Московский авиационный институт
(Национальный исследовательский университет)

Институт: №8 «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Операционные системы»

Лабораторная работа № 3

Тема: Управление потоками в ОС

Студент: Бирюков В. В.
Группа: М80-207Б-19
Преподаватель: Миронов Е. С.
Дата:
Оценка:

Москва, 2020

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.

Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

В отчете привести исследование зависимости ускорения и эффективности алгоритма от входящих данных и количества потоков. Получившиеся результаты необходимо объяснить.

Вариант 7.

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

Алгоритм решения задачи

Элементарной единицей задачи является один эксперимент, в ходе которого необходимо сгенерировать необходимое количество случайных чисел, имитирующих броски костей и определить победителя. Следовательно, в отдельный поток можно вынести проведение кого-то количества экспериментов.

Количество экспериментов, проводимых в одном потоке, вычисляется как частное целочисленного деления количества экспериментов на число потоков (в последнем потоке может быть меньше экспериментов). Результаты – число выигрышей каждого игрока – при этом можно обрабатывать после завершения потоков, а значит сами потоки можно никак не сихронизировать.

Листинг программы

```
// main.cpp
#include "stdio.h"
#include "stdlib.h"
#include "time.h"
#include "pthread.h"

int score1, score2, rounds;

void* experiment(void *arg) {
     int *ex_count = (int *)arg;
     int sc1;
     int sc2;
     int *win = (int *)malloc(sizeof(int) * 2);
     win[0] = 0; win[1] = 0;
     for (int j = 0; j < *ex_count; j++) {
          sc1 = score1;
          sc2 = score2;
          for (int i = 0; i < rounds; ++i) {
               sc1 += rand() % 6 + 1 + rand() % 6 + 1;
               sc2 += rand() % 6 + 1 + rand() % 6 + 1;
          }
          if (sc1 > sc2) {
               win[0]++;
          } else if (sc2 > sc1) {
               win[1]++;
          }
     }
     free(ex_count);
     pthread_exit((void *)win);
     return NULL;
}

int main(int argc, char const *argv[]) {
     if (argc < 2) {
          printf("expected number of threads\n");
          return 1;
     }
     int th_count = atoi(argv[1]);
     if (th_count <= 0) {
          printf("invalid argument\n");
          return 1;
     }

     srand(time(NULL));

     int k, cur, count;
     scanf("%d %d %d %d %d", &k, &cur, &score1, &score2, &count);
     rounds = k - cur + 1;

     if (count < th_count) {
          th_count = count;
```

```
    }

    pthread_t *id = (pthread_t *)malloc(sizeof(pthread_t) *
                                                    th_count);

    int first = 0, second = 0;
    int *win = NULL;
    int *ex_count = NULL;

    for (int i = 0; i < th_count-1; ++i) {
        ex_count = (int *)malloc(sizeof(int));
        *ex_count = count / th_count;
        if (pthread_create(&id[i], NULL, experiment,
                                (void *)ex_count) != 0) {
            perror("pthread create error\n");
            return 1;
        }
    }
    ex_count = (int *)malloc(sizeof(int));
    *ex_count = count / th_count + count % th_count;
    if (pthread_create(&id[th_count - 1], NULL, experiment,
                                    (void *)ex_count) != 0) {
            perror("pthread create error\n");
            return 1;
    }


    for (int i = 0; i < th_count; ++i) {
        pthread_join(id[i], (void *)&win);
        first += win[0];
        second += win[1];
        free(win);
    }

    printf("%.2lf %.2lf\n", (double)first / count,
                        (double)second / count);

    free(id);
    return 0;
}
```

Тесты и протокол исполнения

```
./main 10
16 1 5 6 100
0.30 0.70

./main 10
16 1 5 6 1000
0.41 0.58
```

Ускорение и эффективность

Тестирование проводилось на 10 раундах и 10^6 экспериментах.

Количество потоков	Время выполнения (ms)	Ускорение	Эффективность
1	721	1.00	1.00
2	415	1.73	0.86
3	345	2.08	0.70
4	287	2.51	0.63
5	239	3.02	0.60
6	210	3.43	0.57
7	198	3.64	0.52
8	180	4.01	0.50
9	199	3.62	0.40
10	181	3.98	0.39
11	187	3.85	0.35
12	193	3.74	0.31

Можно заметить, что после 8 потоков время выполнения перестало уменьшаться, а ускорение – увеличиваться. Вероятно это связано с устройством процессора, который имеет 8 ядер.

Выводы

В ходе выполнения лабораторной работы я познакомился с использованием потоков в ОС Linux, а также со способами их синхронизации. Я изучил способы передачи данных в поток и возвращения их оттуда и выбрал тот, который больше подходит в данном случае. В ходе изучения вывода strace я обнаружил, что процессы и потоки в Linux создаются одним и тем же системным вызовом clone. Также оказалось, что виртуальные операционные системы не всегда могут корректно работать с многопоточной программой, поэтому изучение ускорения проводилось на основной системе. Разделение задачи на несколько потоков – это очень удобный способ её ускорения. Иногда вычисления

выполняющиеся в отдельных потоках даже не требуется синхронизировать, что снижает дополнительные временные расходы.

Список литературы

1. Таненбаум Э., Бос Х. *Современные операционные системы. – 4-е изд. –* СПб.: Издательский дом «Питер», 2018.