

**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»**

**Выполнил: М. А. Понизяйкин
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 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. 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: 15

15. Есть колода из 52 карт, рассчитать экспериментально (метод Монте-Карло) вероятность того, что сверху лежат две одинаковых карты. Количество раундов задаётся ключом программы.

Метод решения

Алгоритм решения задачи:

1. Пользователь в консоль вводит два параметра командной строки:

- rounds N — общее количество экспериментов (раундов),
- max-threads M — максимальное число потоков, выполняемых одновременно.

Проверяется корректность входных данных (положительные целые числа).

2. Общее число раундов N равномерно распределяется между M потоками. Остаток от деления $N \bmod M$ добавляется к первому потоку для обеспечения полноты покрытия.

3. Для каждого потока создаётся объект `std::thread`. Каждому потоку передаётся:

- Указатель на функцию обработки (`card_function`),
- Количество раундов для данного потока (типа `long long`)

4. Для каждого раунда:

- Колода из 52 карт (13 рангов × 4 масти) создаётся заново.
- Колода случайным образом перемешивается с использованием `std::shuffle` и генератора `std::mt19937`, инициализированного энтропией из `std::random_device`.
- Проверяется, совпадают ли достоинства первых двух карт.
- При совпадении увеличивается локальный счётчик успешных исходов.

5. После завершения всех раундов поток:

- Блокирует глобальный мьютекс (`std::mutex`) с помощью `std::lock_guard`,
- Обновляет общие счётчики (`total_rounds`, `total_successes`),
- Мьютекс автоматически разблокируется при выходе из области видимости.

6. Главный поток ожидает завершения всех рабочих потоков с помощью метода `.join()`.
Вычисляется экспериментальная вероятность:

$$P = \frac{total_rounds}{total_successes}$$

7. После успешного завершения работы программа выводит:

- Общее число раундов,
- Число успешных исходов,
- Экспериментальную вероятность,
- Время выполнения в наносекундах и секундах (с использованием `std::chrono`).

Архитектура программы:

```
lab2-var15/
├── bin/
├── build/
├── include/
│   ├── deck.h
│   └── monte_carlo_card.h
├── src/
│   ├── deck.cpp
│   └── monte_carlo_card.cpp
├── main.cpp
└── CMakeLists.txt
```

Ссылки:

- <https://en.cppreference.com/w/cpp/chrono.html>
- <https://en.cppreference.com/w/cpp/thread/thread.html>
- <https://en.cppreference.com/w/cpp/thread/mutex.html>
- https://en.cppreference.com/w/cpp/thread/lock_guard.html
- <https://en.cppreference.com/w/cpp/numeric/random/rand.html>
- https://en.cppreference.com/w/cpp/algorithm/random_shuffle.html
- https://en.cppreference.com/w/cpp/numeric/random/mersenne_twister_engine.html
- <https://en.cppreference.com/w/cpp/algorithm.html>
- <https://en.cppreference.com/w/cpp/header/cstdlib.html>

Описание программы

`main.cpp` — точка входа в программу. Выполняет парсинг аргументов командной строки (`-rounds N`, `-max-threads M`), проверку их корректности, распределение вычислительной нагрузки между потоками и запуск многопоточного моделирования. После завершения всех потоков выводит статистику: общее число раундов, количество успешных исходов, экспериментальную вероятность и время выполнения.

`include/deck.cpp` — модуль работы с колодой карт. Содержит объявление структуры `Card` и функцию `CreateDeck()`, создающую стандартную колоду из 52 карт (13 достоинств × 4 масти).

Основные функции:

- `std::vector<Card> CreateDeck();` — создаёт стандартную колоду из 52 карт. Колода представлена вектором объектов `Card`, где каждый объект содержит поле `denomination` (достоинство карты от 0 до 12). Колода упорядочена по достоинствам (по 4 карты каждого ранга).

`include/monte_carlo_card.cpp` — модуль логики моделирования методом Монте-Карло. Содержит глобальное состояние программы и функцию `card_function()`, реализующую вычисления в отдельном потоке.

Основные функции:

- `void card_function(long long rounds);` — выполняет заданное количество раундов в отдельном потоке:
 - Для каждого раунда создаёт копию колоды и перемешивает её с использованием `std::shuffle` и генератора `std::mt19937`.
 - Проверяет, совпадают ли достоинства первых двух карт в перемешанной колоде.
 - При совпадении увеличивает локальный счётчик успешных исходов.
 - После завершения всех раундов обновляет глобальные счётчики `total_successes_rounds` и `total_rounds` под защитой мьютекса.

Результаты

Программа получает на вход два параметра: количество экспериментов (`-rounds N`) и максимальное число потоков (`-max-threads M`). После проверки корректности аргументов она запускает многопоточное моделирование методом Монте-Карло: каждый поток независимо выполняет заданное число раундов, в каждом из которых создаётся и перемешивается колода из 52 карт, после чего проверяется, совпадают ли достоинства двух верхних карт.

По завершении всех вычислений программа выводит в стандартный поток вывода:

- общее число проведённых раундов,
- количество успешных исходов (совпадение достоинств),
- экспериментальную вероятность в виде десятичной дроби,
- время выполнения как в наносекундах, так и в секундах.

Результатом работы является численная оценка вероятности, которая при увеличении числа раундов стремится к теоретическому значению $\frac{3}{51} \approx 0.0588(5.88\%)$.

Программа корректно обрабатывает ошибочные ситуации (некорректные аргументы, недопустимые значения) и завершается с кодом ошибки. В случае корректного запуска все потоки завершаются штатно, ресурсы освобождаются автоматически, а результат выводится без искажений. Реализация на основе `std::thread` обеспечивает кроссплатформенность и безопасную параллельную обработку данных.

График зависимости времени от количества используемых потоков приведен на рисунке 1. (`rounds = 10^7`). Анализ показывает, что при увеличении числа потоков от 1 до 12 наблюдается устойчивое сокращение времени выполнения, что свидетельствует о хорошей параллелизуемости задачи. Ускорение достигает примерно $6.3\times$ ($21.10 / 3.34$) при использовании 12 потоков по сравнению с однопоточной версией.

Минимальное время выполнения достигается при 12 потоках, после чего дальнейшее увеличение числа потоков не приводит к ускорению, а, напротив, вызывает незначительный рост времени. Это объясняется тем, что количество рабочих потоков превышает число

логических ядер процессора, доступных в системе. В результате операционная система вынуждена выполнять переключение контекста между потоками, что порождает дополнительные накладные расходы на управление потоками, конкуренцию за кэш процессора и снижение общей эффективности выполнения.

Таким образом, оптимальное число потоков для данной задачи и аппаратной конфигурации составляет 12, что, соответствует количеству логических ядер центрального процессора. Полученные результаты подтверждают общее правило: для CPU-ограниченных задач максимальная производительность достигается при числе потоков, равном числу логических ядер системы. Дальнейшее увеличение параллелизма нецелесообразно и может привести к деградации производительности.

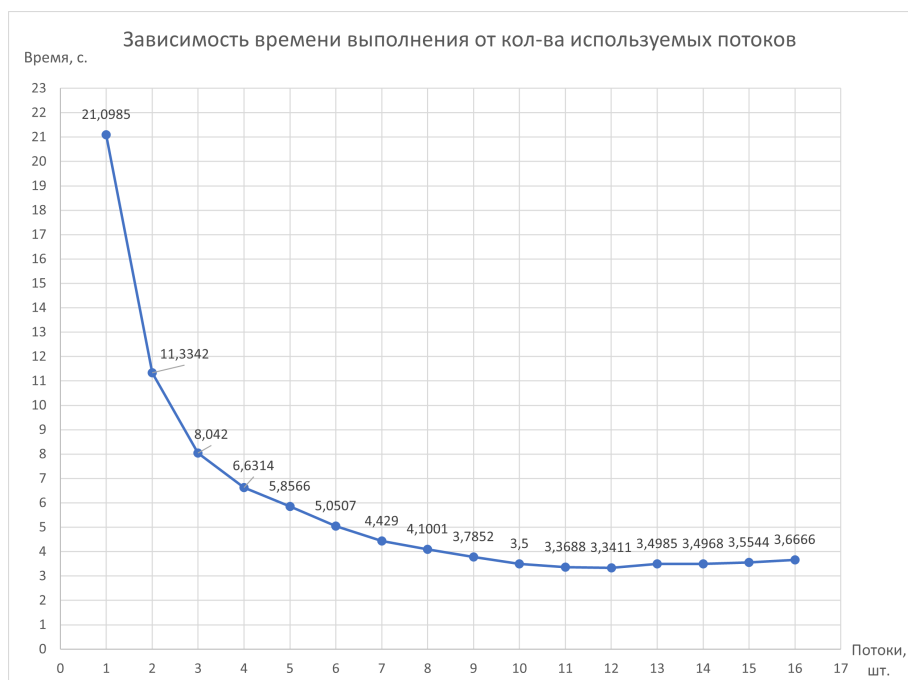


Рис. 1: График зависимости времени выполнения от количества используемых потоков.

Выводы

В ходе выполнения лабораторной работы были приобретены практические навыки в организации многопоточной обработки данных в операционных системах, синхронизации потоков и безопасной работы с разделяемыми ресурсами.

Была составлена и отлажена программа на языке C++, реализующая параллельное моделирование методом Монте-Карло для оценки вероятности совпадения достоинств двух верхних карт в случайно перетасованной колоде. Программа использует стандартные средства многопоточности C++ (`std::thread`, `std::mutex`), что обеспечивает корректную работу на операционных системах семейства Unix (включая Linux) и поддерживает кроссплатформенность.

В результате работы программа запускает указанное пользователем количество потоков, каждый из которых независимо выполняет часть общего числа экспериментов. Обмен данными между потоками сведён к минимуму: результаты агрегируются в глобальные счётчики под защитой мьютекса, что исключает гонки данных и гарантирует корректность итогового результата.

Были обработаны возможные ошибки ввода (некорректные аргументы командной строки), а также обеспечена корректная инициализация и завершение всех потоков. Экспериментально подтверждена эффективность параллельных вычислений: время выполнения сокращается почти пропорционально числу потоков до достижения аппаратного предела (числа логических ядер процессора).

Исходная программа

```
1 | #pragma once
2 |
3 | #include <vector>
4 |
5 | struct Card {
6 |     int denomination;
7 | };
8 |
9 | std::vector<Card> CreateDeck();
```

Листинг 1: include/deck.h

```
1 | #include "deck.h"
2 |
3 | std::vector<Card> CreateDeck() {
4 |     std::vector<Card> deck;
5 |     deck.reserve(52);
6 |     for (int denom = 0; denom < 13; ++denom) {
7 |         for (int suit = 0; suit < 4; ++suit) {
8 |             deck.push_back(Card{denom});
9 |         }
10 |    }
11 |
12 |    return deck;
13 | }
```

Листинг 2: src/deck.cpp

```
1 | #pragma once
2 |
3 | #include <mutex>
4 |
5 | struct GeneralThread {
6 |     long long total_succeses_rounds = 0;
7 |     long long total_rounds = 0;
8 |     std::mutex mutex;
9 | };
10 |
11 | extern GeneralThread gen_thread;
12 |
13 | void card_function(long long rounds);
```

Листинг 3: include/monte_carlo_card.h

```
1 | #include "monte_carlo_card.h"
2 | #include "deck.h"
3 | #include <random>
4 | #include <algorithm>
5 | #include <iostream>
6 | #include <mutex>
7 |
8 | GeneralThread gen_thread;
9 |
10 | void card_function(long long rounds) {
11 |     std::vector<Card> deck = CreateDeck();
```



```

12
13     std::random_device rd;
14     std::mt19937 random_gen(rd());
15
16     long long success_shuffles = 0;
17
18     for (long long i = 0; i < rounds; ++i) {
19         std::shuffle(deck.begin(), deck.end(), random_gen);
20
21         if (deck[0].denomination == deck[1].denomination) {
22             ++success_shuffles;
23         }
24     }
25
26     std::lock_guard<std::mutex> lock(gen_thread.mutex);
27     gen_thread.total_suceses_rounds += success_shuffles;
28     gen_thread.total_rounds += rounds;
29 }

```

Листинг 4: src/monte_carlo_card.cpp

```

1  #include "monte_carlo_card.h"
2  #include "deck.h"
3  #include <iostream>
4  #include <vector>
5  #include <chrono>
6  #include <cstdlib>
7  #include <cstring>
8  #include <thread>
9  #include <mutex>
10
11 constexpr std::size_t MAX_ROUNDS = 1000000;
12
13 int main(int argc, char* argv[]) {
14     if (argc != 5) {
15         std::cerr << "Wrong arguments, expected './main --rounds [N] --max-threads [M\n\n";
16         return 1;
17     }
18
19     long long total_rounds;
20     int max_threads;
21
22     if (std::strcmp(argv[1], "--rounds") != 0) { std::cerr << "Wrong arguments,\n\n"; return 1; }
23     if (argc < 2) { std::cerr << "Wrong number of arguments\n"; return 1; }
24     total_rounds = std::atoll(argv[2]);
25
26     if (std::strcmp(argv[3], "--max-threads") != 0) { std::cerr << "Wrong arguments,\n\n"; return 1; }
27     if (argc < 4) { std::cerr << "wrong number of arguments\n"; return 1; }
28     max_threads = std::atoll(argv[4]);
29
30     if (total_rounds <= 0 || max_threads <= 0) {
31         std::cerr << "Rounds and max-threads number should be positive\n";
32         return 1;
33     }
34

```

```

35     long long rounds_per_thread = total_rounds / max_threads;
36     long long remainder = total_rounds % max_threads;
37
38     auto start_time = std::chrono::steady_clock::now();
39
40     std::vector<std::thread> threads;
41     long long rounds_for_cur_thread;
42     for (int i = 0; i < max_threads; ++i) {
43         rounds_for_cur_thread = rounds_per_thread + (i == 0 ? remainder : 0);
44         threads.emplace_back(card_function, rounds_for_cur_thread);
45     }
46
47     for (auto& t : threads) {
48         t.join();
49     }
50
51     auto end_time = std::chrono::steady_clock::now();
52     double probability = static_cast<double>(gen_thread.total_succeses_rounds) /
53         gen_thread.total_rounds;
54
55     std::cout << "Total rounds: " << total_rounds << "\n";
56     std::cout << "Successful rounds: " << gen_thread.total_succeses_rounds << "\n";
57     std::cout << "Probability: " << probability << " or " << probability * 100 << "%\n";
58     std::cout << "Duration: " << end_time - start_time << " or "
59         << std::chrono::duration_cast<std::chrono::duration<double>>(end_time -
60             start_time) << "\n";
61 }

```

Листинг 5: main.cpp

Strace

```

execve("./main", [ "./main", "--rounds", "10000000", "--max-threads", "12"], 0x7ffe355
brk(NULL)
= 0x55d894574000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7a07b7f8
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=20791, ...}) = 0
mmap(NULL, 20791, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7a07b7f83000
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\0\0\0\0"...
, 832) = 8
fstat(3, {st_mode=S_IFREG|0644, st_size=2592224, ...}) = 0
mmap(NULL, 2609472, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7a07b7c00000
mmap(0x7a07b7c9d000, 1343488, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE
, 0, 0) = 0x7a07b7c9d000
mmap(0x7a07b7de5000, 552960, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e5
, 0) = 0x7a07b7de5000
mmap(0x7a07b7e6c000, 57344, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE
, 0, 0) = 0x7a07b7e6c000
mmap(0x7a07b7e7a000, 12608, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS
, 0, 0) = 0x7a07b7e7a000
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\0\0\0\0"...
, 832) = 8
fstat(3, {st_mode=S_IFREG|0644, st_size=952616, ...}) = 0

```

```

mmap(NULL, 950296, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7a07b7e9a000
mmap(0x7a07b7eaa000, 520192, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE
mmap(0x7a07b7f29000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x8f0
mmap(0x7a07b7f81000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
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\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"..., 832) = 8
fstat(3, {st_mode=S_IFREG|0644, st_size=183024, ...}) = 0
mmap(NULL, 185256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7a07b7bd2000
mmap(0x7a07b7bd6000, 147456, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE
mmap(0x7a07b7bfa000, 16384, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2800
mmap(0x7a07b7bfe000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
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\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"..., 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\0\0"..., 784, 6
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\0\0"..., 784, 6
mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7a07b7800000
mmap(0x7a07b7828000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE
mmap(0x7a07b79b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0
mmap(0x7a07b79ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE
mmap(0x7a07b7a05000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS
close(3) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7a07b7e9
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7a07b7e
arch_prctl(ARCH_SET_FS, 0x7a07b7e95740) = 0
set_tid_address(0x7a07b7e95a10) = 40647
set_robust_list(0x7a07b7e95a20, 24) = 0
rseq(0x7a07b7e96060, 0x20, 0, 0x53053053) = 0
mprotect(0x7a07b79ff000, 16384, PROT_READ) = 0
mprotect(0x7a07b7bfe000, 4096, PROT_READ) = 0
mprotect(0x7a07b7f81000, 4096, PROT_READ) = 0
mprotect(0x7a07b7e6c000, 45056, PROT_READ) = 0
mprotect(0x55d881542000, 4096, PROT_READ) = 0
mprotect(0x7a07b7fc1000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7a07b7f83000, 20791) = 0
futex(0x7a07b7e7a7bc, FUTEX_WAKE_PRIVATE, 2147483647) = 0
getrandom("\x1a\x05\x45\x45\x70\xe1\x35\x67", 8, GRND_NONBLOCK) = 8
brk(NULL) = 0x55d894574000
brk(0x55d894595000) = 0x55d894595000
rt_sigaction(SIGRT_1, {sa_handler=0x7a07b7899530, sa_mask=[], sa_flags=SA_RESTORER|SA
rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7a07b6
mprotect(0x7a07b7000000, 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|

```

```
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7a07b6
mprotect(0x7a07b67ff000, 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|
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7a07b5
mprotect(0x7a07b5ffe000, 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|
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7a07b5
mprotect(0x7a07b57fd000, 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|
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7a07b4
mprotect(0x7a07b4ffc000, 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|
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7a07b4
mprotect(0x7a07b47fb000, 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|
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7a079f
mprotect(0x7a079f800000, 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|
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7a079e
mprotect(0x7a079efff000, 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|
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7a079e
mprotect(0x7a079e7fe000, 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|
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7a079d
mprotect(0x7a079dffd000, 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|
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7a079d
mprotect(0x7a079d7fc000, 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|
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7a079c
mprotect(0x7a079cffb000, 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|
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
futex(0x7a07b77ff990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 40648, NULL, FUTEX_BITS
futex(0x7a07b5ffc990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 40651, NULL, FUTEX_BITS
munmap(0x7a07b6fff000, 8392704) = 0
munmap(0x7a07b67fe000, 8392704) = 0
munmap(0x7a07b5ffd000, 8392704) = 0
munmap(0x7a07b57fc000, 8392704) = 0
munmap(0x7a07b4ffb000, 8392704) = 0
munmap(0x7a07b47fa000, 8392704) = 0
futex(0x7a079dffb990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 40658, NULL, FUTEX_BITS
munmap(0x7a079f7ff000, 8392704) = 0
munmap(0x7a079effe000, 8392704) = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
write(1, "Total rounds: 10000000\n", 23) = 23
write(1, "Successful rounds: 587118\n", 26) = 26
write(1, "Probability: 0.0587118 or 5.8711"... , 35) = 35
futex(0x7a07b7e7a7c8, FUTEX_WAKE_PRIVATE, 2147483647) = 0
write(1, "Duration: 5115134898ns or 5.1151"... , 35) = 35
exit_group(0) = ?
+++ exited with 0 +++

```