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»

Выполнила: В. А. Гузова
Группа: М8О-207БВ-24
Преподаватель: Е. С. Миронов

Москва, 2025

Условие

Цель работы:

Приобретение практических навыков в:

- Управление потоками в ОС
- Обеспечение синхронизации между потоками

Задание:

Составить программу на языке C++, 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.

Вариант: 14

Есть набор 512 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

Метод решения

Программа загружает массив больших целых чисел (512-битных) из файла, распределяет их равномерно между заданным числом потоков и вычисляет сумму параллельно. Каждый поток накапливает локальную частичную сумму. После завершения всех потоков главный поток собирает локальные результаты, вычисляет среднее арифметическое и применяет арифметическое округление.Каждый поток пишет только в свою собственную структуру `ThreadData`, финальное суммирование происходит последовательно в главном потоке.

Описание программы

`main.cpp` — точка входа в программу. Отвечает за парсинг аргументов командной строки. Создает и запускает рабочие потоки, распределяя между ними нагрузку. После завершения всех потоков вычисляет и округляет среднее,выводит результат и замер времени.

`pthread.h` — объявление общего кроссплатформенного интерфейса для работы с потоками.

Основные функции:

- `Thread::Thread(ThreadFunc func)` — конструктор, создаёт объект `ThreadInfo`
- `void Thread::Run(void* data)` — запускает поток.Обёртка над `pthread_create()`.
- `void Thread::Join()` — ожидает завершения потока.`pthread_join()`.
- `Thread::Thread(Thread&& other) noexcept` и `Thread& operator=(Thread&&)` — передают владение потоком от одного объекта к другому, не запуская поток заново.

- `void Thread::Swap(Thread& other)` — обменивает содержимое двух объектов `Thread`.

Главный поток (main) подготавливает вектор ThreadData, содержащий указатель на общий массив чисел и границы поддиапазона `[start, end)` для каждого потока. Затем он создаёт K=min(threadsCount,N) рабочих потоков. Каждый получает указатель на свою структуру `ThreadData` и выполняет функцию `thread_func`.

В `thread_func` каждый поток:

- последовательно складывает все числа из своего диапазона в локальный `BigInt sum` с использованием метода `Add`,
- не синхронизируется с другими потоками во время вычислений,
- завершает работу, оставляя результат в своей `ThreadData.sum`.

После вызова `Join()` для всех потоков главный поток последовательно суммирует все `threadData[i].sum` в `total_sum`, затем вычисляет среднее с округлением — объединение результатов происходит в одном потоке, это исключает гонки и делает мьютексы избыточными.

Результаты

Программа получает на вход три параметра: максимальное число потоков, доступный объём памяти и имя файла с 128-символьными шестнадцатеричными числами. На выходе она выводит одну строку — 128-символьное шестнадцатеричное представление округлённого среднего арифметического всех загруженных чисел, а также время выполнения в миллисекундах. Промежуточные вычисления выполняются с точностью до 512 бит без потерь. Если при создании потоков или чтении чисел из файлов возникают ошибки, программа завершается сообщением в `stderr`.

Выводы

В ходе выполнения лабораторной работы были приобретены практические навыки в организации многопоточной обработки данных в операционных системах, распределения вычислительной нагрузки между потоками и работы с большими объёмами численных данных без разделяемого состояния. Была составлена и отлажена программа на языке C++, реализующая параллельное вычисление среднего арифметического для массива 512-битных целых чисел, представленных в шестнадцатеричном виде. Программа использует средства многопоточности (`pthread_create, pthread_join`), что обеспечивает корректную работу в Unix-подобных операционных системах и поддерживает кросплатформенность. В результате работы программа ограничивает объём загружаемых данных в соответствии с заданным лимитом памяти, распределяет числа между потоками, а каждый поток независимо вычисляет свою частичную сумму. Обмен данными между потоками полностью исключён во время расчётов. Финальное объединение результатов выполняется последовательно в главном потоке. Были обработаны возможные ошибки: некорректные аргументы командной строки, невалидные символы в числах, а также сбои при создании потоков. Экспериментально подтверждена эффективность параллельных вычислений: при увеличении числа потоков (до числа ядер) время выполнения сокращается почти линейно, особенно при обработке больших наборов данных.

Исходная программа

```
#pragma once

#include <iostream>
#include <string>
#include <iomanip>
#include <stdexcept>

struct BigInt {
  private:
    uint64_t parts[8];
  public:
    BigInt();
    BigInt(const BigInt& other);
    BigInt(BigInt&& other) noexcept;
    BigInt& operator=(const BigInt& other) = default;
    BigInt& operator=(BigInt&& other) noexcept;
    ~BigInt() = default;

    void Add(const BigInt& other);
    uint64_t Divide(uint64_t divisor, BigInt& quotient) const;
    void Round();
    int ReadFromHex(const char* hex, size_t len);
    void Print() const;
};
```

Листинг 1: include/bigint.h

```
#include "bigint.h"


BigInt::BigInt() {
    for (int i = 0; i < 8; i++) {
        parts[i] = 0;
    }
}

BigInt::BigInt(const BigInt& other) {
    for (int i = 0; i < 8; i++) {
        parts[i] = other.parts[i];
    }
}

BigInt::BigInt(BigInt&& other) noexcept {
    for (int i = 0; i < 8; i++) {
        parts[i] = other.parts[i];
        other.parts[i] = 0;
    }
}


BigInt& BigInt::operator=(BigInt&& other) noexcept {
    if (this != &other) {
        for (int i = 0; i < 8; i++) {
            parts[i] = other.parts[i];
            other.parts[i] = 0;
```

```
        }
    }
    return *this;
}

void BigInt::Add(const BigInt& other) {
    uint64_t carry = 0;
    for (int i = 7; i >= 0; --i) {
        uint64_t a = parts[i];
        uint64_t b = other.parts[i];
        uint64_t sum_ab = a + b;
        uint64_t carry1 = (sum_ab < a);
        uint64_t sum = sum_ab + carry;
        uint64_t carry2 = (sum < carry);
        carry = carry1 | carry2;
        parts[i] = sum;
    }
}

int BigInt::ReadFromHex(const char* hex, size_t len) {
    for (int i = 0; i < 8; ++i) {
        parts[i] = 0;
    }
    int word_idx = 7;
    int shift = 0;
    for (size_t i = len; i > 0; --i) {
        char c = hex[i - 1];
        unsigned digit;
        if (c >= '0' && c <= '9') {
            digit = c - '0';
        } else if (c >= 'a' && c <= 'f') {
            digit = c - 'a' + 10;
        } else {
            return -1;
        }
        parts[word_idx] |= static_cast<uint64_t>(digit) << shift;
        shift += 4;
        if (shift == 64) {
            shift = 0;
            --word_idx;
            if (word_idx < 0) {
                break;
            }
        }
    }
    return 0;
}

uint64_t BigInt::Divide(uint64_t divisor, BigInt& quotient) const {

    __uint128_t remainder = 0;
    for (int i = 0; i < 8; ++i) {
        __uint128_t temp = (remainder << 64) | parts[i];
        quotient.parts[i] = static_cast<uint64_t>(temp / divisor);
        remainder = temp % divisor;
    }
    return static_cast<uint64_t>(remainder);
}
```

```

void BigInt::Round() {
       uint64_t carry = 1;
    for (int i = 7; i >= 0 && carry; --i) {
        uint64_t old = parts[i];
        parts[i] = old + carry;
        carry = (parts[i] < old);
    }
}

void BigInt::Print() const {
    for (int i = 0; i < 8; ++i) {
        std::cout << std::hex << std::setw(16) << std::setfill('0') << parts[i];
    }
    std::cout << std::dec;
}
```

Листинг 2: src/bigint.cpp

```
namespace thread {
using ThreadFunc = typeof(void*(void*))*;
struct ThreadInfo;
using ThreadHandle = ThreadInfo*;

class Thread {
  private:
    ThreadFunc func_;
    ThreadHandle handle_;
  public:
    Thread(ThreadFunc func);
    Thread(const Thread&) = delete;
    Thread(Thread&& other) noexcept;

    Thread& operator=(const Thread&) = delete;
    Thread& operator=(Thread&& other) noexcept;

    void Swap(Thread& other);
    void Run(void* data);
    void Join();
    ~Thread() noexcept;
};
}
```

Листинг 3: include/thread.h

```
#include <pthread.h>
#include <utility>
#include <system_error>

#include "thread.h"


namespace thread {
struct ThreadInfo {
    pthread_t thread;
```

```
};

Thread::Thread(ThreadFunc func): func_(func) {
    handle_ = new ThreadInfo();
}

Thread::Thread(Thread&& other) noexcept: func_(other.func_), handle_(other.handle_) {
    other.func_ = nullptr;
    other.handle_ = nullptr;
}

Thread::~Thread() noexcept {
    delete handle_;
}

Thread& Thread::operator=(Thread&& other) noexcept {
    if (this == &other) {
        return *this;
    }
    Thread tmp = std::move(other);
    this->Swap(tmp);
    return *this;
}

void Thread::Run(void* data) {
    int res = pthread_create(&(handle_->thread), NULL, func_, data);
    if (res != 0) {
        throw std::system_error(res, std::system_category(), "pthread_create failed");
    }
}

void Thread::Join() {
    int res = pthread_join(handle_->thread, NULL);
    if (res != 0) {
        throw std::system_error(res, std::system_category(), "pthread_join failed");
    }
}

void Thread::Swap(Thread& other) {
    std::swap(func_, other.func_);
    std::swap(handle_, other.handle_);
}

}
```

Листинг 4: src/thread.cpp

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cstring>
#include <cstdlib>
#include <cstdint>
#include <chrono>

#include "thread.h"
#include "bigint.h"
```

```
11
12
13  struct ThreadData {
14      const std::vector<BigInt>* my_numbers;
15      size_t start;
16      size_t end;
17      BigInt sum;
18  };
19
20  void* thread_func(void* arg) {
21      ThreadData* td = static_cast<ThreadData*>(arg);
22      for (size_t i = td->start; i < td->end; ++i) {
23          td->sum.Add((*td->my_numbers)[i]);
24      }
25      return nullptr;
26  }
27
28  int main(int argc, char* argv[]) {
29      const size_t MEMORY_SYS = 20;
30      if (argc != 4) {
31          std::cerr << "Must be 3 argument";
32          return 1;
33      }
34      size_t threadsCount = std::stoul(argv[1]);
35      size_t memory = std::stoul(argv[2]);
36      std::string my_file = argv[3];
37
38      if (threadsCount == 0 || threadsCount > 10000) {
39          std::cerr << "Incorrect threads count" << std::endl;
40          return 1;
41      }
42
43      if (memory == 0 || memory > 10000000) {
44          std::cerr << "Incorrect memory count" << std::endl;
45          return 1;
46      }
47
48      if (memory < MEMORY_SYS) {
49          std::cerr << "Memory limit too small" << std::endl;
50          return 1;
51      }
52      auto start = std::chrono::steady_clock::now();
53
54      const size_t bigint_size = sizeof(BigInt);
55      const size_t usable_memory = memory - MEMORY_SYS;
56      size_t max_numbers = (usable_memory * 1024ULL * 1024ULL) / bigint_size; //  !
57      std::ifstream file(my_file);
58      std::vector<BigInt> numbers;
59      numbers.reserve(usable_memory);
60      std::string line;
61      while (numbers.size() < max_numbers && std::getline(file, line)) {
62          if (!line.empty() && line.back() == '\r') line.pop_back();
63          if (line.empty()) continue;
64          if (line.length() > 128) line.resize(128);
65          if (line.length() < 128) {
66              line = std::string(128 - line.length(), '0') + line;
67          }
68
```

```
69          BigInt num;
70          int res = num.ReadFromHex(line.c_str(), line.length());
71          if (res != 0) {
72              std::cerr << "Error: invalid hex number at line " << numbers.size() + 1 <<
                    "\n";
73              return 1;
74          }
75          numbers.push_back(num);
76      }
77      file.close();
78
79
80      size_t threads_n = std::min(threadsCount, numbers.size());
81      std::vector<thread::Thread> threads;
82      std::vector<ThreadData> threadData(threads_n);
83
84      size_t base = numbers.size() / threads_n;
85      size_t leftover = numbers.size() % threads_n;
86      size_t current = 0;
87
88      for (size_t i = 0; i < threads_n; ++i) {
89          const size_t block = base + (i < leftover ? 1 : 0);
90
91          threadData[i].my_numbers = &numbers;
92          threadData[i].start = current;
93          threadData[i].end = current + block;
94          current += block;
95
96          try {
97              threads.emplace_back(thread_func);
98              threads.back().Run(&threadData[i]);
99          } catch (const std::exception& e) {
100             std::cerr << e.what() << std::endl;
101             return 1;
102         }
103     }
104     BigInt total_sum;
105     size_t total_count = numbers.size();
106     for (size_t i = 0; i < threads_n; ++i) {
107         try {
108             threads[i].Join();
109             total_sum.Add(threadData[i].sum);
110         } catch (const std::exception& e) {
111             std::cerr << e.what() << std::endl;
112             return 1;
113         }
114     }
115     BigInt average;
116     uint64_t remainder = total_sum.Divide(static_cast<uint64_t>(total_count), average)
            ;
117     if (remainder * 2 >= total_count) {
118         average.Round();
119     }
120     average.Print();
121
122     auto end = std::chrono::steady_clock::now();
123
124     std::cout << "Algorithm was done in: " << std::chrono::duration_cast<std::chrono::
```

```
        milliseconds>(end - start).count() << " ms" << std::endl;
    return 0;
}
```

Листинг 5: main.cpp

Strace

```
execve("./main", ["./main", "4", "100", "number.hex"], 0x7fffede89ce8 /* 26
↪  vars */) = 0
brk(NULL)                                  = 0x5e908c261000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffe70625cc0) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
↪  0x7a2157779000
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
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=29000, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 29000, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7a2157771000
close(3)                                   = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libstdc++.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\0\0\0\0\0\0\0\0"...,
↪  832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=2260296, ...}, AT_EMPTY_PATH) =
↪  0
mmap(NULL, 2275520, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7a2157400000
mprotect(0x7a215749a000, 1576960, PROT_NONE) = 0
mmap(0x7a215749a000, 1118208, PROT_READ|PROT_EXEC,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x9a000) = 0x7a215749a000
mmap(0x7a21575ab000, 454656, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
↪  0x1ab000) = 0x7a21575ab000
mmap(0x7a215761b000, 57344, PROT_READ|PROT_WRITE,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x21a000) = 0x7a215761b000
mmap(0x7a2157629000, 10432, PROT_READ|PROT_WRITE,
↪  MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7a2157629000
close(3)                                   = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libgcc_s.so.1", 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\0\0\0\0\0\0\0\0"...,
↪  832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=125488, ...}, AT_EMPTY_PATH) =
↪  0
mmap(NULL, 127720, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7a2157751000
mmap(0x7a2157754000, 94208, PROT_READ|PROT_EXEC,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x3000) = 0x7a2157754000
mmap(0x7a215776b000, 16384, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
↪  0x1a000) = 0x7a215776b000
mmap(0x7a215776f000, 8192, PROT_READ|PROT_WRITE,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1d000) = 0x7a215776f000
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\0P\237\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 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"...,
↪  48, 848) = 48
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0O{\f\225\\=\201\327\312\301P\32\$\23
↪  0\266\235"..., 68, 896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, 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, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7a2157000000
mprotect(0x7a2157028000, 2023424, PROT_NONE) = 0
mmap(0x7a2157028000, 1658880, PROT_READ|PROT_EXEC,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7a2157028000
mmap(0x7a21571bd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
↪  0x1bd000) = 0x7a21571bd000
mmap(0x7a2157216000, 24576, PROT_READ|PROT_WRITE,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7a2157216000
mmap(0x7a215721c000, 52816, PROT_READ|PROT_WRITE,
↪  MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7a215721c000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libm.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\0\0\0\0\0\0\0\0"...,
↪  832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=940560, ...}, AT_EMPTY_PATH) =
↪  0
mmap(NULL, 942344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7a215766a000
mmap(0x7a2157678000, 507904, PROT_READ|PROT_EXEC,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe000) = 0x7a2157678000
mmap(0x7a21576f4000, 372736, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
↪  0x8a000) = 0x7a21576f4000
mmap(0x7a215774f000, 8192, PROT_READ|PROT_WRITE,
↪  MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe4000) = 0x7a215774f000
close(3)                                = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
↪  0x7a2157668000
arch_prctl(ARCH_SET_FS, 0x7a21576693c0) = 0
set_tid_address(0x7a2157669690)         = 2384
set_robust_list(0x7a21576696a0, 24)     = 0
rseq(0x7a2157669d60, 0x20, 0, 0x53053053) = 0
mprotect(0x7a2157216000, 16384, PROT_READ) = 0
mprotect(0x7a215774f000, 4096, PROT_READ) = 0
mprotect(0x7a215776f000, 4096, PROT_READ) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
↪  0x7a2157666000
mprotect(0x7a215761b000, 45056, PROT_READ) = 0
mprotect(0x5e9085560000, 4096, PROT_READ) = 0
mprotect(0x7a21577b3000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
↪  rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7a2157771000, 29000)           = 0
getrandom("\x1f\xa0\xa4\x16\x77\x33\xeb\x7c", 8, GRND_NONBLOCK) = 8
```

```
brk(NULL)                               = 0x5e908c261000
brk(0x5e908c282000)                     = 0x5e908c282000
futex(0x7a215762977c, FUTEX_WAKE_PRIVATE, 2147483647) = 0
openat(AT_FDCWD, "number.hex", O_RDONLY) = 3
read(3, "0000000000000001\r\n00000000000000"..., 8191) = 250
read(3, "", 8191)                       = 0
close(3)                                = 0
rt_sigaction(SIGRT_1, {sa_handler=0x7a2157091870, sa_mask=[],
↪  sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO,
↪  sa_restorer=0x7a2157042520}, 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) =
↪  0x7a21567ff000
mprotect(0x7a2156800000, 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_S⌋
↪  YSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
↪  child_tid=0x7a2156fff910, parent_tid=0x7a2156fff910, exit_signal=0,
↪  stack=0x7a21567ff000, stack_size=0x7fff00, tls=0x7a2156fff640} =>
↪  {parent_tid=[2385]}, 88) = 2385
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
↪  0x7a2155ffe000
mprotect(0x7a2155fff000, 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_S⌋
↪  YSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
↪  child_tid=0x7a21567fe910, parent_tid=0x7a21567fe910, exit_signal=0,
↪  stack=0x7a2155ffe000, stack_size=0x7fff00, tls=0x7a21567fe640} =>
↪  {parent_tid=[2386]}, 88) = 2386
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
↪  0x7a21557fd000
mprotect(0x7a21557fe000, 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_S⌋
↪  YSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
↪  child_tid=0x7a2155ffd910, parent_tid=0x7a2155ffd910, exit_signal=0,
↪  stack=0x7a21557fd000, stack_size=0x7fff00, tls=0x7a2155ffd640} =>
↪  {parent_tid=[2387]}, 88) = 2387
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
↪  0x7a2154ffc000
mprotect(0x7a2154ffd000, 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_S⌋
↪  YSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
↪  child_tid=0x7a21557fc910, parent_tid=0x7a21557fc910, exit_signal=0,
↪  stack=0x7a2154ffc000, stack_size=0x7fff00, tls=0x7a21557fc640} =>
↪  {parent_tid=[0]}, 88) = 2388
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
```

```
newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x3), ...},
↪  AT_EMPTY_PATH) = 0
write(1, "000000000000000000000000000000000"...,
↪  157000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪  000000000000000000000000000000000000000000000000000008Algorithm was done
↪  in: 12 ms
) = 157
exit_group(0)                           = ?
+++ exited with 0 +++
```