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-23

Студент: Ткаченко Е. А.

Преподаватель: Бахарев В.Д.

Оценка:

Дата:

Москва, 2024

Постановка задачи

Вариант 7.

Два человека играют в кости. Правила игры следующие: каждый игрок делает бросок 2-ух костей К раз; побеждает тот, кто выбросил суммарно большее количество очков. Задача программы экспериментально определить шансы на победу каждого из игроков. На вход программе подается К, какой сейчас тур, сколько очков суммарно у каждого из игроков и количество экспериментов, которые должна произвести программа

Общий метод и алгоритм решения

Использованные системные вызовы:

write() – записываем число байт из буфера в указанный файловый дескриптор
pthread_create() – создаем новый поток с атрибутами
pthread_join() – ожидаем завершение потока
pthread_mutex_lock(): блокирует мьютекс, если он свободен. Если мьютекс уже заблокирован, поток будет ждать, пока он не станет доступен.
pthread_mutex_unlock(): освобождает мьютекс, позволяя другим потокам его заблокировать.

Создадим функцию которая будет производить необходимое нам количество экспериментов. Разделим введенное пользователем количество экспериментов между потоками и будем запускать функцию в каждом потоке. Результатом работы функции будет суммарное количество очков за все эксперименты проведенные в потоке. После объединения потоков суммируем счет и считаем вероятности.

Число потоков	Время исполнения (мс)	Ускорение	Эффективность
1	180	1	1
2	103	1,74	0,87
3	73	2,46	0,82
4	74	2,43	0,61
5	69	2,61	0,52
6	68	2,64	0,44

Время выполнения уменьшается при увеличении числа потоков, так как задачи распределяются между несколькими потоками, что позволяет выполнять больше вычислений параллельно. Однако, начиная с определённого количества потоков (около 4 в данном случае), снижение времени останавливается или даже возрастает. Это происходит так как каждый поток должен получать доступ к общим ресурсам. Ускорение сначала увеличивается с ростом числа потоков, поскольку каждый поток обрабатывает свою часть раундов, что ускоряет общую работу программы. Но при большем числе потоков ускорение перестаёт расти из-за увеличивающихся накладных расходов, что приводит к снижению эффективности. Эффективность падает, так как затраты на управление потоками становятся выше, чем прирост в производительности, что видно по снижающемуся значению эффективности при большем числе потоков.

Код программы

main.c

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <pthread.h>
#include <time.h>

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; // Инициализация мьютекса

// Проверка, являются ли все символы строками цифрами
int validate_nums(int amount, char* nums[]) {
    for (int j = 0; j <= amount; ++j) {
        int len_str = strlen(nums[j]);
        for (int i = 0; i < len_str; ++i) {
            if (nums[j][i] < '0' || nums[j][i] > '9') {
                return 0;
            }
        }
    }
    return 1;
}

// Преобразование числа с плавающей точкой в строку
void float_to_string(long double value, char *buffer, int len, int precision) {
    for (int i = 0; i < len; ++i) {
        buffer[i] = '\0';
    }
    if (value < 0) {
        *buffer++ = '-';
        value = -value;
    }
    int integerPart = (int)value;
    float fractionalPart = value - integerPart;

    char *intPtr = buffer;
    if (integerPart == 0) {
        *intPtr++ = '0';
    } else {
        char temp[20];
        int i = 0;
        while (integerPart > 0) {
            temp[i++] = (integerPart % 10) + '0';
            integerPart /= 10;
        }
        while (i > 0) {
            *intPtr++ = temp[--i];
        }
    }

    *intPtr++ = '.';

    for (int i = 0; i < precision; i++) {
```

```
        fractionalPart *= 10;
        int fractionalDigit = (int)fractionalPart;
        *intPtr++ = fractionalDigit + '0';
        fractionalPart -= fractionalDigit;
    }

    *intPtr++ = '%';
    *intPtr = '\n';
}

// Функция, выполняемая потоками
void* calc_score(void* arg) {
    unsigned long current  = time(NULL);
    const unsigned long a = 1664525;
    const unsigned long c = 1013904223;
    const unsigned long m = 4294967296;

    int rounds = ((int*)arg)[0];
    int exp = ((int*)arg)[1];
    int first_score = ((int*)arg)[2];
    int second_score = ((int*)arg)[3];

    int* score = malloc(sizeof(int) * 2);
    long double* result = malloc(sizeof(long double) * 3);
    result[0] = result[1] = result[2] = 0; // Инициализация

    for (int j = 0; j < exp; j++) {
        score[0] = first_score;
        score[1] = second_score;
        for (int i = 0; i < rounds; i++) {
            current = (current * a + c) % m;
            score[0] += current % 6 + 1;
            current = (current * a + c) % m;
            score[0] += current % 6 + 1;

            current = (current * a + c) % m;
            score[1] += current % 6 + 1;
            current = (current * a + c) % m;
            score[1] += current % 6 + 1;
        }

        // Защита доступа к результатам
        pthread_mutex_lock(&mutex);
        if (score[0] > score[1]) {
            result[0] += 1.0;
        } else if (score[0] < score[1]) {
            result[1] += 1.0;
        }
        result[2] += 1.0;
        pthread_mutex_unlock(&mutex);
    }

    free(arg);
    free(score);
```

```
    return result;
}

int main(int argc, char* argv[]) {
    if (argc != 7) {
        char msg[] = "USAGE: ./a.out <total rounds> <current tour> <first_score> <second_score>
<experiments> <threads>\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        return 1;
    }

    if (validate_nums(argc, argv)) {
        char msg[] = "ERROR: all input numbers must be integer and positive\n";

        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        return 2;
    }

    int k = atoi(argv[1]);
    int tour = atoi(argv[2]);
    int first_score = atoi(argv[3]);
    int second_score = atoi(argv[4]);
    int experiments = atoi(argv[5]);
    int num_threads = atoi(argv[6]);

    if (k < tour) {
        char msg[] = "ERROR: total rounds must be greater than current tour\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        return 3;
    }

    if (num_threads < 1) {
        char msg[] = "ERROR: number of threads must be greater than 1\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        return 4;
    }

    pthread_t experiments_threads[num_threads];
    long double total_first_prob = 0.;
    long double total_second_prob = 0.;
    long double total_exp = 0.;

    for (int i = 0; i < num_threads; ++i) {
        int* data_for_calc = malloc(sizeof(int) * 4);
        data_for_calc[0] = k - tour; // Количество бросков
        data_for_calc[1] = experiments / num_threads + (i < experiments % num_threads); // Количество
экспериментов
        data_for_calc[2] = first_score; // Очки первого игрока
        data_for_calc[3] = second_score; // Очки второго игрока

        if (pthread_create(&experiments_threads[i], NULL, calc_score, data_for_calc)) {
            char msg[] = "ERROR: thread cannot be created\n";
            write(STDERR_FILENO, msg, sizeof(msg) - 1);
            return 5;
```

```
        }
    }

    for (int i = 0; i < num_threads; ++i) {
        long double* scores;
        if (pthread_join(experiments_threads[i], (void**)&scores)) {
            char msg[] = "ERROR: thread cannot be joined\n";
            write(STDERR_FILENO, msg, sizeof(msg) - 1);
            return 6;
        }

        // Каждый поток возвращает результаты, мы их суммируем
        total_first_prob += scores[0];
        total_second_prob += scores[1];
        total_exp += scores[2];
        free(scores); // Освобождение памяти после использования
    }
    total_first_prob = total_first_prob / total_exp * 100;
    total_second_prob = total_second_prob / total_exp * 100;

    char num[16];
    float_to_string(total_first_prob, num, 16, 2);

    char msg1[] = "Probability of the first player winnig - ";
    write(STDOUT_FILENO, msg1, sizeof(msg1) - 1);
    write(STDOUT_FILENO, num, sizeof(num) - 1);

    float_to_string(total_second_prob, num, 16, 2);

    char msg2[] = "Probability of the second player winnig - ";
    write(STDOUT_FILENO, msg2, sizeof(msg2) - 1);
    write(STDOUT_FILENO, num, sizeof(num) - 1);
    // Освобождение мьютекса
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

Протокол работы программы

Некорректный ввод:

```
liza@NotebookLizaT:/mnt/c/Users/Лиза/CLionProjects/os/lab2$ ./a.out 10 0 1 1 1000000 -1
ERROR: number of threads must be greater than 1
```

10 раундов, 1 тур, 1 очко у первого, 1 у второго, 10000000 экспериментов, 6 потоков:

```
liza@NotebookLizaT:/mnt/c/Users/Лиза/CLionProjects/os/lab2$ ./a.out 10 0 1 1 1000000 6
Probability of the first player winnig - 50.02%
Probability of the second player winnig - 49.97%
```

Strace:

```
liza@NotebookLizaT:/mnt/c/Users/Лиза/CLionProjects/os/laba2$ strace ./a.out 13 0 2 1 100000 4
execve("./a.out", ["./a.out", "13", "0", "2", "1", "100000", "4"], 0x7fffe3672670 /* 20 vars */) = 0
brk(NULL)                               = 0x7fffe2a64000
arch_prctl(0x3001 /* ARCH_??? */, 0x7fffeb5d8400) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f1312450000
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=16055, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 16055, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f1312455000
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"..., 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\0\226 \25\252\235\23<l\274\3731\3540\5\226\327"..., 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) = 0x7f13121e0000
mprotect(0x7f1312208000, 2023424, PROT_NONE) = 0
```

mmap(0x7f1312208000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f1312208000

mmap(0x7f131239d000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7f131239d000

mmap(0x7f13123f6000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7f13123f6000

mmap(0x7f13123fc000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f13123fc000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f13121d0000

arch_prctl(ARCH_SET_FS, 0x7f13121d0740) = 0

set_tid_address(0x7f13121d0a10) = 553

set_robust_list(0x7f13121d0a20, 24) = 0

rseq(0x7f13121d10e0, 0x20, 0, 0x53053053) = -1 ENOSYS (Function not implemented)

mprotect(0x7f13123f6000, 16384, PROT_READ) = 0

mprotect(0x7f131245d000, 4096, PROT_READ) = 0

mprotect(0x7f1312448000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=8192*1024}) = 0

munmap(0x7f1312455000, 16055) = 0

getrandom("\x4a\xb5\x31\xeb\x90\x67\xc3\x6a", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x7fffe2a64000

brk(0x7fffe2a85000) = 0x7fffe2a85000

rt_sigaction(SIGRT_1, {sa_handler=0x7f1312271870, sa_mask=[], sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x7f1312222520}, 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) = 0x7f13119c0000

mprotect(0x7f13119c1000, 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|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_ti

d=0x7f13121c0910, parent_tid=0x7f13121c0910, exit_signal=0, stack=0x7f13119c0000, stack_size=0x7fff00, tls=0x7f13121c0640}, 88) = -1 ENOSYS (Function not implemented)

clone(child_stack=0x7f13121bfef0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tid=[554], tls=0x7f13121c0640, child_tidptr=0x7f13121c0910) = 554

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f13111b0000

mprotect(0x7f13111b1000, 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|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_ti

d=0x7f13119b0910, parent_tid=0x7f13119b0910, exit_signal=0, stack=0x7f13111b0000, stack_size=0x7fff00, tls=0x7f13119b0640}, 88) = -1 ENOSYS (Function not implemented)

clone(child_stack=0x7f13119afef0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tid=[555], tls=0x7f13119b0640, child_tidptr=0x7f13119b0910) = 555

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f13109a0000

mprotect(0x7f13109a1000, 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_SYS
VSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_ti

d=0x7f13111a0910, parent_tid=0x7f13111a0910, exit_signal=0, stack=0x7f13109a0000,
stack_size=0x7fff00, tls=0x7f13111a0640}, 88) = -1 ENOSYS (Function not implemented)

clone(child_stack=0x7f131119fef0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|C
LONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tid=[556],
tls=0x7f13111a0640, child_tidptr=0x7f13111a0910) = 556

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f1310190000

mprotect(0x7f1310191000, 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_SYS
VSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_ti

d=0x7f1310990910, parent_tid=0x7f1310990910, exit_signal=0, stack=0x7f1310190000,
stack_size=0x7fff00, tls=0x7f1310990640}, 88) = -1 ENOSYS (Function not implemented)

clone(child_stack=0x7f131098fef0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|C
LONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tid=[557],
tls=0x7f1310990640, child_tidptr=0x7f1310990910) = 557

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

futex(0x7f13121c0910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 554, NULL,
FUTEX_BITSET_MATCH_ANY) = 0

futex(0x7f13119b0910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 555, NULL,
FUTEX_BITSET_MATCH_ANY) = 0

futex(0x7f13111a0910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 556, NULL,
FUTEX_BITSET_MATCH_ANY) = 0

futex(0x7f1310990910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 557, NULL,
FUTEX_BITSET_MATCH_ANY) = 0

write(1, "Probability of the first player "..., 41Probability of the first player winnig -) = 41

write(1, "53.38%\n\0\0\0\0\0\0\0\0", 1553.38%

) = 15

write(1, "Probability of the second player"..., 42Probability of the second player winnig -) = 42

write(1, "46.61%\n\0\0\0\0\0\0\0\0", 1546.61%

) = 15

exit_group(0)   = ?

+++ exited with 0 +++

Вывод

Язык Си с поддержкой библиотек позволяет создавать многопоточные приложения, предоставляя инструменты для работы с потоками и механизмы ограничения для обеспечения безопасности. Это делает разработку на Си более разнообразной и увлекательной.