# 第三次作业

## 3.1

(1) 所写 C 程序如下：

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

int main(){
   int array[1000000];
   for (int i = 0; i < 1000000; i++)
      array[i] = i+1;
   long sum = 0;
   struct timespec start, end;
   clock_gettime(CLOCK_MONOTONIC, &start);
   for (int i = 0; i < 1000000; i++)
      sum += array[i];
   clock_gettime(CLOCK_MONOTONIC, &end);
   printf("Sum: %ld\n", sum);
   printf("Time: %ld ns\n", (end.tv_sec - start.tv_sec) * 1000000000 +
       (end.tv_nsec - start.tv_nsec));
   return 0;
}
```

编译运行后，输出结果如下：

```
zhangjiawei@OS:/Users/zhangjiawei/Desktop/Operating_System/hw3$ cd "/Users/zhangjiawei/Desktop/Operating_System/hw3/" && gcc 3_1_1
.c -o 3_1_1 && "/Users/zhangjiawei/Desktop/Operating_System/hw3/"3_1_1
Sum: 500000500000
Time: 2421223 ns
```

图 1. 3_1_1 运行结果

(2) 所写 C 程序如下：

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#define THREADS 4
long sum[THREADS] = {0};
int array[1000000];
```

```c
typedef struct {
   int *array;
   int id;
} thread_data_t;

void* thread_sum(void* arg){
   thread_data_t *data = (thread_data_t*)arg;
   int *array = data->array;
   int id = data->id;
   long local_sum = 0;
   int start = id * 1000000 / THREADS;
   int end = (id + 1) * 1000000 / THREADS;
   for (int i = start; i < end ; i++)
      local_sum += array[i];
   sum[id] = local_sum;
   return NULL;
}

int main(){
   for (int i = 0; i < 1000000; i++)
      array[i] = i+1;

   struct timespec start, end;
   pthread_t threads[THREADS];
   thread_data_t thread_data[THREADS];
   clock_gettime(CLOCK_MONOTONIC, &start);
   for (int i = 0; i < THREADS; i++){
      thread_data[i].array = array;
      thread_data[i].id = i;
      pthread_create(&threads[i], NULL, thread_sum, &thread_data[i]);
   }
   for (int i = 0; i < THREADS; i++)
      pthread_join(threads[i], NULL);

   long total_sum = 0;
   for (int i = 0; i < THREADS; i++)
      total_sum += sum[i];
   clock_gettime(CLOCK_MONOTONIC, &end);
   printf("Sum: %ld\n", total_sum);
   printf("Time: %ld ns\n", (end.tv_sec - start.tv_sec) * 1000000000 +
       (end.tv_nsec - start.tv_nsec));
   return 0;
```

```
}
```

在这个程序中，我将数组分成了四份，分别由四个线程进行求和，最后将四个线程的结果相加得到最终结果。

编译运行后，输出结果如下：

```
zhangjiawei@OS:/Users/zhangjiawei/Desktop/Operating_System/hw3$ cd "/Users/zhangjiawei/Desktop/Operating_System/hw3/" && gcc 3_1_2
.c -o 3_1_2 && "/Users/zhangjiawei/Desktop/Operating_System/hw3/"3_1_2
Sum: 500000500000
Time: 5498159 ns
zhangjiawei@OS:/Users/zhangjiawei/Desktop/Operating_System/hw3$ cd "/Users/zhangjiawei/Desktop/Operating_System/hw3/" && gcc 3_1_2
.c -o 3_1_2 && "/Users/zhangjiawei/Desktop/Operating_System/hw3/"3_1_2
Sum: 500000500000
Time: 5666289 ns
zhangjiawei@OS:/Users/zhangjiawei/Desktop/Operating_System/hw3$ cd "/Users/zhangjiawei/Desktop/Operating_System/hw3/" && gcc 3_1_2
.c -o 3_1_2 && "/Users/zhangjiawei/Desktop/Operating_System/hw3/"3_1_2
Sum: 500000500000
Time: 5573495 ns
```

图 **2.** 3_1_2 运行结果

可以看出，四线程的运行时间大于直接进行求和的运行时间，这是因为线程的创建和销毁也需要时间，而且线程之间的切换也会带来额外的开销。经过进一步试验，发现随着线程数目从 1 开始逐渐增加，程序运行时间先降低后增加，这是因为我的 CPU 是 8 核的，当线程数目增加到 8 时，线程数目超过了 CPU 的核心数，导致线程切换的开销增加，从而使得程序运行时间增加。

(3) 所写 C 程序如下：

```c
#define _GNU_SOURCE
// There may be some wrong with the homework document, if the macro is
    __USE_GNU, the program will not run. After asking Github copilot, I
    change it to _GNU_SOURCE, and the program runs successfully. Others
    also have the same problem, so I hope the teacher can check out the
    problem.
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <sched.h>
#define THREADS 4
int array[1000000];
long sum[THREADS] = {0};

typedef struct {
   int *array;
   int id;
} thread_data_t;
```

```
void* thread_sum(void* arg){
   thread_data_t *data = (thread_data_t*)arg;
   int *array = data->array;
   int id = data->id;
   long local_sum = 0;
   int start = id * 1000000 / THREADS;
   int end = (id + 1) * 1000000 / THREADS;
   for (int i = start; i < end; i++)
      local_sum += array[i];
   sum[id] = local_sum;
   return NULL;
}

void* worker(void* arg){
   thread_data_t *data = (thread_data_t*)arg;
   int core_id = data->id;
   cpu_set_t cpuset;
   CPU_ZERO(&cpuset);
   CPU_SET(core_id, &cpuset);
   sched_setaffinity(0, sizeof(cpuset), &cpuset);
   thread_sum(arg);
   return NULL;
}

int main(){
   for (int i = 0; i < 1000000; i++)
      array[i] = i+1;

   struct timespec start, end;
   pthread_t threads[THREADS];
   thread_data_t thread_data[THREADS];
   clock_gettime(CLOCK_MONOTONIC, &start);
   for (int i = 0; i < THREADS; i++){
      thread_data[i].array = array;
      thread_data[i].id = i;
      pthread_create(&threads[i], NULL, worker, &thread_data[i]);
   }
   for (int i = 0; i < THREADS; i++)
      pthread_join(threads[i], NULL);

   long total_sum = 0;
   for (int i = 0; i < THREADS; i++)
      total_sum += sum[i];
```

```
        clock_gettime(CLOCK_MONOTONIC, &end);
        printf("Sum: %ld\n", total_sum);
        printf("Time: %ld ns\n", (end.tv_sec - start.tv_sec) * 1000000000 +
            (end.tv_nsec - start.tv_nsec));
        return 0;
    }
```

在这个程序中，我将线程绑定到了特定的 CPU 核心上，这样可以避免线程切换的开销。编译运行后，输出结果如下：

```
zhangjiawei@OS:/Users/zhangjiawei/Desktop/Operating_System/hw3$ cd "/Users/zhangjiawei/Desktop/Operating_System/hw3/" && gcc 3_1_3
.c -o 3_1_3 && "/Users/zhangjiawei/Desktop/Operating_System/hw3/"3_1_3
Sum: 500000500000
Time: 5593189 ns
zhangjiawei@OS:/Users/zhangjiawei/Desktop/Operating_System/hw3$ cd "/Users/zhangjiawei/Desktop/Operating_System/hw3/" && gcc 3_1_3
.c -o 3_1_3 && "/Users/zhangjiawei/Desktop/Operating_System/hw3/"3_1_3
Sum: 500000500000
Time: 5693191 ns
zhangjiawei@OS:/Users/zhangjiawei/Desktop/Operating_System/hw3$ cd "/Users/zhangjiawei/Desktop/Operating_System/hw3/" && gcc 3_1_3
.c -o 3_1_3 && "/Users/zhangjiawei/Desktop/Operating_System/hw3/"3_1_3
Sum: 500000500000
Time: 5620856 ns
```

图 3. 3_1_3 运行结果

运行的结果与上一问相差无几，这可能是由于操作系统在上一问中自动进行了线程绑定，所以运行时间没有太大的变化。

### 3.2

所写 C 程序如下：

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#define THREADS 4 // Number of threads
int array[1000000];
struct thread_data_t
{
    int *array;
    int id;
}; // Structure to pass data to threads

// Function to assign values to array
void* thread_assign(void* arg){
    struct thread_data_t *data = (struct thread_data_t*)arg; // Get data from
        argument
    int *array = data->array; // Get array from data
```

```
    int id = data->id; // Get id from data to calculate start and end index of
        current thread
    int start = id * 1000000 / THREADS; // Calculate start index
    int end = (id + 1) * 1000000 / THREADS; // Calculate end index
    for (int i = start; i < end; i++)
       array[i] = i+1;
    return NULL;
}

int main(){
    struct thread_data_t thread_data[THREADS]; // Array of thread data to pass to
        threads
    pthread_t threads[THREADS]; // Array of threads to store thread ids
    for (int i = 0; i < THREADS; i++){
       thread_data[i].array = array;
       thread_data[i].id = i;
       pthread_create(&threads[i], NULL, thread_assign, &thread_data[i]); //
           Create threads and pass data to them
    }
    for (int i = 0; i < THREADS; i++)
       pthread_join(threads[i], NULL); // Wait for threads to finish

    long sum = 0;
    for (int i = 0; i < 1000000; i++)
       sum += array[i];
    printf("Sum: %ld\n", sum);
    return 0;
}
```

在这个程序中，我将数组分成了四份，分别由四个线程进行赋值，最后将整个数组的值相加得到最终结果。编译运行后，输出结果如下：

```
zhangjiawei@OS:/Users/zhangjiawei/Desktop/Operating_System/hw3$ cd "/Users/zhangjiawei/Desktop/Operating_System/hw3/" && gcc 3_2.c
 -o 3_2 && "/Users/zhangjiawei/Desktop/Operating_System/hw3/"3_2
Sum: 500000500000
```

图 4. 3_2 运行结果