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: Лоскутова А. Д.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 05.11.25

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

Использованные системные вызовы:

Для реализации многопоточности, синхронизации и измерения времени в программе используются функции стандартных библиотек Pthreads и Семафоров, которые обращаются к системным вызовам ядра Linux. Для управления временем используется функция *clock_gettime*. Управление потоками осуществляется с помощью двух основных функций. Вызов *pthread_create* создаёт новый поток выполнения (worker_thread_clean). Главный поток ожидает завершения всех рабочих потоков, используя функцию *pthread_join*, чтобы собрать результаты и произвести слияние локальных счетчиков. Функция *sem_init* инициализирует семафор *thread_limit_sem*. По завершении всех расчетов семафор уничтожается функцией *sem_destroy*.

Алгоритм решения:

Алгоритм, реализованный в функциях *sequential_version_clean* и *naive_search_range_clea*n, является наивным алгоритмом поиска подстроки.

1. Сравнение с началом: Алгоритм начинает с первой позиции в тексте.

2. Посимвольное сравнение: Он пытается сравнить каждый символ образца с соответствующим символом текста, начиная с текущей позиции.

3. Успех/Неудача:

   - Если найдено совпадение: Подсчитывается одно совпадение.
   - Если найдено несовпадение: Сравнение немедленно прекращается.

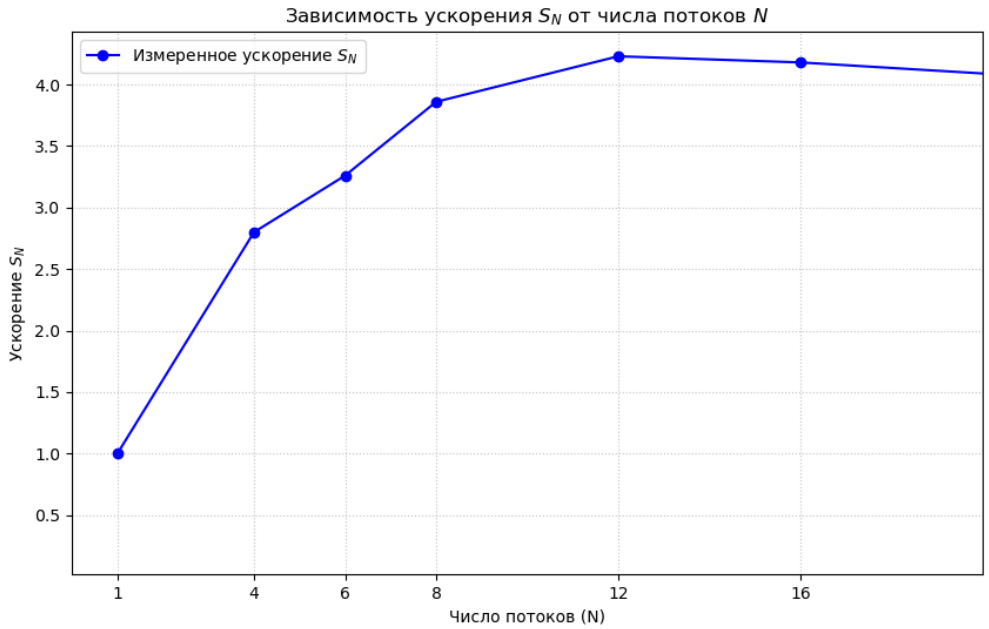
4. Сдвиг: Позиция в тексте сдвигается на один символ вперед, и сравнение начинается снова (для параллельной версии ваш код сдвигается на длину паттерна, что является оптимизацией).

Анализ ускорения и эффективности:

Число потоков	**Время исполнения (мс)**	**Ускорение**	**Эффективность**
1	15.26213	1	1
4	5.65632	2.8	0.7
6	5.17774	3.26	0.55
8	3.90801	3.86	0.52
12	3.72999	4.23	0.38
16	3.57168	4.18	0.27
128	9.95331	1.67	0.02
1024	72.64624	0.22	0.0003

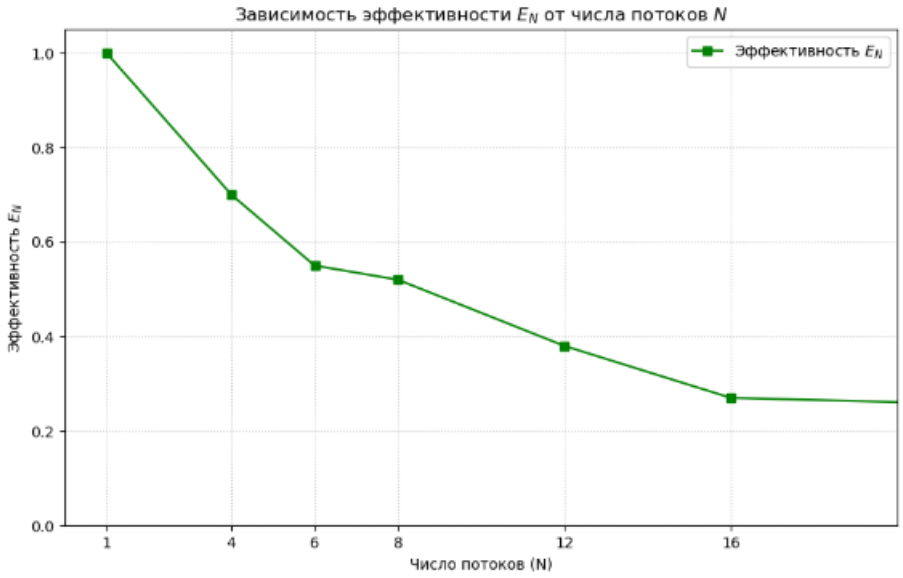
Ускорение показывает во сколько раз применение параллельного алгоритма уменьшает время решения задачи по сравнению с последовательным алгоритмом. Ускорение определяется величиной $S_N=T_1/T_N$, где T_1 - время выполнения на одном потоке, T_N - время выполнения на N потоках.

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



Из графика видно, что ускорение растет с увеличением количества потоков, демонстрируя распараллеливание задачи. На начальных этапах ускорение резко растет за счет эффективного задействования. После N=8 рост замедляется, при этом максимальное ускорение наблюдается при N=12, что соответствует числу логических ядер. При дальнейшем увеличении количества потоков ускорение начинает падать. Это происходит потому, что накладные расходы на управление лишними потоками (сверх 12 логических) превышают любую выгоду от компенсации задержек.

Параллельная программа эффективно масштабируется только до числа логических ядер. Это демонстрирует Закон Амдала: максимальное ускорение ограничено неизбежной долей последовательных операций.



Высокая эффективность на графике была, когда количество потоков было от 1 до 6, при этом постепенно снижаясь, в этой зоне новые потоки приносят выгоду. До 12 потоков эффективность продолжает снижаться, связано падение с накладными расходами на управление потоками и синхронизацию, которые становятся значительными. Падение становится более заметным, когда потоки начинают конкурировать за одни и те же ядра. При N=16 эффективность сильно падает, 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. Использование большего количества потоков неэффективно.

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

Код программы

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

static sem_t thread_limit_sem;

static char *generate_long_text(size_t multiplier, size_t *text_len_out) {
    const char *base_pattern = "AgctGCTGT";
    size_t base_len = strlen(base_pattern);
    size_t total_len = base_len * multiplier;
    char *text = malloc(total_len + 1);

    if (text == NULL) { *text_len_out = 0; return NULL; }

    for (size_t i = 0; i < multiplier; i++) {
        memcpy(text + (i * base_len), base_pattern, base_len);
    }

    text[total_len] = '\0';
    *text_len_out = total_len;
    return text;
```

```
}

typedef struct {
    const char *text;
    const char *pattern;
    size_t pattern_len;
    size_t start_idx;
    size_t end_idx;
} ThreadArgs;

static double sequential_version_clean(const char *text, const char *pattern, size_t
*final_count) {
    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC, &start);

    size_t text_len = strlen(text);
    size_t pattern_len = strlen(pattern);
    size_t count = 0;

    for (size_t i = 0; i < text_len; i++) {
        if (i + pattern_len > text_len) break;
        bool found = true;
        for (size_t j = 0; j < pattern_len; j++) {
            if (text[i + j] != pattern[j]) {
                found = false;
                break;
            }
        }
        if (found) {
            count++;
```

```
            i += pattern_len - 1;
        }
    }

    *final_count = count;

    clock_gettime(CLOCK_MONOTONIC, &end);

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

static size_t naive_search_range_clean(const char *text, size_t pattern_len,
                                       const char *pattern, size_t start_pos, size_t end_pos) {
    size_t local_count = 0;

    for (size_t i = start_pos; i < end_pos; i++) {
        if (i + pattern_len > end_pos) break;
        bool found = true;
        for (size_t j = 0; j < pattern_len; j++) {
            if (text[i + j] != pattern[j]) {
                found = false;
                break;
            }
        }
        if (found) {
            local_count++;
            i += pattern_len - 1;
        }
    }
```

```
    return local_count;
}

static void *worker_thread_clean(void *_args) {
    ThreadArgs *args = (ThreadArgs *)_args;

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

    size_t *result_count = malloc(sizeof(size_t));
    if (result_count == NULL) {
        perror("malloc result error в потоке");
        sem_post(&thread_limit_sem);
        return NULL;
    }

    *result_count = naive_search_range_clean(args->text, args->pattern_len, args->pattern,
                                             args->start_idx, args->end_idx);
    if (sem_post(&thread_limit_sem) != 0) {
        perror("Ошибка sem_post в потоке");
    }
    return (void *)result_count;
}

static double parallel_version_clean(const char *text, const char *pattern, size_t
K_threads, size_t L_limit, size_t *final_count) {
    struct timespec start, end;
```

```
    if (sem_init(&thread_limit_sem, 0, L_limit > 0 ? L_limit : 1) != 0) {
        perror("Ошибка инициализации семафора");
        return -1.0;
    }

    size_t text_len = strlen(text);
    size_t pattern_len = strlen(pattern);
    size_t chunk_size = text_len / 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 управляющей памяти");
        sem_destroy(&thread_limit_sem);
        if (threads) free(threads);
        if (thread_args) free(thread_args);
        return -1.0;
    }
    *final_count = 0;

    clock_gettime(CLOCK_MONOTONIC, &start);

    for (size_t i = 0; i < K_threads; i++) {
        size_t start_idx = i * chunk_size;
        size_t end_idx = (i == K_threads - 1) ? text_len : (i + 1) * chunk_size + pattern_len -
1;
        if (end_idx > text_len) end_idx = text_len;

        thread_args[i] = (ThreadArgs){
            .text = text, .pattern = pattern, .pattern_len = pattern_len,
```

```
            .start_idx = start_idx, .end_idx = end_idx
        };

        if (pthread_create(&threads[i], NULL, worker_thread_clean, &thread_args[i]) != 0)
{
            perror("Ошибка создания потока.");

            for (size_t j = 0; j < i; j++) {
                size_t *local_count_ptr;
                pthread_join(threads[j], (void**)&local_count_ptr);
                if (local_count_ptr) {
                    *final_count += *local_count_ptr;
                    free(local_count_ptr);
                }
            }
            sem_destroy(&thread_limit_sem);
            free(thread_args);
            free(threads);
            return -1.0;
        }
    }

    size_t *local_count_ptr;
    for (size_t i = 0; i < K_threads; i++) {
        if (pthread_join(threads[i], (void **)&local_count_ptr) != 0) {
            perror("Ошибка pthread_join");
        }

        if (local_count_ptr) {
            *final_count += *local_count_ptr;
```

```
            free(local_count_ptr);
        }
    }

    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 <образец> <число_потоков_N>
<лимит_потоков_L>\n", argv[0]);
        return EXIT_FAILURE;
    }
    size_t text_len = 0;
    char *text = generate_long_text(700000, &text_len);

    if (text == NULL || text_len == 0) return EXIT_FAILURE;

    char *pattern = argv[1];
    size_t K_threads = atoi(argv[2]);
    size_t L_limit = atoi(argv[3]);

    if (K_threads == 0 || L_limit == 0 || K_threads < L_limit || strlen(pattern) == 0) {
        fprintf(stderr, "Ошибка: N, L должны быть > 0, N >= L, и паттерн не может быть
пустым.\n");
        free(text);
        return EXIT_FAILURE;
```

```
    }

    size_t expected_count = 0;
    double seq_time_clean = sequential_version_clean(text, pattern, &expected_count);

    size_t par_count = 0;
    double par_time_clean = parallel_version_clean(text, pattern, K_threads, L_limit,
&par_count);

    if (par_time_clean < 0.0) {
        fprintf(stderr, "\n=== ОШИБКА ВЫПОЛНЕНИЯ ПАРАЛЛЕЛЬНОЙ ВЕРСИИ.
РЕСУРСЫ ОСВОБОЖДЕНЫ. ===\n");
        free(text);
        return EXIT_FAILURE;
    }

    printf("Образец: '%s'\n", pattern);
    printf("T1: %.5f мс\n", seq_time_clean);
    printf("Tn: %.5f мс\n", par_time_clean);
    if (expected_count == par_count) {
        printf("Найдено %zu\n", expected_count);
    } else {
        printf("ОШИБКА: Sequential %zu != Parallel %zu.\n", expected_count, par_count);
    }
    free(text);
    return EXIT_SUCCESS;
}
```

Протокол работы программы

Тесты

```
kriasatri@kriasa2006:/mnt/f/2_kurs/1_sem/os/lab_2/src$ gcc main.c -o main -pthread -lm
kriasatri@kriasa2006:/mnt/f/2_kurs/1_sem/os/lab_2/src$ ./main 'GCT' 1 1
Образец: 'GCT'
T1: 14.69495 мс
Tn: 14.64765 мс
Найдено 700000
kriasatri@kriasa2006:/mnt/f/2_kurs/1_sem/os/lab_2/src$ ./main 'GCT' 4 4
Образец: 'GCT'
T1: 15.72726 мс
Tn: 4.41058 мс
Найдено 700000
kriasatri@kriasa2006:/mnt/f/2_kurs/1_sem/os/lab_2/src$ ./main 'GCT' 6 6
Образец: 'GCT'
T1: 15.59199 мс
Tn: 5.14241 мс
Найдено 700000
kriasatri@kriasa2006:/mnt/f/2_kurs/1_sem/os/lab_2/src$ ./main 'GCT' 8 8
Образец: 'GCT'
T1: 15.58391 мс
Tn: 3.72978 мс
Найдено 700000
```

```
kriasatri@kriasa2006:/mnt/f/2_kurs/1_sem/os/lab_2/src$ ./main 'GCT' 12 12
Образец: 'GCT'
T1: 15.98291 мс
Tn: 4.06665 мс
Найдено 700000
kriasatri@kriasa2006:/mnt/f/2_kurs/1_sem/os/lab_2/src$ ./main 'GCT' 16 16
Образец: 'GCT'
T1: 15.33230 мс
Tn: 4.09546 мс
Найдено 700000
kriasatri@kriasa2006:/mnt/f/2_kurs/1_sem/os/lab_2/src$ ./main 'GCT' 128 128
Образец: 'GCT'
T1: 15.51498 мс
Tn: 10.32508 мс
Найдено 700000
kriasatri@kriasa2006:/mnt/f/2_kurs/1_sem/os/lab_2/src$ ./main 'GCT' 1024 1024
Образец: 'GCT'
T1: 15.48482 мс
Tn: 67.89452 мс
Найдено 700000
```

Strace:

kriasatri@kriasa2006:/mnt/f/2_kurs/1_sem/os/lab_2/src$ strace -f ./main 'GCT' 4 4

execve("./main", ["./main", "GCT", "4", "4"], 0x7ffd045f1c20 /* 35 vars */) = 0

brk(NULL) = 0x5dbae555f000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x77c43a4e1000

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=27015, ...}) = 0

mmap(NULL, 27015, PROT_READ, MAP_PRIVATE, 3, 0) = 0x77c43a4da000

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\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) = 0x77c43a200000

mmap(0x77c43a228000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x77c43a228000

mmap(0x77c43a3b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x77c43a3b0000

mmap(0x77c43a3ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x77c43a3ff000

```
mmap(0x77c43a405000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x77c43a405000

close(3)                        = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x77c43a4d7000

arch_prctl(ARCH_SET_FS, 0x77c43a4d7740) = 0

set_tid_address(0x77c43a4d7a10)         = 83583

set_robust_list(0x77c43a4d7a20, 24)     = 0

rseq(0x77c43a4d8060, 0x20, 0, 0x53053053) = 0

mprotect(0x77c43a3ff000, 16384, PROT_READ) = 0

mprotect(0x5dbae32c8000, 4096, PROT_READ) = 0

mprotect(0x77c43a519000, 8192, PROT_READ) = 0

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

munmap(0x77c43a4da000, 27015)           = 0

getrandom("\xbf\x12\x84\x7a\x85\x33\xdd\xeb", 8, GRND_NONBLOCK) = 8

brk(NULL)                               = 0x5dbae555f000

brk(0x5dbae5580000)                     = 0x5dbae5580000

mmap(NULL, 6303744, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x77c439bfd000

rt_sigaction(SIGRT_1, {sa_handler=0x77c43a299530, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO,
sa_restorer=0x77c43a245330}, 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) = 0x77c4393fc000

mprotect(0x77c4393fd000, 8388608, PROT_READ|PROT_WRITE) = 0

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

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x77c439bfc990, parent_tid=0x77c439bfc990, exit_signal=0, stack=0x77c4393fc000, stack_size=0x7fff80, tls=0x77c439bfc6c0}strace: Process 83584 attached

=> {parent_tid=[83584]}, 88) = 83584

[pid 83584] rseq(0x77c439bfcfe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 83583] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 83584] <... rseq resumed>) = 0

[pid 83583] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 83584] set_robust_list(0x77c439bfc9a0, 24 <unfinished ...>

[pid 83583] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>

[pid 83584] <... set_robust_list resumed>) = 0

[pid 83583] <... mmap resumed>) = 0x77c438bfb000

[pid 83584] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 83583] mprotect(0x77c438bfc000, 8388608, PROT_READ|PROT_WRITE <unfinished ...>

[pid 83584] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 83583] <... mprotect resumed>) = 0

[pid 83584] mmap(NULL, 134217728, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>

[pid 83583] rt_sigprocmask(SIG_BLOCK, ~[], <unfinished ...>

[pid 83584] <... mmap resumed>) = 0x77c430bfb000

[pid 83583] <... rt_sigprocmask resumed>[], 8) = 0

[pid 83584] munmap(0x77c430bfb000, 54546432 <unfinished ...>

[pid 83583]
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x77c4393fb990, parent_tid=0x77c4393fb990, exit_signal=0, stack=0x77c438bfb000, stack_size=0x7fff80, tls=0x77c4393fb6c0} <unfinished ...>

[pid 83584] <... munmap resumed>) = 0

[pid 83584] munmap(0x77c438000000, 12562432strace: Process 83585 attached

 <unfinished ...>

[pid 83583] <... clone3 resumed> => {parent_tid=[83585]}, 88) = 83585

[pid 83584] <... munmap resumed>) = 0

[pid 83585] rseq(0x77c4393fbfe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 83583] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 83584] mprotect(0x77c434000000, 135168, PROT_READ|PROT_WRITE <unfinished ...>

[pid 83583] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 83585] <... rseq resumed>) = 0

[pid 83583] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>

[pid 83584] <... mprotect resumed>) = 0

[pid 83583] <... mmap resumed>) = 0x77c4383fa000

[pid 83585] set_robust_list(0x77c4393fb9a0, 24 <unfinished ...>

[pid 83583] mprotect(0x77c4383fb000, 8388608, PROT_READ|PROT_WRITE <unfinished ...>

[pid 83585] <... set_robust_list resumed>) = 0

[pid 83583] <... mprotect resumed>) = 0

[pid 83585] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 83583] rt_sigprocmask(SIG_BLOCK, ~[], <unfinished ...>

[pid 83585] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 83583] <... rt_sigprocmask resumed>[], 8) = 0

[pid 83585] mmap(NULL, 134217728, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>

[pid 83583]
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x77c438bfa990, parent_tid=0x77c438bfa990, exit_signal=0, stack=0x77c4383fa000, stack_size=0x7fff80, tls=0x77c438bfa6c0} <unfinished ...>

[pid 83585] <... mmap resumed>) = 0x77c42c000000

strace: Process 83586 attached

[pid 83585] munmap(0x77c430000000, 67108864 <unfinished ...>

[pid 83583] <... clone3 resumed> => {parent_tid=[83586]}, 88) = 83586

[pid 83586] rseq(0x77c438bfafe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 83585] <... munmap resumed>) = 0

[pid 83583] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 83586] <... rseq resumed>) = 0

[pid 83583] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 83585] mprotect(0x77c42c000000, 135168, PROT_READ|PROT_WRITE
<unfinished ...>

[pid 83583] mmap(NULL, 8392704, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>

[pid 83586] set_robust_list(0x77c438bfa9a0, 24 <unfinished ...>

[pid 83583] <... mmap resumed>) = 0x77c4337ff000

[pid 83585] <... mprotect resumed>) = 0

[pid 83583] mprotect(0x77c433800000, 8388608, PROT_READ|PROT_WRITE
<unfinished ...>

[pid 83586] <... set_robust_list resumed>) = 0

[pid 83583] <... mprotect resumed>) = 0

[pid 83586] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 83583] rt_sigprocmask(SIG_BLOCK, ~[], <unfinished ...>

[pid 83586] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 83583] <... rt_sigprocmask resumed>[], 8) = 0

[pid 83586] mmap(0x77c430000000, 67108864, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>

[pid 83583]
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x77c433fff990,
parent_tid=0x77c433fff990, exit_signal=0, stack=0x77c4337ff000,
stack_size=0x7fff80, tls=0x77c433fff6c0} <unfinished ...>

[pid 83586] <... mmap resumed>)   = 0x77c428000000

[pid 83586] mprotect(0x77c428000000, 135168, PROT_READ|PROT_WRITEstrace: Process 83587 attached

) = 0

[pid 83583] <... clone3 resumed> => {parent_tid=[83587]}, 88) = 83587

[pid 83587] rseq(0x77c433ffffe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 83583] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 83587] <... rseq resumed>)   = 0

[pid 83583] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 83587] set_robust_list(0x77c433fff9a0, 24 <unfinished ...>

[pid 83583] futex(0x77c439bfc990,
FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 83584, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 83587] <... set_robust_list resumed>) = 0

[pid 83587] rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

[pid 83587] mmap(NULL, 134217728, PROT_NONE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x77c420000000

[pid 83587] munmap(0x77c424000000, 67108864) = 0

[pid 83587] mprotect(0x77c420000000, 135168, PROT_READ|PROT_WRITE) = 0

[pid 83584] rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0

[pid 83584] madvise(0x77c4393fc000, 8368128, MADV_DONTNEED <unfinished ...>

[pid 83585] rt_sigprocmask(SIG_BLOCK, ~[RT_1], <unfinished ...>

[pid 83584] <... madvise resumed>)   = 0

[pid 83585] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 83584] exit(0 <unfinished ...>

[pid 83585] madvise(0x77c438bfb000, 8368128, MADV_DONTNEED <unfinished ...>

[pid 83584] <... exit resumed>)   = ?

[pid 83585] <... madvise resumed>)   = 0

[pid 83583] <... futex resumed>)   = 0

[pid 83585] exit(0 <unfinished ...>

[pid 83584] +++ exited with 0 +++

[pid 83583] futex(0x77c4393fb990,
FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 83585, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 83585] <... exit resumed>)   = ?

[pid 83583] <... futex resumed>)   = 0

[pid 83585] +++ exited with 0 +++

[pid 83583] futex(0x77c438bfa990,
FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 83586, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 83586] rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0

[pid 83586] madvise(0x77c4383fa000, 8368128, MADV_DONTNEED <unfinished ...>

[pid 83587] rt_sigprocmask(SIG_BLOCK, ~[RT_1], <unfinished ...>

[pid 83586] <... madvise resumed>)   = 0

[pid 83587] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 83586] exit(0 <unfinished ...>

[pid 83587] madvise(0x77c4337ff000, 8368128, MADV_DONTNEED <unfinished ...>

[pid 83586] <... exit resumed>)   = ?

[pid 83587] <... madvise resumed>)   = 0

[pid 83583] <... futex resumed>)   = 0

[pid 83586] +++ exited with 0 +++

[pid 83583] futex(0x77c433fff990,
FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 83587, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 83587] exit(0)   = ?

[pid 83583] <... futex resumed>)   = 0

[pid 83587] +++ exited with 0 +++

fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x4), ...}) = 0

write(1, "\320\236\320\261\321\200\320\260\320\267\320\265\321\206: 'GCT'\n",
22Образец: 'GCT'

) = 22

write(1, "T1: 15.33343 \320\274\321\201\n", 18T1: 15.33343 мс

) = 18

write(1, "Tn: 10.57445 \320\274\321\201\n", 18Tn: 10.57445 мс

) = 18

write(1, "\320\235\320\260\320\271\320\264\320\265\320\275\320\276 700000\n",
22Найдено 700000

) = 22

munmap(0x77c439bfd000, 6303744)   = 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 наивного алгоритма для поиска образца в строке.