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-213Б-23

Студент: Солодова С. М.

Преподаватель: Бахарев В.Д.

Оценка:_____

Дата: _____

Москва, 2024

Постановка задачи

Вариант 8.

Есть К массивов одинаковой длины. Необходимо сложить эти массивы. Необходимо предусмотреть стратегию, адаптирующуюся под количество массивов и их длину (по количеству операций)

Общий метод и алгоритм решения

Использованные системные вызовы:

- write() – записываем число байт из буфера в указанный файловый дескриптор
- read() – открытие файлов для чтения и записи.
- malloc() – выделение памяти для массивов и структур данных.
- pthread_exit() – завершаем вызывающий поток
- pthread_create() – создаем новый поток с атрибутами
- pthread_join() – ожидаем завершение потока

1. Чтение входных данных:
   - Программа считывает количество массивов и длину каждого массива из входного файла.
   - Затем она считывает все элементы массивов и сохраняет их в одном большом массиве (**flattened array**).
2. Инициализация данных для потоков:
   - Программа выделяет память для результирующего массива, который будет содержать суммы элементов соответствующих массивов.
   - Выделяется память для структур данных потоков (**ThreadData**), которые содержат информацию о том, какие части массивов будет обрабатывать каждый поток.
3. Создание потоков:
   - Программа создает указанное количество потоков с помощью **pthread_create**.
   - Каждый поток выполняет функцию **sum_arrays**, которая суммирует элементы массивов в заданном диапазоне.
4. Распределение работы между потоками:
   - Каждый поток получает уникальный идентификатор (**thread_id**) и общее количество потоков (**num_threads**).
   - На основе этих данных каждый поток вычисляет свой диапазон работы (**start** и **end**), который определяет, какие элементы массивов он будет обрабатывать.
   - Этот диапазон гарантирует, что каждый поток обрабатывает примерно равное количество элементов массивов.
5. Суммирование элементов массивов:
   - В функции **sum_arrays** каждый поток суммирует элементы всех массивов в своем диапазоне и записывает результат в результирующий массив.
   - Поскольку каждый поток работает с уникальным диапазоном элементов, нет необходимости в синхронизации (например, с использованием мьютексов).
6. Ожидание завершения потоков:
   - Основной поток ожидает завершения всех созданных потоков с помощью **pthread_join**.
7. Запись результата в выходной файл:
   - После завершения всех потоков основной поток записывает результирующий массив в выходной файл.

Исследование ускорения и эффективности программы при разном количестве потоков.

Была написана программа на языке Python, который вычисляет ускорение и эффективность программы при разном количестве потоков.

Код программы:

```
import subprocess
import time
import os

def generate_input_file(filename, num_arrays, elements_per_array):
    with open(filename, 'w') as f:
        f.write(f"{num_arrays} {elements_per_array}\n")
        for _ in range(num_arrays):
            array = " ".join(str(x) for x in range(1, elements_per_array + 1))
            f.write(array + "\n")

def run_experiment(executable, input_file, num_threads, output_file):
    start_time = time.time()
    result = subprocess.run([executable, input_file, str(num_threads), output_file],
capture_output=True, text=True)
    end_time = time.time()

    if result.returncode != 0:
        print(f"Error: {result.stderr}")
        return None

    time_elapsed = end_time - start_time
    return time_elapsed

def main():
    executable = "./main"
    thread_counts = [1, 2, 6, 10]
    configurations = [
        (5, 5),
        (10, 100),
        (100, 1000),
    ]
    input_folder = "input_data"
    output_folder = "output_data"
    os.makedirs(input_folder, exist_ok=True)
    os.makedirs(output_folder, exist_ok=True)

    input_files = []
    for num_arrays, elements_per_array in configurations:
        filename = os.path.join(input_folder,
f"input_{num_arrays}_{elements_per_array}.txt")
        generate_input_file(filename, num_arrays, elements_per_array)
        input_files.append(filename)

    results = {}
    for input_file in input_files:
        file_results = {}
        base_name = os.path.splitext(os.path.basename(input_file))[0]
        for threads in thread_counts:
            output_file = os.path.join(output_folder,
f"{base_name}_output_{threads}.txt")
            time_elapsed = run_experiment(executable, input_file, threads, output_file)
```

```
            if time_elapsed is not None:
                file_results[threads] = time_elapsed
            else:
                file_results[threads] = float('inf')

        results[input_file] = file_results

    for input_file, timings in results.items():
        config = os.path.splitext(os.path.basename(input_file))[0].split('_')[1:]
        num_arrays, elements_per_array = map(int, config)
        t1 = timings[1]

        print(f"\nResults for {num_arrays} arrays of size {elements_per_array}:")
        for threads, time_elapsed in sorted(timings.items()):
            if time_elapsed == float('inf'):
                continue
            speedup = t1 / time_elapsed
            efficiency = speedup / threads
            print(f"Threads: {threads}, Time: {time_elapsed:.6f} sec, Speedup:
{speedup:.2f}, Efficiency: {efficiency:.2f}")

if __name__ == "__main__":
    main()
```

Результат выполнения программы:

```
ab2$ python3 research.py

Results for 5 arrays of size 5:
Threads: 1, Time: 0.097317 sec, Speedup: 1.00, Efficiency: 1.00
Threads: 2, Time: 0.085144 sec, Speedup: 1.14, Efficiency: 0.57
Threads: 6, Time: 0.083382 sec, Speedup: 1.17, Efficiency: 0.19
Threads: 10, Time: 0.076831 sec, Speedup: 1.27, Efficiency: 0.13

Results for 10 arrays of size 100:
Threads: 1, Time: 1.189483 sec, Speedup: 1.00, Efficiency: 1.00
Threads: 2, Time: 0.797592 sec, Speedup: 1.49, Efficiency: 0.75
Threads: 6, Time: 0.878026 sec, Speedup: 1.35, Efficiency: 0.23
Threads: 10, Time: 0.837428 sec, Speedup: 1.42, Efficiency: 0.14

Results for 100 arrays of size 1000:
Threads: 1, Time: 97.563452 sec, Speedup: 1.00, Efficiency: 1.00
Threads: 2, Time: 101.013496 sec, Speedup: 0.97, Efficiency: 0.48
Threads: 6, Time: 99.868316 sec, Speedup: 0.98, Efficiency: 0.16
Threads: 10, Time: 95.695844 sec, Speedup: 1.02, Efficiency: 0.10
```

Результаты показали, что ускорение и эффективность алгоритма зависят от количества потоков и входных данных. При увеличении количества потоков не наблюдается значительное ускорения выполнения задачи: замедление возникает из-за накладных расходов на управление потоками и синхронизацию.

Код программы

main.c

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

#define MAX_THREADS 12
#define EOF -1

struct timespec start, end;

typedef struct {
    int thread_id;
    int num_threads;
    int num_arrays;
    int array_length;
    int* arrays;
    int* result;
} ThreadData;

int str_to_int(char* str, int* out) {
    int res = 0;
    int sign = 1;
    int i = 0;
    if (str[0] == '-') {
        sign = -1;
        i++;
    }
    for(; str[i] != '\0'; i++) {
        if(str[i] < '0' || str[i] > '9') {
            return -1;
        }
        res = res * 10 + (str[i] - '0');
    }
    *out = res * sign;
    return 0;
}

int int_to_str(int num, char* str) {
    int i = 0;
    int is_negative = 0;
    if(num == 0){
        str[i++] = '0';
        str[i] = '\0';
        return i;
    }
```

```
    if(num < 0){
        is_negative = 1;
        num = -num;
    }
    while(num > 0){
        str[i++] = (num % 10) + '0';
        num /= 10;
    }
    if(is_negative){
        str[i++] = '-';
    }
    for(int j = 0; j < i / 2; j++) {
        char tmp = str[j];
        str[j] = str[i - j -1];
        str[i - j -1] = tmp;
    }
    str[i] = '\0';
    return i;
}

int read_int(int fd, int* value) {
    char buffer[20];
    int idx = 0;
    char c;
    while(read(fd, &c, 1) == 1 && (c == ' ' || c == '\n' || c == '\t'));
    if(c == EOF || c == 0){
        return -1;
    }
    if (c == '-') {
        buffer[idx++] = c;
    } else if (c >= '0' && c <= '9') {
        buffer[idx++] = c;
    } else {
        return -1;
    }
    while(read(fd, &c, 1) == 1 && (c >= '0' && c <= '9')) {
        buffer[idx++] = c;
        if(idx >= 19) break;
    }
    buffer[idx] = '\0';
    if(!(c >= '0' && c <= '9')){
        lseek(fd, -1, SEEK_CUR);
    }
    if(str_to_int(buffer, value) != 0){
        return -1;
    }
    return 0;
}

void* sum_arrays(void* arg) {
    ThreadData* data = (ThreadData*)arg;
```

```
    int start = (data->thread_id * data->array_length) / data->num_threads;
    int end = ((data->thread_id + 1) * data->array_length) / data->num_threads;

    for(int i = start; i < end; i++) {
        int sum = 0;
        for(int j = 0; j < data->num_arrays; j++) {
            sum += data->arrays[j * data->array_length + i];
        }
        data->result[i] = sum;
    }

    return NULL;
}

int main(int argc, char* argv[]) {
    if(argc != 4) {
        char* msg = "Usage: ./sum_arrays <input_file> <num_threads> <output_file>\n";
        write(STDOUT_FILENO, msg, strlen(msg));
        return EXIT_FAILURE;
    }

    int N;
    if(str_to_int(argv[2], &N) != 0 || N <= 0 || N > MAX_THREADS) {
        char* msg = "Invalid number of threads. Must be between 1 and 8.\n";
        write(STDOUT_FILENO, msg, strlen(msg));
        return EXIT_FAILURE;
    }

    int fd_in = open(argv[1], O_RDONLY);
    if (fd_in == -1) {
        char* msg = "Failed to open input file.\n";
        write(STDOUT_FILENO, msg, strlen(msg));
        return EXIT_FAILURE;
    }

    int num_arrays, array_length;
    if(read_int(fd_in, &num_arrays) == -1 || read_int(fd_in, &array_length) == -1) {
        char* msg = "Failed to read array counts or lengths.\n";
        write(STDOUT_FILENO, msg, strlen(msg));
        close(fd_in);
        return EXIT_FAILURE;
    }

    int* arrays = (int*)malloc(num_arrays * array_length * sizeof(int));
    if(!arrays) {
        char* msg = "Memory allocation failed for arrays.\n";
        write(STDOUT_FILENO, msg, strlen(msg));
        close(fd_in);
        return EXIT_FAILURE;
    }
```

```
for(int i = 0; i < num_arrays; i++) {
    for(int j = 0; j < array_length; j++) {
        if(read_int(fd_in, &arrays[i * array_length + j]) == -1) {
            char* msg = "Failed to read array elements.\n";
            write(STDOUT_FILENO, msg, strlen(msg));
            free(arrays);
            close(fd_in);
            return EXIT_FAILURE;
        }
    }
}

close(fd_in);

int* result = (int*)malloc(array_length * sizeof(int));
if(!result) {
    char* msg = "Memory allocation failed for result array.\n";
    write(STDOUT_FILENO, msg, strlen(msg));
    free(arrays);
    return EXIT_FAILURE;
}
memset(result, 0, array_length * sizeof(int));

pthread_t* threads = (pthread_t*)malloc(N * sizeof(pthread_t));
ThreadData* threads_data = (ThreadData*)malloc(N * sizeof(ThreadData));
if(!threads || !threads_data){
    char* msg = "Memory allocation failed for threads.\n";
    write(STDOUT_FILENO, msg, strlen(msg));
    free(arrays);
    free(result);
    free(threads);
    free(threads_data);
    return EXIT_FAILURE;
}

for (int i = 0; i < N; i++) {
    threads_data[i].thread_id = i;
    threads_data[i].num_threads = N;
    threads_data[i].num_arrays = num_arrays;
    threads_data[i].array_length = array_length;
    threads_data[i].arrays = arrays;
    threads_data[i].result = result;
}

// Создание потоков
for (int i = 0; i < N; i++) {
    if(pthread_create(&threads[i], NULL, &sum_arrays, &threads_data[i]) != 0){
        char* msg = "Failed to create thread.\n";
        write(STDOUT_FILENO, msg, strlen(msg));
        free(arrays);
        free(result);
```

```
            free(threads);
            free(threads_data);
            return EXIT_FAILURE;
        }
    }

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

    int fd_out = open(argv[3], O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if(fd_out == -1){
        char* msg = "Failed to open output file.\n";
        write(STDOUT_FILENO, msg, strlen(msg));
        free(arrays);
        free(result);
        free(threads);
        free(threads_data);
        return EXIT_FAILURE;
    }

    char buffer[20];
    int len = 0;

    // Запись результирующего массива в файл
    for(int i = 0; i < array_length; i++) {
        len = int_to_str(result[i], buffer);
        write(fd_out, buffer, len);
        if(i != array_length -1){
            write(fd_out, " ", 1);
        }
    }
    write(fd_out, "\n", 1);

    close(fd_out);

    free(arrays);
    free(result);
    free(threads);
    free(threads_data);

    return EXIT_SUCCESS;
}
```

Протокол работы программы

Работа программы:

```
wabisabi@wabisabi:/mnt/c/Users/zlata/OneDrive/Рабочий стол/labs/sofa/os_l
ab2$ gcc -o main main.c -pthread
wabisabi@wabisabi:/mnt/c/Users/zlata/OneDrive/Рабочий стол/labs/sofa/os_l
ab2$ ./main data1.txt 4 result.txt
wabisabi@wabisabi:/mnt/c/Users/zlata/OneDrive/Рабочий стол/labs/sofa/os_lab2$
```

Data1.txt:

sofa > os_lab2 > data1.txt

```
1   6 5
2   1 2 3 4 5
3   1 2 3 4 5
4   1 2 3 4 5
5   1 2 3 4 5
6   1 2 3 4 5
7   1 2 3 4 5
```

Result.txt:

sofa > os_lab2 > result.txt

```
1   6 12 18 24 30
2
```

Strace:

```
strace ./main data1.txt 4 result.txt
execve("./main", ["./main", "data1.txt", "4", "result.txt"], 0x7fff7586d438 /* 35 vars */) = 0
brk(NULL)                               = 0x5613931eb000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f9f44e07000
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=20167, ...}) = 0
mmap(NULL, 20167, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f9f44e02000
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) = 0x7f9f44bf0000
mmap(0x7f9f44c18000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f9f44c18000
mmap(0x7f9f44da0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000)
= 0x7f9f44da0000
mmap(0x7f9f44def000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1fe000) = 0x7f9f44def000
mmap(0x7f9f44df5000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f9f44df5000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f9f44bed000
arch_prctl(ARCH_SET_FS, 0x7f9f44bed740) = 0
set_tid_address(0x7f9f44beda10)         = 35199
set_robust_list(0x7f9f44beda20, 24)     = 0
rseq(0x7f9f44bee060, 0x20, 0, 0x53053053) = 0
mprotect(0x7f9f44def000, 16384, PROT_READ) = 0
mprotect(0x561393141000, 4096, PROT_READ) = 0
mprotect(0x7f9f44e3f000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f9f44e02000, 20167)           = 0
openat(AT_FDCWD, "data1.txt", O_RDONLY) = 3
read(3, "6", 1)                         = 1
read(3, " ", 1)                         = 1
lseek(3, -1, SEEK_CUR)                  = 1
read(3, " ", 1)                         = 1
read(3, "5", 1)                         = 1
read(3, "\n", 1)                        = 1
lseek(3, -1, SEEK_CUR)                  = 3
getrandom("\x22\x58\xd1\xa1\x0f\x8c\x1a\xbc", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x5613931eb000
brk(0x56139320c000)                     = 0x56139320c000
read(3, "\n", 1)                        = 1
read(3, "1", 1)                         = 1
read(3, " ", 1)                         = 1
lseek(3, -1, SEEK_CUR)                  = 5
read(3, " ", 1)                         = 1
read(3, "2", 1)                         = 1
read(3, " ", 1)                         = 1
lseek(3, -1, SEEK_CUR)                  = 7
read(3, " ", 1)                         = 1
read(3, "3", 1)                         = 1
read(3, " ", 1)                         = 1
lseek(3, -1, SEEK_CUR)                  = 9
read(3, " ", 1)                         = 1
read(3, "4", 1)                         = 1
```

```
read(3, " ", 1)                 = 1
lseek(3, -1, SEEK_CUR)                  = 11
read(3, " ", 1)                 = 1
read(3, "5", 1)                  = 1
read(3, "\n", 1)                 = 1
lseek(3, -1, SEEK_CUR)                  = 13
read(3, "\n", 1)                 = 1
read(3, "1", 1)                  = 1
read(3, " ", 1)                 = 1
lseek(3, -1, SEEK_CUR)                  = 15
read(3, " ", 1)                 = 1
read(3, "2", 1)                  = 1
read(3, " ", 1)                 = 1
lseek(3, -1, SEEK_CUR)                  = 17
read(3, " ", 1)                 = 1
read(3, "3", 1)                  = 1
read(3, " ", 1)                 = 1
lseek(3, -1, SEEK_CUR)                  = 19
read(3, " ", 1)                 = 1
read(3, "4", 1)                  = 1
read(3, " ", 1)                 = 1
lseek(3, -1, SEEK_CUR)                  = 21
read(3, " ", 1)                 = 1
read(3, "5", 1)                  = 1
read(3, "\n", 1)                 = 1
lseek(3, -1, SEEK_CUR)                  = 23
read(3, "\n", 1)                 = 1
read(3, "1", 1)                  = 1
read(3, " ", 1)                 = 1
lseek(3, -1, SEEK_CUR)                  = 25
read(3, " ", 1)                 = 1
read(3, "2", 1)                  = 1
read(3, " ", 1)                 = 1
lseek(3, -1, SEEK_CUR)                  = 27
read(3, " ", 1)                 = 1
read(3, "3", 1)                  = 1
read(3, " ", 1)                 = 1
lseek(3, -1, SEEK_CUR)                  = 29
read(3, " ", 1)                 = 1
read(3, "4", 1)                  = 1
read(3, " ", 1)                 = 1
lseek(3, -1, SEEK_CUR)                  = 31
read(3, " ", 1)                 = 1
read(3, "5", 1)                  = 1
read(3, "\n", 1)                 = 1
lseek(3, -1, SEEK_CUR)                  = 33
read(3, "\n", 1)                 = 1
read(3, "1", 1)                  = 1
read(3, " ", 1)                 = 1
lseek(3, -1, SEEK_CUR)                  = 35
read(3, " ", 1)                 = 1
read(3, "2", 1)                  = 1
read(3, " ", 1)                 = 1
lseek(3, -1, SEEK_CUR)                  = 37
read(3, " ", 1)                 = 1
read(3, "3", 1)                  = 1
read(3, " ", 1)                 = 1
lseek(3, -1, SEEK_CUR)                  = 39
read(3, " ", 1)                 = 1
read(3, "4", 1)                  = 1
read(3, " ", 1)                 = 1
lseek(3, -1, SEEK_CUR)                  = 41
read(3, " ", 1)                 = 1
```

read(3, "5", 1) = 1
read(3, "\n", 1) = 1
lseek(3, -1, SEEK_CUR) = 43
read(3, "\n", 1) = 1
read(3, "1", 1) = 1
read(3, " ", 1) = 1
lseek(3, -1, SEEK_CUR) = 45
read(3, " ", 1) = 1
read(3, "2", 1) = 1
read(3, " ", 1) = 1
lseek(3, -1, SEEK_CUR) = 47
read(3, " ", 1) = 1
read(3, "3", 1) = 1
read(3, " ", 1) = 1
lseek(3, -1, SEEK_CUR) = 49
read(3, " ", 1) = 1
read(3, "4", 1) = 1
read(3, " ", 1) = 1
lseek(3, -1, SEEK_CUR) = 51
read(3, " ", 1) = 1
read(3, "5", 1) = 1
read(3, "\n", 1) = 1
lseek(3, -1, SEEK_CUR) = 53
read(3, "\n", 1) = 1
read(3, "1", 1) = 1
read(3, " ", 1) = 1
lseek(3, -1, SEEK_CUR) = 55
read(3, " ", 1) = 1
read(3, "2", 1) = 1
read(3, " ", 1) = 1
lseek(3, -1, SEEK_CUR) = 57
read(3, " ", 1) = 1
read(3, "3", 1) = 1
read(3, " ", 1) = 1
lseek(3, -1, SEEK_CUR) = 59
read(3, " ", 1) = 1
read(3, "4", 1) = 1
read(3, " ", 1) = 1
lseek(3, -1, SEEK_CUR) = 61
read(3, " ", 1) = 1
read(3, "5", 1) = 1
read(3, "", 1) = 0
close(3) = 0
rt_sigaction(SIGRT_1, {sa_handler=0x7f9f44c89520, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x7f9f44c35320}, 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) =
0x7f9f443ec000
mprotect(0x7f9f443ed000, 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_SYSVS
EM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f9f44bec990,
parent_tid=0x7f9f44bec990, exit_signal=0, stack=0x7f9f443ec000, stack_size=0x7fff80, tls=0x7f9f44bec6c0} =>
{parent_tid=[35209]}, 88) = 35209
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f9f43beb000
mprotect(0x7f9f43bec000, 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_SYSVS
EM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f9f443eb990,
parent_tid=0x7f9f443eb990, exit_signal=0, stack=0x7f9f43beb000, stack_size=0x7fff80, tls=0x7f9f443eb6c0} =>

{parent_tid=[35210]}, 88) = 35210
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f9f433ea000
mprotect(0x7f9f433eb000, 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=0x7f9f43bea990, parent_tid=0x7f9f43bea990, exit_signal=0, stack=0x7f9f433ea000, stack_size=0x7fff80, tls=0x7f9f43bea6c0} => {parent_tid=[0]}, 88) = 35211
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f9f42be9000
mprotect(0x7f9f42bea000, 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=0x7f9f433e9990, parent_tid=0x7f9f433e9990, exit_signal=0, stack=0x7f9f42be9000, stack_size=0x7fff80, tls=0x7f9f433e96c0} => {parent_tid=[0]}, 88) = 35212
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
openat(AT_FDCWD, "result.txt", O_WRONLY|O_CREAT|O_TRUNC, 0644) = 3
write(3, "6", 1) = 1
write(3, " ", 1) = 1
write(3, "12", 2) = 2
write(3, " ", 1) = 1
write(3, "18", 2) = 2
write(3, " ", 1) = 1
write(3, "24", 2) = 2
write(3, " ", 1) = 1
write(3, "30", 2) = 2
write(3, "\n", 1) = 1
close(3) = 0
exit_group(0) = ?
+++ exited with 0 +++

Вывод

Программа реализует многопоточное суммирование элементов нескольких массивов, считывая данные из входного файла и записывая результат в выходной файл. Основная цель программы — продемонстрировать ускорение вычислений за счет использования многопоточности и оценить эффективность параллельного выполнения. Язык Си с библиотеками предоставляет возможность построения многопоточных приложений, дает инструменты для работы с потоками, их ограничения для безопасности. Все это делает разработку на Си многообразнее и интереснее.