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-214Б-23

Студент: Шведова Е. В.

Препод

ватель: Бахарев В.Д.

Оценка: _____

Дата: 12.11.25

Москва, 2025

Постановка задачи

Вариант 14.

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 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.

Есть набор 128 битных чисел, записанных в шестнадцатеричном представлении, хранящихся в файле. Необходимо посчитать их среднее арифметическое. Округлить результат до целых. Количество используемой оперативной памяти должно задаваться "ключом"

Общий метод и алгоритм решения

Программа начинает выполнение с инициализации таймера и проверки аргументов командной строки. После успешной валидации параметров открывается входной файл, содержащий 128 шестнадцатеричных чисел. На основе переданного объема памяти вычисляется размер буфера для пакетной обработки данных.

Основной цикл программы построчно считывает числа из файла, заполняя выделенный буфер. Когда буфер заполняется или достигается конец файла, данные распределяются между рабочими потоками. Каждый поток получает свой сегмент чисел для вычисления частичной суммы с использованием функции hex_add(), которая корректно обрабатывает сложение больших шестнадцатеричных значений.

После создания и запуска потоков основная программа переходит в режим ожидания через вызовы pthread_join(), обеспечивая синхронизацию и сбор результатов. Полученные частичные суммы агрегируются в общую сумму, а счетчик обработанных чисел обновляется. Этот процесс повторяется для всех пакетов данных до полного чтения файла.

По завершении обработки всех чисел программа вычисляет среднее арифметическое с помощью функции hex_div(), выполняющей целочисленное деление в шестнадцатеричной системе. Финальный этап включает освобождение выделенных ресурсов, вывод итоговых результатов и информации о производительности, после чего программа корректно завершает работу.

Использованные системные вызовы:

- ssize_t write(int fd, const void* buf, size_t count) – записывает данные из буфера в файловый дескриптор. Используется для вывода результатов и сообщений об ошибках.

- void exit(int status) – завершает выполнение процесса с указанным статусом при критических ошибках.
- int clock_gettime(clockid_t clk_id, struct timespec* tp) – получает время от указанных часов (CLOCK_MONOTONIC). Используется для измерения реального времени выполнения программы.
- int pthread_create(pthread_t* thread, const pthread_attr_t* attr, void* (*start_routine) (void*), void* arg) – создает новый поток выполнения для параллельного вычисления частичных сумм.
- int pthread_join(pthread_t thread, void** thread_return) – ожидает завершения указанного потока и получает его возвращаемое значение. Обеспечивает синхронизацию потоков перед агрегацией результатов.
- FILE* fopen(const char* filename, const char* mode) – открывает файл для чтения данных.
- char* fgets(char* str, int num, FILE* stream) – построчно считывает данные из файла в буфер.
- int fclose(FILE* stream) – закрывает файл после завершения работы.

Анализ ускорения и эффективности

Число потоков	Время исполнения (мс)	Ускорение	Эффективность
1	0.588	1	1
2	0.368	1.6	0.8
4	0.595	0.99	0.247
8	0.671	0.88	0.11
16	1.194	0.49	0.031
32	2.473	0.24	0.007
100	4.429	0.13	0.001

Из результатов тестирования четко видна оптимальная зона параллелизации при 2 потоках, где достигается максимальное ускорение 1.60x. При дальнейшем увеличении числа потоков наблюдается резкий спад производительности - ускорение снижается до 0.13x при 100 потоках.

Это характерное поведение для алгоритмов с малым объемом вычислений, где накладные расходы на создание и синхронизацию потоков начинают значительно преобладать над выигрышем от параллелизации. При 128 числах каждый дополнительный поток добавляет больше служебных операций, чем полезной работы.

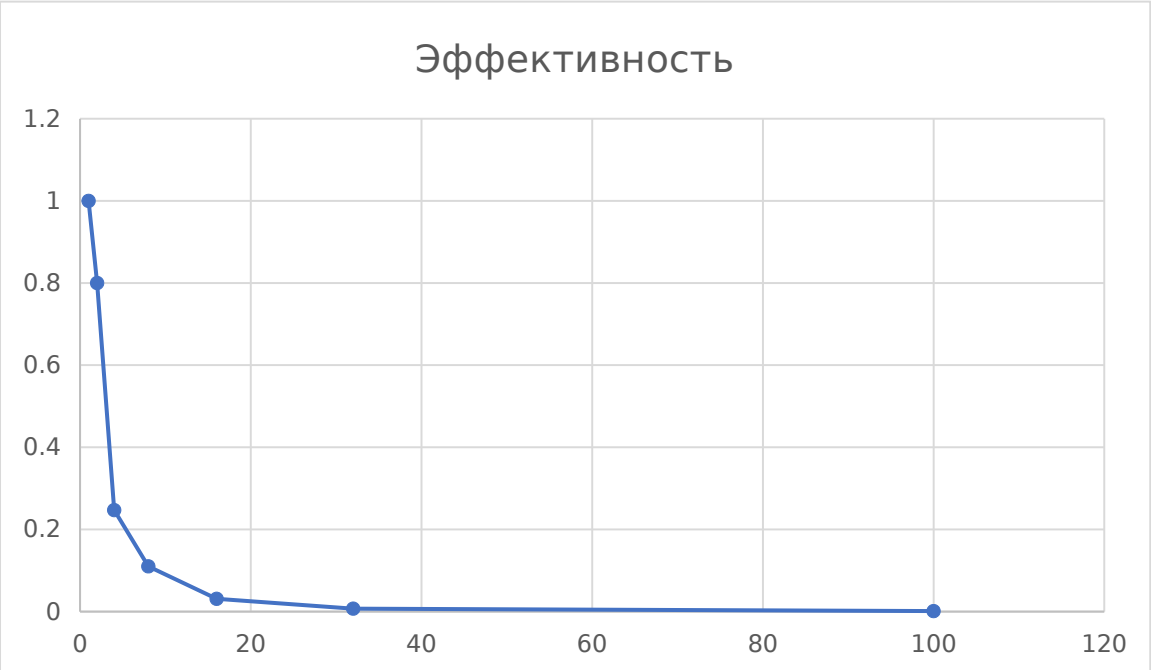

График эффективности демонстрирует резкое падение после 2 потоков - с 80% до всего 0.1% при 100 потоках. Наиболее эффективное использование ресурсов

системы наблюдается именно при 2 потоках, где каждый поток работает с эффективностью 80%.

Дальнейшее увеличение числа потоков приводит к крайне нерациональному использованию вычислительных ресурсов - при 100 потоках эффективность составляет лишь 0.1%, что означает, что 99.9% ресурсов системы тратятся на организацию параллелизма, а не на полезные вычисления.

Данные графика демонстрируют закон Амдала. Закон иллюстрирует ограничение роста производительности вычислительной системы с увеличением числа вычислителей.

Можно сделать вывод, что самым важным является нахождение баланса между ускорением и эффективностью использования ресурсов.

Код программы

hex,h

```
#ifndef HEX_H
#define HEX_H


#include <stdint.h>


char* hex_add(const char* n1, const char* n2);
char* hex_div(const char* hex, uint64_t div);


#endif
```

hex.c

```
#include "hex.h"
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <ctype.h>


char* hex_add(const char* hex1, const char* hex2)
{
    size_t length1 = strlen(hex1);
    size_t length2 = strlen(hex2);
    size_t max_length = length1 > length2 ? length1 : length2;
    size_t output_size = max_length + 2;


    char *output_buffer = malloc(output_size);
    if (!output_buffer)
    {
        char error_msg[] = "Memory allocation failed\n";
```

```
        write(STDOUT_FILENO, error_msg, sizeof(error_msg) - 1);
        return NULL;
    }

    uint8_t overflow = 0;
    uint64_t position = output_size - 2;
    output_buffer[output_size - 1] = '\0';

    for (size_t index = 0; index < max_length; ++index)
    {
        uint8_t digit1 = 0, digit2 = 0;

        if (index < length1)
        {
            char current_char = hex1[length1 - 1 - index];
            digit1 = (current_char >= '0' && current_char <= '9') ?
current_char - '0' : current_char - 'a' + 10;
        }

        if (index < length2)
        {
            char current_char = hex2[length2 - 1 - index];
            digit2 = (current_char >= '0' && current_char <= '9') ?
current_char - '0' : current_char - 'a' + 10;
        }

        uint8_t total_digits = digit1 + digit2 + overflow;
        overflow = total_digits / 16;
        total_digits %= 16;

        output_buffer[position--] = total_digits < 10 ? total_digits + '0' :
total_digits - 10 + 'a';
    }

    if (overflow)
    {
        output_buffer[position--] = overflow < 10 ? overflow + '0' : overflow -
10 + 'a';
    }

    char *final_output = malloc(strlen(output_buffer + position + 1) + 1);
    if (!final_output)
    {
        char error_msg[] = "Memory allocation failed\n";
        write(STDOUT_FILENO, error_msg, sizeof(error_msg) - 1);
        free(output_buffer);
        return NULL;
    }
```

```
    strcpy(final_output, output_buffer + position + 1);
    free(output_buffer);
    return final_output;
}

char* hex_div(const char* hex_value, uint64_t divider)
{
    if (divider == 0)
    {
        return NULL;
    }

    if (strcmp(hex_value, "0") == 0)
    {
        return strdup("0");
    }

    size_t hex_length = strlen(hex_value);
    char *division_result = malloc(hex_length + 1);
    if (!division_result)
    {
        char error_msg[] = "Memory allocation failed\n";
        write(STDOUT_FILENO, error_msg, sizeof(error_msg) - 1);
        return NULL;
    }

    uint64_t remainder_value = 0;
    size_t result_index = 0;

    for (size_t pos = 0; pos < hex_length; ++pos)
    {
        char current_char = hex_value[pos];
        uint8_t digit_value = (current_char >= '0' && current_char <= '9') ? current_char - '0' : current_char - 'a' + 10;

        remainder_value = remainder_value * 16 + digit_value;
        uint64_t quotient_value = remainder_value / divider;
        remainder_value %= divider;

        if (quotient_value > 0 || result_index > 0)
        {
            division_result[result_index++] = quotient_value < 10 ? quotient_value + '0' : quotient_value - 10 + 'a';
        }
    }

    if (result_index == 0)
```

```
    {
        division_result[result_index++] = '0';
    }
    division_result[result_index] = '\0';

    return division_result;
}
```

main.c

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

typedef struct {
    char **values;
    uint64_t total;
    char *partial_result;
    uint64_t processed_count;
} ThreadContext;

void* compute_partial_sum(void* context_ptr) {
    ThreadContext* ctx = (ThreadContext*)context_ptr;
    char* local_sum = strdup("0");
    uint64_t local_counter = 0;

    for (uint64_t idx = 0; idx < ctx->total; ++idx) {
        char* updated_sum = hex_add(local_sum, ctx->values[idx]);
        free(local_sum);
```

```
        local_sum = updated_sum;
        ++local_counter;
    }

    ctx->partial_result = local_sum;
    ctx->processed_count = local_counter;

    return NULL;
}

int main(int argc, char** argv) {
    if (argc != 4) {
        char usage_msg[] = "Usage: ./program <thread_count> <memory_size> <input_file>\n";
        write(STDOUT_FILENO, usage_msg, sizeof(usage_msg) - 1);
        return 1;
    }

    struct timespec begin_time, end_time;
    clock_gettime(CLOCK_MONOTONIC, &begin_time);

    uint64_t num_threads = strtoull(argv[1], NULL, 10);
    uint64_t memory_size = strtoull(argv[2], NULL, 10);

    FILE* input_file = fopen(argv[3], "r");
    if (!input_file) {
        char error_msg[] = "Failed to open input file\n";
        write(STDOUT_FILENO, error_msg, sizeof(error_msg) - 1);
        exit(EXIT_FAILURE);
    }
```

```
uint64_t bytes_per_entry = 32 + 1 + 8;
uint64_t max_entries_per_batch = memory_size / bytes_per_entry;
if (max_entries_per_batch == 0) max_entries_per_batch = 1;

char input_buffer[64];
char** number_array = malloc(sizeof(char*) * max_entries_per_batch);

char* overall_sum = strdup("0");
uint64_t total_numbers = 0;

pthread_t* thread_pool = malloc(sizeof(pthread_t) * num_threads);
ThreadContext* thread_contexts = malloc(sizeof(ThreadContext) * num_threads);

while (1) {
    uint64_t current_batch_size = 0;

    while (current_batch_size < max_entries_per_batch &&
           fgets(input_buffer, sizeof(input_buffer), input_file)) {
        char* newline_pos = input_buffer;
        while (*newline_pos && *newline_pos != '\n') ++newline_pos;
        *newline_pos = '\0';

        if (strlen(input_buffer) > 0) {
            number_array[current_batch_size] = strdup(input_buffer);
            ++current_batch_size;
        }
    }

    if (current_batch_size == 0) break;
```

```
uint64_t base_per_thread = current_batch_size / num_threads;
uint64_t extra_items = current_batch_size % num_threads;

for (uint64_t t = 0; t < num_threads; ++t) {
    uint64_t start_index = t * base_per_thread + (t > 0 ? extra_items : 0);
    uint64_t item_count = base_per_thread + (t == 0 ? extra_items : 0);

    if (item_count > 0) {
        thread_contexts[t].values = number_array + start_index;
        thread_contexts[t].total = item_count;
        thread_contexts[t].partial_result = NULL;
        thread_contexts[t].processed_count = 0;

        pthread_create(&thread_pool[t], NULL, compute_partial_sum, &thread_contexts[t]);
    }
}

for (uint64_t t = 0; t < num_threads; ++t) {
    uint64_t item_count = base_per_thread + (t == 0 ? extra_items : 0);
    if (item_count > 0) {
        pthread_join(thread_pool[t], NULL);
    }
}

for (uint64_t t = 0; t < num_threads; ++t) {
    uint64_t item_count = base_per_thread + (t == 0 ? extra_items : 0);

    if (item_count > 0 && thread_contexts[t].partial_result) {
        char* new_total = hex_add(overall_sum, thread_contexts[t].partial_result);
        free(overall_sum);
```

```
            overall_sum = new_total;
            free(thread_contexts[t].partial_result);
            total_numbers += thread_contexts[t].processed_count;
        }
    }

    for (uint64_t i = 0; i < current_batch_size; ++i) {
        free(number_array[i]);
    }
}

char count_msg[64];
snprintf(count_msg, sizeof(count_msg), "Total numbers processed: %lu\n", total_numbers);
write(STDOUT_FILENO, count_msg, strlen(count_msg));

char sum_msg[64];
snprintf(sum_msg, sizeof(sum_msg), "Cumulative sum: %s\n", overall_sum);
write(STDOUT_FILENO, sum_msg, strlen(sum_msg));

if (total_numbers > 0) {
    char* mean_value = hex_div(overall_sum, total_numbers);
    char avg_msg[64];
    snprintf(avg_msg, sizeof(avg_msg), "Floored hexadecimal average: %s\n", mean_value);
    write(STDOUT_FILENO, avg_msg, strlen(avg_msg));
    free(mean_value);
}

free(number_array);
free(overall_sum);
free(thread_pool);
```

```
    free(thread_contexts);
    fclose(input_file);

    clock_gettime(CLOCK_MONOTONIC, &end_time);

    double time_elapsed = (end_time.tv_sec - begin_time.tv_sec) + (end_time.tv_nsec - begin_time.tv_nsec) / 1e9;

    char time_msg[64];
    snprintf(time_msg, sizeof(time_msg), "\nTotal execution time: %f seconds\n", time_elapsed);
    write(STDOUT_FILENO, time_msg, strlen(time_msg));

    char thread_msg[64];
    snprintf(thread_msg, sizeof(thread_msg), "Threads utilized: %lu\n", num_threads);
    write(STDOUT_FILENO, thread_msg, strlen(thread_msg));

    return 0;
}
```

Протокол работы программы

```
crane@Assus:~/op/Lab2/scr/build$ ./main 1 1048576 numbers.txt
Total numbers processed: 128
Cumulative sum: 2a66ca8b4b04b617b8886ac58b4b04b5ed
Floored hexadecimal average: 54cd951696096c2f7110d58b1696096b

Total execution time: 0.000588 seconds
Threads utilized: 1
crane@Assus:~/op/Lab2/scr/build$ ./main 2 1048576 numbers.txt
Total numbers processed: 128
Cumulative sum: 2a66ca8b4b04b617b8886ac58b4b04b5ed
Floored hexadecimal average: 54cd951696096c2f7110d58b1696096b
```

Total execution time: 0.000368 seconds

Threads utilized: 2

crane@Assus:~/op/Lab2/scr/build$ ./main 4 1048576 numbers.txt

Total numbers processed: 128

Cumulative sum: 2a66ca8b4b04b617b8886ac58b4b04b5ed

Floored hexadecimal average: 54cd951696096c2f7110d58b1696096b

Total execution time: 0.000595 seconds

Threads utilized: 4

crane@Assus:~/op/Lab2/scr/build$ ./main 8 1048576 numbers.txt

Total numbers processed: 128

Cumulative sum: 2a66ca8b4b04b617b8886ac58b4b04b5ed

Floored hexadecimal average: 54cd951696096c2f7110d58b1696096b

Total execution time: 0.000671 seconds

Threads utilized: 8

crane@Assus:~/op/Lab2/scr/build$ ./main 16 1048576 numbers.txt

Total numbers processed: 128

Cumulative sum: 2a66ca8b4b04b617b8886ac58b4b04b5ed

Floored hexadecimal average: 54cd951696096c2f7110d58b1696096b

Total execution time: 0.001194 seconds

Threads utilized: 16

crane@Assus:~/op/Lab2/scr/build$ ./main 32 1048576 numbers.txt

Total numbers processed: 128

Cumulative sum: 2a66ca8b4b04b617b8886ac58b4b04b5ed

Floored hexadecimal average: 54cd951696096c2f7110d58b1696096b

Total execution time: 0.002473 seconds

Threads utilized: 32

crane@Assus:~/op/Lab2/scr/build$ ./main 100 1048576 numbers.txt

Total numbers processed: 128

Cumulative sum: 2a66ca8b4b04b617b8886ac58b4b04b5ed

Floored hexadecimal average: 54cd951696096c2f7110d58b1696096b

Total execution time: 0.004429 seconds

Threads utilized: 100

crane@Assus:~/op/Lab2/scr/build$ strace -f ./main 4 1048576 numbers.txt

execve("./main", ["./main", "4", "1048576", "numbers.txt"], 0x7ffd2634c290 /* 78 vars */) = 0

brk(NULL) = 0x619137d1e000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7b23f4487000

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

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

fstat(3, {st_mode=S_IFREG|0644, st_size=75687, ...}) = 0

mmap(NULL, 75687, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7b23f4474000

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) = 0x7b23f4200000

mmap(0x7b23f4228000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7b23f4228000

mmap(0x7b23f43b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7b23f43b0000

mmap(0x7b23f43ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7b23f43ff000

mmap(0x7b23f4405000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7b23f4405000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7b23f4471000

arch_prctl(ARCH_SET_FS, 0x7b23f4471740) = 0

set_tid_address(0x7b23f4471a10) = 50524

set_robust_list(0x7b23f4471a20, 24) = 0

rseq(0x7b23f4472060, 0x20, 0, 0x53053053) = 0

mprotect(0x7b23f43ff000, 16384, PROT_READ) = 0

mprotect(0x619132910000, 4096, PROT_READ) = 0

mprotect(0x7b23f44c5000, 8192, PROT_READ) = 0

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

munmap(0x7b23f4474000, 75687) = 0

getrandom("\xb8\x3b\x29\x4c\x24\x02\x58\xf8", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x619137d1e000

brk(0x619137d3f000) = 0x619137d3f000

openat(AT_FDCWD, "numbers.txt", O_RDONLY) = 3

mmap(NULL, 204800, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7b23f443f000

fstat(3, {st_mode=S_IFREG|0664, st_size=4224, ...}) = 0

read(3, "1a3f5c7e9b2d4f6a8c1e3a5c7e9b2d4f"..., 4096) = 4096

read(3, "9c1e3a5c7e9a1c5e7a9c1e3a5c7e\n6f8"..., 4096) = 128

read(3, "", 4096) = 0

rt_sigaction(SIGRT_1, {sa_handler=0x7b23f4299530, sa_mask=[], sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x7b23f4245330}, 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) = 0x7b23f39ff000

mprotect(0x7b23f3a00000, 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=0x7b23f41ff990, parent_tid=0x7b23f41ff990, exit_signal=0, stack=0x7b23f39ff000, stack_size=0x7fff80, tls=0x7b23f41ff6c0} => {parent_tid=[50525]}, 88) = 50525

strace: Process 50525 attached

[pid 50524] rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

[pid 50525] rseq(0x7b23f41fffe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 50524] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>

[pid 50525] <... rseq resumed>) = 0

[pid 50524] <... mmap resumed>) = 0x7b23f31fe000

[pid 50525] set_robust_list(0x7b23f41ff9a0, 24 <unfinished ...>

[pid 50524] mprotect(0x7b23f31ff000, 8388608, PROT_READ|PROT_WRITE) = 0

[pid 50525] <... set_robust_list resumed>) = 0

[pid 50524] rt_sigprocmask(SIG_BLOCK, ~[], <unfinished ...>

[pid 50525] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 50524] <... rt_sigprocmask resumed>[], 8) = 0

[pid 50525] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 50524] clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7b23f39fe990, parent_tid=0x7b23f39fe990, exit_signal=0, stack=0x7b23f31fe000, stack_size=0x7fff80, tls=0x7b23f39fe6c0} <unfinished ...>

[pid 50525] mmap(NULL, 134217728, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0strace: Process 50526 attached

<unfinished ...>

[pid 50524] <... clone3 resumed> => {parent_tid=[50526]}, 88) = 50526

[pid 50525] <... mmap resumed>) = 0x7b23eb000000

[pid 50524] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 50526] rseq(0x7b23f39fefe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 50524] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 50526] <... rseq resumed>) = 0

[pid 50525] munmap(0x7b23eb000000, 16777216 <unfinished ...>

[pid 50524] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7b23ea7ff000

[pid 50526] set_robust_list(0x7b23f39fe9a0, 24 <unfinished ...>

[pid 50524] mprotect(0x7b23ea800000, 8388608, PROT_READ|PROT_WRITE <unfinished ...>

[pid 50526] <... set_robust_list resumed>) = 0

[pid 50525] <... munmap resumed>) = 0

[pid 50524] <... mprotect resumed>) = 0

[pid 50526] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 50524] rt_sigprocmask(SIG_BLOCK, ~[], <unfinished ...>

[pid 50526] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 50525] munmap(0x7b23f0000000, 50331648 <unfinished ...>

[pid 50524] <... rt_sigprocmask resumed>[], 8) = 0

[pid 50526] mmap(NULL, 134217728, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>

[pid 50525] <... munmap resumed>) = 0

[pid 50524] clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7b23eafff990, parent_tid=0x7b23eafff990, exit_signal=0, stack=0x7b23ea7ff000, stack_size=0x7fff80, tls=0x7b23eafff6c0} <unfinished ...>

[pid 50526] <... mmap resumed>) = 0x7b23e2600000

[pid 50525] mprotect(0x7b23ec000000, 135168, PROT_READ|PROT_WRITEstrace: Process 50527 attached

<unfinished ...>

[pid 50526] munmap(0x7b23e2600000, 27262976 <unfinished ...>

[pid 50525] <... mprotect resumed>) = 0

[pid 50524] <... clone3 resumed> => {parent_tid=[50527]}, 88) = 50527

[pid 50527] rseq(0x7b23eaffffe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 50526] <... munmap resumed>) = 0

[pid 50525] rt_sigprocmask(SIG_BLOCK, ~[RT_1], <unfinished ...>

[pid 50524] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 50527] <... rseq resumed>) = 0

[pid 50526] munmap(0x7b23e8000000, 39845888 <unfinished ...>

[pid 50524] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 50527] set_robust_list(0x7b23eafff9a0, 24 <unfinished ...>

[pid 50526] <... munmap resumed>) = 0

[pid 50524] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0 <unfinished ...>

[pid 50527] <... set_robust_list resumed>) = 0

[pid 50526] mprotect(0x7b23e4000000, 135168, PROT_READ|PROT_WRITE <unfinished ...>

[pid 50527] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 50524] <... mmap resumed>) = 0x7b23f29fd000

[pid 50527] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 50526] <... mprotect resumed>) = 0

[pid 50525] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 50524] mprotect(0x7b23f29fe000, 8388608, PROT_READ|PROT_WRITE <unfinished ...>

[pid 50525] madvise(0x7b23f39ff000, 8368128, MADV_DONTNEED <unfinished ...>

[pid 50527] rt_sigprocmask(SIG_BLOCK, ~[RT_1], <unfinished ...>

[pid 50524] <... mprotect resumed>)   = 0

[pid 50526] rt_sigprocmask(SIG_BLOCK, ~[RT_1], <unfinished ...>

[pid 50525] <... madvise resumed>)   = 0

[pid 50524] rt_sigprocmask(SIG_BLOCK, ~[], <unfinished ...>

[pid 50527] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 50526] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 50525] exit(0 <unfinished ...>

[pid 50524] <... rt_sigprocmask resumed>[], 8) = 0

[pid 50527] madvise(0x7b23ea7ff000, 8368128, MADV_DONTNEED <unfinished ...>

[pid 50526] madvise(0x7b23f31fe000, 8368128, MADV_DONTNEED) = 0

[pid 50526] exit(0 <unfinished ...>

[pid 50525] <... exit resumed>)   = ?

[pid 50526] <... exit resumed>)   = ?

[pid 50526] +++ exited with 0 +++

[pid 50525] +++ exited with 0 +++

[pid 50524] clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7b23f31fd990, parent_tid=0x7b23f31fd990, exit_signal=0, stack=0x7b23f29fd000, stack_size=0x7fff80, tls=0x7b23f31fd6c0} <unfinished ...>

[pid 50527] <... madvise resumed>)   = 0

[pid 50527] exit(0strace: Process 50528 attached

)   = ?

[pid 50524] <... clone3 resumed> => {parent_tid=[50528]}, 88) = 50528

[pid 50528] rseq(0x7b23f31fdfe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 50527] +++ exited with 0 +++

[pid 50524] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 50528] <... rseq resumed>)   = 0

[pid 50524] <... rt_sigprocmask resumed>NULL, 8) = 0

[pid 50524] futex(0x7b23f31fd990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 50528, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 50528] set_robust_list(0x7b23f31fd9a0, 24) = 0

[pid 50528] rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

[pid 50528] rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0

[pid 50528] madvise(0x7b23f29fd000, 8368128, MADV_DONTNEED) = 0

[pid 50528] exit(0)   = ?

[pid 50524] <... futex resumed>)   = 0

[pid 50528] +++ exited with 0 +++

write(1, "Total numbers processed: 128\n", 29Total numbers processed: 128

) = 29

write(1, "Cumulative sum: 2a66ca8b4b04b617"..., 51Cumulative sum: 2a66ca8b4b04b617b8886ac58b4b04b5ed

) = 51

write(1, "Floored hexadecimal average: 54c"..., 62Floored hexadecimal average: 54cd951696096c2f7110d58b1696096b

) = 62

munmap(0x7b23f443f000, 204800)   = 0

close(3)   = 0

write(1, "\nTotal execution time: 0.007037 "..., 40

Total execution time: 0.007037 seconds

) = 40

write(1, "Threads utilized: 4\n", 20Threads utilized: 4

)   = 20

exit_group(0)   = ?

+++ exited with 0 +++

Вывод

Во время выполнения лабораторной работы произошло изучение и использование основных системных вызовов для работы с процессами в ОС Linux. Была создана программа, демонстрирующая создание потоков и вычисление среднего арифметического числа из файла.