

作业3

[姓名:蔡合森] [学号:2022K8009909004]

3.1pthread 函数库可以用来在 Linux 上创建线程，请调研了解 pthread_create, pthread_join, pthread_exit等API的使用方法，然后完成以下任务：

(1) 写一个C程序，首先创建一个值为1到100万的整数数组，然后对这100万个数求和。请打印最终结果，统计求和操作的耗时并打印。（注：可以使用作业 1中用到的gettimeofday和clock_gettime函数测量耗时）；

(2) 在 (1) 所写程序基础上，在创建完 1到100万的整数数组后，使用pthread函数库创建N个线程（N可以自行决定, 且 $N > 1$ ），由这 N个线程完成100万个数的求和，并打印最终结果。请统计N个线程完成求和所消耗的总时间并打印。和 (1) 的耗费时间相比，你能否解释 (2) 的耗时结果？（注意：可以多运行几次看测量结果）

(3) 在 (2) 所写程序基础上，增加绑核操作，将所创建线程和某个CPU核绑定后运行，并打印最终结果，以及统计N个线程完成求和所消耗的总时间并打印。和 (1)、(2) 的耗费时间相比，你能否解释 (3) 的耗时结果？（注意：可以多运行几次看测量结果）（注意：可以多运行几次看测量结果）

解

(1)

```

#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>

#define SIZE 1000000

int main() {
    int *arr = (int *)malloc(SIZE * sizeof(int));
    for (int i = 0; i < SIZE; i++) {
        arr[i] = i + 1;
    }

    struct timeval start, end;
    long long sum = 0;

    gettimeofday(&start, NULL);

    for (int i = 0; i < SIZE; i++) {
        sum += arr[i];
    }
    gettimeofday(&end, NULL);

    long seconds = end.tv_sec - start.tv_sec;
    long microseconds = end.tv_usec - start.tv_usec;
    double elapsed = seconds + microseconds*1e-6;

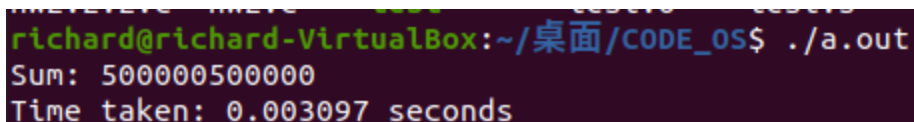
    printf("Sum: %lld\n", sum);
    printf("Time taken: %.6f seconds\n", elapsed);

    free(arr);

    return 0;
}

```

打印结果:



```

richard@richard-VirtualBox:~/桌面/CODE_OS$ ./a.out
Sum: 500000500000
Time taken: 0.003097 seconds

```

(2) 打印结果:

```
richard@richard-VirtualBox:~/桌面/CODE_OS$ ./su
Sum: 500000500000
Time taken with 4 threads: 0.001444 seconds
```

在 (1) 基础上添加主要代码:

```
typedef struct {

    int *array;
    int start;
    int end;
    long long sum;
} ThreadData;

void *sum_array(void *arg) {
    ThreadData *data = (ThreadData *)arg;
    data->sum = 0;
    for (int i = data->start; i < data->end; i++) {
        data->sum += data->array[i];
    }
    pthread_exit(NULL);
}

for (int i = 0; i < NUM_THREADS; i++) {
    thread_data[i].array = arr;
    thread_data[i].start = i * segment_size;
    thread_data[i].end = (i == NUM_THREADS - 1) ? ARRAY_SIZE : (i + 1) * segment_size;
    pthread_create(&threads[i], NULL, sum_array, (void *)&thread_data[i]);
}

for (int i = 0; i < NUM_THREADS; i++) {
    pthread_join(threads[i], NULL);
    total_sum += thread_data[i].sum;
}
```

解释: 单线程只是顺序依次相加数组, 从一到一百万; 多线程将这一百万个数字的相加分成了N=4组来执行, 能够同时进行四组的相加, 减少了耗时。

(3)

打印结果:

```
Time taken with 4 threads (with CPU affinity): 0.002379 seconds
● richard@richard-VirtualBox:~/桌面/CODE_OS$ ./sum_array_multithreaded_affinity
Sum: 500000500000
Time taken with 4 threads (with CPU affinity): 0.001429 seconds
● richard@richard-VirtualBox:~/桌面/CODE_OS$ ./sum_array_multithreaded_affinity
```

仅列出部分重要代码：

```

#define _USE_GNU
#include <sched.h>
#include <pthread.h>

typedef struct {
    int *array;
    int start;
    int end;
    long long sum;
    int cpu_core;
} ThreadData;

//线程执行的函数
void *worker(void *arg){
    cpu_set_t cpuset;    //CPU核的位图
    CPU_ZERO(&cpuset);  //将位图清零
    CPU_SET(N, &cpuset); //设置位图第N位为1，表示与core N绑定。N从0开始计数
    sched_setaffinity(0, sizeof(cpuset), &cpuset); //将当前线程和cpuset位图中指定的核绑定运行

    //其他操作
    data->sum = 0;
    for (int i = data->start; i < data->end; i++) {
        data->sum += data->array[i];
    }
    pthread_exit(NULL);
}

// 创建线程来计算数组部分和，并绑定到特定的CPU核
for (int i = 0; i < NUM_THREADS; i++) {
    thread_data[i].array = arr;
    thread_data[i].start = i * segment_size;
    thread_data[i].end = (i == NUM_THREADS - 1) ? SIZE : (i + 1) * segment_size;
    thread_data[i].cpu_core = i % NUM_THREADS; // 绑定到不同的CPU核
    pthread_create(&threads[i], NULL, sum_array, (void *)&thread_data[i]);
}

// 等待所有线程完成并合并结果
for (int i = 0; i < NUM_THREADS; i++) {
    pthread_join(threads[i], NULL);
    total_sum += thread_data[i].sum;
}

```

解释说明：绑定特定的CPU之后，可以进一步提高，减少线程调度开销，但是多次运行观察到这种优化不稳定，某些情况下，开销时间过长。

3.2 请调研了解pthread_create, pthread_join, pthread_exit等API的使用方法后，完成以下任务：

(1) 写一个C程序，首先创建一个有100万个元素的整数型空数组，然后使用pthread创建N个线程（N可以自行决定, 且 $N > 1$ ），由这N个线程完成前述100万个元素数组的赋值(注意:赋值时第i个元素的值为i)。最后由主进程对该数组的100万个元素求和,并打印结果,验证线程已写入数据。

提交内容：

(1) 所写C程序，打印结果截图,关键代码注释等

解：

```

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/time.h>

#define SIZE 1000000
#define NUM_THREADS 4

typedef struct {
    int *array;
    int start;
    int end;
} ThreadData;

void *assign_values(void *arg) {
    ThreadData *data = (ThreadData *)arg;
    for (int i = data->start; i < data->end; i++) {
        data->array[i] = i;
    }
    pthread_exit(NULL);
}

int main() {
    int *arr = (int *)malloc(SIZE * sizeof(int));
    if (arr == NULL) {
        fprintf(stderr, "内存分配失败\n");
        return 1;
    }

    pthread_t threads[NUM_THREADS];
    ThreadData thread_data[NUM_THREADS];
    int segment_size = SIZE / NUM_THREADS;

    // 创建线程来赋值数组
    for (int i = 0; i < NUM_THREADS; i++) {
        thread_data[i].array = arr;
        thread_data[i].start = i * segment_size;
        thread_data[i].end = (i == NUM_THREADS - 1) ? SIZE : (i + 1) * segment_size;
        pthread_create(&threads[i], NULL, assign_values, (void *)&thread_data[i]);
    }

    // 等待所有线程完成
    for (int i = 0; i < NUM_THREADS; i++) {

```

```
        pthread_join(threads[i], NULL);
    }

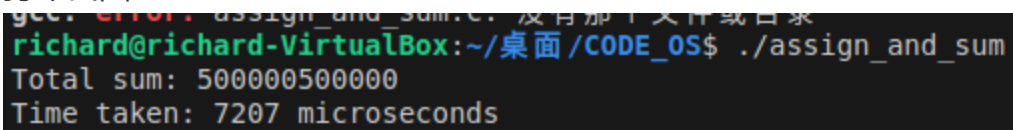
    // 主进程计算数组的和
    long long sum = 0;
    for (int i = 0; i < SIZE; i++) {
        sum += arr[i];
    }

    // 打印结果
    printf("Sum: %lld\n", sum);

    // 释放内存
    free(arr);

    return 0;
}
```

打印截图：



```
gcc: error: assign_and_sum.c: 没有那个文件或目录
richard@richard-VirtualBox:~/桌面/CODE_OS$ ./assign_and_sum
Total sum: 500000500000
Time taken: 7207 microseconds
```