中山大学计算机学院本科生实验报告

课程名称：超级计算机原理与操作　　任课教师：吴迪&黄聃

年级	2019 级	专业（方向）	计算机科学与技术
学号	18329015	姓名	郝裕玮
开始日期	2021.4.10	完成日期	2021.4.11

一、实验题目

使用 pthread 中的 semaphore 计算 π 的值

二、实验内容

①补充部分代码（即 Thread_sum 函数）如下所示（代码分析已全部包含在注释中）：

```
/*-------------------------------------------------------------------*/
void* Thread_sum(void* rank) {
   long my_rank = (long) rank;
   //如果指针类型的大小和表示进程编号的整数类型不同，在编译时就会受到警告
   //在我们使用的机器上，指针类型 64 位，int 类型 32 位
   //为了避免警告，所以我们用 long 型替换 int 型
   double my_sum = 0.0;//每个线程内部的和（最终要汇总到全局变量 sum）
   /*****************************************************************/
   double flag;//用于确定公式中每项的正负
   long long i;
   long long group_num=n/thread_count;//计算每个线程中需要分配的公式项数（尽可能平均）
   long long group_first_i=group_num*my_rank;//计算每个线程中开始累加的第一项
   long long group_last_i=group_first_i+group_num;//计算每个线程中需要累加的最后一项
   if(group_first_i%2==0){
      flag=1.0;
   }
   else{
      flag=-1.0;
   }
   for(i=group_first_i;i<=group_last_i-1;i++){//每个线程内部开始累加
      my_sum+=flag/(2*i+1);//1/4pi 的计算公式
      flag*=-1;//改变下一项的正负
   }
   sem_wait(&sem);//阻塞线程，等待获取信号量
   sum+=my_sum;//获取信号量后可将当前线程的和累加进入全局变量 sum
   sem_post(&sem);//释放信号量
```

```
    /*****************************************************************/
    return NULL;
}  /* Thread_sum */
```

接下来是对其余已完成函数的解析：

②宏定义（代码分析已全部包含在注释中）

```
#define GET_TIME(now) { \
   struct timeval t; \
   gettimeofday(&t, NULL); \
   now = t.tv_sec + t.tv_usec/1000000.0; \
}
//最后一行将 us 转换为 s，统一单位
//该结构体用于计算多线程和串行计算 pi 的运行时间
```

③全局变量（代码分析已全部包含在注释中）

```
const int MAX_THREADS = 1024;//最大线程数

long thread_count;//线程数量
long long n;//公式项数
double sum;//公式最终的总和（多线程计算 pi 的估计量）

sem_t sem;//信号量
```

④主函数（代码分析已全部包含在注释中）

```
int main(int argc, char* argv[]) {
   long       thread;  /* 在 64 位系统中使用 long */
   pthread_t* thread_handles;
   //pthread_t 用于声明线程 ID
   double start, finish, elapsed;

   /* 可选择在这里确定公式项数和线程数量 */
   n = 10000;
   thread_count = 4;

   /* You can also get number of threads from command line */
   //Get_args(argc, argv);

   thread_handles = (pthread_t*) malloc (thread_count*sizeof(pthread_t));//创建线程
数组
   sem_init(&sem, 0, 1);//初始化信号量
   sum = 0.0;//最终的总和初始化为 0

   GET_TIME(start);//得到多线程运行的开始时间
   for (thread = 0; thread < thread_count; thread++)
      pthread_create(&thread_handles[thread], NULL,Thread_sum, (void*)thread);//创
建线程
```

```
      //第一个参数为指向线程标识符的指针。
      //第二个参数用来设置线程属性。
      //第三个参数是线程运行函数的起始地址。
      //最后一个参数是运行函数的参数。

   for (thread = 0; thread < thread_count; thread++)
      pthread_join(thread_handles[thread], NULL);
      //函数 pthread_join 用来等待一个线程的结束。
      //第一个参数为被等待的线程标识符，第二个参数为一个用户定义的指针，它可以用来存储被等待线程的返回值。
      //这个函数是一个线程阻塞的函数，调用它的线程将一直等待到被等待的线程结束为止
      //当函数返回时，被等待线程的资源被收回。
      //也就是说主线程中要是加了这段代码，就会在该代码所处的位置卡住，直到这个线程执行完毕才会继续往下运行。
   GET_TIME(finish);//得到多线程运行的结束时间
   elapsed = finish - start;//计算多线程运行时间

   sum = 4.0*sum;//1/4*pi*4=pi（公式计算的是 1/4*pi）
   printf("With n = %lld terms,\n", n);//公式项数
   printf("   Our estimate of pi = %.15f\n", sum);//多线程估计的 pi 值
   printf("The elapsed time is %e seconds\n", elapsed);//多线程运行时间
   GET_TIME(start);//得到串行计算的开始时间
   sum = Serial_pi(n);//串行计算 pi，也就是单线程
   GET_TIME(finish);//得到串行计算的结束时间
   elapsed = finish - start;//计算串行计算的运行时间
   printf("   Single thread est  = %.15f\n", sum);//串行计算估计的 pi 值
   printf("The elapsed time is %e seconds\n", elapsed);//串行计算运行时间
   printf("                     pi = %.15f\n", 4.0*atan(1.0));//arctan 计算 pi 值

   sem_destroy(&sem);//释放信号量
   free(thread_handles);//释放线程数组
   return 0;
}  /* main */
```

⑤串行计算 pi 值（代码分析已全部包含在注释中）

```
/*--------------------------------------------------------------------
 * Function:   Serial_pi
 * Purpose:    Estimate pi using 1 thread
 * In arg:     n
 * Return val: Estimate of pi using n terms of Maclaurin series
 */
double Serial_pi(long long n) {
   double sum = 0.0;
   long long i;
   double factor = 1.0;
```

```
    for (i = 0; i < n; i++, factor = -factor) {
        sum += factor/(2*i+1);//按照公式循环累加每一项
    }
    return 4.0*sum;//乘 4 后即可得到 pi 值

}  /* Serial_pi */
```

⑥Get_args 和 Usage 为命令行编译程序并输入公式项数和线程数，与本题关系不大，不再赘述（因为主函数中已有可以直接修改这两项值的语句）

```
    /* 可选择在这里确定公式项数和线程数量 */
    n = 10000;
    thread_count = 4;
```

三、实验结果

①设置项数分别为 1000，10000，100000，线程数不变，均为 4：

```
======== ERROR ========

======== OUTPUT ========
With n = 1000 terms,
Our estimate of pi = 3.140592653839791
The elapsed time is 3.230572e-04 seconds
Single thread est = 3.140592653839794
The elapsed time is 5.960464e-06 seconds
pi = 3.141592653589793

======== REPORT ========
```

```
======== ERROR ========

======== OUTPUT ========
With n = 10000 terms,
Our estimate of pi = 3.141492653590044
The elapsed time is 3.361702e-04 seconds
Single thread est = 3.141492653590034
The elapsed time is 6.389618e-05 seconds
pi = 3.141592653589793

======== REPORT ========
```

```
======== ERROR ========

======== OUTPUT ========
With n = 100000 terms,
Our estimate of pi = 3.141582653589787
The elapsed time is 4.639626e-04 seconds
Single thread est = 3.141582653589720
The elapsed time is 6.802082e-04 seconds
pi = 3.141592653589793

======== REPORT ========
```

由上述结果可知，项数增加，多线程和串行计算的运行时间均增加，且多线程运行时间的增长速度比串行慢，所以最后多线程运行时间会从比串行慢到比串行快。

且随着项数的增加，多线程和串行计算的估计值也越来越接近 pi 的真实值（精度越来越高）。

② 设置线程数分别为 4，8，16，项数不变，均为 10000：

```
======== ERROR ========

======== OUTPUT ========
With n = 10000 terms,
Our estimate of pi = 3.141492653590044
The elapsed time is 3.499985e-04 seconds
Single thread est = 3.141492653590034
The elapsed time is 7.200241e-05 seconds
pi = 3.141592653589793

======== REPORT ========
```

```
======== ERROR ========

======== OUTPUT ========
With n = 10000 terms,
Our estimate of pi = 3.141492653590043
The elapsed time is 6.480217e-04 seconds
Single thread est = 3.141492653590034
The elapsed time is 6.413460e-05 seconds
pi = 3.141592653589793

======== REPORT ========
```

```
======== ERROR ========

======== OUTPUT ========
With n = 10000 terms,
Our estimate of pi = 3.141492653590046
The elapsed time is 1.362085e-03 seconds
Single thread est = 3.141492653590034
The elapsed time is 6.413460e-05 seconds
pi = 3.141592653589793

======== REPORT ========
```

由上述结果可知，线程数增加，多线程的运行时间增加，串行计算的运行时间基本不变。同时我们也发现线程数的增加对多线程计算的 pi 估计值并无影响。