# به نا م خدا

الگوریتم های تقسیم و غلبه بیشتر در مرتب سازی آرایه ها استفاده می شوند. الگوریتم هایی شبیه مرتب سازی سریع، مرتب سازی ادغامی و ....

در این پروژه الگوریتم مرتب سازی ادغامی با قابلیت multithreading در C#.NET پیاده سازی شده است.

پروژه از یک کلاس به نام Sorter تشکیل شده است که در این کلاس تابعی به نام ParallelMergeSort وجود دارد. این تابع کار مرتب سازی را به صورت multithreading انجام می دهد. در فایل Program.cs، تابع Main نقطه ی شروع برنامه است که با ساختن یک آرایه بزرگ از اعداد تصادفی و صدا زدن تابع ParallelMergeSort بر روی آن آرایه کار را انجام می دهد.

فایل exe و قابل اجرای پروژه در مسیر

MultithreadedDivideAndConquer\bin\Debug\MultithreadedDivideAndConquer.exe وجود دارد.

در ادامه به توضیح برنامه می پردازیم.

```
1   static void Main(string[] args)
2           {
3               var unsorted = new Queue<Queue<int>>();
4               long listLength = 100000;
5              int maxSize = 1000;
6               Random rand = new Random();
7               Console.WriteLine("Creating list of size " + listLength );
8               for (int i = 0; i < listLength; ++i)
9               {
10                  var l = new Queue<int>();
11                  l.Enqueue(rand.Next(maxSize));
12                  unsorted.Enqueue(l);
13              }
14              Stopwatch parallelSW = new Stopwatch();
15              Console.WriteLine("Starting Parallel MergeSort");
16                  var pending = new Queue<Queue<int>>();
17                  foreach (Queue<int> q in unsorted)
18                  {
19                      pending.Enqueue(new Queue<int>(q));
20                  }
21                  parallelSW.Start();
22                  Queue<int> sortedParallel = Sorter.ParallelMergeSort(pending);
23                  parallelSW.Stop();
24                  Console.WriteLine("ParallelMergeSort -
25   Time Elapsed: " + (parallelSW.ElapsedMilliseconds) + "ms");
26          foreach ( int item in sortedParallel )
27                  {
28                      Console.WriteLine ( item );
29                  }
30              Console.WriteLine();
31              Console.Write("Press any key to exit...");
32              Console.Read();
33          }
```

توضیح خط به خط کد:

1. خط شروع تابع Main که نقطه ی ورودی برنامه است. یعنی برنامه از اینجا شروع می شود.
3. تعریف یک صف به نام unsorted برای نگهداری اعدادی که نیاز به مرتب سازی دارند. عناصر این صف هر کدام خود نیز یک صف هستند: Generic Queue
4. تعریف یک متغیر برای نگه داشتن طول لیست مورد نظر برای مرتب سازی به نام listLenght
5. تعریف یک متغیر به نام maxSize برای مشخص کردن بازه ی اعداد تصادفی مورد نظر برای مربت سازی
6. تعریف یک شی به نام rand از کلاس random که وظیفه ی تولید اعداد تصادفی برای مرتب سازی را دارد.
7. چاپ یک رشته در خروجی برای اطلاع دادن به کاربر از شروع کار. یک رشته ی ثابت به اضافه ی مقدار متغیر listLenght
8. از خط 8 تا 13 یک حلقه ی تکرار وجود دارد که به اندازه ی متغیر listLenght که در بالا تعریف شده است تکرار می شود. این حلقه در هر بار تکرار، یک صف به نام I که عناصرش از نوع int هستند را می سازد (Generic Queue)، یک عدد تصادفی با متد Next مربوط به شی rand تولید می شود و سپس این عدد با متد Enqueue در ابتدای صف تولید شده قرار میدهد و در پایان از آن خود صف وارد شی unsorted می شود.
14. تعریف یک شی از نوع Stopwatch به نام parallelSW. این شی برای اندازه گیری زمان صرف شده است برای مرتب سازی است که بعنوان خروجی نمایش داده خواهد شد.
15. چاپ یک متن در خروجی
16. تعریف یک صف با نام pending با همان ساختار شی unsorted. از این صف برای اضافه کردن عناصر شی unsorted به آن و مرتب کردن آنها استفاده می کنیم.
17. خط 17 تا 20 یک حلقه وجود دارد. این حلقه تک تک عناصر موجود در صف unsorted را گرفته و وارد صف pending می کند. این کار با متد Enqueue مربوط به صف pending اضافه می شود. چون ما می خواهیم صف unsorted دست نخورده باقی بماند، عناصر آن را به یک صف دیگر اضافه می کنیم و سپس صف ساخته شده را به تابع مرتب سازی ارسال می نماییم.
21. Start کردن شی parallelSW که در نقش یک زمان سنج عمل می کند و در واقع برای اندازه گیری زمان مصرف شده ی کار، استفاده می شود.
22. صدا زدن تابع ParallelMergeSort در کلاس Sorter بر روی صف pending و ریختن نتیجه در شی sortedParallel که یک صف از نوع اعداد صحیح است. پس در واقع نتیجه ی مرتب سازی یک صف که هر عنصر آن خود یک صف هستند(صف pending)، در یک صف که عناصر آن اعداد صحیح هستند(صف sortedParallel) ذخیره شد و این کار تابع ParallelMergeSort است که در ادامه توضیح داده خواهد شد.

23. Stop کردن شی parallelSW که همان زمان سنج است.
24. خط 24 و 25 چاپ ی زمان صرف شده با کمک خصوصیت ElapsedMilliseconds مربوط به شی parallelSW که بصورت میلی ثانیه برگردانده می شود.
26. از خط 26 تا 29 برای چاپ محتویات صف sortedParallel استفاده می شود. این صف حاوی اعداد مرتب شده می باشد.
30. از خط 30 تا 32 به ترتیب برای چاپ خط خالی، چاپ یک عبارت، و در پایان منتظر ماندن برای فشرده شدن کلیدی از سوی کاربر استفاده می شود.

```
1           public static Queue<int> ParallelMergeSort(Queue<Queue<int>> pending)
2           {
3               var working = 0;
4               var tasks = Enumerable.Range(0, Environment.ProcessorCount).Select(_ =>
5                   Task.Factory.StartNew(() =>
6                   {
7                       Queue<int> l1, l2, l3;
8                       while (pending.Count >= 2)
9                       {
10                          lock (pending)
11                          {
12                              Interlocked.Increment(ref working);
13                              l1 = pending.Dequeue();
14                              l2 = pending.Dequeue();
15                          }
16                          l3 = new Queue<int>();
17                          while (l1.Count > 0 || l2.Count > 0){
18                              if (l1.Count > 0 && l2.Count > 0){
19                                  if (l1.First() <= l2.First()){
20                                      l3.Enqueue(l1.Dequeue());
21                                  }else{
22                                      l3.Enqueue(l2.Dequeue());
23                                  }
24                              }else if(l1.Count > 0){
25                                  while (l1.Count > 0){
26                                      l3.Enqueue(l1.Dequeue());
27                                  }
28                              }else if(l2.Count > 0){
29                                  while (l2.Count > 0){
30                                      l3.Enqueue(l2.Dequeue());
31                                  }
32                              }
33                          }
34                          lock (pending)
35                          {
36                              pending.Enqueue(l3);
37                              Interlocked.Decrement(ref working);
38                          }
39                          while (Thread.VolatileRead(ref working) > 0 && !(pending.Coun
40  t >= 2))
41                          {
42                              Thread.Sleep(5);
43                          }
44                      }
45
46                  }, TaskCreationOptions.LongRunning)
47              );
48              Task.WaitAll(tasks.ToArray());
49              return pending.Dequeue();
50          }
```

توضیح خط به خط کد

1. این تابع یک صف به نام pending که هر یک از عناصر آن خود یک صف از نوع اعداد صحیح را به عنوان ورودی گرفته و پس از انجام پردازشی روی آن، یک صف واحد از نوع اعداد صحیح بر می گرداند.( تعریف تابع این را نشان می دهد)
3. تعریف یک شی به نام working و ریختن مقدار صفر در آن. این متغیر تعداد thread های در حال کار را نگه می دارد.
4. ساخت شی tasks. این شی در واقع یک لیست از thread ها است. با استفاده از متد Range مربوط به کلاس Enumerable و دادن مقدار صفر به عنوان آرگومان اول و تعداد پردازنده ها (Environment.ProcessorCount) به عنوان آرگومان دوم کار را آغاز می کنیم.
Enumerable.Range در واقع یک لیست از اعداد را با توجه به محدوده ای که به عنوان دو آرگومان ورودی (که در اینجا صفر و تعداد پردازنده های سیستم است) می گیرد، می سازد. با استفاده از دستور Select، تعریف هر یک از این عناصر را مشخص می کنیم. به این معنا عناصرلیست tasks چه هستند. در جلوی دستور select از => برای این کار استفاده شده است. از => برای تعریف یک lambada expression استفاده می شود که در واقع همان تعریف یک تابع بصورت inline است.
5. کار ساخت thread ها و مرتب سازی از خط 5 تا 47 انجام می گیرد که در ادامه تک تک خطوط توضیح داده خواهند شد. در خط 5 یک task جدید با استفاده از Task.Factory.StartNew ساخته می شود. این متد دو آرگومان می گیرد. اولین آرگومان در واقع همان تابع یا خطوط کدی است که این thread باید انجام دهد. برای این کار یک بار دیگر از Lambada Expression به منظور تعریف یک تابع بصورت inline استفاده شده است. این کار را با => انجام داده ایم. جفت پرانتز باز و بسته قبل از => به این معناست که این تابعی که قصد تعریف آن را داریم هیچ پارامتر ورودی دریافت نمی کند. اگر تابع مورد نظر ورودی داشته باشد، باید آنها را در این جفت پرانتز تعریف کنیم.
7. تعریف سه شی به نام های l1، l2، l3 که همگی از نوع صفی هستند که عناصر آن از اعداد صحیح است. (Generic Queue)
8. تعریف یک حلقه while که تا هنگامی که تعداد عناصر موجود در شی pending (پارامتر پاس داده شده به این تابع) بزرگتر از 2 است کدهای موجود در بدنه را اجرا می کند. توضیح این کد ها در ادامه آمده است.
10. یک قفل بر روی شی pending قرار داده شده و در بدنه ی این قفل 3 خط کد اجرا می شود. این قفل به این دلیل بر روی شی pending قرار داده شده که این شی توسط چندین thread مورد دسترسی قرار می گیرد. (مباحث مربوط به همروندی در در س سیستم عامل)
12. مقدار موجود در متغیر working یک واحد اضافه می شود. توجه کنید که این عمل بصورت اتمیک انجام میگیرد یعنی در یک کلاک پالس پردازنده.

13. در خطوط 13 و 14 دو عنصر از اول صف pending برداشته می شود و در شی های l1 و l2 قرار میگرند. اشیا l1 و l2 در واقع صف هایی هستند که عناصرشان اعداد صحیح است و حال آنکه عناصر صف pending نیز خود صف هایی هستند که حاوی اعداد صحیح اند. پس منطقاً درست عمل می شود.
16. تعریف شی l3 از نوع صفی که حاوی اعداد صحیح است.
17. در ادامه یک حلقه ی while وجود دارد که تا زمانی که عنصری درهر یک از صف های l1 و l2 قرار دارد(به شرط OR که با || پیاده سازی شده است دقت کنید)، کد موجود در بدنه ی خود که در واقع همان الگوریتم merge sort که یک الگوریتم تقسیم و غلبه است را اجرا می کند. توضیح این کد در ادامه آمده است.
18. از خطوط 18 تا 33 این اعمال انجام می شود. اگر تعداد عناصر موجود در هر دو صف بزرگتر از صفر بود، این به این معناست که باید عنصر کوچکتر از اول هر دو صف l1 و l2 انتخاب شده در صف هدف که همان l3 است، قرار بگیرد. اگر فقط در l1 عناصری موجود بودند و در l2 عنصری موجود نبود، این به آن معناست که تعداد عناصر لیست l1 بیشتر از لیست l2 بوده و اکنون باید تمامی آن عناصر به آخر صف l3 اضافه شوند. اگر فقط در l2 عناصری موجود بودند و عناصر موجود در l1 به اتمام رسیده بودند باید تمامی آن عناصر به آخر صف l3 اضافه شوند. این در واقع الگوریتم merge sort است. کتاب طراحی الگوریتم این الگوریتم را توضیح داده است.
34. دوباره یک قفل بر روی pending ایجاد شده و صف l3 توسط متد Enqueue به آن اضافه می شود و یک واحد از مقدار ذخیره شده در متغیر working کاسته شده است. تفاوت
;(ref working)Interlocked.Decrement با اینکه به سادگی بنویسیم;--working در این است که کد اول بصورت اتمیک در یک پالس ساعت پردازنده انجام می شود.(مباحث مربوط به multithreading و همروندی در درس سیستم عامل)
39. در ادامه یک حلقه ی While وجود دارد. این حلقه در واقع thread جاریرا در صورت برآورده شدن شرط حلقه به مدت 5 میلی ثانیه معلق یا suspend می کند. شرط حلقه این است که مقدار موجود در متغیر working بزرگتر از صفر و تعداد عناصر صف pending بزگتر یا مساوی 2 باشد. مقدار موجود در متغیر working با استفاده از تابع Thread.VolatileRead خوانده شده است. این تابع آخرین مقدار موجود در یک متغیر را بدون در نظر گرفتن اینکه چه پردازنده ای آن مقدار را نوشته یا اینکه محتویات کش پردازنده چه بوده است را می خواند.
48. در پایان با متد ;(())Task.WaitAll(tasks.ToArray به برنامه می فهمانیم که باید تا پایان یافتن تمامی thread های موجود در لیست tasks که در اول تابع تعریف شد، صبر کند.
49. و در آخر هم عنصر اول صف pending را به عنوان خروجی بر می گردانیم. که همان صف مرتب شده است.

سعی کنید برنامه را با مقادیر مختلفی برای listLength و maxSize اجرا کنید و نتیجه را مشاهده نمایید.