**多个数组排序任务的不均衡案例Pthread**

|  |
| --- |
| C++ //线程参数（线程序号）结构体 typedef struct {  int threadId; } threadParm\_t;  const int ARR\_NUM = 10000; const int ARR\_LEN = 10000; const int THREAD\_NUM = 4; const int seg = ARR\_NUM / THREAD\_NUM; vector<int> arr[ARR\_NUM]; pthread\_mutex\_t mutex; long long head, freq; // timers  void init(void) {  srand(unsigned(time(NULL)));  for (int i = 0; i < ARR\_NUM; i++) {  arr[i].resize(ARR\_LEN);  for (int j = 0; j < ARR\_LEN; j++)  arr[i][j] = rand();  } } //静态任务划分 void \*arr\_sort(void \*parm) {  //定义线程参数  threadParm\_t \*p = (threadParm\_t \*)parm;  int r = p->threadId;  long long tail;   for (int i = r \* seg; i < (r + 1) \* seg; i++)  sort(arr[i].begin(), arr[i].end());  //加互斥锁  pthread\_mutex\_lock(&mutex);  QueryPerformanceCounter((LARGE\_INTEGER \*)&tail);  printf("Thread %d: %lfms.\n", r, (tail - head) \* 1000.0 / freq);  //解锁  pthread\_mutex\_unlock(&mutex);  pthread\_exit(NULL); }  int main(int argc, char \*argv[]) {  QueryPerformanceFrequency((LARGE\_INTEGER \*)&freq);  init();  mutex = PTHREAD\_MUTEX\_INITIALIZER;  pthread\_t thread[THREAD\_NUM];  threadParm\_t threadParm[THREAD\_NUM];  QueryPerformanceCounter((LARGE\_INTEGER \*)&head);   for (int i = 0; i < THREAD\_NUM; i++) {  threadParm[i].threadId = i;  //创建线程  pthread\_create(&thread[i], NULL, arr\_sort, (void \*)&threadParm[i]);  }   for (int i = 0; i < THREAD\_NUM; i++) {  //等待所有线程结束  pthread\_join(thread[i], NULL);  }  //释放互斥锁  pthread\_mutex\_destroy(&mutex);   return 0; } |

|  |
| --- |
| C++ //动态任务划分：每个线程一开始只分配一部分，之后哪个线程先完成，哪个线程继续取任务。 int next\_arr = 0; pthread\_mutex\_t mutex\_task;  void \*arr\_sort\_fine(void \*parm) {  threadParm\_t \*p = (threadParm\_t \*)parm;  int r = p->threadId;  int task = 0;  long long tail;   while (1) {  pthread\_mutex\_lock(&mutex\_task);  task = next\_arr++;  pthread\_mutex\_unlock(&mutex\_task);   if (task >= ARR\_NUM)   break;   stable\_sort(arr[task].begin(), arr[task].end());  }   pthread\_mutex\_lock(&mutex);  QueryPerformanceCounter((LARGE\_INTEGER \*)&tail);  printf("Thread %d: %lfms.\n", r, (tail - head) \* 1000.0 / freq);  pthread\_mutex\_unlock(&mutex);   pthread\_exit(NULL); } //可以缓解负载不均衡问题 |