

Федеральное агентство связи
Сибирский государственный университет телекоммуникаций и
информатики

Лабораторная работа №10

Выполнил: студент группы ИП-211

Оганесян Альберт

Лацук Андрей

Проверил:

Профессор кафедры ПМиК

Малков Е. А.

Новосибирск 2024

Задание: протестируйте программы лабораторных 8 и 9, используя программную реализацию алгоритма Петерсона, запуская их на одном, двух и нескольких ядрах. Протестируйте модифицированный на основе атомарных функций код алгоритма Петерсона используя различные модели упорядочения выполнения инструкций кода.

Цель: знакомство с атомарными функциями.

1. Добавим в программу из 8 лабораторной алгоритм Петерсона:

```
#include <stdlib.h>

#include <pthread.h>

#include <time.h>

#define ARRAY_SIZE 1000000

#define NUM_THREADS 2

typedef struct
{
    int start;

    int end;

    int *array;

    long long partial_sum;
} ThreadData;

volatile int flag[NUM_THREADS];

volatile int turn;
```

```
long long total_sum = 0;

void *calculate_partial_sum(void *arg)
{
    ThreadData *data = (ThreadData *)arg;

    data->partial_sum = 0;

    for (int i = data->start; i < data->end; i++)
    {
        data->partial_sum += data->array[i];
    }

    int thread_id = data->start / (ARRAY_SIZE / NUM_THREADS);
    flag[thread_id] = 1;
    turn = thread_id;

    for (int i = 0; i < NUM_THREADS; i++)
    {
        if (i == thread_id)
            continue;

        while (flag[i] && turn == thread_id)
            ;
    }

    total_sum += data->partial_sum;
```

```
    flag[thread_id] = 0;

    return NULL;
}

int main()
{
    int array[ARRAY_SIZE];

    pthread_t threads[NUM_THREADS];

    ThreadData thread_data[NUM_THREADS];

    clock_t start_time, end_time;

    for (int i = 0; i < ARRAY_SIZE; i++)
    {
        array[i] = 1;
    }

    for (int i = 0; i < NUM_THREADS; i++)
    {
        flag[i] = 0;
    }

    turn = 0;
```

```
start_time = clock();

int segment_size = ARRAY_SIZE / NUM_THREADS;

for (int i = 0; i < NUM_THREADS; i++)
{
    thread_data[i].start = i * segment_size;

    thread_data[i].end = (i == NUM_THREADS - 1) ? ARRAY_SIZE : (i
+ 1) * segment_size;

    thread_data[i].array = array;

    pthread_create(&threads[i], NULL, calculate_partial_sum,
&thread_data[i]);
}

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

end_time = clock();

printf("Сумма элементов массива: %lld\n", total_sum);

printf("Время выполнения: %.6f секунд\n", (double)(end_time -
start_time) / CLOCKS_PER_SEC);
```

```
return 0;

}
```

2. Запустим программы из 8-й и 9-й лабораторной на разном количестве потоков при помощи команды

```
taskset --cpu-list *потоки* ./lab8
```

3. Заметим, что код лабораторной 8 выдает различные значения при одних и тех же тестах (табл. 3.1). Предположительно, это связано тем, что потоки работают с одним и тем же адресом данных, так как в лабораторной 9, где мы использовали спинлок такого не наблюдается (табл. 3.2).

Номер попытки	8 лабораторная число ядер - 1	8 лабораторная число ядер - 2	8 лабораторная число ядер - неограниченно
1	0.001864	0.000108	0.005218
2	0.001529	0.002205	0.000205
3	0.000974	0.002438	0.007018
4	0.000958	0.000128	0.000161
5	0.001319	0.000087	0.000185
6	0.000958	0.011330	0.003736
7	0.001319	0.003133	0.000161

8	0.001128	0.000159	0.004488
9	0.001322	0.006680	0.000138
10	0.001571	0.001764	0.000185

3.1 Таблица со значениями затрат по времени программы по подсчету суммы массива на 1000000 элементах

Номер попытки	9 лабораторная число ядер - 1	9 лабораторная число ядер - 2	9 лабораторная число ядер - неограниченно
1	102	0	0
2	101	0	0
3	97.5	0	0

3.2 Таблица со значениями затрат по времени на 100 итерациях

4. Дополним алгоритм Петерсона атомарными функциями:

```
#include <stdio.h>

#include <stdlib.h>
```

```
#include <pthread.h>

#include <signal.h>

#include <string.h>

#include <unistd.h>


#define __RELAXED __ATOMIC_RELAXED

#define __ACQUIRE __ATOMIC_ACQUIRE

#define __RELEASE __ATOMIC_RELEASE

#define __SEQ_CST __ATOMIC_SEQ_CST


volatile int running = 1;

char sh[256];

volatile int flag[2] = {0, 0};

volatile int turn = 0;


void *Thread(void *pParams);


void handle_sigint(int sig)

{
    __atomic_store_n(&running, 0, __RELAXED);
}


int main(void)

{
```



```
pthread_t thread_id;

signal(SIGINT, handle_sigint);

pthread_create(&thread_id, NULL, &Thread, NULL);

while (__atomic_load_n(&running, __RELAXED))
{
    __atomic_store_n(&flag[0], 1, __RELAXED);
    __atomic_store_n(&turn, 1, __RELAXED);

    while (__atomic_load_n(&flag[1], __RELAXED) &&
__atomic_load_n(&turn, __RELAXED) == 1)
    {

    }

    printf("%s", sh);

    fflush(stdout);

    __atomic_store_n(&flag[0], 0, __RELAXED);
}

pthread_cancel(thread_id);

pthread_join(thread_id, NULL);

return 0;
```

```

}

void *Thread(void *pParams)
{
    int counter = 0;

    while (__atomic_load_n(&running, __RELAXED))
    {
        __atomic_store_n(&flag[1], 1, __RELAXED);
        __atomic_store_n(&turn, 0, __RELAXED);

        while (__atomic_load_n(&flag[0], __RELAXED) &&
__atomic_load_n(&turn, __RELAXED) == 0)
        {

        }

        if (counter % 2)
        {
            strcpy(sh,
"+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+\n");

        }
        else
        {

```

