作业 3

3.1 pthread 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）

提示： cpu_set_t 类型，CPU_ZERO、CPU_SET 宏，以及 sched_setaffinity 函数可以用来进行绑核操作，它们的定义在 sched.h 文件中。请调研了解上述绑核操作。以下是一个参考示例。
假设你的电脑有两个核 core 0 和 core1,同时你创建了两个线程 thread1 和 thread2，则可以用以下代码在线程执行的函数中进行绑核操作。
示例代码：

```
//需要引入的头文件和宏定义
#define __USE_GNU
#include <sched.h>
#include <pthread.h>

//线程执行的函数
void *worker(void *arg){
   cpu_set_t cpuset;   //CPU 核的位图
   CPU_ZERO(&cpuset);  //图将位图清零
   CPU_SET(N, &cpuset);  //设置位第 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
}
```

提交内容：
（1） 所写 C 程序，打印结果截图等
（2） 所写 C 程序，打印结果截图，分析说明等
（3） 所写 C 程序，打印结果截图，分析说明等

答：（1）C 程序如下：

hw.c

home > ubuntu > Desktop > C hw.c

```c
#include<time.h>
#include<stdio.h>
#include<stdlib.h>
#define MAXLEN 1000000

struct timespec tv0,tv1;
long nsec;
int main(int argc,char *argv[])
{
   int a[MAXLEN];
   for(int i=0;i<MAXLEN;i++){
      a[i] = i+1;
   }
   clock_gettime(CLOCK_REALTIME,&tv0);
   long sum=0;
   for(int j=0;j<MAXLEN;j++){
      sum+=a[j];
   }
   clock_gettime(CLOCK_REALTIME,&tv1);
   printf("The time elapse of summing is %ld ns\n",tv1.tv_nsec-tv0.tv_nsec);
   printf("The sum of 1-1000000 is %ld\n",sum);
   return 0;
}
```

运行结果如下：

```
The time elapse of summing is 8182606 ns
The sum of 1-1000000 is 500000500000
```

（2）答：C 程序如下：

```c
#include<time.h>
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<sched.h>
#define MAXLEN 1000000
#define Nthread 2

//gcc hw.c -lpthread -o hw

struct timespec tv0,tv1;
long nsec;
int a[MAXLEN];
long sum_array[Nthread];

void* summing(void* arg){
    long num_thread = (long)arg;
    long psum=0;
    for(int i=0;i<MAXLEN;i+=Nthread){
        psum+=a[i];
    }
    sum_array[num_thread]=psum;
}

```

```c
int main(int argc,char *argv[])
{
    pthread_t thread[Nthread];
    long sum;
   for(int i=0;i<MAXLEN;i++){
       a[i] = i+1;
   }
   clock_gettime(CLOCK_REALTIME,&tv0);
   for(int i=0;i<Nthread;i++){
       pthread_create(thread+i,NULL,summing,(void*)(long)i);
   }
   for(int i=0;i<Nthread;i++){
       pthread_join(thread[i],NULL);
   }
   for(int i=0;i<Nthread;i++){
       sum+=sum_array[i];
   }
   clock_gettime(CLOCK_REALTIME,&tv1);
   printf("The time elapse of summing is %ld ns\n",tv1.tv_nsec-tv0.tv_nsec);
   printf("The sum of 1-1000000 is %ld\n",sum);
   return 0;
}
```

运行 3 次结果如下：

```
ubuntu@ubuntu:~/Desktop$ ./hw
The time elapse of summing is 7335438 ns
The sum of 1-1000000 is 500000000000
ubuntu@ubuntu:~/Desktop$ ./hw
The time elapse of summing is 7659594 ns
The sum of 1-1000000 is 500000000000
ubuntu@ubuntu:~/Desktop$ ./hw
The time elapse of summing is 7214854 ns
The sum of 1-1000000 is 500000000000
```

说明：我们可以看到开 2 线程之后运行时间有缩短，但是加速比并不高（不到 20%），理论上双线程应当有较高加速比，猜测应当是与线程之间上下文切换、CPU 核资源调度所耗费资源太多有关。

（3）**答：C 程序如下：**

```c
#define __USE_GNU
#define _GNU_SOURCE
#include<time.h>
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<sched.h>
#define MAXLEN 1000000
#define Nthread 2

//gcc hw.c -lpthread -o hw

struct timespec tv0,tv1;
long nsec;
int a[MAXLEN];
long sum_array[Nthread];

void* summing(void* arg){
    long num_thread = (long)arg;
    long psum=0;

    cpu_set_t cpuset;   //CPU核的位图
    CPU_ZERO(&cpuset);  //图将位图清零
    CPU_SET(num_thread, &cpuset);  //设置位第num_thread位为1，表示与core num_thread绑定。num_thread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

    for(int i=0;i<MAXLEN;i+=Nthread){
        psum+=a[i];
    }
    sum_array[num_thread]=psum;
}

int main(int argc,char *argv[])
{
    pthread_t thread[Nthread];
    long sum;
    for(int i=0;i<MAXLEN;i++){
        a[i] = i+1;
    }
    clock_gettime(CLOCK_REALTIME,&tv0);
    for(int i=0;i<Nthread;i++){
        pthread_create(thread+i,NULL,summing,(void*)(long)i);
    }
    for(int i=0;i<Nthread;i++){
        pthread_join(thread[i],NULL);
    }
    for(int i=0;i<Nthread;i++){
        sum+=sum_array[i];
    }
    clock_gettime(CLOCK_REALTIME,&tv1);
    printf("The time elapse of summing is %ld ns\n",tv1.tv_nsec-tv0.tv_nsec);
    printf("The sum of 1-1000000 is %ld\n",sum);
    return 0;
}
```

运行结果如下：

```
ubuntu@ubuntu:~/Desktop$ gcc hw.c -lpthread -o hw
ubuntu@ubuntu:~/Desktop$ ./hw
The time elapse of summing is 1112751 ns
The sum of 1-1000000 is 500000000000
ubuntu@ubuntu:~/Desktop$ ./hw
The time elapse of summing is 1054252 ns
The sum of 1-1000000 is 500000000000
ubuntu@ubuntu:~/Desktop$ ./hw
The time elapse of summing is 1091356 ns
The sum of 1-1000000 is 500000000000
ubuntu@ubuntu:~/Desktop$ ./hw
The time elapse of summing is 1179210 ns
The sum of 1-1000000 is 500000000000
```

分析：相比单纯开双线程而言，运行时间显著缩短，缩短为 1/7，性能加速比极高！至于为什么会出现这样的结果，推断是因为绑核让线程单独占用 CPU 资源，硬件资源利用率提高，至于单独占用 CPU 资源竟能获得如此之高的加速比，也侧面印证了（2）中性能受限很大程度是源于 CPU 资源调度和分配。

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

答：C 程序如下：

```c
#include<time.h>
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#define MAXLEN 1000000
#define Nthread 2 //num of threads

//gcc hw.c -lpthread -o hw

int a[MAXLEN];
long sum_array[Nthread];

void* value(void* arg){//value the array a[MAXLEN]
    long num_thread = (long)arg;

    for(int i=num_thread;i<MAXLEN;i+=Nthread){//step is Nthread
        a[i]=i+1;
    }
}

```

```c

int main(int argc,char *argv[])
{
    pthread_t thread[Nthread];
    long sum;
   for(int i=0;i<Nthread;i++){
       pthread_create(thread+i,NULL,value,(void*)(long)i);//void* is 8 bytes, so must be long instead of int!!!  otherwise error!!
   }
   for(int i=0;i<Nthread;i++){
       pthread_join(thread[i],NULL);
   }
   for(int i=0;i<MAXLEN;i++){
       sum+=a[i];
   }
   printf("The sum of 1-1000000 is %ld\n",sum);
   return 0;
}
```

打印结果截图：

```
ubuntu@ubuntu:~/Desktop$ gcc hw.c -lpthread -o hw
ubuntu@ubuntu:~/Desktop$ ./hw
The sum of 1-1000000 is 500000500000
ubuntu@ubuntu:~/Desktop$
```

验证线程成功写入数据！