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-210Б-23

Студент: Мишин С.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 28.11.24

Москва, 2024

Постановка задачи

Вариант 8.

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.

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

Использованные системные вызовы:

- int write(int filedes, const void *buf, size_t nbyte); – записывает данные из буфера в файловый дескриптор.
- int read(int filedes, void *buf, size_t nbyte); – читает данные из файлового дескриптора в буфер.
- int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void *), void *arg); – создаёт новый поток.

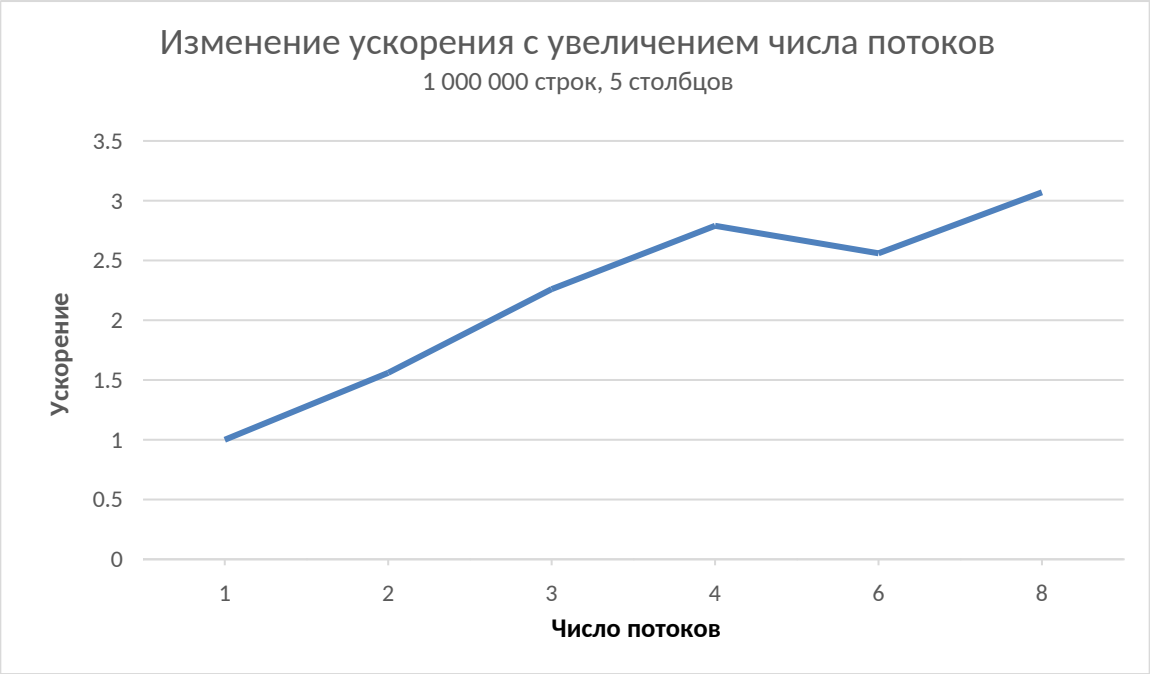
void

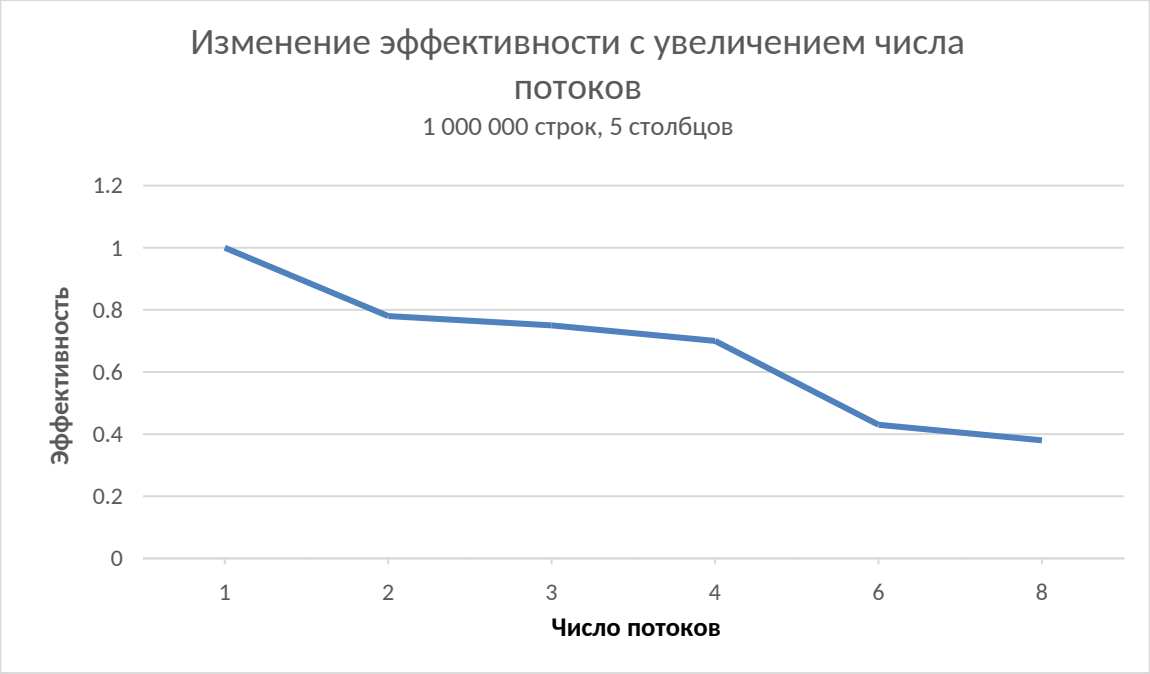
- void pthread_exit(void *value_ptr); – завершает поток.
- int pthread_join(pthread_t thread, void **value_ptr); – ожидает завершения потока.

Программа принимает на вход число строк k и ограничение числа создаваемых потоков, считывает массивы из стандартного ввода. Рассчитывается соотношение между количеством строк и столбцов. Если строк больше, запускается алгоритм разбиения массивов по строкам. Если столбцов больше, массивы разбиваются по столбцам. Каждый поток работает над своим разбиением. Программа ожидает завершения всех потоков и выводит результат суммирования.

Программа была протестирована на следующих входных данных: 1 000 000 массивов из 5 случайных элементов:

Число потоков	**Время исполнения (мс)**	**Ускорение**	**Эффективность**
1	12.43	1	1
2	7.96	1.56	0.78
3	5.5	2.26	0.75
4	4.46	2.79	0.7
6	4.86	2.56	0.43
8	4.05	3.07	0.38



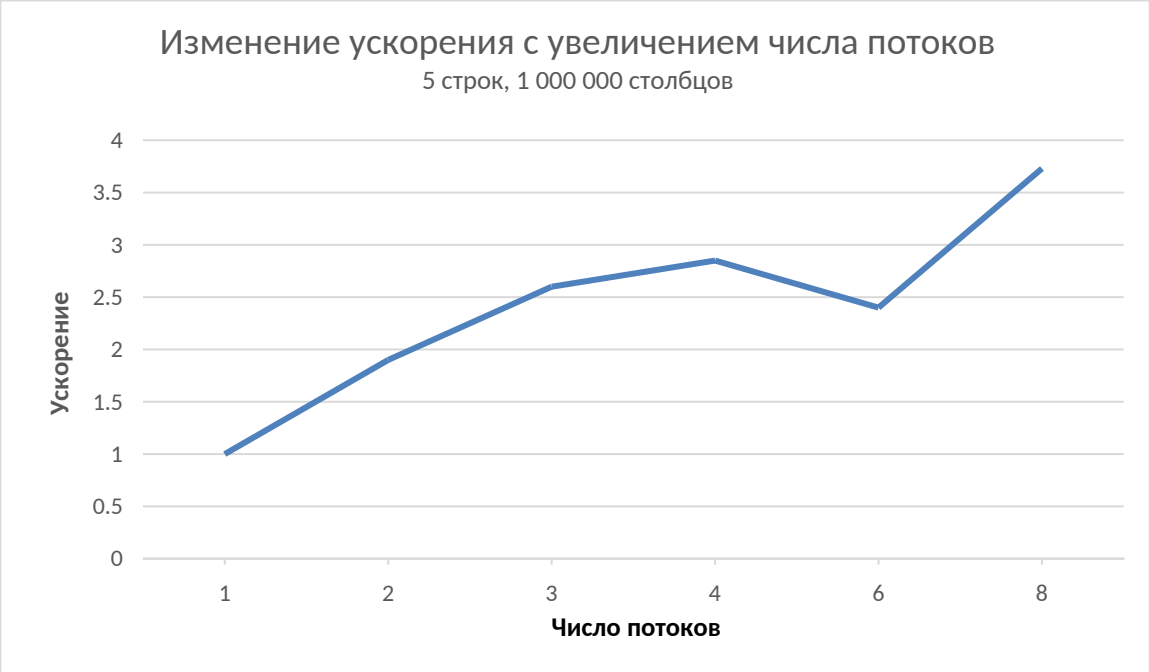


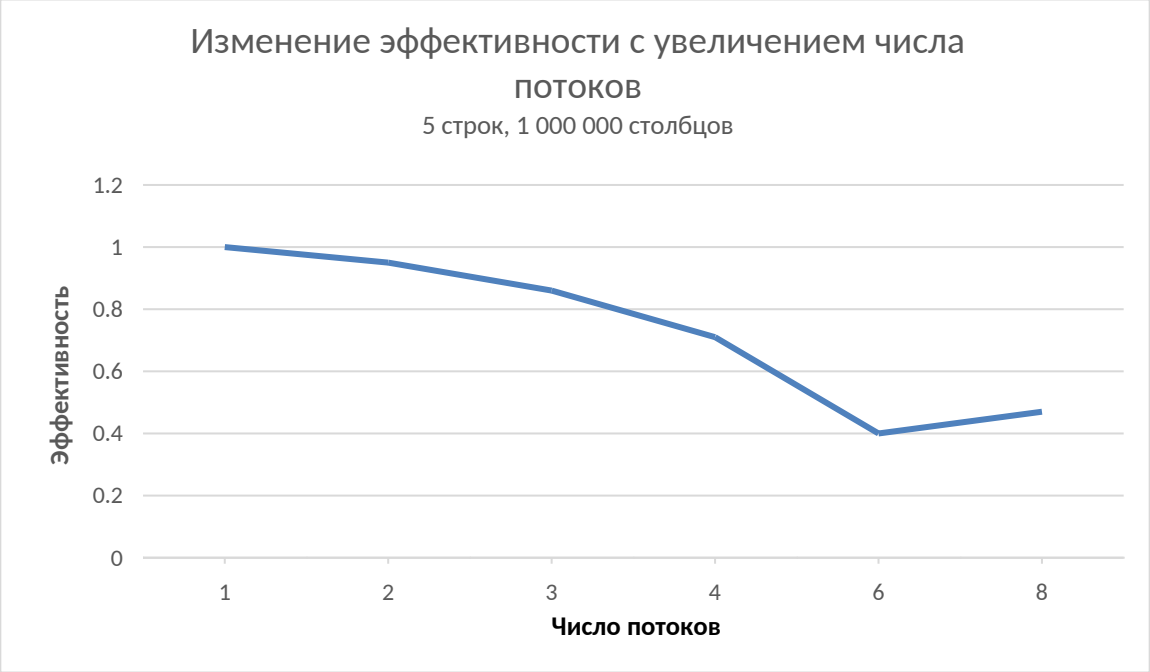
Ожидаемо, ускорение выросло с увеличением числа потоков. Но с пятью и более потоками темп роста ускорения снижается, а эффективность падает. Вероятно, это проявление особенностей архитектуры Apple M1: процессор имеет 4 производительных и 4 энерго-эффективных ядра.

Протестируем на 5 строках и 1 000 000 столбцов случайных чисел:

Число потоков	Время исполнения (мс)	Ускорение	Эффективность
1	5.44	1	1
2	2.86	1.9	0.95
3	2.09	2.6	0.86
4	1.91	2.85	0.71
6	2.26	2.4	0.4

8	1.46	3.73	0.47





Видно, что при разбиении по столбцам ускорение выше, а время выполнения меньше, чем при разбиении по столбцам. Это может быть связано с тем, что процессору быстрее обращаться к меньшему количеству массивов.

Код программы

main.cpp

```
#include <format>
#include <sstream>
#include <string>
#include <vector>
```

```
#include "column_split.h"
#include "posix_buf.h"
#include "row_split.h"

using std::vector;

int main(int argc, char* argv[]) {
        if (argc != 3) {
                perr << std::format(
                    "usage: {} <k> <threads>\n"
                    "\n"
                    "Sums array columns using multiple threads.\n"
                    "\n"
                    "  <k>       -- number of arrays\n"
                    "  <threads> -- number of threads\n",
                    argv[0]);
                return 1;
        }

        size_t k;
        {
                std::istringstream stream(argv[1]);
                if (!(stream >> k)) {
                        perr << "Invalid `k`; malformed number\n";
                        return 1;
                }
        }

        size_t threads;
        {
                std::istringstream stream(argv[2]);
                if (!(stream >> threads)) {
                        perr << "Invalid `threads`; malformed number\n";
                        return 1;
                }
        }

        pout << "Input `k` number arrays with the same lengths; "
                "one array per line, numbers are separated with spaces"
             << std::endl;
```

```
vector<vector<long>> arrays(k);


for (size_t i = 0; i != k; ++i) {
        std::string line;
        std::getline(pin, line);


        std::istringstream stream(line);
        long number;


        while (stream >> number) {
                arrays[i].push_back(number);
        }
}


for (size_t i = 1; i != k; ++i) {
        if (arrays[i].size() != arrays[i - 1].size()) {
                perr << "Arrays have different lengths :/ "
                        "Perhaps you supplied an incorrect `k`"
                     << std::endl;
                return 1;
        }
}


vector<long> result;


double rowsToColumns =
    static_cast<double>(arrays.size()) / static_cast<double>(arrays[0].size());
pout << "row / column ratio: " << rowsToColumns << std::endl;


if (rowsToColumns > 2.0) {
        pout << "  ==> using row_split" << std::endl;
        result = row_split::sum(arrays, threads);
} else {
        pout << "  ==> using column_split" << std::endl;
        result = column_split::sum(arrays, threads);
}


for (long item : result) {
        pout << item << " ";
}
```

```
	return 0;
}
```

row_split.cpp

```
#include "row_split.h"

#include <format>

#include "posix_buf.h"

using std::chrono::high_resolution_clock;
using std::chrono::time_point;

namespace row_split {

struct pthread_args {
	const vector<vector<long>>& arrays;

	/** Owned vector with temporary result for the row.*/
	vector<long> result;

	size_t start;
	size_t end;
};

void* pthread_func(void* argsPtr) {
	auto args = static_cast<pthread_args*>(argsPtr);
	// Length is the same for all arrays.
	size_t length = args->arrays[0].size();

	for (size_t i = args->start; i != args->end; ++i) {
		for (size_t j = 0; j != length; ++j) {
			args->result[j] += args->arrays[i][j];
		}
	}

	pthread_exit(argsPtr);
}

vector<long> sum(const vector<vector<long>>& arrays, size_t threads) {
	size_t length = arrays[0].size();
	size_t height = arrays.size();

	// Clamp threads to row amount.
	threads = std::min(height, threads);

	vector<long> result(length, 0);
	vector<pthread_t> pthreads;
	vector<pthread_args> args;

	// Reserve capacity for all threads, so that the vector isn't reallocated.
	args.reserve(threads);
```

```
	time_point timeStart = high_resolution_clock::now();

	for (size_t i = 0; i != threads; ++i) {
		size_t start = i * height / threads;
		size_t end = (i + 1) * height / threads;

		// If there's nothing to sum, don't start the thread, it won't do any
		// calculations.
		if (end - start == 1) {
			for (size_t j = 0; j != length; ++j) {
				result[j] += arrays[start][j];
			}
			continue;
		}

		vector<long> tempResult(length, 0);
		args.emplace_back(arrays, std::move(tempResult), start, end);

		pthread_t pthread;
		if (pthread_create(&pthread, nullptr, &pthread_func, &args.back())) {
			throw std::runtime_error("can't create thread");
		}

		pthreads.push_back(pthread);
	}

	for (pthread_t& thread : pthreads) {
		pthread_args* threadResult;
		if (pthread_join(thread, reinterpret_cast<void**>(&threadResult))) {
			throw std::runtime_error("can't join thread");
		}

		for (size_t i = 0; i != length; ++i) {
			result[i] += threadResult->result[i];
		}
	}

	time_point timeEnd = high_resolution_clock::now();

	auto ns = (timeEnd - timeStart).count();
	auto ms = static_cast<double>(ns) / 1000000.0;
	pout << std::format("Took {}ns ({}ms)\n", ns, ms);

	return result;
}

}  // namespace row_split
```

column_split.cpp

```
#include "column_split.h"

#include <format>
```

```
#include "posix_buf.h"

using std::chrono::high_resolution_clock;
using std::chrono::time_point;

namespace column_split {

struct pthread_args {
	const vector<vector<long>>& arrays;

	vector<long>& result;

	size_t start;
	size_t end;
};

void* pthread_func(void* argsPtr) {
	auto args = static_cast<const pthread_args*>(argsPtr);

	for (size_t i = args->start; i != args->end; ++i) {
		for (const vector<long>& array : args->arrays) {
			args->result[i] += array[i];
		}
	}

	pthread_exit(nullptr);
}

vector<long> sum(const vector<vector<long>>& arrays, size_t threads) {
	size_t length = arrays[0].size();

	// Clamp threads to column amount.
	threads = std::min(length, threads);

	vector<long> result(length, 0);
	vector<pthread_t> pthreads;
	vector<pthread_args> args;

	args.reserve(threads);

	time_point timeStart = high_resolution_clock::now();

	for (size_t i = 0; i != threads; ++i) {
		size_t start = i * length / threads;
		size_t end = (i + 1) * length / threads;

		args.emplace_back(arrays, result, start, end);

		pthread_t pthread;
```

```
                if (pthread_create(&pthread, nullptr, &pthread_func, &args.back())) {
                        throw std::runtime_error("can't create thread");
                }

                pthreads.push_back(pthread);
        }

        for (pthread_t& thread : pthreads) {
                pthread_join(thread, nullptr);
        }

        time_point timeEnd = high_resolution_clock::now();

        auto ns = (timeEnd - timeStart).count();
        auto ms = static_cast<double>(ns) / 1000000.0;
        pout << std::format("Took {}ns ({}ms)\n", ns, ms);

        return result;
}

}  // namespace column_split
```

Протокол работы программы

Тестирование:

```
skidunion@apollo lab-2 % ./lab_2 1000000 1 < ~/random_5_many_rows.txt
Input `k` number arrays with the same lengths; one array per line, numbers are separated with spaces
row / column ratio: 200000
  ==> using row_split
Took 12727250ns (12.72725ms)
-134431797 183680042 216098632 659643770 -68949719
skidunion@apollo lab-2 % ./lab_2 1000000 8 < ~/random_5_many_rows.txt
Input `k` number arrays with the same lengths; one array per line, numbers are separated with spaces
row / column ratio: 200000
  ==> using row_split
Took 4240125ns (4.240125ms)
-134431797 183680042 216098632 659643770 -68949719
skidunion@apollo lab-2 % ./lab_2 5 1 < ~/random_5_many_columns.txt | head -c 400
Input `k` number arrays with the same lengths; one array per line, numbers are separated with spaces
row / column ratio: 5e-06
  ==> using column_split
Took 6017541ns (6.017541ms)
-2831941 218028 -616552 -1441237 1200702 -1506676 ...
skidunion@apollo lab-2 % ./lab_2 5 8 < ~/random_5_many_columns.txt | head -c 400
```

```
      Input `k` number arrays with the same lengths; one array per line, numbers are separated with
spaces

      row / column ratio: 5e-06

        ==> using column_split

      Took 1487584ns (1.487584ms)

      -2831941 218028 -616552 -1441237 1200702 -1506676 ...
```

Dtrace:

```
             PID/THRD  RELATIVE  ELAPSD    CPU SYSCALL(args)              = return

       1675/0x2f05:       219:        0:       0 fork()            = 0 0

       1675/0x2f05:       228      44      8 munmap(0x102CA8000, 0x8C000)         = 0 0

       1675/0x2f05:       232       4      2 munmap(0x102D34000, 0x8000)          = 0 0

       1675/0x2f05:       234       2      1 munmap(0x102D3C000, 0x4000)          = 0 0

       1675/0x2f05:       237       3      2 munmap(0x102D40000, 0x4000)          = 0 0

       1675/0x2f05:       239       2      1 munmap(0x102D44000, 0x50000)         = 0 0

       1675/0x2f05:       256       2      0 crossarch_trap(0x0, 0x0, 0x0)                = -1 Err#45

       1675/0x2f05:       285      27     26 open(".\0", 0x100000, 0x0)           = 3 0

       1675/0x2f05:       290      39      4 fcntl(0x3, 0x32, 0x16D3132B8)                = 0 0

       1675/0x2f05:       293       2      1 close(0x3)            = 0 0

       1675/0x2f05:       302       9      7 fsgetpath(0x16D3132C8, 0x400, 0x16D3132A8)

       1675/0x2f05:       327      25     23 fsgetpath(0x16D3132D8, 0x400, 0x16D3132B8)

       1675/0x2f05:       334       1      0 csrctl(0x0, 0x16D3136DC, 0x4)                = -1 Err#1

       1675/0x2f05:       339       5      4 __mac_syscall(0x18A971ACF, 0x2, 0x16D313620)

       1675/0x2f05:       340       1      0 csrctl(0x0, 0x16D3136CC, 0x4)                = -1 Err#1

       1675/0x2f05:       345       3      2 __mac_syscall(0x18A96E902, 0x5A, 0x16D313660)

      Input `k` number arrays with the same lengths; one array per line, numbers are separated with spaces

      row / column ratio: 2

        ==> using column_split

      Took 134667ns (0.134667ms)

      1753362 1472868 2978162 -1105757 -112654  1675/0x2f05:       364       9      6 sysctl([unknown, 3,
0, 0, 0, 0] (2), 0x16D312BE8, 0x16D312BE0, 0x18A970553, 0xD)         = 0 0

       1675/0x2f05:      366      2     1 sysctl([CTL_KERN, 140, 0, 0, 0, 0] (2), 0x16D312C98, 0x16D312C90,
0x0, 0x0)               = 0 0

       1675/0x2f05:       374       7      6 open("/\0", 0x20100000, 0x0)         = 3 0

       1675/0x2f05:       395      20     18 openat(0x3, "System/Cryptexes/OS\0", 0x100000, 0x0)
= 4 0

       1675/0x2f05:       396       1      0 dup(0x4, 0x0, 0x0)           = 5 0

       1675/0x2f05:       403       6      5 fstatat64(0x4, 0x16D312771, 0x16D3126E0)            = 0 0

       1675/0x2f05:       410       7      6 openat(0x4, "System/Library/dyld/\0", 0x100000, 0x0)
         = 6 0

       1675/0x2f05:       412       1      1 fcntl(0x6, 0x32, 0x16D312770)                = 0 0

       1675/0x2f05:       413       1      0 dup(0x6, 0x0, 0x0)           = 7 0

       1675/0x2f05:       414       1      0 dup(0x5, 0x0, 0x0)           = 8 0
```

```
       1675/0x2f05:       415       1      0 close(0x3)            = 0 0

       1675/0x2f05:       416       1      0 close(0x5)            = 0 0

       1675/0x2f05:       416       0      0 close(0x4)            = 0 0

       1675/0x2f05:       417       0      0 close(0x6)            = 0 0

       1675/0x2f05:       419       3      1 __mac_syscall(0x18A971ACF, 0x2, 0x16D313160)

       1675/0x2f05:       422       3      1 shared_region_check_np(0x16D312D80, 0x0, 0x0)

       1675/0x2f05:       427       4      3 fsgetpath(0x16D3132E0, 0x400, 0x16D313228)

       1675/0x2f05:       429       1      0 fcntl(0x8, 0x32, 0x16D3132E0)               = 0 0

       1675/0x2f05:       430       1      0 close(0x8)            = 0 0

       1675/0x2f05:       431       1      0 close(0x7)            = 0 0

       1675/0x2f05:       440       4      2 getfsstat64(0x0, 0x0, 0x2)           = 10 0

       1675/0x2f05:       448       9      7 getfsstat64(0x102F220D0, 0x54B0, 0x2)               = 10 0

       1675/0x2f05:       455       7      6 getattrlist("/\0", 0x16D313210, 0x16D313180)

       1675/0x2f05:                              469                             9                         8
stat64("/System/Volumes/Preboot/Cryptexes/OS/System/Library/dyld/dyld_shared_cache_arm64e\0",
0x16D313570, 0x0)              = 0 0

      dtrace: error on enabled probe ID 1690 (ID 845: syscall::stat64:return): invalid address (0x0) in
action #12 at DIF offset 12

       1675/0x2f05:      526      5     4 stat64("/Users/skidunion/Work/Depot/Personal/mai-os-3-sem/cmake-
build-release/src/lab-2/lab_2\0", 0x16D312A20, 0x0)                  = 0 0

       1675/0x2f05:       551      24     23 open("/Users/skidunion/Work/Depot/Personal/mai-os-3-sem/cmake-
build-release/src/lab-2/lab_2\0", 0x0, 0x0)    = 3 0

       1675/0x2f05:       562      13     11 mmap(0x0, 0x248F8, 0x1, 0x40002, 0x3, 0x0)         2F64000 0

       1675/0x2f05:       564       2      0 fcntl(0x3, 0x32, 0x16D312B38)               = 0 0

       1675/0x2f05:       565       1      0 close(0x3)            = 0 0

       1675/0x2f05:       569       2      1 munmap(0x102F64000, 0x248F8)         = 0 0

       1675/0x2f05:      572      3     2 stat64("/Users/skidunion/Work/Depot/Personal/mai-os-3-sem/cmake-
build-release/src/lab-2/lab_2\0", 0x16D312F90, 0x0)                  = 0 0

       1675/0x2f05:       601       4      3 stat64("/usr/lib/libc++.1.dylib\0", 0x16D311ED0, 0x0)
         = -1 Err#2

       1675/0x2f05:                              609                             4                         3
stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/libc++.1.dylib\0", 0x16D311E80, 0x0)          =    -1
Err#2

       1675/0x2f05:      656      2     1 stat64("/usr/lib/system/libdispatch.dylib\0", 0x16D30FAF0, 0x0)
                = -1 Err#2

       1675/0x2f05:                              659                             3                         2
stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/system/libdispatch.dylib\0", 0x16D30FAA0, 0x0)
         = -1 Err#2

       1675/0x2f05:      660      1     0 stat64("/usr/lib/system/libdispatch.dylib\0", 0x16D30FAF0, 0x0)
                = -1 Err#2

       1675/0x2f05:       673       2      1 stat64("/usr/lib/libSystem.B.dylib\0", 0x16D311ED0, 0x0)
         = -1 Err#2

       1675/0x2f05:                              678                             2                         1
stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/libSystem.B.dylib\0", 0x16D311E80, 0x0)
= -1 Err#2

       1675/0x2f05:       837      65     29 open("/dev/dtracehelper\0", 0x2, 0x0)              = 3 0

       1675/0x2f05:      1200     364    363 ioctl(0x3, 0x80086804, 0x16D311AC8)         = 0 0

       1675/0x2f05:      1203       3      2 close(0x3)            = 0 0
```

```
      1675/0x2f05:     1484      13     12 open("/Users/skidunion/Work/Depot/Personal/mai-os-3-sem/cmake-
build-release/src/lab-2/lab_2\0", 0x0, 0x0)    = 3 0

      1675/0x2f05:     1487       4      2 __mac_syscall(0x18A971ACF, 0x2, 0x16D3111D0)

      1675/0x2f05:     1501      12     10 map_with_linking_np(0x16D310D30, 0x1, 0x16D310D60) = 0 0

      1675/0x2f05:     1502       1      0 close(0x3)            = 0 0

      1675/0x2f05:     1505       3      2 mprotect(0x102B04000, 0x4000, 0x1)          = 0 0

      1675/0x2f05:     1535       2      0 shared_region_check_np(0xFFFFFFFFFFFFFFFF, 0x0, 0x0)
        = 0 0

      1675/0x2f05:     1538       2      1 mprotect(0x102F20000, 0x40000, 0x1)         = 0 0

      1675/0x2f05:     1555       3      2 access("/AppleInternal/XBS/.isChrooted\0", 0x0, 0x0)
        = -1 Err#2

      1675/0x2f05:     1576       3      1 bsdthread_register(0x18AC74D2C, 0x18AC74D20, 0x4000)
        = 1073746399 0

      1675/0x2f05:     1613       1      0 getpid(0x0, 0x0, 0x0)                = 1675 0

      1675/0x2f05:     1618       4      3 shm_open(0x18AB0FF51, 0x0, 0xFFFFFFFFF7A13AF8)

      1675/0x2f05:     1620       1      0 fstat64(0x3, 0x16D311E60, 0x0)              = 0 0

      1675/0x2f05:     1623       5      3 mmap(0x0, 0x4000, 0x1, 0x40001, 0x3, 0x0)          2F6C000 0

      1675/0x2f05:     1626       1      0 close(0x3)            = 0 0

      1675/0x2f05:     1637       1      0 ioctl(0x2, 0x4004667A, 0x16D311F0C)         = 0 0

      1675/0x2f05:     1646       2      1 mprotect(0x102F78000, 0x4000, 0x0)          = 0 0

      1675/0x2f05:     1647       1      0 mprotect(0x102F84000, 0x4000, 0x0)          = 0 0

      1675/0x2f05:     1651       1      0 mprotect(0x102F88000, 0x4000, 0x0)          = 0 0

      1675/0x2f05:     1652       1      0 mprotect(0x102F94000, 0x4000, 0x0)          = 0 0

      1675/0x2f05:     1657       1      0 mprotect(0x102F98000, 0x4000, 0x0)          = 0 0

      1675/0x2f05:     1658       1      0 mprotect(0x102FA4000, 0x4000, 0x0)          = 0 0

      1675/0x2f05:     1664       1      0 mprotect(0x102F70000, 0xA0, 0x1)            = 0 0

      1675/0x2f05:     1666       2      1 mprotect(0x102F70000, 0xA0, 0x3)            = 0 0

      1675/0x2f05:     1669       1      0 mprotect(0x102F70000, 0xA0, 0x1)            = 0 0

      1675/0x2f05:     1672       1      0 mprotect(0x102FA8000, 0x4000, 0x1)          = 0 0

      1675/0x2f05:     1676       1      0 mprotect(0x102FAC000, 0xA0, 0x1)            = 0 0

      1675/0x2f05:     1684       2      1 mprotect(0x102FAC000, 0xA0, 0x3)            = 0 0

      1675/0x2f05:     1686       1      0 mprotect(0x102FAC000, 0xA0, 0x1)            = 0 0

      1675/0x2f05:     1687       2      1 mprotect(0x102F70000, 0xA0, 0x3)            = 0 0

      1675/0x2f05:     1690       1      0 mprotect(0x102F70000, 0xA0, 0x1)            = 0 0

      1675/0x2f05:     1691       1      0 mprotect(0x102FA8000, 0x4000, 0x3)          = 0 0

      1675/0x2f05:     1693       1      0 mprotect(0x102FA8000, 0x4000, 0x1)          = 0 0

      1675/0x2f05:     1695       1      0 mprotect(0x102F20000, 0x40000, 0x3)         = 0 0

      1675/0x2f05:     1699       1      0 mprotect(0x102F20000, 0x40000, 0x1)         = 0 0

      1675/0x2f05:      1712       2       0 objc_bp_assist_cfg_np(0x18A8A1000, 0x80000018001C1048, 0x0)
               = -1 Err#5

      1675/0x2f05:     1715       1      0 issetugid(0x0, 0x0, 0x0)             = 0 0

      1675/0x2f05:     1740       1      0 mprotect(0x102F20000, 0x40000, 0x3)         = 0 0

      1675/0x2f05:     1786       8      6 getentropy(0x16D311578, 0x20, 0x0)          = 0 0
```

```
       1675/0x2f05:      1828       2      0 mprotect(0x102F20000, 0x40000, 0x1)         = 0 0

       1675/0x2f05:      1850       1      0 mprotect(0x102F20000, 0x40000, 0x3)         = 0 0

       1675/0x2f05:      1854       1      0 mprotect(0x102F20000, 0x40000, 0x1)         = 0 0

       1675/0x2f05:      1862       7      6 getattrlist("/Users/skidunion/Work/Depot/Personal/mai-os-3-
sem/cmake-build-release/src/lab-2/lab_2\0", 0x16D311DF0, 0x16D311E08)               = 0 0

      1675/0x2f05:     1874     10     9 access("/Users/skidunion/Work/Depot/Personal/mai-os-3-sem/cmake-
build-release/src/lab-2\0", 0x4, 0x0)                = 0 0

      1675/0x2f05:     1884     10     9 open("/Users/skidunion/Work/Depot/Personal/mai-os-3-sem/cmake-
build-release/src/lab-2\0", 0x0, 0x0)                = 3 0

       1675/0x2f05:      1885       1      0 fstat64(0x3, 0x154E04450, 0x0)              = 0 0

       1675/0x2f05:      1886       1      0 csrctl(0x0, 0x16D31201C, 0x4)               = 0 0

       1675/0x2f05:      1888       2      1 fcntl(0x3, 0x32, 0x16D311CD8)               = 0 0

       1675/0x2f05:      1891       1      0 close(0x3)            = 0 0

      1675/0x2f05:     1902      8     7 open("/Users/skidunion/Work/Depot/Personal/mai-os-3-sem/cmake-
build-release/src/lab-2/Info.plist\0", 0x0, 0x0)             = -1 Err#2

       1675/0x2f05:      1918       2      1 proc_info(0x2, 0x68B, 0xD)           = 64 0

       1675/0x2f05:      1933       3      2 csops_audittoken(0x68B, 0x10, 0x16D312060)

      1675/0x2f05:     1993      7     5 sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16D3123B8, 0x16D3123B0,
0x18E0A6D3D, 0x15)      = 0 0

      1675/0x2f05:     1995     3     1 sysctl([CTL_KERN, 138, 0, 0, 0, 0] (2), 0x16D312448, 0x16D312440,
0x0, 0x0)               = 0 0

       1675/0x2f05:      1997       1      0 csops(0x68B, 0x0, 0x16D3124EC)              = 0 0

       1675/0x2f05:      2001       1      0 mprotect(0x102F20000, 0x40000, 0x3)         = 0 0

      1675/0x2f05:      2097       4      3 write(0x1, "Input `k` number arrays with the same lengths; one
array per line, numbers are separated with spaces\n\0", 0x65)               = 101 0

      1675/0x2f05:      2107       8      7 read(0x0, "253961 -915234 791928 686246 -397831\n-897873 -843275
560403 -105247 -266075\n593448 803935 746530 -732356 593569\n634330 645272 74884 -362824 -196424\n202535
515467 -402451 -897495 -322973\n-30886 -787023 289243 -496126 229208\n-936157 175134 -625562 293519 130",
0x1000)               = 4096 0

      1675/0x2f05:      2132       2      1 write(0x1, "row / column ratio: 2\n064\n-77180 -474819 839480
121573 -110032\n947750 991120 -882874 745873 441166\n569015 745331 -135069 452033 742164\n592855 951357 -
118840 799771 -432538\n940933 -239532 -431289 262730 149329\n-56953 505342 -248213 946037 503652\n-507794
-30", 0x16)            = 22 0

      1675/0x2f05:      2133       1      0 write(0x1, "  ==> using column_split\n9063 -777000\n84632 959412
-817631 998487 123194\n890102 234172 151340 -2898 445515\n366951 650002 715679 619634 399344\n612732 -
451101 264380 -779789 591379\n-119530 -505244 -988138 59425 -38927\n563839 -630215 -546345 -658392
309006\n40", 0x19)             = 25 0

       1675/0x2f05:      2159      15     13 bsdthread_create(0x102AEFFE4, 0x6000017E4080, 0x16D39B000)
= 1832497152 0

       1675/0x2f0a:         5:        0:       0 fork()            = 0 0

       1675/0x2f05:      2168       6      4 bsdthread_create(0x102AEFFE4, 0x6000017E40A0, 0x16D427000)
= 1833070592 0

       1675/0x2f0b:         2:        0:       0 fork()            = 0 0

       1675/0x2f05:      2175       5      3 bsdthread_create(0x102AEFFE4, 0x6000017E40C0, 0x16D4B3000)
= 1833644032 0

       1675/0x2f0b:         2       2      0 thread_selfid(0x0, 0x0, 0x0)         = 12043 0

       1675/0x2f0b:         3       2      0 __disable_threadsignal(0x1, 0x0, 0x0)               = 0 0

       1675/0x2f0c:         4:        0:       0 fork()            = 0 0

       1675/0x2f05:      2183       6      4 bsdthread_create(0x102AEFFE4, 0x6000017E40E0, 0x16D53F000)
= 1834217472 0

       1675/0x2f0a:         5      37      0 thread_selfid(0x0, 0x0, 0x0)         = 12042 0
```

```
 1675/0x2f0a:         6       2      0 __disable_threadsignal(0x1, 0x0, 0x0)               = 0 0
 1675/0x2f0d:        13:        0:       0 fork()            = 0 0
 1675/0x2f0a:        15       4      2 ulock_wake(0x1000002, 0x16D39B034, 0x0)             = 0 0
 1675/0x2f0c:         4      36      0 thread_selfid(0x0, 0x0, 0x0)         = 12044 0
 1675/0x2f0d:        14       1      0 thread_selfid(0x0, 0x0, 0x0)         = 12045 0
 1675/0x2f05:      2192      36      8 ulock_wait(0x1020002, 0x16D39B034, 0xA07)
 1675/0x2f0d:        14       2      0 __disable_threadsignal(0x1, 0x0, 0x0)               = 0 0
 1675/0x2f0c:         5       2      0 __disable_threadsignal(0x1, 0x0, 0x0)               = 0 0
 1675/0x2f0c:        13       3      2 ulock_wake(0x1000002, 0x16D4B3034, 0x0)             = 0 0
 1675/0x2f05:      2203      12      4 ulock_wait(0x1020002, 0x16D4B3034, 0x1C03)
 1675/0x2f05:      2225       3      2 write(0x1, "Took 134667ns (0.134667ms)\n1753362 1472868 2978162
-1105757 -112654 \0", 0x44)            = 68 0


      CALL                                        COUNT
      bsdthread_register                              1
      crossarch_trap                                  1
      csops                                           1
      csops_audittoken                                1
      exit                                            1
      fstatat64                                       1
      getentropy                                      1
      getpid                                          1
      issetugid                                       1
      map_with_linking_np                             1
      objc_bp_assist_cfg_np                           1
      proc_info                                       1
      read                                            1
      shm_open                                        1
      access                                          2
      fstat64                                         2
      getattrlist                                     2
      getfsstat64                                     2
      ioctl                                           2
      mmap                                            2
      openat                                          2
      shared_region_check_np                          2
      ulock_wait                                      2
      ulock_wake                                      2
      csrctl                                          3
      dup                                             3
      fsgetpath                                       3
```

__disable_threadsignal	4
__mac_syscall	4
bsdthread_create	**4**
bsdthread_terminate	**4**
sysctl	4
thread_selfid	4
write	**4**
fcntl	5
munmap	6
open	7
stat64	11
close	12
mprotect	26

Вывод

В ходе выполнения этой лабораторной работы я научился работать с потоками в POSIX. Я написал обёртку файловых потоков на C++ для более удобной работы с ними. Хотя они тут не понадобились, я ознакомился со способами синхронизации данных между потоками. При выполнении задания я столкнулся с небольшой проблемой: массив аргументов к потоку расширялся во время выполнения, из-за чего поток возвращал провисшый указатель. Отладить это помог ThreadSanitizer.