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-214БВ-25

Студент: Александров М.С.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 19.11.25

Москва, 2025

Постановка задачи

Вариант 17.

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 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.

Найти в большом целочисленном массиве минимальный и максимальный элементы.

Общий метод и алгоритм решения

Использованные системные вызовы:

- *int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void*), void *arg);* – создаёт новый поток, который начинает выполнение функции start_routine с аргументом arg.
- *int pthread_join(pthread_t thread, void **retval);* – ожидает завершения указанного потока и возвращает его результат.
- *int clock_gettime(clockid_t clockid, struct timespec *tp);* – получает текущее время, указанное идентификатором *clock_id* и записывает его в структуру *struct timespec*, на которую указывает параметр *tp*.

Алгоритм решения:

1. Генерация данных: создаётся массив, размер которого передаётся через аргументы командной строки, заполненный случайными числами от -10^9 до 10^9.

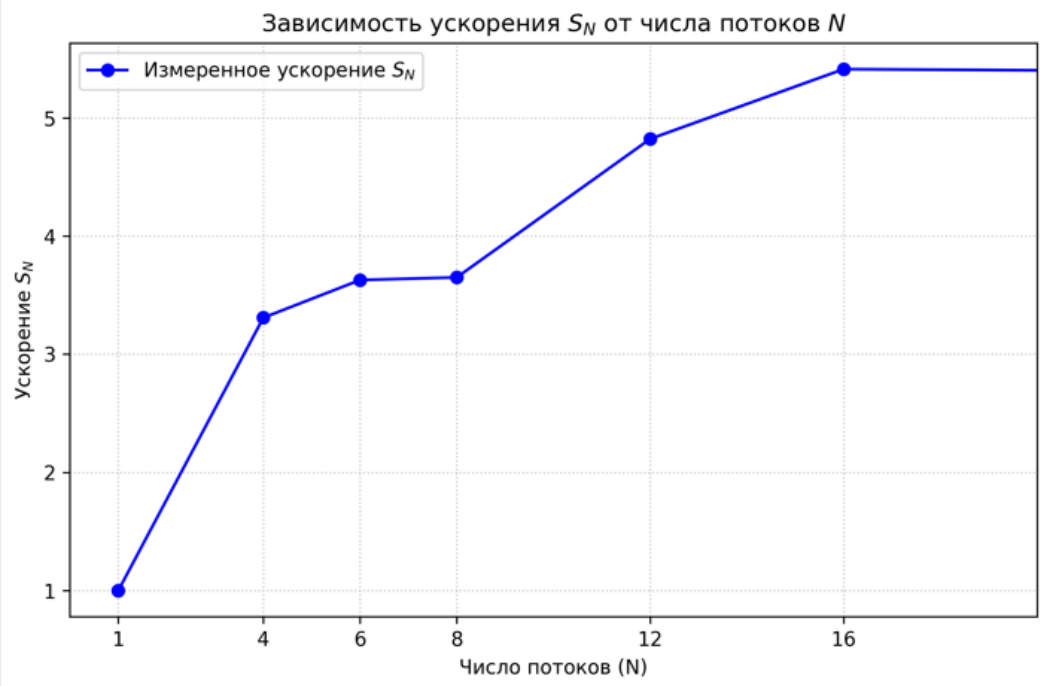
2. В последовательной версии один поток последовательно проходит по всем элементам массива, находя максимальный и минимальный элемент в массиве.
3. В параллельной версии массив делится на равное число непересекающихся частей (равное числу потоков), каждый поток обрабатывает свой диапазон массива, находя в своём куске локальный минимум и максимум.
4. Главный поток ожидает завершения всех дочерних и выполняет сводку, находя глобальный минимум и максимум.

Анализ ускорения и эффективности

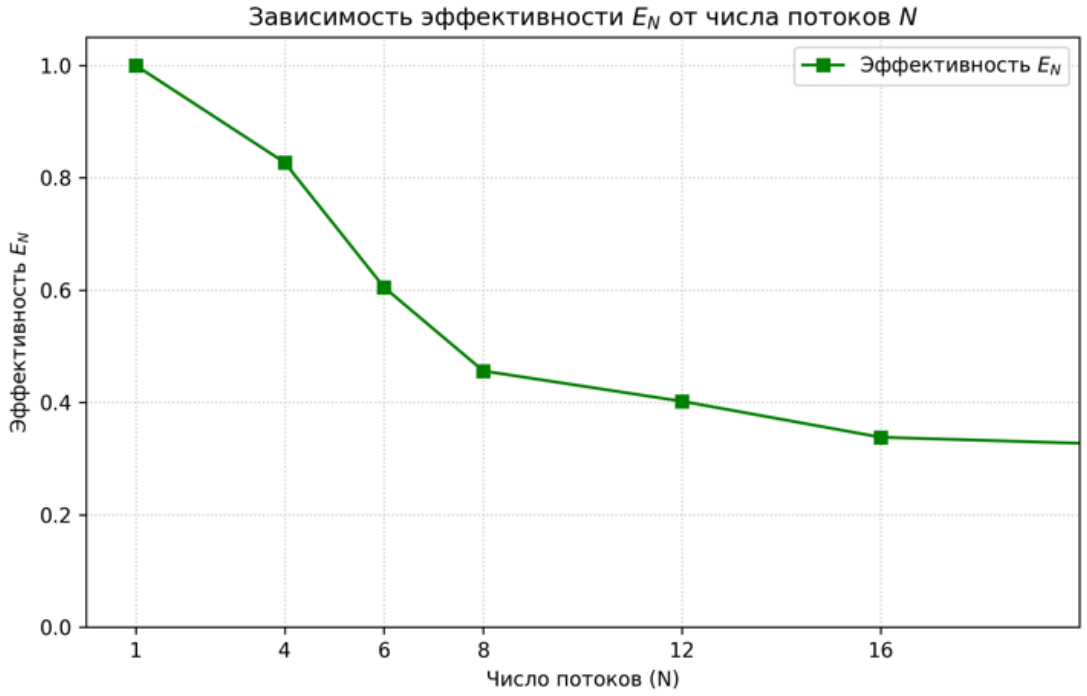
Число потоков	Время исполнения (мс)	Ускорение	Эффективность
1	240.669	1	1
4	72.729	3.309	0.827
6	66.341	3.628	0.605
8	65.940	3.650	0.456
12	49.907	4.822	0.402
16	44.467	5.412	0.338
128	46.833	5.139	0.040
1024	116.903	2.059	0.002

Ускорение показывает во сколько раз применение параллельного алгоритма уменьшает время решения задачи по сравнению с последовательным алгоритмом. Ускорение определяется величиной $S_N=T_1/T_N$, где T_1 - время выполнения на одном потоке, T_N - время выполнения на N потоках.

Эффективность - величина $E_N=S_N/N$, где S_N - ускорение, N - количество используемых потоков.



Из анализа графиков зависимости ускорения и эффективности от числа потоков видно, что параллельная реализация задачи поиска минимума и максимума в массиве демонстрирует хорошее масштабирование при увеличении числа потоков N до значения, равного числу логических ядер процессора (N = 16). На начальных этапах (N = 1–8) наблюдается резкий рост ускорения: при N = 4 оно достигает 3.31, а при N = 8 — 3.65, что свидетельствует об эффективном распараллеливании вычислений. Максимальное ускорение (5.41) достигнуто при N = 16, что соответствует числу логических ядер системы и подтверждает теоретические ожидания.



Высокая эффективность наблюдалась при числе потоков от 1 до 6 — в этой зоне каждый новый поток приносит значительную выгоду. До 12 потоков эффективность постепенно снижается — это связано с ростом накладных расходов на управление потоками и синхронизацию. Падение становится более заметным, когда потоки начинают конкурировать за одни и те же ядра. При N=16 эффективность падает до 0.338, каждый дополнительный поток тратит много времени на оверхед и синхронизацию.

Для данной программы оптимальное число потоков лежит в диапазоне N=4 или N=8, где сохраняется хорошая эффективность и высокое ускорение. Использование большего количества потоков неэффективно.

Графики наглядно демонстрируют закон Амдала: ускорение ограничено долей последовательного кода.

Код программы

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


sem_t thread_limit_sem;

typedef struct {
    const int *arr;
    size_t start_idx;
    size_t end_idx;
    int local_min;
    int local_max;
} ThreadArgs;

void *worker_thread(void *raw_args) {
    ThreadArgs *args = (ThreadArgs *)raw_args;

    if (sem_wait(&thread_limit_sem) != 0) {
        perror("Ошибка sem_wait в потоке");
        return NULL;
    }

    args->local_min = args->local_max = args->arr[args->start_idx];

    for (size_t i = args->start_idx + 1; i < args->end_idx; ++i) {
```

```
        if (args->arr[i] < args->local_min) args->local_min = args->arr[i];
        if (args->arr[i] > args->local_max) args->local_max = args->arr[i];
    }

    if (sem_post(&thread_limit_sem) != 0) {
        perror("Ошибка sem_post в потоке");
        return NULL;
    }

    return NULL;
}


int *generate_large_array(size_t n, size_t *size_out) {
    int *arr = malloc(n * sizeof(int));
    if (!arr) { *size_out = 0; return NULL; }
    int A = -1e9, B = 1e9;
    for (size_t i = 0; i < n; ++i) {
        arr[i] = A + rand() % (B - A + 1);
    }
    *size_out = n;
    return arr;
}

double sequential_version(const int *arr, size_t n, int *p_min, int *p_max) {
    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC, &start);

    *p_min = *p_max = arr[0];
```

```
    for (size_t i = 1; i < n; ++i) {
        if (arr[i] < *p_min) *p_min = arr[i];
        if (arr[i] > *p_max) *p_max = arr[i];
    }

    clock_gettime(CLOCK_MONOTONIC, &end);
    return (end.tv_sec - start.tv_sec) * 1000.0 +
        (end.tv_nsec - start.tv_nsec) / 1000000.0;
}

double parallel_version(const int* arr, size_t n, size_t K_threads, size_t L_limit, int*
p_min, int* p_max) {
    struct timespec start, end;

    if (sem_init(&thread_limit_sem, 0, (unsigned int)L_limit) != 0) {
        perror("Ошибка инициализации семафора");
        return -1.0;
    }

    size_t part_size = n / K_threads;

    pthread_t *threads = malloc(K_threads * sizeof(pthread_t));
    ThreadArgs *thread_args = malloc(K_threads * sizeof(ThreadArgs));

        if (!threads || !thread_args) {
        perror("Ошибка выделения памяти");
        sem_destroy(&thread_limit_sem);
        free(threads);
        free(thread_args);
        return -1.0;
```

```
}

clock_gettime(CLOCK_MONOTONIC, &start);

for (size_t i = 0; i < K_threads; ++i) {
    size_t start_idx = i * part_size;
    size_t end_idx = (i == K_threads - 1) ? n : (i + 1) * part_size;

    thread_args[i].arr = arr;
    thread_args[i].start_idx = start_idx;
    thread_args[i].end_idx = end_idx;

    if (pthread_create(&threads[i], NULL, worker_thread, &thread_args[i]) != 0) {
        perror("Ошибка создания потока.");
        // ждём уже запущенные потоки
        for (size_t j = 0; j < i; ++j) {
            pthread_join(threads[j], NULL);
        }

        sem_destroy(&thread_limit_sem);
        free(threads);
        free(thread_args);
        return -1.0;
    }
}

// ждём завершения всех потоков
for (size_t i = 0; i < K_threads; ++i) {
    if (pthread_join(threads[i], NULL) != 0) {
```

```
            perror("ошибка pthread_join");
        }
    }

    *p_min = thread_args[0].local_min;
    *p_max = thread_args[0].local_max;

    for (size_t i = 1; i < K_threads; ++i) {
        if (thread_args[i].local_min < *p_min) *p_min = thread_args[i].local_min;
        if (thread_args[i].local_max > *p_max) *p_max = thread_args[i].local_max;
    }

    clock_gettime(CLOCK_MONOTONIC, &end);

    sem_destroy(&thread_limit_sem);
    free(thread_args);
    free(threads);
    return (end.tv_sec - start.tv_sec) * 1000.0 +
           (end.tv_nsec - start.tv_nsec) / 1000000.0;
}


int main(int argc, char *argv[]) {
    if (argc != 4) {
        fprintf(stderr, "Использование: %s <размер_массива_N> <число_потоков_K>
<лимит_потоков_L>\n", argv[0]);
        return EXIT_FAILURE;
    }

    size_t N = atoi(argv[1]);
```

```
    size_t K_threads = atoi(argv[2]);
    size_t L_limit = atoi(argv[3]);

    if (N < 3 || K_threads == 0 || L_limit == 0 || L_limit > K_threads) {
        fprintf(stderr, "Ошибка: N>=3, K>0, L>0, L<=K\n");
        return EXIT_FAILURE;
    }



    size_t arr_size = 0;
    int *arr = generate_large_array(N, &arr_size);
    if (!arr || arr_size == 0) {
        fprintf(stderr, "Ошибка при создании массива чисел\n");
        return EXIT_FAILURE;
    }



    // Последовательная версия
    int min_seq, max_seq;
    double t_seq = sequential_version(arr, arr_size, &min_seq, &max_seq);
    if (t_seq < 0) {
        fprintf(stderr, "Ошибка в последовательной версии\n");
        free(arr);
        return EXIT_FAILURE;
    }


        // Параллельная версия
    int min_par, max_par;
    double t_par = parallel_version(arr, arr_size, K_threads, L_limit, &min_par,
&max_par);
```

```
    if (t_par < 0) {
        fprintf(stderr, "Ошибка в параллельной версии\n");
        free(arr);
        return EXIT_FAILURE;
    }



    printf("N=%zu\n", N);
    printf("K=%zu (всего потоков)\n", K_threads);
    printf("L=%zu (макс. одновременно работающих)\n", L_limit);
    printf("t_seq (последовательно): %.3f мс\n", t_seq);
    printf("t_par (параллельно):    %.3f мс\n", t_par);
    printf("Ускорение S = t_seq/t_par: %.3f\n", t_seq / t_par);
    printf("Эффективность E = S/K: %.3f\n", (t_seq / t_par) / K_threads);


    if (min_seq == min_par && max_seq == max_par) {
        printf("Найдено (min, max) = (%d, %d)\n", min_seq, max_seq);
    } else {
        printf("ОШИБКА: Sequential (%d, %d) != Parallel (%d, %d).\n", min_seq,
max_seq, min_par, max_par);
    }
    free(arr);
    return EXIT_SUCCESS;
}
```

Протокол работы программы

Тесты

```
maks-alex@DESKTOP-QFPFVP1:~/test$ ./lab2 100000000 4 4
N=100000000
K=4 (всего потоков)
L=4 (макс. одновременно работающих)
t_seq (последовательно): 197.662 мс
t_par (параллельно):    70.026 мс
Ускорение S = t_seq/t_par: 2.823
Эффективность E = S/K: 0.706
Найдено (min, max) = (-999999993, 999999981)
maks-alex@DESKTOP-QFPFVP1:~/test$ ./lab2 100000000 8 8
N=100000000
K=8 (всего потоков)
L=8 (макс. одновременно работающих)
t_seq (последовательно): 200.133 мс
t_par (параллельно):    55.240 мс
Ускорение S = t_seq/t_par: 3.623
Эффективность E = S/K: 0.453
Найдено (min, max) = (-999999993, 999999981)
```

```
maks-alex@DESKTOP-QFPFVP1:~/test$ ./lab2 100000000 16 16
N=100000000
K=16 (всего потоков)
L=16 (макс. одновременно работающих)
t_seq (последовательно): 198.226 мс
t_par (параллельно):    41.012 мс
Ускорение S = t_seq/t_par: 4.833
Эффективность E = S/K: 0.302
Найдено (min, max) = (-999999993, 999999981)
maks-alex@DESKTOP-QFPFVP1:~/test$ ./lab2 100000000 128 16
N=100000000
K=128 (всего потоков)
L=16 (макс. одновременно работающих)
t_seq (последовательно): 199.437 мс
t_par (параллельно):    47.135 мс
Ускорение S = t_seq/t_par: 4.231
Эффективность E = S/K: 0.033
Найдено (min, max) = (-999999993, 999999981)
```

Strace:

execve("./lab2", ["./lab2", "1000000", "4", "4"], 0x7ffee92c5670 /* 28 vars */) = 0

brk(NULL) = 0x564e96311000

mmap(NULL, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f1fe4af1000

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=20003, ...}) = 0

mmap(NULL, 20003, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f1fe4aec000

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"..., 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"..., 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"..., 784, 64) =
784

mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f1fe48da000

mmap(0x7f1fe4902000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f1fe4902000

mmap(0x7f1fe4a8a000, 323584, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7f1fe4a8a000

mmap(0x7f1fe4ad9000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7f1fe4ad9000

mmap(0x7f1fe4adf000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f1fe4adf000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f1fe48d7000

arch_prctl(ARCH_SET_FS, 0x7f1fe48d7740) = 0

set_tid_address(0x7f1fe48d7a10)   = 81051

set_robust_list(0x7f1fe48d7a20, 24)   = 0

rseq(0x7f1fe48d8060, 0x20, 0, 0x53053053) = 0

mprotect(0x7f1fe4ad9000, 16384, PROT_READ) = 0

mprotect(0x564e91ae0000, 4096, PROT_READ) = 0

mprotect(0x7f1fe4b29000, 8192, PROT_READ) = 0

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

munmap(0x7f1fe4aec000, 20003)   = 0

getrandom("\xd9\xcc\x97\x41\xad\xb9\xf1\xc3", 8, GRND_NONBLOCK) = 8

brk(NULL)   = 0x564e96311000

brk(0x564e96332000)   = 0x564e96332000

mmap(NULL, 4001792, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f1fe4506000

rt_sigaction(SIGRT_1, {sa_handler=0x7f1fe4973530, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO,
sa_restorer=0x7f1fe491f330}, 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) = 0x7f1fe3d05000

mprotect(0x7f1fe3d06000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

**clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLO
NE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|
CLONE_CHILD_CLEARTID, child_tid=0x7f1fe4505990,
parent_tid=0x7f1fe4505990, exit_signal=0, stack=0x7f1fe3d05000,
stack_size=0x7fff80, tls=0x7f1fe45056c0}strace: Process 81052 attached**

=> {parent_tid=[81052]}, 88) = 81052

[pid 81052] rseq(0x7f1fe4505fe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 81051] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 81052] <... rseq resumed>) = 0

[pid 81051] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 81052] set_robust_list(0x7f1fe45059a0, 24 <unfinished ...>

[pid 81051] mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>

[pid 81052] <... set_robust_list resumed>) = 0

[pid 81051] <... mmap resumed>) = 0x7f1fe3504000

[pid 81052] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 81051] mprotect(0x7f1fe3505000, 8388608, PROT_READ|PROT_WRITE
<unfinished ...>

[pid 81052] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 81051] <... mprotect resumed>) = 0

[pid 81051] rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

**[pid 81051]
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLO
NE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|
CLONE_CHILD_CLEARTID, child_tid=0x7f1fe3d04990,
parent_tid=0x7f1fe3d04990, exit_signal=0, stack=0x7f1fe3504000,
stack_size=0x7fff80, tls=0x7f1fe3d046c0}strace: Process 81053 attached**

=> {parent_tid=[81053]}, 88) = 81053

[pid 81053] rseq(0x7f1fe3d04fe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 81051] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 81053] <... rseq resumed>) = 0

[pid 81052] rt_sigprocmask(SIG_BLOCK, ~[RT_1], <unfinished ...>

[pid 81051] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 81053] set_robust_list(0x7f1fe3d049a0, 24 <unfinished ...>

[pid 81051] mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>

[pid 81052] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 81051] <... mmap resumed>) = 0x7f1fe2d03000

[pid 81053] <... set_robust_list resumed>) = 0

[pid 81051] mprotect(0x7f1fe2d04000, 8388608, PROT_READ|PROT_WRITE
<unfinished ...>

[pid 81052] madvise(0x7f1fe3d05000, 8368128, MADV_DONTNEED <unfinished ...>

[pid 81051] <... mprotect resumed>) = 0

[pid 81053] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 81051] rt_sigprocmask(SIG_BLOCK, ~[], <unfinished ...>

[pid 81052] <... madvise resumed>) = 0

[pid 81051] <... rt_sigprocmask resumed>[], 8) = 0

[pid 81053] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 81051]
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLO
NE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|
CLONE_CHILD_CLEARTID, child_tid=0x7f1fe3503990,
parent_tid=0x7f1fe3503990, exit_signal=0, stack=0x7f1fe2d03000,
stack_size=0x7fff80, tls=0x7f1fe35036c0} <unfinished ...>

[pid 81052] exit(0strace: Process 81054 attached

) = ?

[pid 81051] <... clone3 resumed> => {parent_tid=[81054]}, 88) = 81054

[pid 81054] rseq(0x7f1fe3503fe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 81051] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 81052] +++ exited with 0 +++

[pid 81054] <... rseq resumed>) = 0

[pid 81051] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 81054] set_robust_list(0x7f1fe35039a0, 24 <unfinished ...>

[pid 81051] mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>

[pid 81054] <... set_robust_list resumed>) = 0

[pid 81051] <... mmap resumed>) = 0x7f1fe2502000

[pid 81053] rt_sigprocmask(SIG_BLOCK, ~[RT_1], <unfinished ...>

[pid 81051] mprotect(0x7f1fe2503000, 8388608, PROT_READ|PROT_WRITE
<unfinished ...>

[pid 81054] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 81051] <... mprotect resumed>) = 0

[pid 81053] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 81051] rt_sigprocmask(SIG_BLOCK, ~[], <unfinished ...>

[pid 81054] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 81051] <... rt_sigprocmask resumed>[], 8) = 0

[pid 81053] madvise(0x7f1fe3504000, 8368128, MADV_DONTNEED <unfinished ...>

**[pid 81051]
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLO
NE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|
CLONE_CHILD_CLEARTID, child_tid=0x7f1fe2d02990,
parent_tid=0x7f1fe2d02990, exit_signal=0, stack=0x7f1fe2502000,
stack_size=0x7fff80, tls=0x7f1fe2d026c0} <unfinished ...>**

[pid 81053] <... madvise resumed>) = 0

[pid 81053] exit(0strace: Process 81055 attached

) = ?

[pid 81051] <... clone3 resumed> => {parent_tid=[81055]}, 88) = 81055

[pid 81055] rseq(0x7f1fe2d02fe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 81051] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 81053] +++ exited with 0 +++

[pid 81051] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 81055] <... rseq resumed>) = 0

[pid 81054] rt_sigprocmask(SIG_BLOCK, ~[RT_1], <unfinished ...>

[pid 81051] futex(0x7f1fe3503990,
FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 81054, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 81055] set_robust_list(0x7f1fe2d029a0, 24 <unfinished ...>

[pid 81054] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 81055] <... set_robust_list resumed>) = 0

[pid 81054] madvise(0x7f1fe2d03000, 8368128, MADV_DONTNEED <unfinished ...>

[pid 81055] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 81054] <... madvise resumed>)   = 0

[pid 81055] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 81054] exit(0)   = ?

[pid 81051] <... futex resumed>)   = 0

[pid 81054] +++ exited with 0 +++

[pid 81051] futex(0x7f1fe2d02990,
FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 81055, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 81055] rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0

[pid 81055] madvise(0x7f1fe2502000, 8368128, MADV_DONTNEED) = 0

[pid 81055] exit(0)   = ?

[pid 81051] <... futex resumed>)   = 0

[pid 81055] +++ exited with 0 +++

fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x2), ...}) = 0

write(1, "N=1000000\n", 10N=1000000

)   = 10

write(1, "K=4 (\320\262\321\201\320\265\320\263\320\276
\320\277\320\276\321\202\320\276\320\272\320\276\320\262)\n", 32K=4 (всего потоков)

) = 32

write(1, "L=4 (\320\274\320\260\320\272\321\201.
\320\276\320\264\320\275\320\276\320\262\321\200\320\265\320\274\320"..., 62L=4 (макс. одновременно работающих)

) = 62

write(1, "t_seq
(\320\277\320\276\321\201\320\273\320\265\320\264\320\276\320\262\320\260\321\202
\320\265\320\273\321"..., 51t_seq (последовательно): 5.888 мс

) = 51

write(1, "t_par
(\320\277\320\260\321\200\320\260\320\273\320\273\320\265\320\273\321\214\320\275
\320\276): "..., 47t_par (параллельно):   23.427 мс

) = 47

write(1, "\320\243\321\201\320\272\320\276\321\200\320\265\320\275\320\270\320\265
S = t_seq/t_p"..., 42Ускорение S = t_seq/t_par: 0.251

) = 42

write(1,
"\320\255\321\204\321\204\320\265\320\272\321\202\320\270\320\262\320\275\320\276
\321\201\321\202\321\214 E = S"..., 42Эффективность E = S/K: 0.063

) = 42

write(1, "\320\235\320\260\320\271\320\264\320\265\320\275\320\276 (min, max) = (-
99"..., 52Найдено (min, max) = (-999999421, 999999981)

) = 52

munmap(0x7f1fe4506000, 4001792)   = 0

exit_group(0)   = ?

+++ exited with 0 +++

Вывод

В ходе выполнения лабораторной работы были успешно изучены и применены основные системные вызовы для работы с потоками в ОС Linux. Была реализована программа, демонстрирующая работу поиска максимального и минимального элементов в целочисленном массиве.