Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

Студент: Смирнов А.В.

Группа: М8О-207Б-21

Вариант: 14

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

<https://github.com/Liguha/OS>

**Постановка задачи**

**Цель работы**

Целью является приобретение практических навыков в:

Управление потоками в ОС  
Обеспечение синхронизации между потоками

**Задание**

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.  
 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.

**Общие сведения о программе**

Программа компилируется из threads.c, использует заголовочные файлы “stdio.h”, “stdlib.h” и “pthread.h”. В программе присутствует следующие библиотечные функции:

1. pthread\_create() – создание нового потока
2. pthread\_join() – ожидание окончания потока
3. pthread\_mutex\_init() – инициализация мьютекса
4. pthread\_mutex\_lock() – блокировка мьютекса
5. pthread\_mutex\_unlock() – разблокировка мьютекса
6. pthread\_mutex\_destroy() – удаление мьютекса

**Общий метод и алгоритм решения**

После получения входных данных программа создаёт N потоков, где N = min(K, T), где K – количество раундов, а T – максимальное количество потоков. В функцию, выполняемую потоком, передаётся 2 значение: количество раундов, моделируемых потоком, и ключ генерации случайных чисел. Количество раундов потока вычисляется как K/T + 1 или K/T, прибавка единицы зависит от номера потока и остатка от деления K на T.

Моделирование карт: очевидно, что нам важны лишь 2 верхние карты, поэтому вместо генерации полной колоды, будем генерировать 2 числа a и b, a ∈ [0, 51], b ∈ [0, 50], если b ≥ a, то прибавим к b 1. Так мы получим 2 разных числа от 0 до 51, каждое из которых означает карту. Т.к. карт каждой масти в колоде поровну (52 / 4 = 13), то можно считать, что наверху 2 одинаковые карты, если их остатки деления на 13 равны.

После требуемого числа раундов функция потока прибавит количество переменных к счётчику и завершит работу. Когда основной поток дождётся всех остальных он должен получить вероятность – просто разделить счётчик на K.

Учитывая подобный метод распараллеливания, получится, что программа с T потоками в примерно T раз быстрее программы с 1 потоком. Однако это ускорение имеет предел, равный количеству процессорных ядер, т.к. в случае, если потоков больше, чем ядер, то как минимум 1 ядро возьмёт на себя 2 потока и будет переключаться между ними, что только ухудшит производительность.

**Исходный код**

|  |
| --- |
| **threads.c** |
| #include "stdio.h"  #include "stdlib.h"  #include "pthread.h"  #define DECK\_SIZE 52  pthread\_mutex\_t mutex;  long long counter = 0;  long long min(long long l, long long r)  {  if (l > r)  return r;  return l;  }  void\* make\_rounds(void\* args)  {  long long n = \*(long long\*)args;  unsigned int seed = (unsigned int)\*(long long\*)(args + sizeof(void\*));  int res = 0;  for (int i = 0; i < n; i++)  {  int a = rand\_r(&seed) % DECK\_SIZE;  int b = rand\_r(&seed) % (DECK\_SIZE - 1);  b += (b >= a);  res += (a % (DECK\_SIZE / 4)) == (b % (DECK\_SIZE / 4));  }  long long er;  if (er = pthread\_mutex\_lock(&mutex))  return (void\*)er;  counter += res;  if (er = pthread\_mutex\_unlock(&mutex))  return (void\*)er;  return NULL;  }  int main(int argc, char\* argv[])  {  long long rounds = 1;  int max\_threads = 0;  unsigned int seed = time(NULL);  int er;  scanf("%d %lld", &max\_threads ,&rounds);  int d = rounds % max\_threads;  long long\*\* data = (long long\*\*)calloc(max\_threads, sizeof(long long\*));  if (data == NULL)  {  printf("Alloc error");  return -1;  }  for (int i = 0; i < max\_threads; i++)  {  data[i] = (long long\*)calloc(2, sizeof(long long));  if (data[i] == NULL)  {  printf("Alloc error");  return -1;  }  }  pthread\_t\* threads = (pthread\_t\*)calloc(max\_threads, sizeof(pthread\_t));  if (threads == NULL)  {  printf("Alloc error");  return -1;  }  if (er = pthread\_mutex\_init(&mutex, NULL))  {  printf("Mutex init error: %d", er);  return -1;  }  for (int i = 0; i < min(max\_threads, rounds); i++)  {  data[i][0] = rounds / max\_threads + (d > 0);  d--;  data[i][1] = seed + i;  if (er = pthread\_create(&threads[i], NULL, make\_rounds, (void\*)data[i]))  {  printf("Thread create error: %d", er);  return -1;  }  }  for (int i = 0; i < min(max\_threads, rounds); i++)  {  long long out = 0;  if (er = pthread\_join(threads[i], (void\*\*)&out))  {  printf("Thread join error: %d", er);  return -1;  }  if (out)  {  printf("Mutex lock/unlock error: %lld", out);  return -1;  }  }  if (er = pthread\_mutex\_destroy(&mutex))  {  printf("Mutex destroy error: %d", er);  return -1;  }  free((void\*)threads);  for (int i = 0; i < max\_threads; i++)  free((void\*)data[i]);  free((void\*)data);  double res = (double)counter / rounds;  printf("%lf\n", res);  } |

**Демонстрация работы программы**

liguha@Laptop:~/OS/LR3/build$ cat test1.txt

1 2000000000

liguha@Laptop:~/OS/LR3/build$ cat test12.txt

12 2000000000

liguha@Laptop:~/OS/LR3/build$ time ./a.out < test1.txt

0.058810

real 0m23,544s

user 0m23,512s

sys 0m0,004s

liguha@Laptop:~/OS/LR3/build$ time ./a.out < test12.txt

0.058804

real 0m4,045s

user 0m46,225s

sys 0m0,060s

liguha@Laptop:~/OS/LR3/build$ strace -f -e trace="exit" ./a.out < test12.txt

strace: Process 6009 attached

strace: Process 6010 attached

strace: Process 6011 attached

strace: Process 6012 attached

strace: Process 6013 attached

strace: Process 6014 attached

strace: Process 6015 attached

strace: Process 6016 attached

strace: Process 6017 attached

strace: Process 6018 attached

strace: Process 6019 attached

strace: Process 6020 attached

[pid 6020] exit(0) = ?

[pid 6020] +++ exited with 0 +++

[pid 6018] exit(0) = ?

[pid 6018] +++ exited with 0 +++

[pid 6012] exit(0) = ?

[pid 6012] +++ exited with 0 +++

[pid 6009] exit(0) = ?

[pid 6009] +++ exited with 0 +++

[pid 6015] exit(0) = ?

[pid 6015] +++ exited with 0 +++

[pid 6019] exit(0) = ?

[pid 6019] +++ exited with 0 +++

[pid 6016] exit(0) = ?

[pid 6016] +++ exited with 0 +++

[pid 6013] exit(0) = ?

[pid 6013] +++ exited with 0 +++

[pid 6017] exit(0) = ?

[pid 6017] +++ exited with 0 +++

[pid 6011] exit(0) = ?

[pid 6011] +++ exited with 0 +++

[pid 6010] exit(0) = ?

[pid 6010] +++ exited with 0 +++

[pid 6014] exit(0) = ?

[pid 6014] +++ exited with 0 +++

0.058799

+++ exited with 0 +++

**Выводы**

Составлена и отлажена многопоточная программа на языке Си, рассчитывающая вероятность методом Монте-Карло. Тем самым, приобретены навыки в распараллеливании вычислений, управлении потоками и обеспечении синхронизации между ними.