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-214Б-24

Студент: Дылдин С.В.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 17.11.25

Москва, 2025

Постановка задачи

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

Общий метод и алгоритм решения

ПОТОКИ:

- int pthread_create() - создаёт новый поток
- *int pthread_join(pthread_t, thread void retval) - ожидает завершения потока
- pthread_t - тип идентификатора потока

Описание:

Взаимодействие двух процессов.

- Программа распределяет входные данные по указанному кол-ву потоков и создает их. Каждый поток тасует колоду карт, проверяет на совпадение рангов 2 верхних карт. Производит Суммирование полученных локальных результатов с глобальным результатом при помощи блокировки мьютекса для избежания гонки данных. Выводит отформатированный ответ.

Код программы

main.c (без примитивов синхронизации)

```c
#include <pthread.h>
#include <stdio.h>
#include "add.h"
#include <stdlib.h>
#include <time.h>
#include <string.h>

// т.к. общий add.h, а там прописан extern
long global_matches = 0;
pthread_mutex_t result_mutex = PTHREAD_MUTEX_INITIALIZER;

int main(int argc, char **argv) {
    long roundsAmount = 0;
    int threadsAmount = 0;

    int parsed = parseInput(argc, argv, &roundsAmount, &threadsAmount);
    if (parsed == 0) {
        return 1;
    }
    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC, &start);
```

```
    pthread_t *threads = (pthread_t*)malloc(sizeof(pthread_t) * threadsAmount);
    if (threads == NULL) {
        printf("memory allocation error\n");
        return 1;
    }
    threadData *data = (threadData*)malloc(sizeof(threadData) * threadsAmount);
    if (data == NULL) {
        printf("memory allocation error\n");
        return 1;
    }

    long roundsPerThread = roundsAmount / threadsAmount;
    long roundsRemain = roundsAmount % threadsAmount;

    for (int i = 0; i < threadsAmount ; i++) {
        data[i].rounds = roundsPerThread;
        if (i == 0) {
            data[i].rounds += roundsRemain;
        }
        data[i].seed = (unsigned int)(time(NULL) ^ (i * 12345));
        data[i].matches = 0;

        if (pthread_create(&threads[i], NULL, process_no_mutex, &data[i]) != 0) {
            printf("Error creating thread\n");
            return 1;
        }
    }

    for (int i = 0; i < threadsAmount; i++) {
        pthread_join(threads[i], NULL);
    }


    long total_matches = 0;
    for (int i = 0; i < threadsAmount; i++) {
        total_matches += data[i].matches;
    }

    clock_gettime(CLOCK_MONOTONIC, &end);
    double elapsed = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) /
1e9;

    printf("\nTotal rounds: %ld\n", roundsAmount);
    printf("Matches: %ld\n", total_matches);
    printf("Probability: %.6f\n", (double)total_matches / roundsAmount);
    printf("Time: %.3f sec\n", elapsed);

    free(threads);
    free(data);
```

```
    return 0;
}
```

main.c (с мьютексом)

```
#include <pthread.h>
#include <stdio.h>
#include "add.h"
#include <stdlib.h>
#include <time.h>
#include <string.h>



long global_matches = 0;
pthread_mutex_t result_mutex = PTHREAD_MUTEX_INITIALIZER;


int main(int argc, char **argv) {
    long roundsAmount = 0;
    int threadsAmount = 0;
    int parsed = parseInput(argc, argv, &roundsAmount, &threadsAmount);
    if (parsed == 0) {
        return 1;
    }
    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC, &start);

    pthread_t *threads = (pthread_t*)malloc(sizeof(pthread_t) * threadsAmount);
    if (threads == NULL) {
        printf("memory allocation error\n");
        return 1;
    }
    threadData *data = (threadData*)malloc(sizeof(threadData) * threadsAmount);
    if (data == NULL) {
        printf("memory allocation error\n");
        return 1;
    }

    long roundsPerThread = roundsAmount / threadsAmount;
    long roundsRemain = roundsAmount % threadsAmount;
    for (int i = 0; i < threadsAmount ; i++) {
        data[i].rounds = roundsPerThread;
        if (i == 0) {
            data[i].rounds += roundsRemain;
        }
        data[i].seed = (unsigned int)(time(NULL) ^ (i * 12345));

        if (pthread_create(&threads[i], NULL, process, &data[i]) != 0) {
```

```
            printf("error while creating thread\n");
            return 1;
        }
    }

    for (int i = 0; i < threadsAmount; i++) {
        pthread_join(threads[i], NULL);
    }
    clock_gettime(CLOCK_MONOTONIC, &end);
    double elapsed = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1e9;

    printf("\nTotal rounds: %ld\n", roundsAmount);
    printf("Matches: %ld\n", global_matches);
    printf("Probability: %.6f\n", (double)global_matches / roundsAmount);
    printf("Time: %.3f sec\n", elapsed);

    free(threads);
    free(data);
    pthread_mutex_destroy(&result_mutex);

    return 0;

}
```

funcs.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "add.h"

void shuffle(int *deck, unsigned int *seed) {
    for (int i = 51; i > 0; i--) {
        int j = rand_r(seed) % (i + 1);
        int temp = deck[i];
        deck[i] = deck[j];
        deck[j] = temp;
    }
}


int parseInput(int argc, char **argv, long *rounds, int *threads) {
    for (int i = 1; i < argc; i++) {
        if (strcmp(argv[i], "-n") == 0 && i + 1 < argc) {
            i++;
            *rounds = atol(argv[i]);
            if (*rounds <= 0) {
                printf("Enter positive amount of rounds\n");
```

```
                return 0;
            }
        } else if (strcmp(argv[i], "-t") == 0 && i + 1 < argc) {
            i++;
            *threads = atoi(argv[i]);
            if (*threads <= 0) {
                printf("Enter positive amount of theads\n");
                return 0;
            }
        } else {
            printf("Wtf you entered?\n");
            return 0;
        }
    }
    return 1;
}

//кол-во раундов и кол-во потоков
void *process(void *something) {
    threadData *data = (threadData*)something;
    int deck[52];
    long local_matches = 0;
    for (int i = 0; i < 52; i++) {
        deck[i] = i;
    }

    for (int i = 0; i < data->rounds; i++) {
        shuffle(deck, &data->seed);
        if (deck[0] % 13 == deck[1] % 13) {
            local_matches++;
        }
    }
    pthread_mutex_lock(&result_mutex);
    global_matches += local_matches;
    pthread_mutex_unlock(&result_mutex);
    return NULL;
}

//рещение без мьютексов, используются локальные данные
void *process_no_mutex(void *something) {
    threadData *data = (threadData*)something;
    int deck[52];
    long local_matches = 0;

    for (int i = 0; i < 52; i++) {
        deck[i] = i;
    }

    for (int i = 0; i < data->rounds; i++) {
```

```
        shuffle(deck, &data->seed);
        if (deck[0] % 13 == deck[1] % 13) {
            local_matches++;
        }
    }

    data->matches = local_matches;

    return NULL;
}
```

add.h

```
#pragma once

#include <pthread.h>

typedef struct {
    long rounds;
    unsigned int seed;
    long matches;
} __attribute__((aligned(64))) threadData;

void shuffle(int *deck, unsigned int *seed);
int parseInput(int argc, char **argv, long *rounds, int *threads);
void *process(void *something);
extern long global_matches;
extern pthread_mutex_t result_mutex;
void *process_no_mutex(void *something);
```

Протокол работы программы

С мьютексом
1_000_000

Кол-во потоков	Время	Ускорение	Эффективность
1	0.193	1.000	1.0
2	0.096	2.010	1.005
8	0.044	4.386	0.548
12(лог ядра)	0.034	5.676	0.473
1024	0.056	3.446	0.003
4096	0.135	1.430	0.0003

Без примитивов синхронизации
1_000_000

Кол-во потоков	Время	Ускорение	Эффективность
1	**0.193**	**1.000**	**1.0**
2	**0.097**	**1.990**	**0.995**
8	**0.043**	**4.488**	**0.561**
12(лог ядра)	**0.039**	**4.949**	**0.412**
1024	**0.053**	**3.642**	**0.004**
4096	**0.131**	**1.473**	**0.0004**

Наибольшая производительность была в диапазоне 12 потоков, быстрее примерно в 4-6 раз однопоточной версии. На маленьких данных различий между версией с мьютексом и без почти нет. При большом кол-ве потоков, например 4096 - наблюдается деградация производительности из-за большого кол-ва накладных ресурсов на создание потоков и из-за меньшего кол-ва логических ядер - псевдопараллелизм.

Протокол работы программы

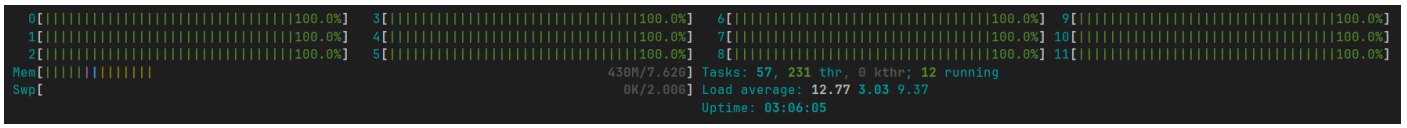
С мьютексом
100_000_000

Кол-во потоков	Время	Ускорение	Эффективность
1	**19.049**	**1.000**	**1.0**
2	**9.954**	**1.914**	**0.957**
8	**4.437**	**3.363**	**0.841**
12(лог ядра)	**4.194**	**4.542**	**0.379**
1024	**4.045**	**4.709**	**0.005**
4096	**4.133**	**4.609**	**0.001**

Без примитивов синхронизации
100_000_000

Кол-во потоков	Время	Ускорение	Эффективность
1	19.797	1.000	1.000
2	10.213	1.938	0.969
8	4.632	4.274	0.534
12(лог ядра)	4.438	4.461	0.372
1024	4.289	4.616	0.005
4096	4.449	4.450	0.001

Максимальное ускорение примерно 4-5 раз, При большом кол-ве потоков значения почти не отличаются, видимо потому что накладные расходы не особо большие, по сравнению с кол-вом вычислений. Версия с мьютексом и без практически одинаковы.



Тестирование:

С мьютексом

/mnt/d/MAI/3semestr/OSI/laba2 → for t in 1 2 4 8 12 128 1024 4096; do echo "Threads: $t"; ./main -n 1000000 -t $t; echo "----------"; done
Threads: 1

Total rounds: 1000000
Matches: 58817
Probability: 0.058817
Time: 0.193 sec

Threads: 2

Total rounds: 1000000
Matches: 58980
Probability: 0.058980

Time: 0.096 sec

Threads: 4

Total rounds: 1000000
Matches: 58901
Probability: 0.058901
Time: 0.054 sec

Threads: 8

Total rounds: 1000000
Matches: 59054
Probability: 0.059054
Time: 0.044 sec

Threads: 12

Total rounds: 1000000
Matches: 58954
Probability: 0.058954
Time: 0.034 sec

Threads: 128

Total rounds: 1000000
Matches: 59286
Probability: 0.059286
Time: 0.037 sec

Threads: 1024

Total rounds: 1000000
Matches: 58507
Probability: 0.058507
Time: 0.056 sec

Threads: 4096

Total rounds: 1000000
Matches: 58860
Probability: 0.058860
Time: 0.135 sec

Без примитивов синхронизации

```
/mnt/d/MAI/3semestr/OSI/laba2 → for t in 1 2 4 8 12 128 1024 4096; do echo "Threads: $t"; ./main_new -n 1000000 -t $t; echo "----------------"; done
Threads: 1

Total rounds: 1000000
Matches: 58880
Probability: 0.058880
Time: 0.193 sec
----------------
Threads: 2

Total rounds: 1000000
Matches: 59168
Probability: 0.059168
Time: 0.097 sec
----------------
Threads: 4

Total rounds: 1000000
Matches: 58880
Probability: 0.058880
Time: 0.054 sec
----------------
Threads: 8

Total rounds: 1000000
Matches: 58676
Probability: 0.058676
Time: 0.043 sec
----------------
Threads: 12

Total rounds: 1000000
Matches: 58811
Probability: 0.058811
Time: 0.039 sec
----------------
Threads: 128

Total rounds: 1000000
Matches: 58565
Probability: 0.058565
Time: 0.033 sec
----------------
Threads: 1024
```

```
Total rounds: 1000000
Matches: 58485
Probability: 0.058485
Time: 0.053 sec
----------
Threads: 4096

Total rounds: 1000000
Matches: 59113
Probability: 0.059113
Time: 0.131 sec
----------
/mnt/d/MAI/3semestr/OSI/laba2 →
```

strace

```
/mnt/d/MAI/3semestr/OSI/laba2 → strace ./main_new -n 10000 -t 1
execve("./main_new", ["./main_new", "-n", "10000", "-t", "1"], 0x7ffd0dd6b040 /* 30 vars */) = 0
brk(NULL)                               = 0x5c9584da8000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
 0) = 0x733c72981000
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=34311, ...}) = 0
mmap(NULL, 34311, PROT_READ, MAP_PRIVATE, 3, 0) = 0x733c72978000
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\0\220\243\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
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"..., 784, 64) = 784

mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x733c72600000
mmap(0x733c72628000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x733c72628000
mmap(0x733c727b0000, 323584, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x733c727b0000

mmap(0x733c727ff000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x733c727ff000
mmap(0x733c72805000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x733c72805000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
```

```
0) = 0x733c72975000
arch_prctl(ARCH_SET_FS, 0x733c72975740) = 0
set_tid_address(0x733c72975a10)         = 828864
set_robust_list(0x733c72975a20, 24)     = 0
rseq(0x733c72976060, 0x20, 0, 0x53053053) = 0
mprotect(0x733c727ff000, 16384, PROT_READ) = 0
mprotect(0x5c954737e000, 4096, PROT_READ) = 0
mprotect(0x733c729b9000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) =
0
munmap(0x733c72978000, 34311)           = 0
getrandom("\x5d\xea\x3b\x5b\xe7\xf1\xbb\x8a", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x5c9584da8000
brk(0x5c9584dc9000)                     = 0x5c9584dc9000
rt_sigaction(SIGRT_1, {sa_handler=0x733c72699530, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO,
sa_restorer=0x733c72645330}, 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) = 0x733c71dff000
mprotect(0x733c71e00000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREA
D|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEAR
TID, child_tid=0x733c725ff990, parent_tid=0x733c725ff990, exit_signal=0,
stack=0x733c71dff000, stack_size=0x7fff80, tls=0x733c725ff6c0} => {parent_tid=[828865]}, 88)
=
828865
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
futex(0x733c725ff990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 828865, NULL,
FUTEX_BITSET_MATCH_ANY) = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x5), ...}) = 0
write(1, "\n", 1
)                       = 1
write(1, "Total rounds: 10000\n", 20Total rounds: 10000
)   = 20
write(1, "Matches: 597\n", 13Matches: 597
)         = 13
write(1, "Probability: 0.059700\n", 22Probability: 0.059700
) = 22
write(1, "Time: 0.004 sec\n", 16Time: 0.004 sec
)      = 16
exit_group(0)                           = ?
+++ exited with 0 +++
/mnt/d/MAI/3semestr/OSI/laba2 →
```

100млн раундов без примитивов

/mnt/d/MAI/3semestr/OSI/laba2 → for t in 1 2 4 8 12 128 1024 4096; do echo "Threads: $t"; ./main_new -n 100000000 -t $t; echo "----------"; done

Threads: 1

Total rounds: 100000000
Matches: 5884799
Probability: 0.058848
Time: 19.797 sec

Threads: 2

Total rounds: 100000000
Matches: 5883185
Probability: 0.058832
Time: 10.213 sec

Threads: 4

Total rounds: 100000000
Matches: 5883417
Probability: 0.058834
Time: 6.187 sec

Threads: 8

Total rounds: 100000000
Matches: 5879591
Probability: 0.058796
Time: 4.632 sec

Threads: 12

Total rounds: 100000000
Matches: 5885380
Probability: 0.058854
Time: 4.438 sec

Threads: 128

Total rounds: 100000000
Matches: 5879479
Probability: 0.058795
Time: 4.245 sec

Threads: 1024

Total rounds: 100000000
Matches: 5880552
Probability: 0.058806
Time: 4.289 sec

Threads: 4096

Total rounds: 100000000
Matches: 5878836
Probability: 0.058788
Time: 4.449 sec

100млн с мьютексом

/mnt/d/MAI/3semestr/OSI/laba2 → for t in 1 2 4 8 12 128 1024 4096; do echo "Threads: $t"; ./main -n 100000000 -t $t; echo "----------"; done
Threads: 1

Total rounds: 100000000
Matches: 5878291
Probability: 0.058783
Time: 19.049 sec

Threads: 2

Total rounds: 100000000
Matches: 5879707
Probability: 0.058797
Time: 9.954 sec

Threads: 4

Total rounds: 100000000
Matches: 5879131
Probability: 0.058791
Time: 5.664 sec

Threads: 8

Total rounds: 100000000
Matches: 5880084

Probability: 0.058801
Time: 4.437 sec

Threads: 12

Total rounds: 100000000
Matches: 5880639
Probability: 0.058806
Time: 4.194 sec

Threads: 128

Total rounds: 100000000
Matches: 5881755
Probability: 0.058818
Time: 3.963 sec

Threads: 1024

Total rounds: 100000000
Matches: 5881212
Probability: 0.058812
Time: 4.045 sec

Threads: 4096

Total rounds: 100000000
Matches: 5880314
Probability: 0.058803
Time: 4.133 sec

/mnt/d/MAI/3semestr/OSI/laba2 →

Вывод

Научился создавать потоки в Си, распределять нагрузку между ними, использовать примитивы синхронизации и понимать, когда они нужны, а когда - нет.