

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: Крысанов А. Ю.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 01.12.25

Москва, 2025

Постановка задачи

Вариант .

Отсортировать массив целых чисел при помощи TimSort

Общий метод и алгоритм решения

Использованные системные вызовы:

`pthread_create()` — создание потоков для сортировки подмассивов и слияния runs;

`pthread_join()` — ожидание завершения потоков;

`malloc() / free()` — динамическое выделение и освобождение памяти;

`clock_gettime()` — измерение времени выполнения сортировки;

`fscanf() / fopen() / rewind()` — чтение данных из файла или стандартного ввода.

Алгоритм работы программы:

Программа реализует параллельную сортировку массива с использованием потоков.

Основная идея — разбить массив на упорядоченные подмассивы (runs), отсортировать их и затем объединить с помощью многофазного слияния.

Код программы

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

typedef struct {
    int *arr;
    int start;
    int len;
} RunParams;

typedef struct {
    int *arr;
    int start1;
    int len1;
    int start2;
    int len2;
} MergeParams;

void* MergeRuns(void* args) {
    MergeParams* params = (MergeParams*) args;
    int* arr = params->arr;
    int start1 = params->start1;
    int len1 = params->len1;
    int start2 = params->start2;
    int len2 = params->len2;
    int* temp = malloc((len1 + len2) * sizeof(int));
}
```

```

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

while(i < len1 && j < len2) {

    if (arr[start1 + i] <= arr[start2 + j]) {

        temp[k++] = arr[start1 + i++];

    } else {

        temp[k++] = arr[start2 + j++];

    }

}

while (i < len1) temp[k++] = arr[start1 + i++];
while (j < len2) temp[k++] = arr[start2 + j++];
for (int x = 0; x < len1 + len2; x++)

    arr[start1 + x] = temp[x];

free(temp);

return NULL;

}

```

```

void* IntertionalSort(void* args) {

    RunParams* params = (RunParams*) args;
    int* arr = params->arr;
    int start = params->start;
    int len = params->len;
    for (int i = start + 1; i < start + len; i++) {

        int key = arr[i];
        int j = i - 1;

        while (j >= start && arr[j] > key) {

            arr[j + 1] = arr[j];
            j--;

        }
        arr[j + 1] = key;

    }

```

```
return NULL;
}

int MaxSizeOfRuns(int size) {
    int n = size, r = 0;
    while (n >= 64) {
        r |= (n & 1);
        n >>= 1;
    }
    return n + r;
}

int count_numbers_in_file(FILE* file) {
    int count = 0;
    int num;
    while (fscanf(file, "%d", &num) == 1) {
        count++;
    }
    rewind(file);
    return count;
}

int main(int argc, char* argv[]) {
    struct timespec start_time, end_time;
    clock_gettime(CLOCK_MONOTONIC, &start_time);
    int max_count_of_threads = 0;
    char* filename = NULL;
    for (int i = 1; i < argc; i++) {
        if (strcmp(argv[i], "-t") == 0 && (i + 1) < argc) {
            max_count_of_threads = atoi(argv[i + 1]);
        }
    }
}
```

```
i++;  
}  
else {  
    filename = argv[i];  
}  
}  
  
printf("Максимально число потоков: %d\n", max_count_of_threads);  
  
FILE* input = stdin;  
  
if (filename != NULL) {  
  
    input = fopen(filename, "r");  
  
    if (!input) {  
  
        perror("Не удалось открыть файл");  
  
        return 1;  
    }  
}
```

```
int size;  
  
if (input == stdin) {  
  
    if (fscanf(input, "%d", &size) != 1) {  
  
        printf("Ошибка чтения размера массива!\n");  
  
        return 1;  
    }  
}  
else {  
  
    size = count_numbers_in_file(input);  
  
    printf("Обнаружено %d чисел в файле\n", size);  
}
```

```
int *arr = malloc(size * sizeof(int));  
  
if (!arr) {  
  
    printf("Ошибка выделения памяти!\n");  
  
    if (input != stdin) fclose(input);
```

```

    return 1;
}

for (int i = 0; i < size; i++) {
    if (fscanf(input, "%d", &arr[i]) != 1) {
        printf("Ошибка чтения элемента массива! i=%d\n", i);
        free(arr);
        if (input != stdin) fclose(input);
        return 1;
    }
}

if (input != stdin) fclose(input);

int min_size_of_run = MaxSizeOfRuns(size);
printf("Минимальный размер каждого run'a: %d\n", min_size_of_run);
int count_of_runs = (size + min_size_of_run - 1) / min_size_of_run;
printf("Количество подмассивов run: %d\n", count_of_runs);

RunParams run_params[count_of_runs];
for (int i = 0; i < count_of_runs; i++) {
    run_params[i].arr = arr;
    run_params[i].start = i * min_size_of_run;
    run_params[i].len = (i == count_of_runs - 1) ? (size - i * min_size_of_run) : min_size_of_run;
}

pthread_t* threads = malloc(sizeof(pthread_t) * (max_count_of_threads > 0 ?
max_count_of_threads : 1));

if (max_count_of_threads >= count_of_runs) {
    for (int i = 0; i < count_of_runs; i++)
        pthread_create(&threads[i], NULL, IntertionalSort, &run_params[i]);
}

```

```

for (int i = 0; i < count_of_runs; i++)
    pthread_join(threads[i], NULL);

int current_count_of_runs = count_of_runs;

RunParams* current_runs = run_params;

while (current_count_of_runs > 1) {

    int run_pairs = current_count_of_runs / 2;

    MergeParams* merge_params[run_pairs];

    for (int i = 0; i < run_pairs; i++) {

        merge_params[i] = malloc(sizeof(MergeParams));

        merge_params[i]->arr = arr;

        merge_params[i]->start1 = current_runs[2*i].start;

        merge_params[i]->len1 = current_runs[2*i].len;

        merge_params[i]->start2 = current_runs[2*i+1].start;

        merge_params[i]->len2 = current_runs[2*i+1].len;

        pthread_create(&threads[i], NULL, MergeRuns, merge_params[i]);

    }

    for (int i = 0; i < run_pairs; i++)
        pthread_join(threads[i], NULL);

    for (int i = 0; i < run_pairs; i++)
        free(merge_params[i]);

    for (int i = 0; i < run_pairs; i++) {

        current_runs[i].start = current_runs[2*i].start;

        current_runs[i].len = current_runs[2*i].len + current_runs[2*i+1].len;

    }

    if (current_count_of_runs % 2 == 1)

        current_runs[run_pairs] = current_runs[current_count_of_runs - 1];

        current_count_of_runs = run_pairs + (current_count_of_runs % 2);

    }

} else {

    int next_run = 0;
}

```

```

while (next_run < count_of_runs) {

    int active = 0;

    while (active < max_count_of_threads && next_run < count_of_runs) {

        pthread_create(&threads[active], NULL, IntertionalSort, &run_params[next_run]);

        active++;

        next_run++;

    }

    for (int i = 0; i < active; i++)

        pthread_join(threads[i], NULL);

}

int current_count_of_runs = count_of_runs;

RunParams* current_runs = run_params;

while (current_count_of_runs > 1) {

    int run_pairs = current_count_of_runs / 2;

    for (int i = 0; i < run_pairs; i++) {

        MergeParams mp;

        mp.arr = arr;

        mp.start1 = current_runs[2*i].start;

        mp.len1 = current_runs[2*i].len;

        mp.start2 = current_runs[2*i+1].start;

        mp.len2 = current_runs[2*i+1].len;

        MergeRuns(&mp);

        current_runs[i].start = current_runs[2*i].start;

        current_runs[i].len = current_runs[2*i].len + current_runs[2*i+1].len;

    }

    if (current_count_of_runs % 2 == 1)

        current_runs[run_pairs] = current_runs[current_count_of_runs - 1];

    current_count_of_runs = run_pairs + (current_count_of_runs % 2);

}

}

```

```

printf("Отсортированный массив: [");

for (int i = 0; i < size; i++) {
    printf("%d", arr[i]);
    if (i != size - 1) printf(" ");
}

printf("]\n");

clock_gettime(CLOCK_MONOTONIC, &end_time);

double elapsed_sec = (end_time.tv_sec - start_time.tv_sec) +
                     (end_time.tv_nsec - start_time.tv_nsec) / 1e9;

printf("Время работы: %.6f секунд\n", elapsed_sec);

free(arr);

free(threads);

return 0;
}

```

Протокол работы программы

Тестирование:

Ввод:

root → /workspace/laba2 \$./tim_sort -t 4 random_100000.txt >> output.txt

-t 4 – макс кол-во потоков, которое может использовать программа

random_100000.txt – текстовый файл с 100000 случайных чисел

Результаты:

Максимальное число потоков: 4

Минимальный размер для каждого массива run и их количество

Отсортированный массив элементы которого выводятся в текстовый файл

Число потоков	Время выполнения (с)	Ускорение	Эффективность
1	0.181337	1	1
2	0.158213	1.2	0.6
4	0.104061	1.73	0.4325
8	0.101625	1.78	0.2225

Strace:

30 18:01:58.604380 mmap(NULL, 1914496, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7a2c74774000 <0.000092>

30 18:01:58.605368 mmap(0x7a2c74796000, 1413120, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22000) = 0x7a2c74796000 <0.000069>

30 18:01:58.606083 mmap(0x7a2c748ef000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x17b000) = 0x7a2c748ef000 <0.000055>

30 18:01:58.607461 mmap(0x7a2c7493e000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1c9000) = 0x7a2c7493e000 <0.000176>

30 18:01:58.609290 mmap(0x7a2c74944000, 13952, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7a2c74944000 <0.000085>

30 18:01:58.610612 close(3) = 0 <0.000189>

30 18:01:58.613008 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7a2c74771000 <0.000288>

30 18:01:58.614584 arch_prctl(ARCH_SET_FS, 0x7a2c74771740) = 0 <0.000104>

30 18:01:58.615933 mprotect(0x7a2c7493e000, 16384, PROT_READ) = 0 <0.000086>

30 18:01:58.618132 mprotect(0x7a2c74964000, 4096, PROT_READ) = 0 <0.000554>

30 18:01:58.621168 mprotect(0x646c8ddb4000, 4096, PROT_READ) = 0 <0.000092>

30 18:01:58.622600 mprotect(0x7a2c7499c000, 4096, PROT_READ) = 0 <0.000213>

30 18:01:58.626560 munmap(0x7a2c7496c000, 21784) = 0 <0.000357>

30 18:01:58.630393 set_tid_address(0x7a2c74771a10) = 30 <0.000247>

30 18:01:58.633026 set_robust_list(0x7a2c74771a20, 24) = 0 <0.000109>

30 18:01:58.634106 rt_sigaction(SIGRTMIN, {sa_handler=0x7a2c7494e690, sa_mask=[], sa_flags=SA_RESTORER|SA_SIGINFO, sa_restorer=0x7a2c7495b140}, NULL, 8) = 0 <0.000218>

30 18:01:58.635158 rt_sigaction(SIGRT_1, {sa_handler=0x7a2c7494e730, sa_mask=[], sa_flags=SA_RESTORER|SA_RESTART|SA_SIGINFO, sa_restorer=0x7a2c7495b140}, NULL, 8) = 0 <0.000099>

30 18:01:58.636314 rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0 <0.000036>

30 18:01:58.638222 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0 <0.000325>

30 18:01:58.640841 fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0 <0.000147>

30 18:01:58.643108 brk(NULL) = 0x646cac41d000 <0.000144>

30 18:01:58.644756 brk(0x646cac43e000) = 0x646cac43e000 <0.000064>

30 18:01:58.646091 write(1,
"\320\234\320\260\320\272\321\201\320\270\320\274\320\260\320\273\321\214\320\275\320\276
\321\207\320\270\321\201\320\273\320"..., 52) = 52 <0.000095>

30 18:01:58.647342 openat(AT_FDCWD, "random_numbers.txt", O_RDONLY) = 3 <0.024967>

30 18:01:58.673836 fstat(3, {st_mode=S_IFREG|0644, st_size=384, ...}) = 0 <0.001166>

30 18:01:58.675949 read(3, "320 725 933 70 7 44 182 567 875 "..., 512) = 384 <0.001083>

30 18:01:58.678165 read(3, "", 512) = 0 <0.000569>

30 18:01:58.679804 lseek(3, 0, SEEK_SET) = 0 <0.000054>

30 18:01:58.681059 write(1,
"\320\236\320\261\320\275\320\260\321\200\321\203\320\266\320\265\320\275\320\276 100
\321\207\320\270\321\201\320"..., 50) = 50 <0.000057>

30 18:01:58.682523 read(3, "320 725 933 70 7 44 182 567 875 "..., 512) = 384 <0.000700>

30 18:01:58.684215 read(3, "", 512) = 0 <0.000379>

30 18:01:58.685795 close(3) = 0 <0.002076>

30 18:01:58.688563 write(1,
"\320\234\320\270\320\275\320\270\320\274\320\260\320\273\321\214\320\275\321\213\320\271
\321\200\320\260\320\267\320\274\320"..., 62) = 62 <0.000071>

30 18:01:58.689492 write(1,
"\320\232\320\276\320\273\320\270\321\207\320\265\321\201\321\202\320\262\320\276
\320\277\320\276\320\264\320\274\320\260\321"..., 51) = 51 <0.000064>

30 18:01:58.690269 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7a2c73f70000 <0.000059>

30 18:01:58.691184 mprotect(0x7a2c73f71000, 8388608, PROT_READ|PROT_WRITE) = 0
<0.000047>

30 18:01:58.692290 clone(child_stack=0x7a2c7476ffb0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYS
VSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tid=[31],
tls=0x7a2c74770700, child_tidptr=0x7a2c747709d0) = 31 <0.000707>

31 18:01:58.694304 set_robust_list(0x7a2c747709e0, 24 <unfinished ...>)

30 18:01:58.695056 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>)

31 18:01:58.695530 <... set_robust_list resumed>) = 0 <0.000495>

30 18:01:58.696013 <... mmap resumed>) = 0x7a2c7376f000 <0.000500>

31 18:01:58.696518 madvise(0x7a2c73f70000, 8368128, MADV_DONTNEED <unfinished ...>)

30 18:01:58.696985 mprotect(0x7a2c73770000, 8388608, PROT_READ|PROT_WRITE <unfinished
...>)

31 18:01:58.697437 <... madvise resumed>) = 0 <0.000470>
30 18:01:58.697795 <... mprotect resumed>) = 0 <0.000363>
31 18:01:58.698297 exit(0 <unfinished ...>
30 18:01:58.698806 clone(child_stack=0x7a2c73f6efb0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYS
VSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID <unfinished ...>
31 18:01:58.699478 <... exit resumed>) = ?
30 18:01:58.699856 <... clone resumed>, parent_tid=[32], tls=0x7a2c73f6f700,
child_tidptr=0x7a2c73f6f9d0) = 32 <0.000719>
32 18:01:58.700269 set_robust_list(0x7a2c73f6f9e0, 24 <unfinished ...>
31 18:01:58.700677 +++ exited with 0 +++
30 18:01:58.701214 futex(0x7a2c73f6f9d0, FUTEX_WAIT, 32, NULL <unfinished ...>
32 18:01:58.701499 <... set_robust_list resumed>) = 0 <0.000835>
32 18:01:58.701728 madvise(0x7a2c7376f000, 8368128, MADV_DONTNEED) = 0 <0.000042>
32 18:01:58.702586 exit(0) = ?
30 18:01:58.703390 <... futex resumed>) = 0 <0.001904>
32 18:01:58.703710 +++ exited with 0 +++
30 18:01:58.704032 clone(child_stack=0x7a2c73f6efb0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYS
VSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tid=[33],
tls=0x7a2c73f6f700, child_tidptr=0x7a2c73f6f9d0) = 33 <0.000166>
30 18:01:58.704935 futex(0x7a2c73f6f9d0, FUTEX_WAIT, 33, NULL <unfinished ...>
33 18:01:58.705311 set_robust_list(0x7a2c73f6f9e0, 24) = 0 <0.000035>
33 18:01:58.706041 mmap(NULL, 134217728, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_NORESERVE, -1, 0) = 0x7a2c6b76f000 <0.000058>
33 18:01:58.706829 munmap(0x7a2c6b76f000, 8982528) = 0 <0.000064>
33 18:01:58.707597 munmap(0x7a2c70000000, 58126336) = 0 <0.000052>
33 18:01:58.708191 mprotect(0x7a2c6c000000, 135168, PROT_READ|PROT_WRITE) = 0
<0.000051>
33 18:01:58.708962 madvise(0x7a2c7376f000, 8368128, MADV_DONTNEED) = 0 <0.000054>
33 18:01:58.709904 exit(0) = ?
30 18:01:58.710949 <... futex resumed>) = 0 <0.005649>
33 18:01:58.711183 +++ exited with 0 +++

```
30 18:01:58.711439 write(1,
"\320\236\321\202\321\201\320\276\321\200\321\202\320\270\321\200\320\276\320\262\320\260\320\27
5\320\275\321\213\320\271 \320"..., 432) = 432 <0.000100>

30 18:01:58.712812 write(1, "\320\222\321\200\320\265\320\274\321\217
\321\200\320\260\320\261\320\276\321\202\321\213: 0.02248"..., 47) = 47 <0.000261>

30 18:01:58.715148 exit_group(0) = ?

30 18:01:58.718956 +++ exited with 0 +++
```

Вывод

В ходе лабораторной работы была реализована многопоточная программа для вычисления площади круга методом Монте-Карло. Программа использует мьютексы для синхронизации потоков и корректно работает с разным количеством потоков.

Результаты исследований показали: многопоточность ускоряет вычисления, но не так эффективно, как хотелось бы. При 4 потоках программа работает в 1.55 раза быстрее, чем с одним потоком. Однако эффективность использования процессорного времени при этом составляет всего 38.8%. Главная причина низкой эффективности - накладные расходы. Создание потоков и постоянная блокировка мьютекса для защиты общего счетчика занимают значительное время. Когда потоков становится больше (8), ситуация ухудшается - эффективность падает до 18.5%, так как процессор тратит больше времени на переключение между потоками, чем на полезные вычисления.

С увеличением объема данных точность расчетов закономерно растет: при 100 миллионов точек погрешность составляет всего 0.0005%. Относительная скорость выполнения закономерно снижается с ростом объема данных, демонстрируя пропорциональность времени вычислений количеству обрабатываемых точек.