Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №3
по курсу «Операционные системы»
III Семестр

Вариант 14

Студент:	Короткевич Л. В.
Группа:	М80-208Б-19
Преподаватель:	Миронов Е.С
Оценка:	
Дата:	

Москва 2020

1. Постановка задачи

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение потоков должно быть задано ключом запуска вашей программы.

Привести исследование зависимости ускорения и эффективности алгоритма от входящих данных и количества потоков. Получившиеся результаты объяснить.

Вариант 14:

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

2. Метод решения

Используемые системные и библиотечные вызовы для выполнения работы:

int pthread_mutex_init(pthread_mutex_t *restrict mutex, const pthread_mutexattr_t *restrict attr);	Функция pthread_mutex_init() инициализирует мьютекс с атрибутами, заданными attr. Если значение attr равно NULL, то используются атрибуты мьютекса по умолчанию
int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg);	Функция pthread_create() запускает новый поток в вызывающем процессе. Новый поток начинает выполнение, вызывая start_routine(); arg передается в качестве единственного аргумента start_routine().
int pthread_join(pthread_t thread, void **retval);	Функция pthread_join() ожидает завершения указанного потока. Если этот поток уже завершен, то функция thread_join() немедленно возвращается.

int pthread_mutex_destroy (pthread_mutex_t *mutex);	Функция pthread_mutex_destroy() уничтожает мьютекс; мьютекс становится, по сути, неинициализированным.

Краткий алгоритм решения:

1) Получение числа раундов, кол-ва потоков в кач-ве аргументов при запуске программы.
2) Инициализация мьютекса.
3) Создание заданного числа потоков.
    1) Моделирование перемешивания колоды карт в функции потоков.
    2) Заблокировать мьютекс.
    3) Инкрементировать результат, переменную count в случае успешного результата эксперимента.
    4) Разблокировать мьютекс.
4) Ожидание завершения потоков.
5) Уничтожение мьютекса.

*Несколько подробней об отдельной функции void *thread_func(void *arg):* Функция получает на вход единственный аргумент – число раундов. Создается массив целых чисел размера 52. Заполняется числами от 0 до 12 (каждого из числа в массиве будет ровно 4, сколько и мастей в стандартной колоде). Устанавливается в качестве базы для rand() текущее время. После чего нехитрым алгоритмом случайным образом перемешивается исходный массив. Далее мы сравниваем две крайние карты и, соответственно, увеличиваем в случае положительного рез-тата глобальную переменную count на 1, прежде заблокировав мьютекс во избежание “гонки”. Наконец, разблокируем мьютекс и по мере завершения необх. числа раундов завершим работу определенной нити.

Вычисление теоретической вероятности для последующего сравнения точности полученных результатов при проведении экспериментов:
На первое место ставится одна из 52 карт. Следом за ней - одна из 3 оставшихся с равным первой значением (масти полагаем незначащим параметром). Следом - 50 любых карт в случайном порядке. Итого: 52 * 3 * 50! удовл. вариантов. Всего же – 52!. Вероятность равняется 52 * 3 * 50! / 52! = 3/51 – приблизительно 5.9%.

Метрики параллельных вычислений:

Метрики параллельных вычислений используется для оценки роста производительности, получаемого при параллельном решении задачи на n процессорах. Также они позволяют определить необходимое кол-во процессоров, используемых для решения конкретной задачи.

- Tp – время выполнения на p различных вычислительных ядрах
- Ускорение: Sp = T1/Tp, Sp < p
- Эффективность/загруженность: Xp = Sp/p, Xp < 1
- Закон Амдала (здесь ускорение показывает во сколько раз меньше времени потребуется параллельной прог-е для выполнения):

  $$Sp = 1 / (g + (1 - g) / p)$$

  g – доля последовательных расчётов в программе,

  p – количество процессоров,

  Sp – производительность.
- Закон Густавсона — Барсиса (здесь укорение -- это увеличение объема выполненной задачи за постоянный промежуток времени)

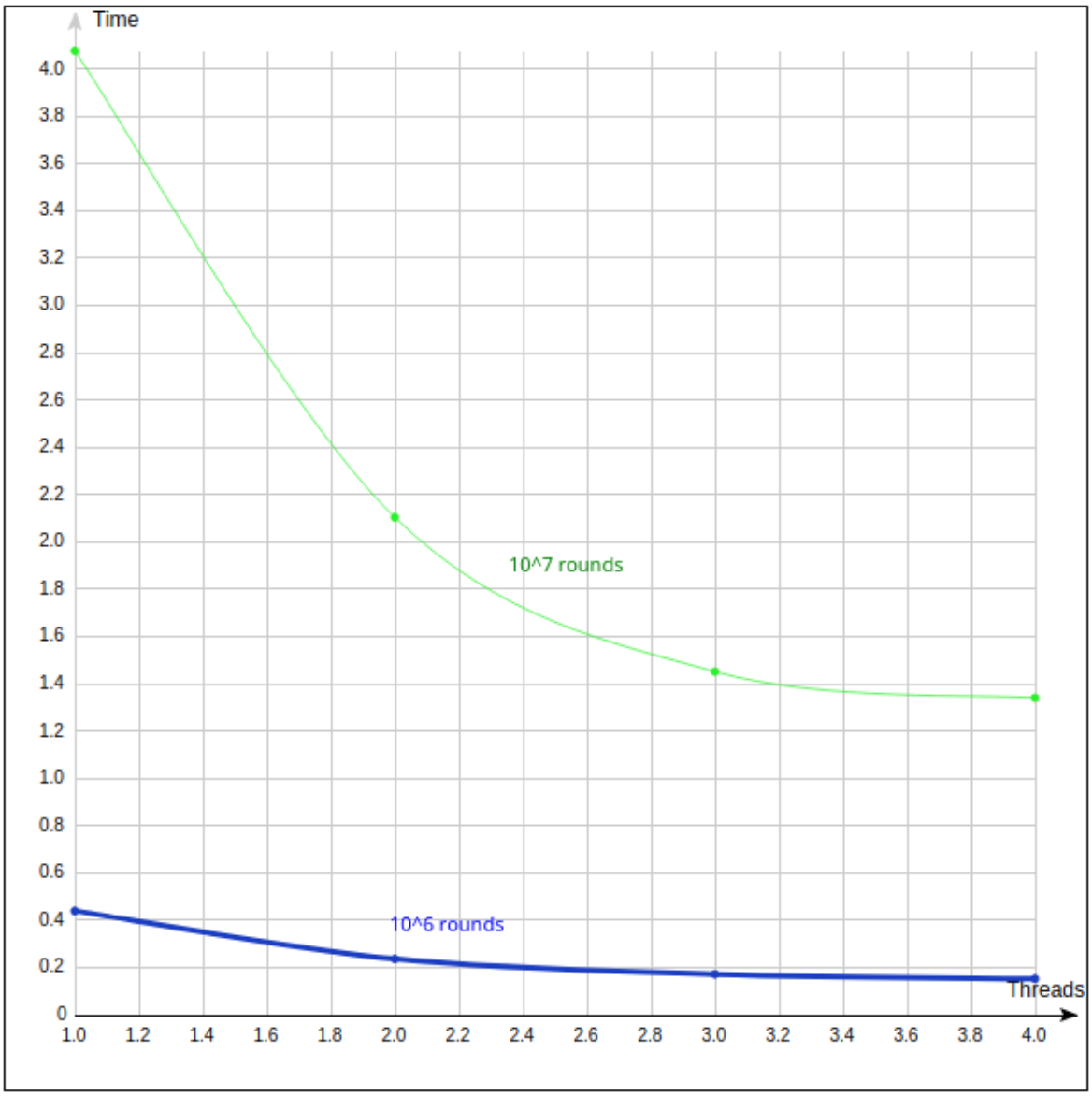
  $$Sp = g + (1 - g)p = p + (1 - p)g$$

  g – доля последовательных расчётов в программе,

  p – количество процессоров.

3. Тестирование

График зависимости времени от числа потоков при фиксированнмом числе раундов:



Теперь определим метрики параллельных вычислений на примере 10^6 раундов.

p	T1 (мс.)	Tp (мс.)	Ускорение (Sp=T1/Tp)	Эффективность (Xp=Sp/p)
1	4085278	4085278	1	1
2	4085278	2103742	1,941910177	0,9709550886
3	4085278	1451379	2,814756173	0,9382520578
4	4085278	1340375	3,047861979	0,7619654947

Консоль:

/ using threads */*
[leo@pc final]$ ./main 1000000 1
Probability equals to 0.059
Time spent: 0.406224 seconds
[leo@pc final]$ ./main 1000000 2
Probability equals to 0.058
Time spent: 0.192576 seconds
[leo@pc final]$ ./main 1000000 3
Probability equals to 0.058
Time spent: 0.135162 seconds
[leo@pc final]$ ./main 1000000 4
Probability equals to 0.059
Time spent: 0.150846 seconds

Strace

Сразу продемонстрирую лог с ключами -f для отслеживания всех потоков и -r для демонстрации «относительного времени» вызовов:

```
    0.000000 execve("./main", ["./main", "3", "1"], 0x7ffc61ba48d0 /* 58 vars */) = 0
    0.000966 brk(NULL)                  = 0x562b34237000
    0.000164 arch_prctl(0x3001 /* ARCH_??? */, 0x7ffdfecb8d90) = -1 EINVAL (Invalid argument)
    0.000510 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
    0.000311 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
    0.000378 fstat(3, {st_mode=S_IFREG|0644, st_size=208296, ...}) = 0
    0.000251 mmap(NULL, 208296, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fc19acdc000
    0.000151 close(3)                   = 0
    0.000115 openat(AT_FDCWD, "/usr/lib/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
    0.000139 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0000\201\0\0\0\0\0\0"..., 832) = 832
    0.000092 pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\307Y\373z\3054\277z\21\35\225\341\273\304<\223"..., 68,
824) = 68
    0.000093 fstat(3, {st_mode=S_IFREG|0755, st_size=158744, ...}) = 0
    0.000092 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fc19acda000
    0.000107 mmap(NULL, 135600, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fc19acb8000
    0.000091 mmap(0x7fc19acbf000, 65536, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x7000) = 0x7fc19acbf000
    0.000108 mmap(0x7fc19accf000, 20480, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x17000) = 0x7fc19accf000
    0.000093 mmap(0x7fc19acd4000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x1b000) = 0x7fc19acd4000
    0.000140 mmap(0x7fc19acd6000, 12720, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0x7fc19acd6000
    0.000379 close(3)                   = 0
    0.000233 openat(AT_FDCWD, "/usr/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
    0.000235 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\202\2\0\0\0\0\0"..., 832) = 832
    0.000222 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, 64) = 784
    0.000164 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
    0.000209 pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\364[g\253(\257\25\201\313\250\344q>\17\323\262"..., 68,
880) = 68
    0.000206 fstat(3, {st_mode=S_IFREG|0755, st_size=2159552, ...}) = 0
    0.000213 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, 64) = 784
    0.000098 mmap(NULL, 1868448, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fc19aaef000
    0.000097 mmap(0x7fc19ab15000, 1363968, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x26000) = 0x7fc19ab15000
    0.000109 mmap(0x7fc19ac62000, 311296, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x173000) = 0x7fc19ac62000
    0.000094 mmap(0x7fc19acae000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x1be000) = 0x7fc19acae000
```

0.000150 mmap(0x7fc19acb4000, 12960, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fc19acb4000
0.000160 close(3) = 0
0.000227 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fc19aaec000
0.000221 arch_prctl(ARCH_SET_FS, 0x7fc19aaec740) = 0
0.000287 mprotect(0x7fc19acae000, 12288, PROT_READ) = 0
0.000305 mprotect(0x7fc19acd4000, 4096, PROT_READ) = 0
0.000116 mprotect(0x562b33e8b000, 4096, PROT_READ) = 0
0.000095 mprotect(0x7fc19ad3b000, 4096, PROT_READ) = 0
0.000137 munmap(0x7fc19acdc000, 208296) = 0
0.000132 set_tid_address(0x7fc19aaeca10) = 26146
0.000080 set_robust_list(0x7fc19aaeca20, 24) = 0
0.000097 rt_sigaction(SIGRTMIN, {sa_handler=0x7fc19acbfb90, sa_mask=[], sa_flags=SA_RESTORER|SA_SIGINFO, sa_restorer=0x7fc19accc0f0}, NULL, 8) = 0
0.000114 rt_sigaction(SIGRT_1, {sa_handler=0x7fc19acbfc30, sa_mask=[], sa_flags=SA_RESTORER|SA_RESTART|SA_SIGINFO, sa_restorer=0x7fc19accc0f0}, NULL, 8) = 0
0.000096 rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0
0.000099 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
0.000166 clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=3803310}) = 0
0.000114 mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7fc19a2eb000
0.000094 mprotect(0x7fc19a2ec000, 8388608, PROT_READ|PROT_WRITE) = 0
0.000197 brk(NULL) = 0x562b34237000
0.000079 brk(0x562b34258000) = 0x562b34258000
0.000097 rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
0.000114 clone(child_stack=0x7fc19aaeaef0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTIDstrace: Process 26147 attached
, parent_tid=[26147], tls=0x7fc19aaeb640, child_tidptr=0x7fc19aaeb910) = 26147
[pid 26147] 0.000174 set_robust_list(0x7fc19aaeb920, 24 <unfinished ...>
[pid 26146] 0.000036 rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>
[pid 26147] 0.000041 <... set_robust_list resumed>) = 0
[pid 26146] 0.000026 <... rt_sigprocmask resumed>NULL, 8) = 0
[pid 26147] 0.000027 rt_sigprocmask(SIG_SETMASK, [], <unfinished ...>
[pid 26146] 0.000040 futex(0x7fc19aaeb910, FUTEX_WAIT, 26147, NULL <unfinished ...>
[pid 26147] 0.000037 <... rt_sigprocmask resumed>NULL, 8) = 0
[pid 26147] 0.000044 clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=4344600}) = 0
[pid 26147] 0.000102 clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=4386645}) = 0
[pid 26147] 0.000097 clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=4424800}) = 0
[pid 26147] 0.000108 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
[pid 26147] 0.000107 fstat(3, {st_mode=S_IFREG|0644, st_size=208296, ...}) = 0
[pid 26147] 0.000094 mmap(NULL, 208296, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fc19acdc000
[pid 26147] 0.000090 close(3) = 0
[pid 26147] 0.000111 mmap(NULL, 134217728, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_NORESERVE, -1, 0) = 0x7fc1922eb000
[pid 26147] 0.000088 munmap(0x7fc1922eb000, 30494720) = 0
[pid 26147] 0.000091 munmap(0x7fc198000000, 36614144) = 0
[pid 26147] 0.000085 mprotect(0x7fc194000000, 135168, PROT_READ|PROT_WRITE) = 0
[pid 26147] 0.000099 openat(AT_FDCWD, "/usr/lib/libgcc_s.so.1", O_RDONLY|O_CLOEXEC) = 3
[pid 26147] 0.000099 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"..., 832) = 832
[pid 26147] 0.000096 fstat(3, {st_mode=S_IFREG|0644, st_size=594704, ...}) = 0
[pid 26147] 0.000132 mmap(NULL, 103144, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fc19a2d1000
[pid 26147] 0.000096 mmap(0x7fc19a2d4000, 69632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x3000) = 0x7fc19a2d4000
[pid 26147] 0.000185 mmap(0x7fc19a2e5000, 16384, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x14000) = 0x7fc19a2e5000
[pid 26147] 0.000093 mmap(0x7fc19a2e9000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x17000) = 0x7fc19a2e9000
[pid 26147] 0.000130 close(3) = 0
[pid 26147] 0.000129 mprotect(0x7fc19a2e9000, 4096, PROT_READ) = 0
[pid 26147] 0.000108 munmap(0x7fc19acdc000, 208296) = 0

```
[pid 26147]     0.000134 futex(0x7fc19a2ea080, FUTEX_WAKE_PRIVATE, 2147483647) = 0
[pid 26147]     0.000099 rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
[pid 26147]     0.000109 madvise(0x7fc19a2eb000, 8368128, MADV_DONTNEED) = 0
[pid 26147]     0.000089 exit(0)        = ?
[pid 26146]     0.000106 <... futex resumed>) = 0
[pid 26147]     0.000020 +++ exited with 0 +++
    0.000021 clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=5531068}) = 0
    0.000125 fstat(1, {st_mode=S_IFCHR|0600, st_rdev=makedev(0x88, 0x1), ...}) = 0
    0.000111 write(1, "Probability equals to 0.000\n", 28Probability equals to 0.000
) = 28
    0.000107 write(1, "Time spent: 0.001728 seconds\n", 29Time spent: 0.001728 seconds
) = 29
    0.000102 exit_group(0)            = ?
    0.000263 +++ exited with 0 +++
```

Наблюдательно, но не удивительно: мы запустили 3 потока, и, как и должно быть, наблюдаем 3 вызова функции clock_gettime (выделены жирным).

Видим также сис. вызов clone с ключом CLONE_THREAD.

```
[leo@pc final]$ strace -f -r -y -c ./main 3 1
strace: -r/--relative-timestamps has no effect with -c/--summary-only
strace: -y/--decode-fds has no effect with -c/--summary-only
strace: Process 27692 attached
Probability equals to 0.000
Time spent: 0.000486 seconds
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0.00    0.000000           0         3           read
  0.00    0.000000           0         2           write
  0.00    0.000000           0         5           close
  0.00    0.000000           0         6           fstat
  0.00    0.000000           0        20           mmap
  0.00    0.000000           0         7           mprotect
  0.00    0.000000           0         4           munmap
  0.00    0.000000           0         3           brk
  0.00    0.000000           0         2           rt_sigaction
  0.00    0.000000           0         5           rt_sigprocmask
  0.00    0.000000           0         5           pread64
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           madvise
  0.00    0.000000           0         1           clone
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         2         1 arch_prctl
  0.00    0.000000           0         2           futex
  0.00    0.000000           0         1           set_tid_address
  0.00    0.000000           0         5           clock_gettime
  0.00    0.000000           0         5           openat
  0.00    0.000000           0         2           set_robust_list
  0.00    0.000000           0         1           prlimit64
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000           0        84         2 total
```

4. Листинг программы

main.c:

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

const int DECK_SIZE = 52;

static inline void swap(int *a, int *b)
{
   int tmp = *a;
   *a = *b;
   *b = tmp;
}

static unsigned long long g_seed;

// Used to seed the generator.
static inline void fast_srand(int seed)
{
   g_seed = seed;
}

// Compute a pseudorandom integer.
// Output value in range [0, 32767]
unsigned long long fast_rand(void)
{
   g_seed = (214013ull * g_seed + 2531011ull);
   return (g_seed >> 16ull) & 0x7FFFull;
}

int count = 0;
pthread_mutex_t mutex;

void *thread_func(void *arg)
{
   int localCount = 0, rounds = *((int *)arg);
   int deck[DECK_SIZE];
   int i, j, k;

   for (k = 0; k < rounds; ++k)
   {
      for (i = 0; i < DECK_SIZE; ++i)
      {
         deck[i] = (i + 1) % 13; // значения карт варьируются от 2 до 14 == от 0 до 12
      }

      for (i = DECK_SIZE - 1; i >= 0; --i)
      {
         j = fast_rand() % (i + 1);
         swap(&deck[i], &deck[j]);
      }

      if (deck[0] == deck[1])
      {
         ++localCount;
      }
   }

   pthread_mutex_lock(&mutex);
   count += localCount;
   pthread_mutex_unlock(&mutex);

   pthread_exit(0);
}

int main(int argc, char *argv[])
{
   fast_srand(time(NULL));
```

```
    if (argc != 3)
    {
        perror("Usage: ./main <number of rounds> <number of threads>\n");
        exit(1);
    }



    // for time-measurement
    struct timespec start, finish;
    double elapsed;

    // start time point
    clock_gettime(CLOCK_MONOTONIC, &start);



    // threads initialization
    pthread_t tid[atoi(argv[2])];

    if (pthread_mutex_init(&mutex, NULL) < 0)
    {
        perror("Mutex init error");
        exit(1);
    }

    // starting threads
    int rounds_per_thread = atoi(argv[1]) / atoi(argv[2]);
    for (int i = 0; i < atoi(argv[2]); ++i)
    {
        if (pthread_create(&tid[i], NULL, thread_func, &rounds_per_thread) != 0)
        {
            perror("Can't create thread\n");
            exit(1);
        }
    }

    // destroying threads
    for (int i = 0; i < atoi(argv[2]); ++i)
    {
        if (pthread_join(tid[i], NULL) != 0)
        {
            perror("Can't join threads");
            exit(1);
        }
    }

    if (pthread_mutex_destroy(&mutex) < 0)
    {
        perror("Mutex destroy error");
        exit(1);
    }



    // end time point
    clock_gettime(CLOCK_MONOTONIC, &finish);
    elapsed = (finish.tv_sec - start.tv_sec);
    elapsed += (finish.tv_nsec - start.tv_nsec) / 1000000000.0;



    printf("Probability equals to %.3lf\n", (double)count / atoi(argv[1]));
    printf("Time spent: %lf seconds\n", elapsed);

    return 0;
}
```

Вывод

По мере выполнения данной лабораторной работы я познал преимущества и недостатки многопоточности; освоил работу с мьютексами, их создание/удаление, использование во избежание некорректностей, нарушения последовательности выполнения задач потоками.

Логичным было наблюдать превосходство многопоточности над многопроцессорностью: несколько процессов запускаются с целью повышения отказоустойчивости приложения, а также с целью повышения безопасности, но не эффективности. Многопоточность же - вариант реализации вычислений, при котором для решения некоторой прикладной задачи запускаются и выполняются несколько независимых потоков вычислений, имеющих общую память; причём выполнение происходит одновременно или псевдоодновременно, что дает выигрыш во времени.

Также потоки быстрее создаются и уничтожаются, чем процессы (в некоторых системах в 10-100 раз). Это особенно критично в задачах, требующих частое изменение количества потоков.

Потоки одного процесса имеют полный доступ к адресному пространству всего процесса, в том числе таблице дескрипторов, сигналы и даже к данным других потоков. Один поток, например, может напрямую записывать данные в стек другого потока. Этого никто не запрещает.

Однако, потоки имеют следующий недостаток: при параллельной работе нескольких потоков может произойти ситауция, когда потоки одновременно обращаются к одному ресурсу. И, если оба из них просто "читают", то вроде бы и ничего плохого в этом нет, но если один из них ведет запись, то у потоков может оказаться неактуальная информация, а следовательно, дальнейшая работа программмы будет некорректной.

Для того, чтобы справиться с данной проблемой были придуманы примитивы синхронизации. В них входят критичиские области, барьеры, спин-блокировки, семафоры и т.д.

Наблюдательно, что для точной оценки вероятности (схожей с теоретической до сотых) достаточно всего одной тысячи раундов. Я был приятно удивлен тому, как просто проверять правдивость тех или иных теоретический утверждений с помощью компьютера. Полагаю, без многопоточности сей процесс занял бы большое время.

Интересно также было реализовывать метод Монте-Карло: полное моделирование ситуации, эксперимента для последующего получения результатов. Хотя в данной ЛР мне кажется избыточным полное моделирование процесса (можно было бы обойтись случайной генерацией двух крайних карт), в других случаях, как, например, вычисление числа «пи» с точностью до n знаков и других вычислительных задачах без этого никуда!