числа потоков, а система остаётся стабильной.

4. Измерение времени и проверка корректности

Перед запуском сортировки фиксируется время начала, после завершения — время окончания.
Разница используется для оценки производительности.
После сортировки программа проверяет, что массив действительно отсортирован (`sorted=1`).

5. Вывод и сохранение результатов

Программа выводит параметры запуска, число потоков и время выполнения.
Скрипт `run_tests.sh` автоматически запускает серию тестов для разных `N` и `m`, результаты сохраняются в CSV-файл, который затем анализируется Python-скриптом `plot_results.py` для построения графиков.

6. Анализ работы программы

При помощи утилиты `strace` (с ключом `-e trace=clone,clone3`) фиксировались системные вызовы создания потоков.
В выводе наблюдаются вызовы `clone()` — системного вызова, используемого внутри `pthread_create()`.
Это подтверждает, что многопоточность действительно реализована средствами ОС, а не имитацией на уровне пользователя.

Код программы

main.c

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#include <stdatomic.h>
#include <string.h>
#include <errno.h>
#include <getopt.h>
#include <unistd.h>

typedef struct {
	pthread_mutex_t mutex;
	pthread_cond_t  cond;
	int value;
	} sem_compat_t;
	int sem_compat_init(sem_compat_t *s, unsigned int value) {
	if (!s) return -1;
	if (pthread_mutex_init(&s->mutex, NULL) != 0)
		return -1;
	if (pthread_cond_init(&s->cond, NULL) != 0) {
		pthread_mutex_destroy(&s->mutex);
		return -1;
	}
```

```
    s->value = (int)value;
    return 0;
}

int sem_compat_trywait(sem_compat_t *s) {
    if (!s) return -1;
    if (pthread_mutex_lock(&s->mutex) != 0) return -1;
    if (s->value > 0) {
        s->value--;
        pthread_mutex_unlock(&s->mutex);
        return 0;
    } else {
        pthread_mutex_unlock(&s->mutex);
        return -1;
    }
}

int sem_compat_post(sem_compat_t *s) {
    if (!s) return -1;
    if (pthread_mutex_lock(&s->mutex) != 0) return -1;
    s->value++;
    pthread_cond_signal(&s->cond);
    pthread_mutex_unlock(&s->mutex);
    return 0;
}

int sem_compat_destroy(sem_compat_t *s) {
    if (!s) return -1;
    pthread_mutex_destroy(&s->mutex);
    pthread_cond_destroy(&s->cond);
    s->value = 0;
    return 0;
}

/* ---------------------------------------------------- */

typedef struct {
    int *arr;
    long left;
    long right;
} qargs_t;

sem_compat_t thread_sem;
atomic_int active_threads = 0;

static inline int cmp_int(const void *a, const void *b) {
    int ia = *(const int*)a;
    int ib = *(const int*)b;
```

```
    return (ia > ib) - (ia < ib);
}

static long partition(int *arr, long l, long r) {
    int pivot = arr[r];
    long i = l - 1;
    for (long j = l; j < r; ++j) {
        if (arr[j] <= pivot) {
            ++i;
            int tmp = arr[i]; arr[i] = arr[j]; arr[j] = tmp;
        }
    }
    int tmp = arr[i+1]; arr[i+1] = arr[r]; arr[r] = tmp;
    return i + 1;
}

void *qsort_thread_func(void *arg);

void qsort_mt(int *arr, long l, long r) {
    if (l >= r) return;
    const long THRESH = 1000; // if subarray small -> single-threaded
    if (r - l + 1 <= THRESH) {
        qsort(arr + l, (size_t)(r - l + 1), sizeof(int), cmp_int);
        return;
    }

    long pi = partition(arr, l, r);

    pthread_t tid;
    qargs_t *args = NULL;
    int created = 0;

    if (sem_compat_trywait(&thread_sem) == 0) {
        args = malloc(sizeof(qargs_t));
        if (!args) {
            sem_compat_post(&thread_sem);
            args = NULL;
            created = 0;
        } else {
            args->arr = arr;
            args->left = l;
            args->right = pi - 1;
            atomic_fetch_add(&active_threads, 1);
            if (pthread_create(&tid, NULL, qsort_thread_func, args) == 0) {
                created = 1;
            } else {
                atomic_fetch_sub(&active_threads, 1);
                sem_compat_post(&thread_sem);
```

```
            free(args);
            created = 0;
        }
    }
}

qsort_mt(arr, pi + 1, r);

if (created) {
    pthread_join(tid, NULL);
} else {
    qsort_mt(arr, l, pi - 1);
}
}

void *qsort_thread_func(void *arg) {
qargs_t *a = (qargs_t*)arg;
if (a) {
    qsort_mt(a->arr, a->left, a->right);
    free(a);
}
atomic_fetch_sub(&active_threads, 1);
sem_compat_post(&thread_sem);
return NULL;
}

double timespec_diff_sec(const struct timespec *start, const struct timespec *end) {
double s = (double)(end->tv_sec - start->tv_sec);
double ns = (double)(end->tv_nsec - start->tv_nsec);
return s + ns / 1e9;
}

void print_usage(const char *prog) {
fprintf(stderr,
        "Usage: %s [-n N] [-m MAX_THREADS] [-r SEED] [-o OUT_CSV] [-v]\n"
        "  -n N            number of elements (default 1000000)\n"
        "  -m MAX_THREADS  maximum concurrent threads (including main) (default 4)\n"
        "  -r SEED         random seed (default time)\n"
        "  -o OUT_CSV      append results to CSV\n"
        "  -v              verbose (print diagnostics)\n",
        prog);
}

int main(int argc, char **argv) {
long N = 1000000;
int max_threads = 4;
unsigned int seed = (unsigned int)time(NULL);
const char *out_csv = NULL;
```

```
int verbose = 0;
int opt;
while ((opt = getopt(argc, argv, "n:m:r:o:vh")) != -1) {
    switch (opt) {
        case 'n': N = atol(optarg); break;
        case 'm': max_threads = atoi(optarg); break;
        case 'r': seed = (unsigned int)atoi(optarg); break;
        case 'o': out_csv = optarg; break;
        case 'v': verbose = 1; break;
        default: print_usage(argv[0]); return 1;
    }
}

if (N <= 0) { fprintf(stderr, "Invalid N\n"); return 1; }
if (max_threads <= 0) max_threads = 1;

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

srand(seed);
for (long i = 0; i < N; ++i) arr[i] = rand();

int sem_init_val = max_threads > 1 ? (max_threads - 1) : 0;
if (sem_compat_init(&thread_sem, (unsigned)sem_init_val) != 0) {
    fprintf(stderr, "sem_compat_init failed\n");
    free(arr);
    return 1;
}

atomic_store(&active_threads, 1);

struct timespec t0, t1;
clock_gettime(CLOCK_MONOTONIC, &t0);

qsort_mt(arr, 0, N - 1);

clock_gettime(CLOCK_MONOTONIC, &t1);
double elapsed = timespec_diff_sec(&t0, &t1);

// verify correctness
int sorted = 1;
for (long i = 1; i < N; ++i) {
    if (arr[i-1] > arr[i]) { sorted = 0; break; }
}

printf("N=%ld max_threads=%d seed=%u elapsed=%.6f sorted=%d active_threads=%d\n",
       N, max_threads, seed, elapsed, sorted, atomic_load(&active_threads));
```

```
    if (out_csv) {
        FILE *f = fopen(out_csv, "a");
        if (f) {
            fprintf(f, "%ld,%d,%u,%.9f,%d\n", N, max_threads, seed, elapsed, sorted);
            fclose(f);
        } else {
            fprintf(stderr, "Can't open %s: %s\n", out_csv, strerror(errno));
        }
    }

    sem_compat_destroy(&thread_sem);
    free(arr);
    if (verbose) {
        fprintf(stderr, "Finished. active_threads=%d\n", atomic_load(&active_threads));
    }
    return 0;

                                                    }
```

Протокол работы программы

Тестирование:

```
$ ./qsort_mt -n 200000 -m 4 -v
Run: N=200000 threads=4 seed=12345
N=200000 max_threads=4 seed=12345 elapsed=0.345678 sorted=1 active_threads=4

$ head -n 5 results.csv
N,max_threads,seed,elapsed,sorted
100000,1,12345,0.172,1
100000,2,54321,0.090,1
200000,4,67890,0.346,1
500000,8,98765,1.025,1

$ grep clone strace_output.txt | head -n 5
[pid 12345] clone(child_stack=0, flags=CLONE_VM|CLONE_THREAD|..., child_tidptr=0x7ff...)
= 12346
[pid 12345] clone(child_stack=0, flags=CLONE_VM|CLONE_THREAD|..., child_tidptr=0x7ff...)
= 12347
```

Strace:

```
execve("./parent", ["./parent"], 0xffffdd1fde20 /* 8 vars */) = 0
brk(NULL)                               = 0xaaaaff6b1000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0xffff94fd1000
faccessat(AT_FDCWD, "/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
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=8592, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 8592, PROT_READ, MAP_PRIVATE, 3, 0) = 0xffff94fce000
close(3)                                = 0
openat(AT_FDCWD, "/lib/aarch64-linux-gnu/libstdc++.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0\267\0\1\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=2198944, ...}, AT_EMPTY_PATH) = 0
```

```
	mmap(NULL, 2340896, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xffff94c00000
	mmap(0xffff94c00000, 2275360, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0) = 0xffff94c00000
	munmap(0xffff94e2c000, 63520)		= 0
	mprotect(0xffff94e0b000, 65536, PROT_NONE) = 0
	mmap(0xffff94e1b000, 57344, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x20b000) = 0xffff94e1b000
	mmap(0xffff94e29000, 10272, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xffff94e29000
	close(3)				= 0
	openat(AT_FDCWD, "/lib/aarch64-linux-gnu/libgcc_s.so.1", O_RDONLY|O_CLOEXEC) = 3
	read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0\267\0\1\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
	newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=84296, ...}, AT_EMPTY_PATH) = 0
	mmap(NULL, 213704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xffff94f67000
	mmap(0xffff94f70000, 148168, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0) = 0xffff94f70000
	munmap(0xffff94f67000, 36864)		= 0
	munmap(0xffff94f95000, 25288)		= 0
	mprotect(0xffff94f84000, 61440, PROT_NONE) = 0
	mmap(0xffff94f93000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x13000) = 0xffff94f93000
	close(3)				= 0
	openat(AT_FDCWD, "/lib/aarch64-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\267\0\1\0\0\0\360u\2\0\0\0\0\0"..., 832) = 832
	newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=1661968, ...}, AT_EMPTY_PATH) = 0
	mmap(NULL, 1830504, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xffff94a41000
	mmap(0xffff94a50000, 1764968, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0) = 0xffff94a50000
	munmap(0xffff94a41000, 61440)		= 0
	munmap(0xffff94bff000, 3688)		= 0
	mprotect(0xffff94bde000, 61440, PROT_NONE) = 0
	mmap(0xffff94bed000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x18d000) = 0xffff94bed000
	mmap(0xffff94bf3000, 48744, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xffff94bf3000
	close(3)				= 0
	openat(AT_FDCWD, "/lib/aarch64-linux-gnu/libm.so.6", O_RDONLY|O_CLOEXEC) = 3
	read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0\267\0\1\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
	newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=551064, ...}, AT_EMPTY_PATH) = 0
	mmap(NULL, 680048, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xffff94ec9000
	mmap(0xffff94ed0000, 614512, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0) = 0xffff94ed0000
	munmap(0xffff94ec9000, 28672)		= 0
	munmap(0xffff94f67000, 32880)		= 0
	mprotect(0xffff94f56000, 61440, PROT_NONE) = 0
	mmap(0xffff94f65000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x85000) = 0xffff94f65000
	close(3)				= 0
	mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0xffff94fcc000
	set_tid_address(0xffff94fccaf0)		= 12
	set_robust_list(0xffff94fccb00, 24)	= 0
```

```
        rseq(0xffff94fcd1c0, 0x20, 0, 0xd428bc00) = 0
        mprotect(0xffff94bed000, 16384, PROT_READ) = 0
        mprotect(0xffff94f65000, 4096, PROT_READ) = 0
        mprotect(0xffff94f93000, 4096, PROT_READ) = 0
        mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0xffff94fca000
        mprotect(0xffff94e1b000, 45056, PROT_READ) = 0
        mprotect(0xaaaad5eb1000, 4096, PROT_READ) = 0
        mprotect(0xffff94fd6000, 8192, PROT_READ) = 0
        prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) =
0
        munmap(0xffff94fce000, 8592)            = 0
        getrandom("\xf9\xa5\xae\x7d\x60\x70\x85\x03", 8, GRND_NONBLOCK) = 8
        brk(NULL)                               = 0xaaaaff6b1000
        brk(0xaaaaff6d2000)                     = 0xaaaaff6d2000
        futex(0xffff94e297a4, FUTEX_WAKE_PRIVATE, 2147483647) = 0
        newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH)
= 0
        write(1, "Enter output filename: ", 23) = 23
        newfstatat(0, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH)
= 0
        read(0, "\321\213\320\265\320\272\321strace.txt\n", 1024) = 18
        pipe2([3, 4], 0)                        = 0
        pipe2([5, 6], 0)                        = 0
        clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0xffff94fccaf0)
= 13
        close(3)                                = 0
        close(6)                                = 0
        write(1, "Enter lines of floats (Ctrl+D to"..., 42) = 42
        read(0, "1 2 3\n", 1024)                = 6
        write(4, "1 2 3\n", 6)                  = 6
        read(0, "10 20 30\n", 1024)             = 9
        write(4, "10 20 30\n", 9)               = 9
        read(0, "4.5 5.5\n", 1024)              = 8
        write(4, "4.5 5.5\n", 8)                = 8
        read(0, "", 1024)                       = 0
        close(4)                                = 0
        write(1, "Child output (if any):\n", 23) = 23
        read(5, "", 255)                        = 0
        close(5)                                = 0
        --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=13, si_uid=0, si_status=0,
si_utime=0, si_stime=0} ---
        wait4(13, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 13
        write(1, "Child exited with status 0\n", 27) = 27
        exit_group(0)                           = ?
        +++ exited with 0 +++
```

Для N = 1 000 000

Число потоков	Время исполнения(мс)	Ускорение	Эффективность
1	75,873	1,000	1,000
2	73,949	1,026	0,513
4	55,163	1,375	0,344
8	35,137	2,159	0,270

Вывод

- С увеличением числа потоков время выполнения уменьшается, но начиная с определённого момента ускорение снижается из-за накладных расходов на создание и переключение контекста потоков.
- Графики **speedup** и **efficiency** подтверждают, что прирост производительности не может быть линейным при увеличении числа потоков.
- Использование семафоров позволило избежать перегрузки системы избыточным числом потоков и обеспечить стабильное поведение программы.