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-210Б-23

Студент: Жданович Е.Т.

Преподаватель: Бахарев В.Д.

Оценка:

Дата: 26.12.24

Москва, 2024

Постановка задачи

Вариант 7.

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы. Два человека играют в кости. Правила игры следующие: каждый игрок делает бросок 2-ух костей K раз; побеждает тот, кто выбросил суммарно большее количество очков. Задача программы экспериментально определить шансы на победу каждого из игроков. На вход программе подается K, 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:

- **`pthread_create`**:
  Создаёт новый поток для выполнения функции.
- **`pthread_join`**:
  Ожидает завершения потока и блокирует выполнение до его завершения.
- **`pthread_mutex_lock` и `pthread_mutex_unlock`**:
  Захватывает и освобождает мьютекс для синхронизации доступа к общим данным между потоками.
- **`rand` и `srand`**:
  Генерируют случайные числа. `srand` используется для инициализации генератора случайных чисел, а `rand` для получения случайных чисел.
- **`scanf` и `printf`**:

  Используются для ввода и вывода данных, взаимодействуя с пользователем.
- **`time`**:
  Получает текущее время, которое может быть использовано, например, для инициализации генератора случайных чисел.

- **`perror`**:
  Выводит сообщение об ошибке, связанной с последней системной ошибкой.

Алгоритм работы программы

1. **Подготовка к работе:**

Пользователь вводит исходные данные: количество бросков кубика на одного игрока; номер текущего раунда; начальные очки игроков (score1 и score2); максимальное количество потоков для выполнения экспериментов (max_threads). Генератор случайных чисел инициализируется текущим временем для обеспечения случайности бросков кубика.

2. **Инициализация потоков:**

Создаётся структура для хранения данных игры (количество бросков, текущий раунд и очки игроков). Объявляется массив потоков, каждый из которых будет выполнять эксперимент.

3. **Эксперимент (выполняется в потоках):**

Для каждого потока подсчитываются очки игроков:
Игрок 1 делает K бросков, результаты суммируются.
Игрок 2 делает K бросков, результаты суммируются.
Общие очки игроков увеличиваются на результаты текущего эксперимента.
Увеличивается счётчик раундов, если игрок 1 набрал больше очков, чем игрок 2.

4. **Ожидание завершения потоков:**

Главный поток ожидает завершения всех созданных потоков, используя.

5. **Вывод результатов:**

После завершения всех потоков программа выводит:
Общие очки игроков.
Имя победителя или сообщение о ничьей.

6. **Завершение программы:**

Все ресурсы освобождаются, программа завершает выполнение.

Расчет ускорения: Ускорение можно рассчитать как отношение времени выполнения программы с одним потоком к времени выполнения программы с n потоками.
Расчет эффективности: Эффективность можно рассчитать как отношение ускорения к количеству потоков.

Число потоков Время выполнения, сек Ускорение Эффективность

1	0.120227	1	1
2	0.057596	2.09	1.05
3	0.054611	2.20	0.73
4	0.053328	2.25	0.56
6	0.051822	2.32	0.39
8	0.048285	2.49	0.31
10	0.044956	2.67	0.27
15	0.039868	3.01	0.20
25	0.034953	3.44	0.14
30	0.037524	3.21	0.11

Код программы

Main.c

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

#define MAX_THREADS 4

typedef struct {
    int K;
    int round;
    int score1;
    int score2;
} GameData;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

int roll_die() {
    return rand() % 6 + 1;
}

void* experiment(void* arg) {
    GameData* gameData = (GameData*)arg;

    int player1_score = 0, player2_score = 0;

    for (int i = 0; i < gameData->K; i++) {
        player1_score += roll_die() + roll_die();
    }
```

```
    for (int i = 0; i < gameData->K; i++) {
        player2_score += roll_die() + roll_die();
    }

    pthread_mutex_lock(&mutex);



    gameData->score1 += player1_score;
    gameData->score2 += player2_score;

    if (player1_score > player2_score) {
        gameData->round++;
    }

    pthread_mutex_unlock(&mutex);

    pthread_exit(NULL);
}

int main() {
    int K, round, score1, score2, max_threads;



    printf("Enter the number of rolls per player (K): ");
    scanf("%d", &K);

    printf("Enter the current round number: ");
    scanf("%d", &round);

    printf("Enter the initial score for player 1: ");
    scanf("%d", &score1);

    printf("Enter the initial score for player 2: ");
    scanf("%d", &score2);

    printf("Enter the maximum number of threads (experiments) to run
simultaneously: ");
    scanf("%d", &max_threads);

    srand(time(NULL));

    pthread_t threads[max_threads];
    GameData gameData = {K, round, score1, score2};
```

```
    for (int i = 0; i < max_threads; i++) {
        if (pthread_create(&threads[i], NULL, experiment, (void*)&gameData) !=
0) {
            perror("Error creating thread");
            return 1;
        }
    }


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

    printf("\nTotal score of player 1: %d\n", gameData.score1);
    printf("Total score of player 2: %d\n", gameData.score2);
    if (gameData.score1 > gameData.score2) {
        printf("Player 1 wins!\n");
    } else if (gameData.score1 < gameData.score2) {
        printf("Player 2 wins!\n");
    } else {
        printf("It's a draw!\n");
    }

    return 0;
}
```

Протокол работы программы

Тестирование:

```
lizka@LizaAlisa:~/ЛАБЫ_ОС/Лаба
2$ ./main Enter the number of
rolls per player (K): 5 Enter
the current round number: 34
Enter the initial score
for player 1: 6 Enter the
initial score for player
2: 8
Enter the maximum number of threads (experiments) to run simultaneously: 400

Total score of player 1: 14020
Total score of player 2: 14128
```

```
Player 2 wins!
lizka@LizaAlisa:~/ЛАБЫ_ОС/Лаба
2$ ./main Enter the number of
rolls per player (K): 23 Enter
the current round number: 56
Enter the initial score
for player 1: 7 Enter the
initial score for player
2: 8
Enter the maximum number of threads (experiments) to run simultaneously: 79

Total score of player 1: 12657
Total score of player 2: 12792
Player 2 wins!
```

Strace:

execve("./main", ["./main"], 0x7ffec8bd4898 /* 28 vars /) = 0 brk(NULL) = 0x5637d4ae7000

brk(NULL) = 0x562d0e9ee000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f3be79bc000

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=20335, ...}) = 0

mmap(NULL, 20335, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f3be79b7000

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"..., 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) = 0x7f3be77a5000

mmap(0x7f3be77cd000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f3be77cd000

mmap(0x7f3be7955000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7f3be7955000

mmap(0x7f3be79a4000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7f3be79a4000

mmap(0x7f3be79aa000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f3be79aa000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f3be77a2000

arch_prctl(ARCH_SET_FS, 0x7f3be77a2740) = 0

set_tid_address(0x7f3be77a2a10) = 3844

set_robust_list(0x7f3be77a2a20, 24) = 0

rseq(0x7f3be77a3060, 0x20, 0, 0x53053053) = 0

mprotect(0x7f3be79a4000, 16384, PROT_READ) = 0

mprotect(0x562ce2969000, 4096, PROT_READ) = 0

mprotect(0x7f3be79f4000, 8192, PROT_READ) = 0

*prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0x7f3be79b7000, 20335) = 0

getrandom("\x01\xae\x69\x0d\x8f\x91\x76\x2f", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x562d0e9ee000 brk(0x562d0ea0f000) = 0x562d0ea0f000

rt_sigaction(SIGRT_1, {sa_handler=0x7f3be783e520, sa_mask=[], sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x7f3be77ea320}, 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) = 0x7f3be6fa1000

mprotect(0x7f3be6fa2000, 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_tid=0x7f3be77a1990, parent_tid=0x7f3be77a1990, exit_signal=0, stack=0x7f3be6fa1000, stack_size=0x7fff80, tls=0x7f3be77a16c0} => {parent_tid=[3845]}, 88) = 3845

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f3be67a0000

mprotect(0x7f3be67a1000, 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_tid=0x7f3be6fa0990, parent_tid=0x7f3be6fa0990, exit_signal=0, stack=0x7f3be67a0000, stack_size=0x7fff80, tls=0x7f3be6fa06c0} => {parent_tid=[3846]}, 88) = 3846

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

fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0

write(1, "Enter the number of rolls per pl"..., 42 Enter the number of rolls per player

(K):) = 42

read(0, "\n", 1024) = 1

read(0, "\n", 1024) = 1

read(0, "\n", 1024) = 1

read(0, "\n", 1024) = 1

read(0, "", 1024) = 0

write(1, "Enter the current round number: "..., 209Enter the current round number:

Enter the initial score for player 1:

Enter the initial score for player 2:

Enter the maximum number of threads (experiments) to run simultaneously:

Total score of player 1: 0) = 209

write(1, "Total score of player 2: 0\n", 27Total score of player 2: 0) = 27

write(1, "It's a draw!\n", 13It's a draw!) = 13

exit_group(0) = ? +++

exited with 0 +++

Вывод

Во время выполнения лабораторной работы я разработал программу, которая использует многопоточность для симуляции игры с подбрасыванием кубиков.
Основная сложность возникла из-за работы с общими данными между потоками. Поскольку несколько потоков одновременно изменяли общие переменные, возникала угроза гонки данных, что могло привести к некорректным результатам. Я решил эту проблему с помощью мьютексов, которые обеспечили безопасный доступ к данным в критических секциях программы.

Кроме того, возникли вопросы, связанные с генерацией случайных чисел в многопоточном контексте. Я использовал стандартный генератор случайных чисел `rand()`, однако в будущем хотелось бы рассмотреть использование более устойчивых и потокобезопасных методов генерации случайных чисел, чтобы избежать неожиданных результатов.

В процессе работы я также столкнулся с необходимостью тщательно контролировать синхронизацию потоков и корректное завершение всех потоков, что потребовало дополнительного внимания к использованию функций `pthread_join`.

В целом, работа была полезной и помогла мне лучше понять основы многопоточности в C, синхронизацию потоков с использованием мьютексов, а также особенности работы с общей памятью в многозадачной среде.