操作系统 作业 3

宋婉婷 2022K8009929009

3.1 数组求和

3.1.1 题目要求

对 1 到 100 0000 求和，分别使用单线程、多线程和多核方式，比较所用时间。

3.1.2 单进程

编写 c 程序如下：

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

#define SIZE 1000000

int main(){
    int *array = malloc(SIZE * sizeof(int));
    for(int i = 0; i < SIZE; i++){
        array[i] = i + 1;
    }
    long sum = 0;
    struct timespec t1 = {0, 0};
    struct timespec t2 = {0, 0};

    clock_gettime(CLOCK_MONOTONIC, &t1);
    for(int i = 0; i < SIZE; i++){
        sum += array[i];
    }
    clock_gettime(CLOCK_MONOTONIC, &t2);

    printf("result is %ld\n", sum);
    long sec = t2.tv_sec - t1.tv_sec;
    long nsec = t2.tv_nsec - t1.tv_nsec;
    if(nsec < 0){
        sec--;
        nsec += 1000000000;
    }
    printf("time = %ld s %ld ns\n", sec, nsec);

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

运行结果示例：

```
# ./pht
result is 500000500000
time = 0 s 1910632 ns
# ./pht
result is 500000500000
time = 0 s 1846374 ns
# ./pht
result is 500000500000
time = 0 s 1928280 ns
```

3.1.3 多线程

在单线程程序上进行修改如下：

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

#define SIZE 1000000
#define NUM_THREADS 2  // 总线程数，可修改为 1000000 的其他约数
int *array;

void* compute_sum(void *arg){
    int my_arg = *(int *)arg;
    long res = 0;
    for(int i = my_arg; i < SIZE; i += NUM_THREADS){
        res += array[i];
    }
    long *result = malloc(sizeof(long));
    *result = res;
    return (void *)result;
}

int main(){
    array = malloc(SIZE * sizeof(int));
    for(int i = 0; i < SIZE; i++){
        array[i] = i + 1;
    }
    long sum = 0;
    struct timespec t1 = {0, 0};
    struct timespec t2 = {0, 0};
    int thread_args[NUM_THREADS];
    void* thread_rets[NUM_THREADS];
    for(int i = 0; i < NUM_THREADS; i++){
        thread_args[i] = i;
    }

    clock_gettime(CLOCK_MONOTONIC, &t1);
    pthread_t threads[NUM_THREADS];
    for(int i = 0; i < NUM_THREADS; i++){
        pthread_create(&threads[i], NULL, compute_sum, &thread_args[i]);
    }
    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], &thread_rets[i]);
        sum += *(long *)thread_rets[i];
    }
    clock_gettime(CLOCK_MONOTONIC, &t2);

    printf("result is %ld\n", sum);
    long sec = t2.tv_sec - t1.tv_sec;
    long nsec = t2.tv_nsec - t1.tv_nsec;
    if(nsec < 0){
        sec--;
        nsec += 1000000000;
    }
    printf("time = %ld s %ld ns\n", sec, nsec);

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

运行结果示例（线程数为 2）：

```
# ./pht2
result is 500000500000
time = 0 s 1435266 ns
# ./pht2
result is 500000500000
time = 0 s 1442225 ns
# ./pht2
result is 500000500000
time = 0 s 1446922 ns
```

3.1.4 双核分配

在多线程程序里增加对 cpu 的分配：

```
#define _GNU_SOURCE
#include <sched.h>
...
void* compute_sum(void *arg){
    int my_arg = *(int *)arg;
    long res = 0;
    cpu_set_t cpuset;   //CPU 核的位图
    CPU_ZERO(&cpuset);  // 将位图清零
    CPU_SET(my_arg, &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
```

因为我的程序运行环境在 Docker 内，可能由于一些版本问题，使用 __USE_GNU 会报错无法链接，在查阅资料后换用了_GNU_SOURCE 成功解决。

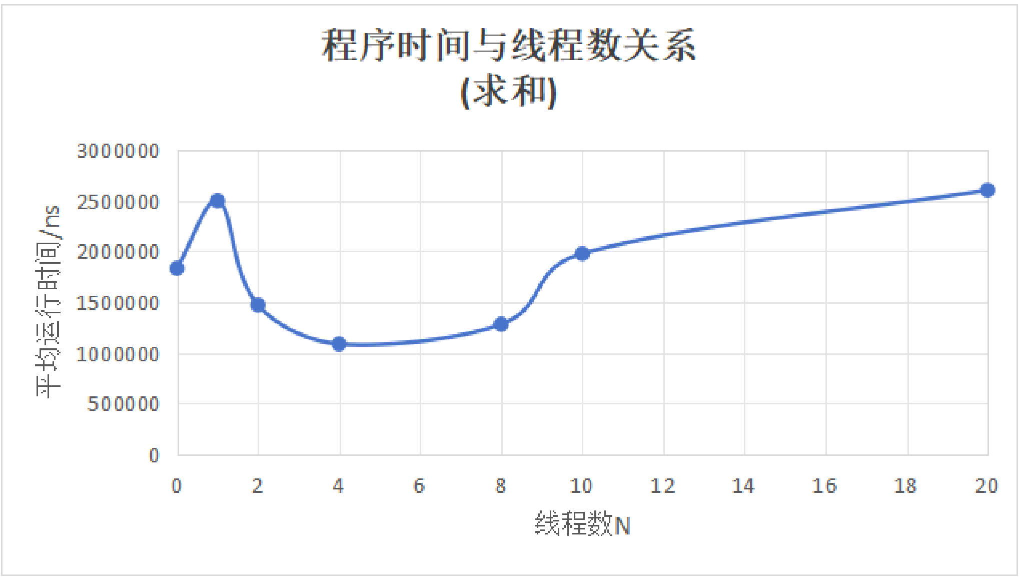
运行结果示例（线程数为 2）：

```
# ./pht2
result is 500000500000
time = 0 s 1331883 ns
# ./pht2
result is 500000500000
time = 0 s 1555563 ns
# ./pht2
result is 500000500000
time = 0 s 1542560 ns
# ./pht2
result is 500000500000
time = 0 s 1381487 ns
```

3.1.5 结果分析

让单线程和多线程程序分别运行 10 次，统计平均时间，得到如下数据：

0(单进程)	1	2	4	8	10	20
1832212.3	2499539.7	1470778.3	1087187.5	1281193.9	1977450.6	2604508.9



从图表中可看出：

1. 多线程的运行时间总体小于单线程（图中的”0“对照），与理论上多线程降低程序开销相符合，只在线程数较大时有不同，猜测此时较多线程管理影响了程序运行。
2. 多线程的的运行时间随线程数增多先减少后增加，并不是简单的 N 越大、时间越短关系，而且理论上应与线程数成反比，但实际结果并不是线性关系，猜想这是因为随线程数增多系统调用的开销也增大。

再分配 cpu core 后运行几次双线程程序，结果如下：

no-core	2 core
1439406	1352398
1320712	1860834
1798972	1378381

no-core	**2 core**
1311911	1325357
1763197	1315670
avg 1526839.6	**avg** 1446528

可以看出将两个线程分开放到两个核上对程序运行有一定提速，但效果并不明显，可能由于 cpu 同时运行着其他程序，或者这个计算程序性能不佳。

3.2 数组最大值

3.2.1 题目要求

创建一个有 1 万个元素的整数型空数组，然后初始化该数组，数组的每一个元素为 [1,100000] 区间的一个随机整数。分别使用以下两种方式找出该数组的最大值。

3.2.2 单进程

编写程序如下：

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

#define SIZE 10000
int *array;
int main(){
    srand(time(NULL));

    int global_max = 0;
    struct timespec t1 = {0, 0};
    struct timespec t2 = {0, 0};
    array = malloc(SIZE * sizeof(int));
    for(int i = 0; i < SIZE; i++){
        array[i] = (rand() % 10000);
    }

    clock_gettime(CLOCK_MONOTONIC, &t1);
    for(int i = 0; i < SIZE; i++){
        if(global_max < array[i]) global_max = array[i];
    }
    clock_gettime(CLOCK_MONOTONIC, &t2);

    printf("max num is %d\n", global_max);
    long sec = t2.tv_sec - t1.tv_sec;
    long nsec = t2.tv_nsec - t1.tv_nsec;
    if(nsec < 0){
        sec--;
        nsec += 1000000000;
    }
    printf("time = %ld s %ld ns\n", sec, nsec);
    // printf("%ld\n",nsec);
    free(array);
    return 0;
}
```

运行结果示例（线程数为 4）：

```
t# ./pht31
max num is 9996
time = 0 s 22111 ns
# ./pht31
max num is 9999
time = 0 s 22613 ns
# ./pht31
max num is 9999
time = 0 s 38245 ns
```

3.2.3 多线程

编写程序如下：

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

#define SIZE 10000
#define NUM_THREADS 4

int *array;
int *results;

void* compute_max(void *arg) {
    int my_arg = *(int *)arg;
    int local_max = 0;
    for(int i = my_arg; i < SIZE; i += NUM_THREADS){
        if(array[i] > local_max) local_max = array[i];
    }
    results[my_arg] = local_max;
    return NULL;
}

int main(){
    srand(time(NULL));

    int global_max = 0;
    struct timespec t1 = {0, 0};
    struct timespec t2 = {0, 0};
    int thread_args[NUM_THREADS];
    array = malloc(SIZE * sizeof(int));
    results = malloc(NUM_THREADS * sizeof(int));
    for(int i = 0; i < SIZE; i++){
        array[i] = (rand() % 10000);
    }

    clock_gettime(CLOCK_MONOTONIC, &t1);
    pthread_t threads[NUM_THREADS];
    for(int i = 0; i < NUM_THREADS; i++){
        thread_args[i] = i;
        pthread_create(&threads[i], NULL, compute_max, &thread_args[i]);
    }
    for(int i = 0; i < NUM_THREADS; i++){
        pthread_join(threads[i], NULL);
    }
    for(int i = 0; i < NUM_THREADS; i++){
        if(global_max < results[i]) global_max = results[i];
    }
    clock_gettime(CLOCK_MONOTONIC, &t2);

    printf("max num is %d\n", global_max);
    long sec = t2.tv_sec - t1.tv_sec;
    long nsec = t2.tv_nsec - t1.tv_nsec;
    if(nsec < 0){
        sec--;
        nsec += 1000000000;
    }
    printf("time = %ld s %ld ns\n", sec, nsec);
    // printf("%ld\n",nsec);

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

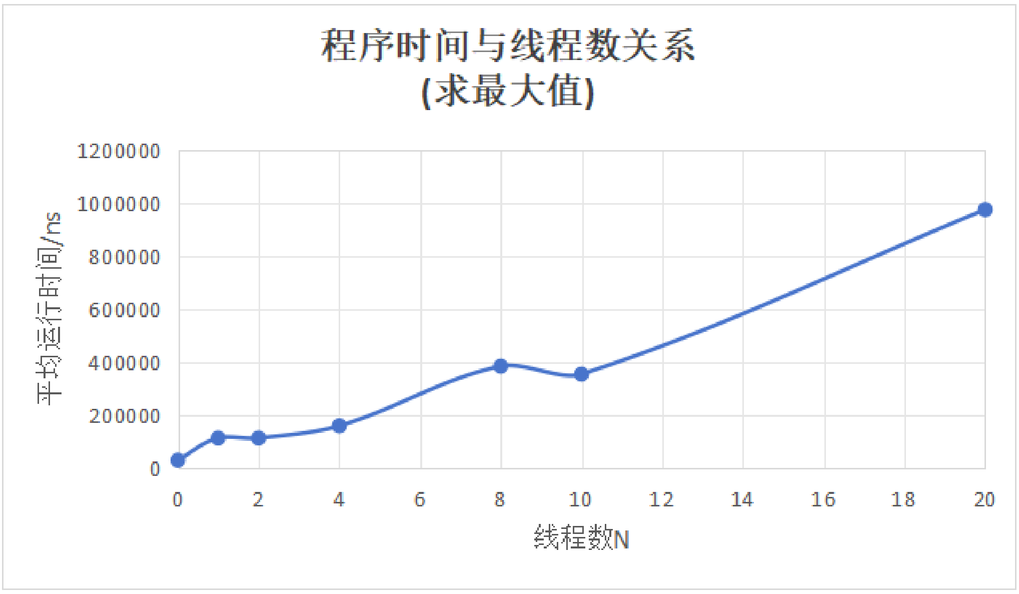
运行结果示例（线程数为 4）：

```
# ./pht3
max num is 9999
time = 0 s 165057 ns
# ./pht3
max num is 9999
time = 0 s 158627 ns
# ./pht3
max num is 9999
time = 0 s 144260 ns
```

3.1.4 结果分析

让单线程和多线程程序分别运行 10 次，统计平均时间，得到如下数据：

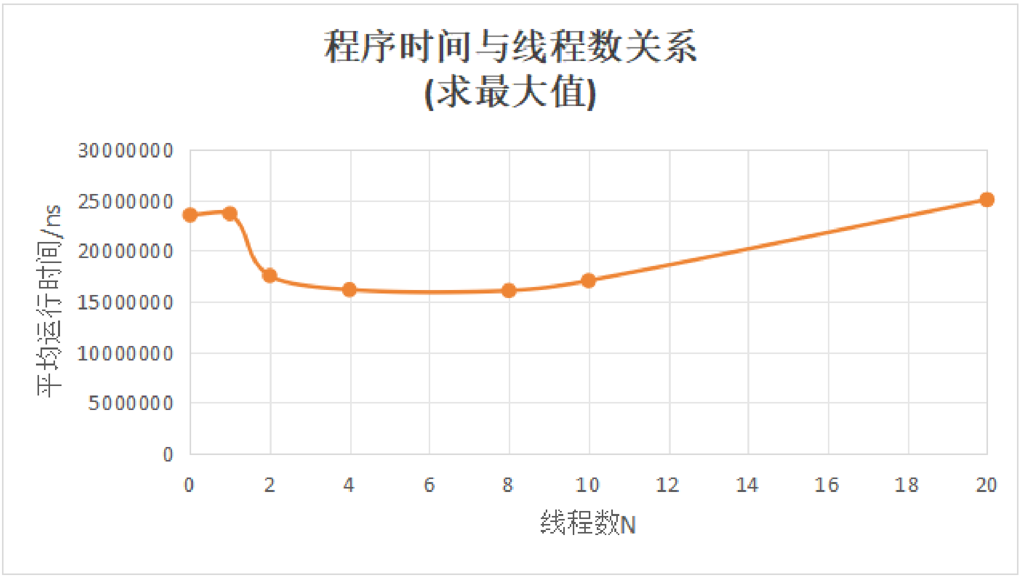
0(单进程)	1	2	4	8	10	20
28391	112301	112341	158476.4	384122.9	354021.9	975898.5



结果非常有意思，与求和时不同，求最大值的单进程程序耗时最短，而多线程程序所用时间大致与线程数量成正比。猜想求最大值时间复杂度较高，系统调用步骤更多，反而不调用线程操作性能更好。

当数组大小从 1 万调整为 1000 万时：

0(单进程)	1	2	4	8	10	20
23534136.2	23667362.4	17537196.9	16146366.7	16058863.1	17044104.3	25051020



可以发现该结果与求和时趋势又差不多一致。所以可以得出此时程序内系统调用占比较少，数据操作较多，因此多线程性能更好。