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-209БВ-24

Студент: Крюков Д. М.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 01.12.25

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 параллельной сортировки слиянием

Общий метод и алгоритм решения

Использованные системные вызовы:

- 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_trywait(), а после завершения — sem_post().
- sem_trywait(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 считывает из файла размер массива и его элементы. Максимальное число потоков передаётся через аргумент командной строки.
2. **Параллельная сортировка**: реализован рекурсивный алгоритм merge sort. Если доступен свободный поток (семафор разрешает), то левая половина массива сортируется в отдельном потоке, правая — в текущем. Если потоков недостаточно — сортировка выполняется последовательно.
3. **Синхронизация**: семафор ограничивает число одновременно работающих потоков.
4. **Измерение времени**: фиксируется время начала и конца сортировки.

Код программы

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <sys/time.h>

sem_t thread_limit;

typedef struct {
    int* arr;
    int left;
    int right;
} thread_data;

double get_time_ms() {
    struct timeval tv;
    gettimeofday(&tv, NULL);
    return tv.tv_sec * 1000.0 + tv.tv_usec / 1000.0;
}

void merge(int* arr, int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int* L = malloc(n1 * sizeof(int));
    int* R = malloc(n2 * sizeof(int));

    for (int i = 0; i < n1; i++) L[i] = arr[left + i];
```

```
    for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;

    while (i < n1 && j < n2) {
        arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];
    }
    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];

    free(L);
    free(R);
}

void merge_sort(int* arr, int left, int right) {
    if (left >= right) return;
    int mid = left + (right - left) / 2;
    merge_sort(arr, left, mid);
    merge_sort(arr, mid + 1, right);
    merge(arr, left, mid, right);
}

void* thread_merge_sort(void* arg) {
    thread_data* data = (thread_data*)arg;
    merge_sort(data->arr, data->left, data->right);
    sem_post(&thread_limit);
    free(data);
    return NULL;
}
```

```
void parallel_merge_sort(int* arr, int left, int right) {
    if (left >= right) return;

    int mid = left + (right - left) / 2;

    pthread_t t1, t2;
    int created1 = 0, created2 = 0;

    if (sem_trywait(&thread_limit) == 0) {
        thread_data* d = malloc(sizeof(thread_data));
        d->arr = arr;
        d->left = left;
        d->right = mid;
        if (pthread_create(&t1, NULL, thread_merge_sort, d) == 0) {
            created1 = 1;
        } else {
            perror("pthread_create");
            free(d);
            sem_post(&thread_limit);
        }
    }

    if (sem_trywait(&thread_limit) == 0) {
        thread_data* d = malloc(sizeof(thread_data));
        d->arr = arr;
        d->left = mid + 1;
        d->right = right;
        if (pthread_create(&t2, NULL, thread_merge_sort, d) == 0) {
            created2 = 1;
        } else {
```

```
            perror("pthread_create");
            free(d);
            sem_post(&thread_limit);
        }
    }

    if (!created1) merge_sort(arr, left, mid);
    if (!created2) merge_sort(arr, mid + 1, right);

    // Ждём созданные потоки
    if (created1) pthread_join(t1, NULL);
    if (created2) pthread_join(t2, NULL);

    merge(arr, left, mid, right);
}

int main(int argc, char* argv[]) {
    if (argc < 2) {
        printf("Usage: %s <max_threads>\n", argv[0]);
        return 1;
    }

    int max_threads = atoi(argv[1]);
    sem_init(&thread_limit, 0, max_threads);

    FILE* f = fopen("input.txt", "r");
    if (!f) {
        perror("Failed to open file");
        return 1;
    }
```

```
    int n;
    fscanf(f, "%d", &n);

    int* arr = malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) fscanf(f, "%d", &arr[i]);
    fclose(f);

    printf("Array size: %d\n", n);
    printf("Max threads: %d\n", max_threads);

    double start_time = get_time_ms();

    if (max_threads > 1)
        parallel_merge_sort(arr, 0, n - 1);
    else
        merge_sort(arr, 0, n - 1);

    double end_time = get_time_ms();
    printf("Sorting time: %.2f ms\n", end_time - start_time);

    // for (int i = 0; i < n; i++) printf("%d ", arr[i]);
    // printf("\n");

    free(arr);
    sem_destroy(&thread_limit);

    return 0;
}
```

Протокол работы программы

Тестирование:

root@01d40df164ff:/workspace/lab2/src# ./main 2
Array size: 1000000
Max threads: 2 Sorting time:
78.12 ms

Strace:

```
1868  execve("./main", ["./main", "4"], 0x7ffe00d84e80 /* 29 vars */) = 0
1868  brk(NULL)                         = 0x2d6e000
1868  mmap(NULL, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7520af8e6000
1868  access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
1868  openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
1868  fstat(3, {st_mode=S_IFREG|0644, st_size=29003, ...}) = 0
1868  mmap(NULL, 29003, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7520af8de000
1868  close(3)                          = 0
1868  openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
1868  read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0p\236\2\0\0\0\0"..., 832) = 832
1868  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"..., 840, 64) =
840
1868  fstat(3, {st_mode=S_IFREG|0755, st_size=2003408, ...}) = 0
1868  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"..., 840, 64) =
840
1868  mmap(NULL, 2055800, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7520af6e8000
1868  mmap(0x7520af710000, 1462272, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7520af710000
1868  mmap(0x7520af875000, 352256, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x18d000) = 0x7520af875000
1868  mmap(0x7520af8cb000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e2000) = 0x7520af8cb000
1868  mmap(0x7520af8d1000, 52856, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7520af8d1000
1868  close(3)                          = 0
1868  mmap(NULL, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7520af6e5000
1868  arch_prctl(ARCH_SET_FS, 0x7520af6e5740) = 0
1868  set_tid_address(0x7520af6e5a10)   = 1868
1868  set_robust_list(0x7520af6e5a20, 24) = 0
1868  rseq(0x7520af6e5680, 0x20, 0, 0x53053053) = 0
1868  mprotect(0x7520af8cb000, 16384, PROT_READ) = 0
1868  mprotect(0x403000, 4096, PROT_READ) = 0
1868  mprotect(0x7520af91c000, 8192, PROT_READ) = 0
1868  prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0
1868  munmap(0x7520af8de000, 29003)     = 0
1868  getrandom("\xb0\xe2\x12\x3f\xe6\xc4\x03\x0f", 8, GRND_NONBLOCK) = 8
1868  brk(NULL)                         = 0x2d6e000
1868  brk(0x2d8f000)                    = 0x2d8f000
1868  openat(AT_FDCWD, "input.txt", O_RDONLY) = 3
```

```
1868  fstat(3, {st_mode=S_IFREG|0777, st_size=4889439, ...}) = 0
1868  read(3, "1000000\n3474\n6147\n9845\n8169\n4229"..., 4096) = 4096
1868  mmap(NULL, 4001792, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7520af314000
1868  read(3, "283\n3655\n1424\n2507\n6244\n4414\n944"..., 4096) = 4096
1868  read(3, "\n159\n3085\n9104\n1258\n6105\n3751\n56"..., 4096) = 4096
1868  read(3, "9\n1330\n1931\n7076\n3286\n420\n1001\n4"..., 4096) = 4096
1868  read(3, "\n8077\n5729\n4390\n3250\n2891\n3690\n2"..., 4096) = 2911
1868  close(3)                          = 0
1868  fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
1868  write(1, "Array size: 1000000\n", 20) = 20
1868  write(1, "Max threads: 4\n", 15)  = 15
1868  rt_sigaction(SIGRT_1, {sa_handler=0x7520af7784b0, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO,
sa_restorer=0x7520af727df0}, NULL, 8) = 0
1868  rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0
1868  mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7520aeb13000
1868  mprotect(0x7520aeb14000, 8388608, PROT_READ|PROT_WRITE) = 0
1868  rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
1868
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLO
NE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7520af313990, parent_tid=0x7520af313990, exit_signal=0, stack=0x7520aeb13000,
stack_size=0x7fff80, tls=0x7520af3136c0}, 88) = -1 ENOSYS (Function not implemented)
1868  clone(child_stack=0x7520af312f70,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE
_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
parent_tid=[1870], tls=0x7520af3136c0, child_tidptr=0x7520af313990) = 1870
1870  rseq(0x7520af313600, 0x20, 0, 0x53053053 <unfinished ...>
1868  rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
1870  <... rseq resumed>)               = 0
1868  <... rt_sigprocmask resumed>NULL, 8) = 0
1870  set_robust_list(0x7520af3139a0, 24 <unfinished ...>
1868  mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>
1870  <... set_robust_list resumed>)    = 0
1868  <... mmap resumed>)               = 0x7520ae312000
1870  rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
1868  mprotect(0x7520ae313000, 8388608, PROT_READ|PROT_WRITE <unfinished ...>
1870  <... rt_sigprocmask resumed>NULL, 8) = 0
1868  <... mprotect resumed>)           = 0
1870  mmap(NULL, 134217728, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0
<unfinished ...>
1868  rt_sigprocmask(SIG_BLOCK, ~[],  <unfinished ...>
1870  <... mmap resumed>)               = 0x7520a6312000
1868  <... rt_sigprocmask resumed>[], 8) = 0
1870  munmap(0x7520a6312000, 30334976 <unfinished ...>
1868  clone(child_stack=0x7520aeb11f70,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE
_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID
<unfinished ...>
1870  <... munmap resumed>)             = 0
1868  <... clone resumed>, parent_tid=[1871], tls=0x7520aeb126c0,
child_tidptr=0x7520aeb12990) = 1871
1870  munmap(0x7520ac000000, 36773888 <unfinished ...>
```

```
1868  rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
1871  rseq(0x7520aeb12600, 0x20, 0, 0x53053053 <unfinished ...>
1868  <... rt_sigprocmask resumed>NULL, 8) = 0
1870  <... munmap resumed>)            = 0
1868  futex(0x7520af313990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 1870,
NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>
1871  <... rseq resumed>)              = 0
1870  mprotect(0x7520a8000000, 135168, PROT_READ|PROT_WRITE <unfinished ...>
1871  set_robust_list(0x7520aeb129a0, 24 <unfinished ...>
1870  <... mprotect resumed>)          = 0
1871  <... set_robust_list resumed>)   = 0
1871  rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
1871  mmap(NULL, 134217728, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7520a0000000
1871  munmap(0x7520a4000000, 67108864)  = 0
1871  mprotect(0x7520a0000000, 135168, PROT_READ|PROT_WRITE) = 0
1870  mprotect(0x7520a8021000, 122880, PROT_READ|PROT_WRITE) = 0
1870  openat(AT_FDCWD, "/proc/sys/vm/overcommit_memory", O_RDONLY|O_CLOEXEC) =
3
1870  read(3, "1", 1)                  = 1
1870  close(3)                         = 0
1870  madvise(0x7520a8022000, 118784, MADV_DONTNEED <unfinished ...>
1871  mprotect(0x7520a0021000, 122880, PROT_READ|PROT_WRITE <unfinished ...>
1870  <... madvise resumed>)           = 0
1871  <... mprotect resumed>)          = 0
1871  madvise(0x7520a0022000, 118784, MADV_DONTNEED) = 0
1870  madvise(0x7520a8022000, 118784, MADV_DONTNEED) = 0
1870  mmap(NULL, 253952, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7520ae2d4000
1870  mmap(NULL, 253952, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>
1871  madvise(0x7520a0022000, 118784, MADV_DONTNEED <unfinished ...>
1870  <... mmap resumed>)              = 0x7520ae296000
1871  <... madvise resumed>)           = 0
1871  mmap(NULL, 253952, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7520ae258000
1870  munmap(0x7520ae2d4000, 253952)    = 0
1870  munmap(0x7520ae296000, 253952 <unfinished ...>
1871  munmap(0x7520ae258000, 253952 <unfinished ...>
1870  <... munmap resumed>)            = 0
1871  <... munmap resumed>)            = 0
1871  mprotect(0x7520a003f000, 249856, PROT_READ|PROT_WRITE) = 0
1870  mprotect(0x7520a803f000, 249856, PROT_READ|PROT_WRITE) = 0
1871  mmap(NULL, 503808, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7520ae297000
1870  mmap(NULL, 503808, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7520ae21c000
1871  munmap(0x7520ae297000, 503808)    = 0
1870  munmap(0x7520ae21c000, 503808)    = 0
1871  mprotect(0x7520a007c000, 503808, PROT_READ|PROT_WRITE <unfinished ...>
1870  mprotect(0x7520a807c000, 503808, PROT_READ|PROT_WRITE <unfinished ...>
1871  <... mprotect resumed>)          = 0
1870  <... mprotect resumed>)          = 0
1871  mmap(NULL, 1003520, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7520ae21d000
```

```
1870  mmap(NULL, 1003520, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7520ae128000
1871  munmap(0x7520ae21d000, 1003520)   = 0
1871  rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0
1870  munmap(0x7520ae128000, 1003520 <unfinished ...>
1871  madvise(0x7520ae312000, 8368128, MADV_DONTNEED) = 0
1870  <... munmap resumed>)            = 0
1871  exit(0 <unfinished ...>
1870  rt_sigprocmask(SIG_BLOCK, ~[RT_1],  <unfinished ...>
1871  <... exit resumed>)              = ?
1870  <... rt_sigprocmask resumed>NULL, 8) = 0
1871  +++ exited with 0 +++
1870  madvise(0x7520aeb13000, 8368128, MADV_DONTNEED) = 0
1870  exit(0)                          = ?
1868  <... futex resumed>)             = 0
1870  +++ exited with 0 +++
1868  mmap(NULL, 2002944, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7520ae129000
1868  mmap(NULL, 2002944, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7520adf40000
1868  munmap(0x7520ae129000, 2002944)   = 0
1868  munmap(0x7520adf40000, 2002944)   = 0
1868  write(1, "Sorting time: 63.26 ms\n", 23) = 23
1868  munmap(0x7520af314000, 4001792)   = 0
1868  exit_group(0)                    = ?
1868  +++ exited with 0 +++
```

Для N = 1 000 000

Число потоков	Время исполнения(мс)	Ускорение	Эффективность
1	78,78	1,00	1,00
2	49,73	1,58	0,79
4	46,31	1,70	0,43
8	44,5	1,77	0,22

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.
- Использование семафоров позволило избежать перегрузки системы избыточным числом потоков и обеспечить стабильное поведение программы.
- Для мелких или средних массивов накладные расходы на управление потоками могут превышать выгоду от параллельных вычислений.