**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ  
ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)**

**ЛАБОРАТОРНАЯ РАБОТА №3**

**по курсу операционные системы I семестр, 2021/22  
уч. год**

Студент: Белоусов Егор Владимирович

Группа: М8О-207Б-20

Преподаватель: Миронов Е. С.

Вариант: 17

Дата:

Оценка:

**Цель работы:**

Приобретение практических навыков в:

* Управление потоками в ОС
* Обеспечение синхронизации между потоками

**Введение**

*Вариант 17. Найти в большом целочисленном массиве минимальный элемент*

Программе подается 2 ключа N и T. Первое – размер входных данных, второе – количество потоков. Входные данные представляются в виде массива размера N, затем массив разбивается на T блоков размера N / T каждый. В случае, если N не делится на K нацело, то в последний блоков добавляются еще все дополнительные элементы. Затем каждый поток находит минимум в своем полуинтервале . Понятно, что в таком случае не будет с проблем сихронизацией, так как полуинтервалы не пересекаются. В конце просто берем минимум из всех T полуинтервалов.

**Системные вызовы**

**int pthread\_create(pthread\_t \*restrict** *thread***,**

**const pthread\_attr\_t \*restrict** *attr***,**

**void \*(\****start\_routine***)(void \*),**

**void \*restrict** *arg***);** - создает новый поток в вызывающем процессе, в *\*thread* записывается ID созданного потока, *attr* задает атрибуты для создания процесса(например приоритет или размер стэка), *\*start\_routine* представляет собой функцию, которую вызывает поток, *arg* соответственно аргументы для функции.

**int pthread\_join(pthread\_t** *thread***, void \*\****retval***);** - ожидает, пока поток *thread* завершится, в *\*\*retval* записывается статус выхода потока (только если *retval != NULL)*.

**Исходный код**

**main.c**

#include "stdio.h"

#include "stdlib.h"

#include "string.h"

#include "pthread.h"

#include "time.h"

int min(int a, int b) {

    return (a < b ? a : b);

}

typedef struct \_\_args {

    int \*values;

    int l, r;

    int \*result;

} args\_t;

void \*routine(void \*\_args) {

    args\_t \*args = (args\_t \*) \_args;

    int \*values = args->values;

    int l = args->l, r = args->r;

    int \*res = args->result;

    int flag = 1;

    for (int i = l; i < r; ++i) {

        if (flag) {

            \*res = values[i];

            flag = 0;

        } else {

            \*res = min(\*res, values[i]);

        }

    }

    return NULL;

}

int main(int argc, char \*argv[]) {

    if (argc != 3) {

        printf("Not enough values\n");

        exit(1);

    }

    const int ARRAY\_SIZE = atoi(argv[1]);

    const int THREAD\_NUMBER = min(ARRAY\_SIZE, atoi(argv[2]));

    const int BLOCK\_SIZE = ARRAY\_SIZE / THREAD\_NUMBER;

    int \*values = (int \*) calloc(ARRAY\_SIZE, sizeof(int));

    pthread\_t \*threads = (pthread\_t \*) calloc(THREAD\_NUMBER, sizeof(pthread\_t));

    args\_t \*args = (args\_t \*) calloc(THREAD\_NUMBER, sizeof(args\_t));

    for (int i = 0; i < ARRAY\_SIZE; ++i) {

        scanf("%d", &values[i]);

    }

    for (int i = 0; i < THREAD\_NUMBER; ++i) {

        args[i].result = (int \*) calloc(1, sizeof(int));

        args[i].values = values;

        args[i].l = i \* BLOCK\_SIZE;

        if (i == THREAD\_NUMBER - 1) {

            args[i].r = ARRAY\_SIZE;

        } else {

            args[i].r = BLOCK\_SIZE \* (i + 1);

        }

        pthread\_create(&threads[i], NULL, routine, (void \*) &args[i]);

    }

    int result = -1;

    int flag = 1;

    for (int i = 0; i < THREAD\_NUMBER; ++i) {

        pthread\_join(threads[i], NULL);

        if (flag) {

            result = \*(args[i].result);

            flag = 0;

        } else {

            result = min(result, \*(args[i].result));

        }

    }

    printf("Result is: %d\n", result);

    free(values);

    free(threads);

    free(args);

    return 0;

}

**Пример работы**

*egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab3$ gcc -pthread main.c -o main*

*egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab3$ ./main 2*

*Not enough values*

*egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab3$ ./main 5 1*

*3 2 1 2 3*

*Result is: 1*

*egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab3$ ./main 5 5*

*2 2 2 2 2*

*Result is: 2*

*egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab3$ ./main 10 2*

*0 2 3 100 100 1000 10 5 5 5 10*

*Result is: 0*

*egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab3$ ./main 10 2*

*10 10 5 5 5 2 3 0 100 1000*

*Result is: 0*

*egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab3$ ./main 10 3*

*10 10 5 5 5 2 3 0 100 1000*

*Result is: 0*

*egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab3$ ./main 3 2*

*100 100 100*

*Result is: 100*

*egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab3$ ./main 3 2*

*1000 200 5000*

*Result is: 200*

*egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab3$ ./main 3 2*

*200 1000 5000*

*Result is: 200*

*egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab3$ ./main 3 2*

*1000 5000 200*

*Result is: 200*

**Ускорение и эффективность.**

Для демонстрации эффективности многопоточной программы я использовал очень большие тесты, состоящие из элементов. Стоит отметить, что я запускал тесты используя WSL, ввиду этого эффективность и ускорение может быть многим больше, если запускать программу на основной системе UNIX.

|  |  |  |  |
| --- | --- | --- | --- |
| **T** | **Время выполнения** | **Ускорение** | **Эффективность** |
| 1 | 180 ms | 1.000 | 1.000 |
| 2 | 140 ms | 1.286 | 0.643 |
| 4 | 80 ms | 2.250 | 0.563 |
| 6 | 70 ms | 2.571 | 0.429 |
| 8 | 80 ms | 2.250 | 0.281 |
| 10 | 70 ms | 2.571 | 0.257 |
| 12 | 100 ms | 1.80 | 0.150 |
| 14 | 80 ms | 2.250 | 0.161 |
| 16 | 60 ms | 3.000 | 0.188 |

Видно, что после 6 потоков ускорение перестало расти и стало более случайным, это можно объяснить тем, что на моем устройстве 4 ядра и 8 потоков.

**Strace**

В данной лабораторной работе нас больше всего интересуют системные вызовы, связанные с потоками.

*egorb@LAPTOP-TIFMO0IA:/mnt/d/os/lab3$ strace -f ./main 10 2*

*execve("./main", ["./main", "10", "2"], 0x7ffe98b236e8 /\* 29 vars \*/) = 0*

*brk(NULL) = 0x558923f36000*

*arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffdbb26dc00) = -1 EINVAL (Invalid argument)*

*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=35431, ...}) = 0*

*mmap(NULL, 35431, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f85ec9ef000*

*close(3) = 0*

*openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libpthread.so.0", O\_RDONLY|O\_CLOEXEC) = 3*

*read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\201\0\0\0\0\0\0"..., 832) = 832*

*pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\345Ga\367\265T\320\374\301V)Yf]\223\337"..., 68, 824) = 68*

*fstat(3, {st\_mode=S\_IFREG|0755, st\_size=157224, ...}) = 0*

*mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f85ec9ed000*

*pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\345Ga\367\265T\320\374\301V)Yf]\223\337"..., 68, 824) = 68*

*mmap(NULL, 140408, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f85ec9ca000*

*mmap(0x7f85ec9d1000, 69632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x7000) = 0x7f85ec9d1000*

*mmap(0x7f85ec9e2000, 20480, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x18000) = 0x7f85ec9e2000*

*mmap(0x7f85ec9e7000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1c000) = 0x7f85ec9e7000*

*mmap(0x7f85ec9e9000, 13432, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f85ec9e9000*

*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\360q\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*

*pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32*

*pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\t\233\222%\274\260\320\31\331\326\10\204\276X>\263"..., 68, 880) = 68*

*fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2029224, ...}) = 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*

*pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32*

*pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\t\233\222%\274\260\320\31\331\326\10\204\276X>\263"..., 68, 880) = 68*

*mmap(NULL, 2036952, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f85ec7d8000*

*mprotect(0x7f85ec7fd000, 1847296, PROT\_NONE) = 0*

*mmap(0x7f85ec7fd000, 1540096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x25000) = 0x7f85ec7fd000*

*mmap(0x7f85ec975000, 303104, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x19d000) = 0x7f85ec975000*

*mmap(0x7f85ec9c0000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1e7000) = 0x7f85ec9c0000*

*mmap(0x7f85ec9c6000, 13528, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f85ec9c6000*

*close(3) = 0*

*mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f85ec7d5000*

*arch\_prctl(ARCH\_SET\_FS, 0x7f85ec7d5740) = 0*

*mprotect(0x7f85ec9c0000, 12288, PROT\_READ) = 0*

*mprotect(0x7f85ec9e7000, 4096, PROT\_READ) = 0*

*mprotect(0x558923423000, 4096, PROT\_READ) = 0*

*mprotect(0x7f85eca25000, 4096, PROT\_READ) = 0*

*munmap(0x7f85ec9ef000, 35431) = 0*

*set\_tid\_address(0x7f85ec7d5a10) = 722*

*set\_robust\_list(0x7f85ec7d5a20, 24) = 0*

*rt\_sigaction(SIGRTMIN, {sa\_handler=0x7f85ec9d1bf0, sa\_mask=[], sa\_flags=SA\_RESTORER|SA\_SIGINFO, sa\_restorer=0x7f85ec9df3c0}, NULL, 8) = 0*

*rt\_sigaction(SIGRT\_1, {sa\_handler=0x7f85ec9d1c90, sa\_mask=[], sa\_flags=SA\_RESTORER|SA\_RESTART|SA\_SIGINFO, sa\_restorer=0x7f85ec9df3c0}, NULL, 8) = 0*

*rt\_sigprocmask(SIG\_UNBLOCK, [RTMIN RT\_1], NULL, 8) = 0*

*prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0*

*brk(NULL) = 0x558923f36000*

*brk(0x558923f57000) = 0x558923f57000*

*fstat(0, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0x1), ...}) = 0*

*read(0,*

*2 5 3 1 4 6 6 6 6 6 "\n", 1024) = 1*

*read(0,*

*"2 5 3 1 4 6 6 6 6 6 \n", 1024) = 21*

*mmap(NULL, 8392704, PROT\_NONE, MAP\_PRIVATE|MAP\_ANONYMOUS|MAP\_STACK, -1, 0) = 0x7f85ebfd4000*

*mprotect(0x7f85ebfd5000, 8388608, PROT\_READ|PROT\_WRITE) = 0*

*clone(child\_stack=0x7f85ec7d3fb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTIDstrace: Process 723 attached*

*, parent\_tid=[723], tls=0x7f85ec7d4700, child\_tidptr=0x7f85ec7d49d0) = 723*

*[pid 723] set\_robust\_list(0x7f85ec7d49e0, 24 <unfinished ...>*

*[pid 722] mmap(NULL, 8392704, PROT\_NONE, MAP\_PRIVATE|MAP\_ANONYMOUS|MAP\_STACK, -1, 0 <unfinished ...>*

*[pid 723] <... set\_robust\_list resumed>) = 0*

*[pid 722] <... mmap resumed>) = 0x7f85eb7d3000*

*[pid 723] madvise(0x7f85ebfd4000, 8368128, MADV\_DONTNEED <unfinished ...>*

*[pid 722] mprotect(0x7f85eb7d4000, 8388608, PROT\_READ|PROT\_WRITE <unfinished ...>*

*[pid 723] <... madvise resumed>) = 0*

*[pid 722] <... mprotect resumed>) = 0*

*[pid 723] exit(0 <unfinished ...>*

*[pid 722] clone(child\_stack=0x7f85ebfd2fb0, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID <unfinished ...>*

*[pid 723] <... exit resumed>) = ?*

*strace: Process 724 attached*

*[pid 723] +++ exited with 0 +++*

*[pid 722] <... clone resumed>, parent\_tid=[724], tls=0x7f85ebfd3700, child\_tidptr=0x7f85ebfd39d0) = 724*

*[pid 724] set\_robust\_list(0x7f85ebfd39e0, 24 <unfinished ...>*

*[pid 722] futex(0x7f85ebfd39d0, FUTEX\_WAIT, 724, NULL <unfinished ...>*

*[pid 724] <... set\_robust\_list resumed>) = 0*

*[pid 724] madvise(0x7f85eb7d3000, 8368128, MADV\_DONTNEED) = 0*

*[pid 724] exit(0) = ?*

*[pid 722] <... futex resumed>) = 0*

*[pid 724] +++ exited with 0 +++*

*fstat(1, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0x1), ...}) = 0*

*write(1, "Result is: 1\n", 13Result is: 1*

*) = 13*

*lseek(0, -2, SEEK\_CUR) = -1 ESPIPE (Illegal seek)*

*exit\_group(0) = ?*

*+++ exited with 0 +++*

**Выводы**

В данной лабораторной работе я научился работать с потоками на ОС UNIX. Я понял, что разделение программы на более мелкие части является чаще всего более оптимальным вариантом, чем исполнение всего в одной большой программе. Использование потоков позволяет ускорить работу даже в однопоточном режиме, так как во время простоя одного потока, другой может выполнять свою работу. В случае многопоточного варианта ускорение является намного более наглядным и очевидным.