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: Шитов Н.В.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 12.11.25

Москва, 2025

Постановка задачи

Вариант 18.

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 образец в строке наивным алгоритмом.

Общий метод и алгоритм решения

Общий метод решения:

Для поиска всех вхождений заданного образца в большом тексте применяется наивный (brute-force) алгоритм, распараллеленный с помощью POSIX-потоков (pthread). Текст разбивается на непересекающиеся (с перекрытием на длину образца) диапазоны, каждый из которых обрабатывается отдельным потоком.
Для предотвращения перегрузки системы введено ограничение на число одновременно выполняющихся потоков, реализованное с помощью семафора POSIX. Производительность оценивается через измерение времени выполнения и расчёт ускорения по сравнению с последовательной версией.

Алгоритм работы программы

1. Генерация данных:
   Создаётся длинная строка (≈9 МБ) путём повторения фиксированного фрагмента ("AGCTagctGCTGT"), имитирующая биологическую последовательность.

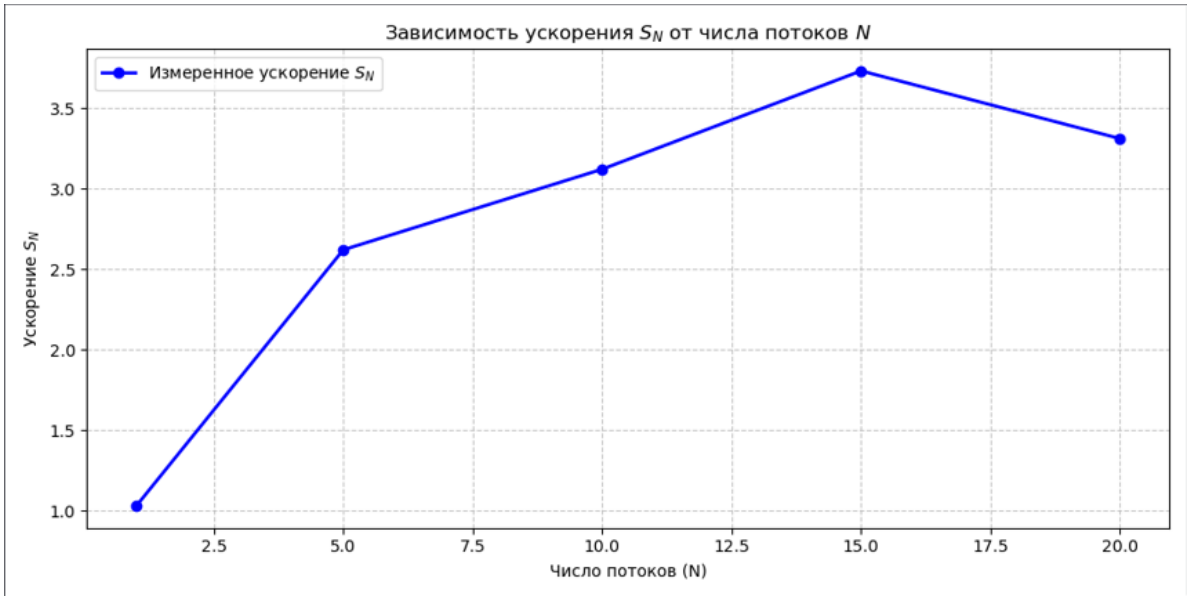
2. Последовательный поиск (базовый случай):
    - Производится полный перебор всех позиций в тексте.
    - Для каждой позиции выполняется посимвольное сравнение с образцом.
    - Фиксируется время выполнения (T_1) и количество вхождений.
3. Параллельный поиск:
    - Текст делится на N частей по длине.
    - Для каждой части создаётся поток, выполняющий наивный поиск в своём диапазоне (с перекрытием на len(pattern) − 1, чтобы не пропустить совпадения на границах).
    - Перед началом работы каждый поток ожидает разрешения семафора (sem_wait), инициализированного значением L (макс. число одновременных потоков).
    - После завершения поиска поток освобождает семафор (sem_post).
    - Главный поток собирает и суммирует результаты всех рабочих потоков.
    - Фиксируется время выполнения (T_N).
4. Оценка эффективности:
    - Вычисляется ускорение: $S_N=T_1/T_N$
    - Проверяется корректность: сравнение count_seq == count_par.
    - Анализируется зависимость S(N, L) при фиксированном тексте и образце.

Анализ ускорения и эффективности:

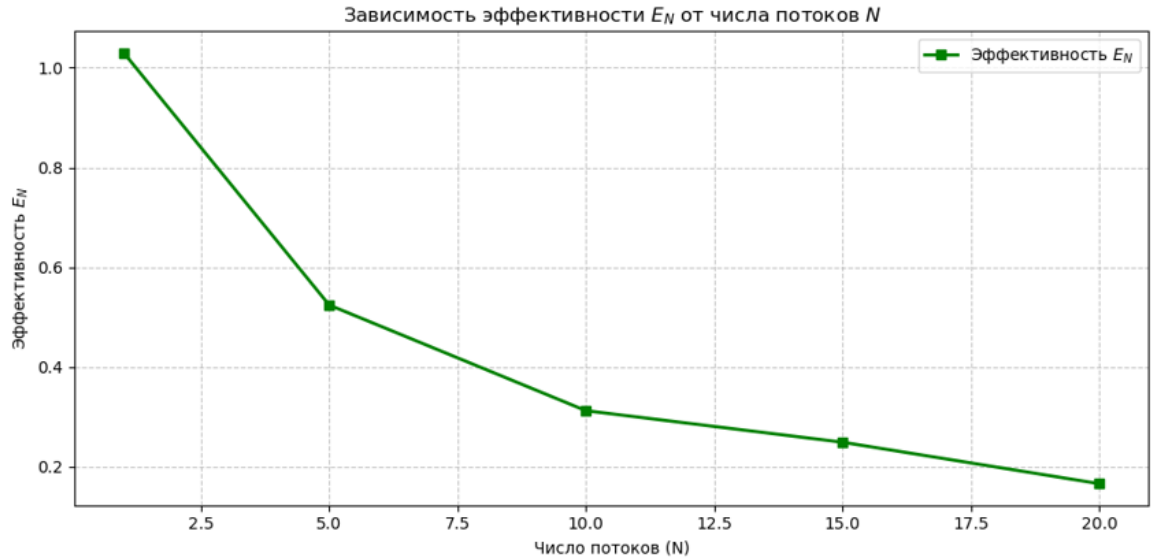
Число потоков	**Время исполнения (мс)**	**Ускорение**	**Эффективность**
1	34.7795	1.03	1.03
5	13.5627	2.62	0.524
10	11.4549	3.12	0.312
15	9.6451	3.73	0.249
20	10.7387	3.31	0.166
100	10.2834	3.46	0.035
500	15.1247	2.35	0.005

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 графика видно, что Рост до максимума (при N=15) программа эффективно использует ресурсы CPU. Падение после N=15 накладные расходы на создание/переключение потоков начинают перевешивать пользу от параллелизма. Максимум 3.73 система не может эффективно обрабатывать больше 15 потоков одновременно.



Быстрое падение с 1.0 до 0.5 при N=5 уже половина потоков простаивает или тратит время на ожидание. Дальше медленное снижение чем больше потоков, тем меньше пользы от каждого. При N=20 только 12% времени каждый поток работает полезно — остальное уходит на управление.

Ускорение растёт с увеличением числа потоков до определённого предела, после чего начинает снижаться из-за накладных расходов. Эффективность же монотонно падает, что свидетельствует о том, что при большом количестве потоков значительная часть вычислительных ресурсов тратится не на поиск, а на управление параллелизмом. Оптимальное число потоков для данной задачи около 15, что соответствует числу логических ядер в системе.

Код программы

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


static sem_t active_thread_cap;

static char* make_genome(size_t repeats, size_t *out_len) {
    const char motif[] = "AGCTagctGCTGT";
    const size_t motif_len = sizeof(motif) - 1;
```

```c
    const size_t total = motif_len * repeats;

    char *buf = malloc(total + 1);
    if (!buf) return NULL;

    for (size_t i = 0; i < repeats; ++i) {
        memcpy(buf + i * motif_len, motif, motif_len);
    }
    buf[total] = '\0';
    *out_len = total;
    return buf;
}


static size_t count_matches(const char *text, const char *pattern,
                            size_t text_start, size_t text_end) {
    const size_t pat_len = strlen(pattern);
    if (pat_len == 0 || text_end <= text_start) return 0;

    size_t hits = 0;
    const size_t last_start = (text_end >= pat_len) ? (text_end - pat_len) : 0;

    for (size_t i = text_start; i <= last_start; ++i) {
        if (memcmp(&text[i], pattern, pat_len) == 0) {
            hits++;
        }
    }
    return hits;
}

typedef struct {
    const char *text;
    const char *pattern;
    size_t from;
    size_t to;
} task_t;

static void* thread_worker(void *arg) {
    task_t *job = (task_t*)arg;

    if (sem_wait(&active_thread_cap) != 0) {
        perror("sem_wait");
        return NULL;
    }

    size_t *result = malloc(sizeof(size_t));
    if (!result) {
        perror("malloc result");
        sem_post(&active_thread_cap);
        return NULL;
    }

    *result = count_matches(job->text, job->pattern, job->from, job->to);
```

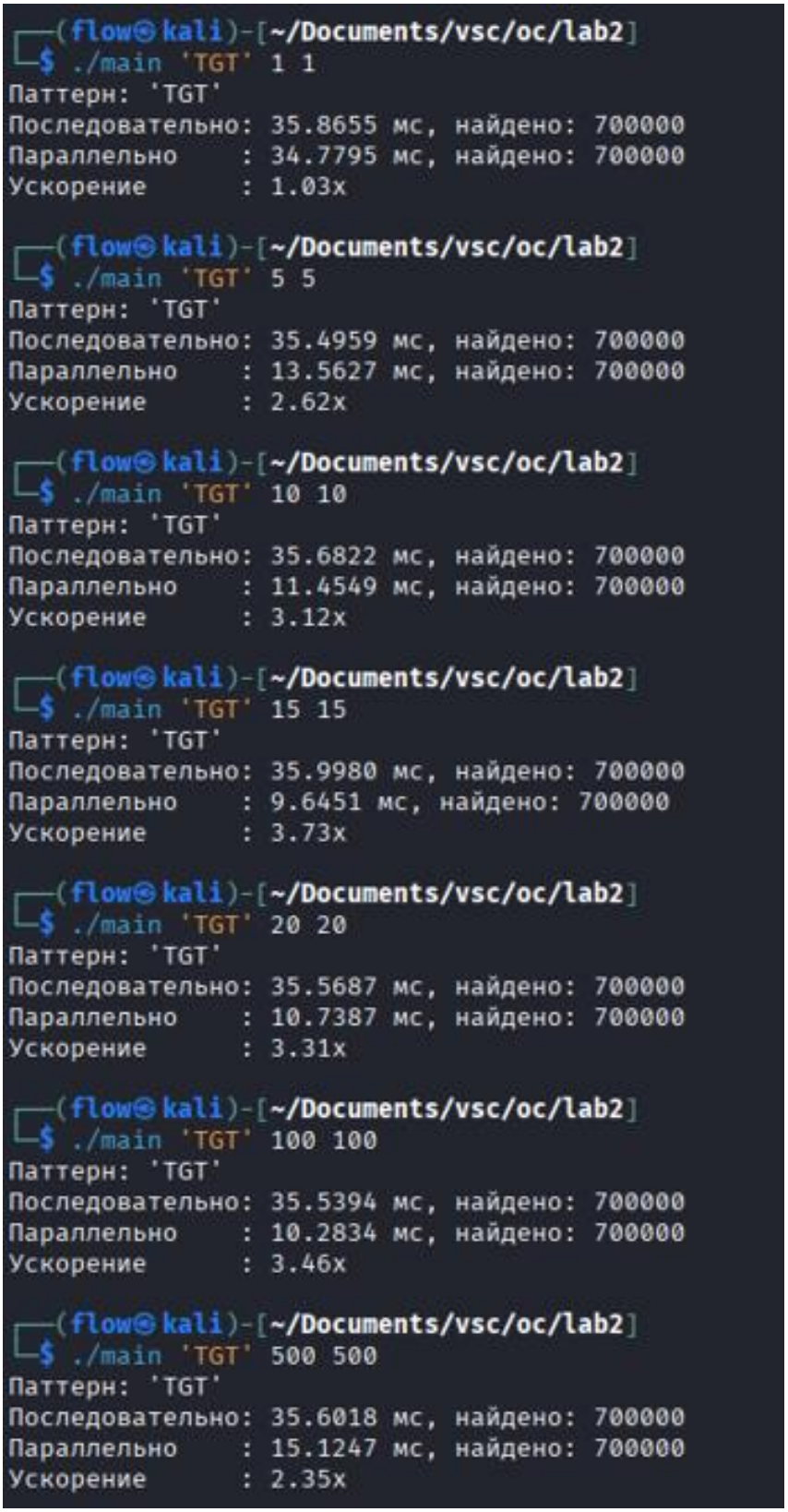
```
    if (sem_post(&active_thread_cap) != 0) {
        perror("sem_post");
    }

    return result;
}

static double run_parallel(const char *text, const char *pattern,
                           size_t num_threads, size_t max_active,
                           size_t *out_total) {
    const size_t txt_len = strlen(text);
    const size_t pat_len = strlen(pattern);

    const unsigned int sem_init_val = (max_active == 0) ? num_threads : (unsigned
int)max_active;
    if (sem_init(&active_thread_cap, 0, sem_init_val) != 0) {
        perror("sem_init");
        return -1.0;
    }

    pthread_t *pool = calloc(num_threads, sizeof(pthread_t));
    task_t *jobs = calloc(num_threads, sizeof(task_t));
    if (!pool || !jobs) {
        perror("calloc");
        sem_destroy(&active_thread_cap);
        free(pool);
        free(jobs);
        return -1.0;
    }

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

    const size_t base_chunk = txt_len / num_threads;
    for (size_t i = 0; i < num_threads; ++i) {
        jobs[i].text = text;
        jobs[i].pattern = pattern;
        jobs[i].from = i * base_chunk;

        if (i == num_threads - 1) {
            jobs[i].to = txt_len;
        } else {
            jobs[i].to = (i + 1) * base_chunk + pat_len - 1;
            if (jobs[i].to > txt_len) jobs[i].to = txt_len;
        }

        if (pthread_create(&pool[i], NULL, thread_worker, &jobs[i]) != 0) {
            perror("pthread_create");
            for (size_t j = 0; j < i; ++j) {
                void *res;
                pthread_join(pool[j], &res);
                free(res);
            }
            sem_destroy(&active_thread_cap);
```

```
            free(pool);
            free(jobs);
            return -1.0;
        }
    }

    *out_total = 0;
    for (size_t i = 0; i < num_threads; ++i) {
        void *res;
        if (pthread_join(pool[i], &res) == 0 && res) {
            *out_total += *(size_t*)res;
            free(res);
        }
    }

    clock_gettime(CLOCK_MONOTONIC, &t1);
    sem_destroy(&active_thread_cap);
    free(pool);
    free(jobs);

    double ms = (t1.tv_sec - t0.tv_sec) * 1000.0;
    ms += (t1.tv_nsec - t0.tv_nsec) / 1e6;
    return ms;
}

static double run_sequential(const char *text, const char *pattern, size_t *out_count) {
    struct timespec t0, t1;
    clock_gettime(CLOCK_MONOTONIC, &t0);
    *out_count = count_matches(text, pattern, 0, strlen(text));
    clock_gettime(CLOCK_MONOTONIC, &t1);

    double ms = (t1.tv_sec - t0.tv_sec) * 1000.0;
    ms += (t1.tv_nsec - t0.tv_nsec) / 1e6;
    return ms;
}

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

    const char *pattern = argv[1];
    const size_t N = strtoul(argv[2], NULL, 10);
    const size_t L = strtoul(argv[3], NULL, 10);

    if (N == 0 || strlen(pattern) == 0) {
        fprintf(stderr, "Ошибка: N > 0, паттерн не пуст.\n");
        return EXIT_FAILURE;
    }

    size_t genome_len = 0;
    char *dna = make_genome(700000, &genome_len);
```

```
    if (!dna) {
        fprintf(stderr, "Не удалось выделить память под геном.\n");
        return EXIT_FAILURE;
    }

    size_t seq_hits = 0;
    double t_seq = run_sequential(dna, pattern, &seq_hits);
    if (t_seq < 0) {
        free(dna);
        return EXIT_FAILURE;
    }

    size_t par_hits = 0;
    double t_par = run_parallel(dna, pattern, N, L, &par_hits);
    if (t_par < 0) {
        fprintf(stderr, "Ошибка в параллельной версии.\n");
        free(dna);
        return EXIT_FAILURE;
    }

    printf("Паттерн: '%s'\n", pattern);
    printf("Последовательно: %.4f мс, найдено: %zu\n", t_seq, seq_hits);
    printf("Параллельно    : %.4f мс, найдено: %zu\n", t_par, par_hits);
    printf("Ускорение      : %.2fx\n", t_seq / t_par);

    if (seq_hits != par_hits) {
        printf("результаты не совпадают!\n");
    }

    free(dna);
    return EXIT_SUCCESS;
}
```

Протокол работы программы

Тесты



Strace:

8561 execve("./main", ["./main", "TGT", "5", "5"], 0x7fffa5a87940 /* 55 vars */) = 0

8561 brk(NULL) = 0x55ee6c408000

8561 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fa6b7697000

8561 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или каталога)

8561 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

8561 fstat(3, {st_mode=S_IFREG|0644, st_size=95907, ...}) = 0

8561 mmap(NULL, 95907, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fa6b767f000

8561 close(3) = 0

8561 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

8561 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\0"..., 832) = 832

8561 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

8561 fstat(3, {st_mode=S_IFREG|0755, st_size=2003408, ...}) = 0

8561 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

8561 mmap(NULL, 2055800, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fa6b7489000

8561 mmap(0x7fa6b74b1000, 1462272, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7fa6b74b1000

8561 mmap(0x7fa6b7616000, 352256, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x18d000) = 0x7fa6b7616000

8561 mmap(0x7fa6b766c000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e2000) = 0x7fa6b766c000

8561 mmap(0x7fa6b7672000, 52856, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fa6b7672000

8561 close(3) = 0

8561 mmap(NULL, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fa6b7486000

8561 arch_prctl(ARCH_SET_FS, 0x7fa6b7486740) = 0

8561 set_tid_address(0x7fa6b7486a10) = 8561

8561 set_robust_list(0x7fa6b7486a20, 24) = 0

8561 rseq(0x7fa6b7486680, 0x20, 0, 0x53053053) = 0

8561 mprotect(0x7fa6b766c000, 16384, PROT_READ) = 0

8561 mprotect(0x55ee64617000, 4096, PROT_READ) = 0

8561 mprotect(0x7fa6b76d5000, 8192, PROT_READ) = 0

8561 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0

8561 munmap(0x7fa6b767f000, 95907) = 0

8561 getrandom("\xdd\xf3\x07\x72\xd1\x12\xf2\xab", 8, GRND_NONBLOCK) = 8

8561 brk(NULL) = 0x55ee6c408000

8561 brk(0x55ee6c429000) = 0x55ee6c429000

8561 mmap(NULL, 9101312, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fa6b6bd8000

8561 rt_sigaction(SIGRT_1, {sa_handler=0x7fa6b75194b0, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO,
sa_restorer=0x7fa6b74c8df0}, NULL, 8) = 0

8561 rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0

8561 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7fa6b63d7000

8561 mprotect(0x7fa6b63d8000, 8388608, PROT_READ|PROT_WRITE) = 0

8561 rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

8561
**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=0x7fa6b6bd7990,
parent_tid=0x7fa6b6bd7990, exit_signal=0, stack=0x7fa6b63d7000,
stack_size=0x7fff80, tls=0x7fa6b6bd76c0} => {parent_tid=[8562]}, 88) = 8562**

8561 rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

8561 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7fa6b5bd6000

8561 mprotect(0x7fa6b5bd7000, 8388608, PROT_READ|PROT_WRITE <unfinished
...>

8562 rseq(0x7fa6b6bd7600, 0x20, 0, 0x53053053 <unfinished ...>

8561 <... mprotect resumed>) = 0

8561 rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

8561
**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=0x7fa6b63d6990,
parent_tid=0x7fa6b63d6990, exit_signal=0, stack=0x7fa6b5bd6000,
stack_size=0x7fff80, tls=0x7fa6b63d66c0} <unfinished ...>**

8562 <... rseq resumed>) = 0

8561 <... clone3 resumed> => {parent_tid=[8563]}, 88) = 8563

8562 set_robust_list(0x7fa6b6bd79a0, 24 <unfinished ...>

8561 rt_sigprocmask(SIG_SETMASK, [] <unfinished ...>

8562 <... set_robust_list resumed>) = 0

8561 <... rt_sigprocmask resumed>, NULL, 8) = 0

8562 rt_sigprocmask(SIG_SETMASK, [] <unfinished ...>

8561 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>

8562 <... rt_sigprocmask resumed>, NULL, 8) = 0

8561 <... mmap resumed>) = 0x7fa6b53d5000

8563 rseq(0x7fa6b63d6600, 0x20, 0, 0x53053053 <unfinished ...>

8561 mprotect(0x7fa6b53d6000, 8388608, PROT_READ|PROT_WRITE <unfinished
...>

8562 mmap(NULL, 134217728, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>

8563 <... rseq resumed>) = 0

8561 <... mprotect resumed>) = 0

8561 rt_sigprocmask(SIG_BLOCK, ~[] <unfinished ...>

8562 <... mmap resumed>) = 0x7fa6ad200000

8561 <... rt_sigprocmask resumed>, [], 8) = 0

8562 munmap(0x7fa6ad200000, 48234496 <unfinished ...>

8561
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7fa6b5bd5990, parent_tid=0x7fa6b5bd5990, exit_signal=0, stack=0x7fa6b53d5000, stack_size=0x7fff80, tls=0x7fa6b5bd56c0} <unfinished ...>

8562 <... munmap resumed>) = 0

8562 munmap(0x7fa6b4000000, 18874368 <unfinished ...>

8563 set_robust_list(0x7fa6b63d69a0, 24) = 0

8561 <... clone3 resumed> => {parent_tid=[8564]}, 88) = 8564

8562 <... munmap resumed>) = 0

8564 rseq(0x7fa6b5bd5600, 0x20, 0, 0x53053053 <unfinished ...>

8563 rt_sigprocmask(SIG_SETMASK, [] <unfinished ...>

8561 rt_sigprocmask(SIG_SETMASK, [] <unfinished ...>

8564 <... rseq resumed>) = 0

8562 mprotect(0x7fa6b0000000, 135168, PROT_READ|PROT_WRITE <unfinished ...>

8561 <... rt_sigprocmask resumed>, NULL, 8) = 0

8564 set_robust_list(0x7fa6b5bd59a0, 24 <unfinished ...>

8563 <... rt_sigprocmask resumed>, NULL, 8) = 0

8561 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>

8564 <... set_robust_list resumed>) = 0

8562 <... mprotect resumed>) = 0

8564 rt_sigprocmask(SIG_SETMASK, [] <unfinished ...>

8563 mmap(NULL, 134217728, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>

8561 <... mmap resumed>) = 0x7fa6b4bd4000

8564 <... rt_sigprocmask resumed>, NULL, 8) = 0

8563 <... mmap resumed>) = 0x7fa6a8000000

8561 mprotect(0x7fa6b4bd5000, 8388608, PROT_READ|PROT_WRITE <unfinished
...>

8564 mmap(NULL, 134217728, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>

8563 munmap(0x7fa6ac000000, 67108864 <unfinished ...>

8561 <... mprotect resumed>) = 0

8564 <... mmap resumed>) = 0x7fa6a0000000

8563 <... munmap resumed>) = 0

8564 munmap(0x7fa6a4000000, 67108864 <unfinished ...>

8561 rt_sigprocmask(SIG_BLOCK, ~[] <unfinished ...>

8563 mprotect(0x7fa6a8000000, 135168, PROT_READ|PROT_WRITE <unfinished ...>

8561 <... rt_sigprocmask resumed>, [], 8) = 0

8564 <... munmap resumed>) = 0

8561
**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=0x7fa6b53d4990,
parent_tid=0x7fa6b53d4990, exit_signal=0, stack=0x7fa6b4bd4000,
stack_size=0x7fff80, tls=0x7fa6b53d46c0} <unfinished ...>**

8563 <... mprotect resumed>) = 0

8564 mprotect(0x7fa6a0000000, 135168, PROT_READ|PROT_WRITE) = 0

8561 <... clone3 resumed> => {parent_tid=[8565]}, 88) = 8565

8565 rseq(0x7fa6b53d4600, 0x20, 0, 0x53053053 <unfinished ...>

8561 rt_sigprocmask(SIG_SETMASK, [] <unfinished ...>

8565 <... rseq resumed>) = 0

8561 <... rt_sigprocmask resumed>, NULL, 8) = 0

8565 set_robust_list(0x7fa6b53d49a0, 24 <unfinished ...>

8561 mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>

8565 <... set_robust_list resumed>) = 0

8561 <... mmap resumed>) = 0x7fa6b43d3000

8565 rt_sigprocmask(SIG_SETMASK, [] <unfinished ...>

8561 mprotect(0x7fa6b43d4000, 8388608, PROT_READ|PROT_WRITE <unfinished
...>

8565 <... rt_sigprocmask resumed>, NULL, 8) = 0

8561 <... mprotect resumed>) = 0

8565 mmap(0x7fa6a4000000, 67108864, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>

8561 rt_sigprocmask(SIG_BLOCK, ~[] <unfinished ...>

8565 <... mmap resumed>) = 0x7fa6a4000000

8561 <... rt_sigprocmask resumed>, [], 8) = 0

8565 mprotect(0x7fa6a4000000, 135168, PROT_READ|PROT_WRITE <unfinished ...>

8561
**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=0x7fa6b4bd3990,
parent_tid=0x7fa6b4bd3990, exit_signal=0, stack=0x7fa6b43d3000,
stack_size=0x7fff80, tls=0x7fa6b4bd36c0} <unfinished ...>**

8565 <... mprotect resumed>) = 0

8561 <... clone3 resumed> => {parent_tid=[8566]}, 88) = 8566

8566 rseq(0x7fa6b4bd3600, 0x20, 0, 0x53053053 <unfinished ...>

8561 rt_sigprocmask(SIG_SETMASK, [] <unfinished ...>

8566 <... rseq resumed>) = 0

8561 <... rt_sigprocmask resumed>, NULL, 8) = 0

```
8566  set_robust_list(0x7fa6b4bd39a0, 24 <unfinished ...>
8561  futex(0x7fa6b6bd7990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME,
8562, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>
8566  <... set_robust_list resumed>)    = 0
8566  rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
8566  mmap(NULL, 134217728, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fa698000000
8566  munmap(0x7fa69c000000, 67108864)  = 0
8566  mprotect(0x7fa698000000, 135168, PROT_READ|PROT_WRITE) = 0
8565  rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0
8565  madvise(0x7fa6b4bd4000, 8368128, MADV_DONTNEED) = 0
8565  exit(0)                           = ?
8565  +++ exited with 0 +++
8563  rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0
8563  madvise(0x7fa6b5bd6000, 8368128, MADV_DONTNEED) = 0
8563  exit(0)                           = ?
8563  +++ exited with 0 +++
8562  rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0
8564  rt_sigprocmask(SIG_BLOCK, ~[RT_1] <unfinished ...>
8562  madvise(0x7fa6b63d7000, 8368128, MADV_DONTNEED <unfinished ...>
8564  <... rt_sigprocmask resumed>, NULL, 8) = 0
8562  <... madvise resumed>)            = 0
8564  madvise(0x7fa6b53d5000, 8368128, MADV_DONTNEED <unfinished ...>
8562  exit(0 <unfinished ...>
8564  <... madvise resumed>)            = 0
8562  <... exit resumed>)               = ?
8564  exit(0)                           = ?
8561  <... futex resumed>)              = 0
```

```
8562  +++ exited with 0 +++

8564  +++ exited with 0 +++

8561  futex(0x7fa6b4bd3990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME,
8566, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

8566  rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0

8566  madvise(0x7fa6b43d3000, 8368128, MADV_DONTNEED) = 0

8566  exit(0)                           = ?

8561  <... futex resumed>)              = 0

8566  +++ exited with 0 +++

8561  munmap(0x7fa6b63d7000, 8392704)   = 0

8561  fstat(1, {st_mode=S_IFCHR|0600, st_rdev=makedev(0x88, 0), ...}) = 0

8561  write(1, "\320\237\320\260\321\202\321\202\320\265\321\200\320\275: 'TGT'\n",
22) = 22

8561  write(1,
"\320\237\320\276\321\201\320\273\320\265\320\264\320\276\320\262\320\260\321\202
\320\265\320\273\321\214\320\275\320\276: "..., 69) = 69

8561  write(1,
"\320\237\320\260\321\200\320\260\320\273\320\273\320\265\320\273\321\214\320\275
\320\276    : 12.1"..., 65) = 65

8561  write(1,
"\320\243\321\201\320\272\320\276\321\200\320\265\320\275\320\270\320\265      :
2.80x\n", 32) = 32

8561  munmap(0x7fa6b6bd8000, 9101312)   = 0

8561  exit_group(0)                     = ?

8561  +++ 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 наивного алгоритма для поиска образца в строке.