

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: М80-206Б-22
Студентка: Коломытцева Е. А.
Преподаватель: Миронов Е.С.
Оценка: _____
Дата: 28.12.23

Москва, 2023

Постановка задачи

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке 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. Также 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.

Вариант 8

Есть K 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

Используемые системные вызовы:

- **pthread_create(&threadID, NULL, pfunction, &pdata))** - создает новый поток с номером threadID, запускает потоковую функцию pfunction и передает в нее данные pdata.
- **pthread_join(threadID, NULL)** - ждет завершение текущего потока.
- **pthread_exit(0)** - завершает вызывающий поток. Функция призвана обеспечить потокобезопасность.

Программа состоит из 1 файла main.c

Команды для запуска программы:

gcc main.c -o main

./main n, где n - количество потоков

Пример ввода и вывода:

./main 2

length of arrays N: 3

number of arrays K: 4

1 2 3

3 4 5

6 5 4

5 6 7

horizontal

15 17 19

./main 4

length of arrays N: 5

number of arrays K: 2

1 2 3 4 5

6 5 4 3 2

vertical

7 7 7 7 7

Описание программы:

Данный код представляет собой программу на языке программирования C, которая выполняет сложение массивов. Программа имеет возможность адаптироваться к количеству массивов и их длине, выбирая стратегию сложения в зависимости от соотношения количества массивов и их длины.

Программа использует многопоточность для ускорения вычислений. Есть две функции для суммирования массивов: `vertical_sum_arrays` и `horizontal_sum_arrays`. Выбор между ними осуществляется на основе соотношения длины массивов (N) и их количества (K). Если N больше чем $K * 2$, используется вертикальное сложение (`vertical_sum_arrays`), иначе горизонтальное (`horizontal_sum_arrays`). При горизонтальном сложении используется мьютекс для синхронизации.

Программа принимает аргумент командной строки для указания количества потоков. Затем запрашивает пользователя ввод длины массивов N и количества массивов K. После этого пользователь должен ввести значения элементов массивов.

Далее программа создает потоки для выполнения сложения и ждет, пока все потоки завершат свою работу. Результат сложения выводится на экран.

Код программы

main.c

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

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

pthread_mutex_t mutex;

typedef struct {
    int** arr;
    int* res;
    int start;
    int steps;
    int K;
    int N;
} ThreadToken;

int min(int a, int b) {
    if (a < b) return a;
    return b;
}
```

```

void exit_with_msg(const char* msg, int return_code) {
    printf("%s\n", msg);
    exit(return_code);
}

void* vertical_sum_arrays(void* arg) {
    ThreadToken token = *((ThreadToken*)arg);
    // printf("(%d;%d)\n", token.start, token.start + token.steps);
    for (int i = token.start; i < token.start + token.steps; ++i) {
        int c = 0;
        for (int j = 0; j < token.K; ++j) {
            c += token.arr[j][i];
        }
        token.res[i] = c;
    }
    return arg;
}

void* horizontal_sum_arrays(void* arg) {
    ThreadToken token = *((ThreadToken*)arg);
    // printf("(%d;%d)\n", token.start, token.start + token.steps);
    for (int i = 0; i < token.N; ++i) {
        int c = 0;
        for (int j = token.start; j < token.start + token.steps; ++j) {
            c += token.arr[j][i];
        }
        pthread_mutex_lock(&mutex);
        token.res[i] += c;
        pthread_mutex_unlock(&mutex);
    }
    return arg;
}

int main(int argc, const char** argv) {
    int CountThreads = 0;
    if (argc < 2) {
        exit_with_msg("missing arguments", -1);
    }
    // str to int
    for (int i = 0; argv[1][i] > 0; ++i) {
        if (argv[1][i] >= '0' && argv[1][i] <= '9') {
            CountThreads = CountThreads * 10 + argv[1][i] - '0';
        }
    }
    int N, K;
    printf("length of arrays N: ");
    if (scanf("%d", &N) == EOF) {
        exit_with_msg("data cannot be read", 1);
    }
}

```

```

printf("number of arrays K: ");
if (scanf("%d", &K) == EOF) {
    exit_with_msg("data cannot be read", 1);
}
int** all = malloc(sizeof(int*) * K);
if (all == NULL) {
    exit_with_msg("cannot allocate memory", -5);
}
for (int i = 0; i < K; ++i) {
    all[i] = malloc(sizeof(int) * N);
    if (all[i] == NULL) {
        for (int j = 0; j < i; ++j) {
            free(all[j]);
            all[j] = NULL;
        }
        free(all);
        all = NULL;
        exit_with_msg("cannot allocate memory", 1);
    }
    for (int j = 0; j < N; ++j) {
        if (scanf("%d", &all[i][j]) == EOF) {
            exit_with_msg("data cannot be read", 1);
        }
    }
}

void* (*function)(void*);
int end;
// some kind of ratio of N and K to choose sum way
if (N > K * 2) {
    function = &vertical_sum_arrays;
    end = N;
    printf("vertical\n");
} else {
    function = &horizontal_sum_arrays;
    end = K;
    printf("horizontal\n");
    // only in horizontal mode we need to use mutex
    if (pthread_mutex_init(&mutex, NULL) != 0) {
        exit_with_msg("cannot init mutex", 1);
    }
}
// if we have only 5 arrays and 10 threads?
CountThreads = min(CountThreads, end);

// create arrays of threads and tokens
pthread_t* th = malloc(sizeof(pthread_t) * CountThreads);
ThreadToken* token = malloc(sizeof(ThreadToken) * CountThreads);

```

```

// result will be placed here
int* result = malloc(sizeof(int) * N);

if (th == NULL || token == NULL || result == NULL) {
    exit_with_msg("cannot allocate memory", 1);
}
// init result with 0 value
for (int i = 0; i < N; ++i) {
    result[i] = 0;
}
// Start and End indexes for each thread
int start = 0;
int steps = (end + CountThreads - 1) / CountThreads;
// fill token data for each thread
for (int i = 0; i < CountThreads; ++i) {
    token[i].arr = all;
    token[i].res = result;
    token[i].start = start;
    token[i].K = K;
    token[i].N = N;
    token[i].steps = min(end - start, steps);
    start += steps;
}

// start threads
for (int i = 0; i < CountThreads; ++i) {
    if (pthread_create(&th[i], NULL, function, &token[i]) != 0) {
        exit_with_msg("cannot create thread", 2);
    }
}
// join threads (wait for all of them to end calculations)
for (int i = 0; i < CountThreads; ++i) {
    if (pthread_join(th[i], NULL) != 0) {
        exit_with_msg("cannot join threads", 3);
    }
}
for (int i = 0; i < N; ++i) {
    printf("%d ", result[i]);
}
printf("\n");
for (int i = 0; i < K; ++i) {
    free(all[i]);
    all[i] = NULL;
}
if (end == K) {
    // if we really have init it
    pthread_mutex_destroy(&mutex);
}
free(all);

```

```
free(token);
free(th);
free(result);
return 0;
```

Протокол работы программы

Тестирование:

Посмотрим на зависимость времени работы программы от количества потоков:

./a.out 2

length of arrays N: 2

number of arrays K: 3

1 2

3 4

5 6

horizontal

Execution time: 0.000216 seconds

9 12

./a.out 3

length of arrays N: 2

number of arrays K: 3

1 2

3 4

5 6

horizontal

Execution time: 0.000309 seconds

9 12

./a.out 4

length of arrays N: 2

number of arrays K: 3

1 2

3 4

5 6

horizontal

Execution time: 0.000353 seconds

9 12

Видим, что при увеличении количества потоков увеличивается и время выполнения работы, однако мы ждали обратное. Тем самым можно сказать, что в данной программе использование потоков является неэффективным подходом, так как их обслуживание только увеличивает время работы программы.

Strace:

```

$ strace -f ./main

execve("./main", [ "./main", "2"], 0x7ffd2d985720 /* 47 vars */) = 0

brk(NULL)                                = 0x55b21ad69000

arch_prctl(0x3001 /* ARCH_??? */, 0x7fff360e4370) = -1 EINVAL (Недопустимый аргумент)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fb302c55000

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

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=119923, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 119923, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fb302c37000

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\0\3\0>\0\1\0\0\0P<\2\0\0\0\0"... , 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"... , 784, 64)
= 784

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=2072888, ...}, AT_EMPTY_PATH) = 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, 2117488, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fb302a00000

mmap(0x7fb302a22000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x22000) = 0x7fb302a22000

mmap(0x7fb302b9a000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x19a000) = 0x7fb302b9a000

mmap(0x7fb302bf2000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1f1000) = 0x7fb302bf2000

mmap(0x7fb302bf8000, 53104, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x7fb302bf8000

close(3)                                  = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fb302c34000

arch_prctl(ARCH_SET_FS, 0x7fb302c34740) = 0

set_tid_address(0x7fb302c34a10)           = 25689

set_robust_list(0x7fb302c34a20, 24)       = 0

rseq(0x7fb302c35060, 0x20, 0, 0x53053053) = 0

mprotect(0x7fb302bf2000, 16384, PROT_READ) = 0

mprotect(0x55b219bd8000, 4096, PROT_READ) = 0

mprotect(0x7fb302c8a000, 8192, PROT_READ) = 0

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

munmap(0x7fb302c37000, 119923)            = 0

```



```

newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x1), ...},
AT_EMPTY_PATH) = 0

getrandom("\x4b\x6e\x76\xba\xc4\x9a\xf8\x91", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x55b21ad69000

brk(0x55b21ad8a000) = 0x55b21ad8a000

newfstatat(0, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x1), ...},
AT_EMPTY_PATH) = 0

write(1, "length of arrays N: ", 20length of arrays N: ) = 20

read(0, 3
"3\n", 1024) = 2

write(1, "number of arrays K: ", 20number of arrays K: ) = 20

read(0, 4
"4\n", 1024) = 2

read(0, 1 2 3
"1 2 3\n", 1024) = 6

read(0, 3 4 5
"3 4 5\n", 1024) = 6

read(0, 6 5 4
"6 5 4\n", 1024) = 6

read(0, 5 6 7
"5 6 7\n", 1024) = 6

write(1, "horizontal\n", 11horizontal
) = 11

rt_sigaction(SIGRT_1, {sa_handler=0x7fb302a8c450, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x7fb302a3c460}, 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) =
0x7fb3021ff000

mprotect(0x7fb302200000, 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|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7fb3029ff990,
parent_tid=0x7fb3029ff990, exit_signal=0, stack=0x7fb3021ff000, stack_size=0x7fff80,
tls=0x7fb3029ff6c0}strace: Process 25737 attached

=> {parent_tid=[25737]}, 88) = 25737

[pid 25737] rseq(0x7fb3029fffe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 25689] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>

[pid 25737] <... rseq resumed> = 0

```

```

[pid 25689] <... rt_sigprocmask resumed>NULL, 8) = 0
[pid 25737] set_robust_list(0x7fb3029ff9a0, 24 <unfinished ...>
[pid 25689] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0
<unfinished ...>
[pid 25737] <... set_robust_list resumed>) = 0
[pid 25689] <... mmap resumed>          = 0x7fb3019fe000
[pid 25737] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>
[pid 25689] mprotect(0x7fb3019ff000, 8388608, PROT_READ|PROT_WRITE <unfinished ...>
[pid 25737] <... rt_sigprocmask resumed>NULL, 8) = 0
[pid 25689] <... mprotect resumed>      = 0
[pid 25689] rt_sigprocmask(SIG_BLOCK, ~[], <unfinished ...>
[pid 25737] rt_sigprocmask(SIG_BLOCK, ~[RT_1], <unfinished ...>
[pid 25689] <... rt_sigprocmask resumed>[], 8) = 0
[pid 25737] <... rt_sigprocmask resumed>NULL, 8) = 0
[pid 25689]
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_S
ETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEAR_TID, child_tid=0x7fb3021fe990,
parent_tid=0x7fb3021fe990, exit_signal=0, stack=0x7fb3019fe000, stack_size=0x7fff80,
tls=0x7fb3021fe6c0} <unfinished ...>
[pid 25737] madvise(0x7fb3021ff000, 8368128, MADV_DONTNEED) = 0
strace: Process 25738 attached
[pid 25689] <... clone3 resumed> => {parent_tid=[25738]}, 88) = 25738
[pid 25737] exit(0 <unfinished ...>
[pid 25689] rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>
[pid 25737] <... exit resumed>          = ?
[pid 25689] <... rt_sigprocmask resumed>NULL, 8) = 0
[pid 25738] rseq(0x7fb3021fefe0, 0x20, 0, 0x53053053 <unfinished ...>
[pid 25689] futex(0x7fb3021fe990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 25738, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
[pid 25737] +++ exited with 0 +++
[pid 25738] <... rseq resumed>          = 0
[pid 25738] set_robust_list(0x7fb3021fe9a0, 24) = 0
[pid 25738] rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
[pid 25738] rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0
[pid 25738] madvise(0x7fb3019fe000, 8368128, MADV_DONTNEED) = 0
[pid 25738] exit(0)                    = ?
[pid 25689] <... futex resumed>        = 0
[pid 25738] +++ exited with 0 +++

```

```

write(1, "Execution time: 0.001985 seconds"... , 33Execution time: 0.001985 seconds
) = 33
write(1, "15 17 19 \n", 1015 17 19
) = 10
lseek(0, -1, SEEK_CUR) = -1 ESPIPE (Недопустимая операция смещения)
exit_group(0) = ?
+++ exited with 0 +++

```

Вывод

В данной лабораторной работе была разработана программа на языке программирования C, предназначенная для сложения массивов различной длины. Программа способна адаптироваться под количество массивов и выбирать стратегию сложения в зависимости от соотношения длины массивов и их количества. Для ускорения вычислений использована многопоточность с возможностью выбора между вертикальным и горизонтальным методами сложения.

Однако, важно отметить, что в данном случае использование многопоточности не всегда приводит к улучшению производительности. Наоборот, при увеличении числа потоков наблюдается увеличение времени выполнения программы. Это может быть связано с избыточным использованием ресурсов, так как создание и управление потоками требует определенных затрат.

Эффективность использования многопоточности зависит от конкретной задачи, характеристик системы и реализации алгоритма.