Лабораторная работа №8

Целью лабораторной работы является:

- Знакомство с параллельным программированием в C++.

1 Описание

Параллельное программирование – это техника программирования, которая использует преимущества многоядерных или многопроцесорных компьютеров и является подмножеством более широкого понятия многопоточности (multithreading).
Параллельное программирование включает три понятия:

- Определение параллелизма: анализ задачи с целью выделить подзадачи, которые могут выполняться одновременно.
- Выявление параллелизма: изменение структуры задачи таким образом, чтобы можно было эффективно выполнять подзадачи. Для этого часто требуется найти зависимости между подзадачами и организовать исходный код так, чтобы ими можно было эффективно управлять.
- Выражение параллелизма: реализация параллельного алгоритма в исходном коде с помощью системы обозначений параллельного программирования.

2 Исходный код

Описание классов фигур и методов класса-контейнера, определенных раннее, остается неизменным.

```
1
2  template <class T>
3  std::shared_ptr<TListItem<T>> TList<T>::PSort(std::shared_ptr<TListItem<T>> &head)
4  {
5      if (head == nullptr || head->GetNext() == nullptr) {
6          return head;
7      }
8
9      std::shared_ptr<TListItem<T>> partitionedEl = Partition(head);
10     std::shared_ptr<TListItem<T>> leftPartition = partitionedEl->GetNext();
11     std::shared_ptr<TListItem<T>> rightPartition = head;
12
13     partitionedEl->SetNext(nullptr);
```

```
14
15      if (leftPartition == nullptr) {
16          leftPartition = head;
17          rightPartition = head->GetNext();
18          head->SetNext(nullptr);
19      }
20
21      rightPartition = PSort(rightPartition);
22      leftPartition = PSort(leftPartition);
23      std::shared_ptr<TListItem<T>> iter = leftPartition;
24      while (iter->GetNext() != nullptr) {
25          iter = iter->GetNext();
26      }
27
28      iter->SetNext(rightPartition);
29
30      return leftPartition;
31  }
32
33  template <class T>
34  std::shared_ptr<TListItem<T>> TList<T>::Partition(std::shared_ptr<TListItem<T>> &head)
35  {
36      std::lock_guard<std::mutex> lock(mutex);
37      if (head->GetNext()->GetNext() = nullptr) {
38          if (head->GetNext()->GetFigure()->Square() > head->GetFigure()->Square()) {
39              return head->GetNext();
40          } else {
41              return head;
42          }
43      } else {
44          std::shared_ptr<TListItem<T>> i = head->GetNext();
45          std::shared_ptr<TListItem<T>> pivot = head;
46          std::shared_ptr<TListItem<T>> lastElSwapped = (pivot->GetNext()->GetFigure()->
                Square() >= pivot->GetFigure()->Square()) ? pivot->GetNext() : pivot;
47
48          while ((i != nullptr) && (i->GetNext() != nullptr)) {
49              if (i->GetNext()->GetFigure()->Square() >= pivot->GetFigure()->Square()) {
50                  if (i->GetNext() == lastElSwapped->GetNext()) {
51                      lastElSwapped = lastElSwapped->GetNext();
52                  } else {
53                      std::shared_ptr<TListItem<T>> tmp = lastElSwapped->GetNext();
54                      lastElSwapped->SetNext(i->GetNext());
55                      i->SetNext(i->GetNext()->GetNext());
56                      lastElSwapped = lastElSwapped->GetNext();
57                      lastElSwapped->SetNext(tmp);
58                  }
59              }
60              i = i->GetNext();
61          }
```

```
62          return lastElSwapped;
63      }
64  }
65
66  template <class T>
67  void TList<T>::Sort()
68  {
69      if (head == nullptr)
70          return;
71      std::shared_ptr<TListItem<T>> tmp = head->GetNext();
72      head->SetNext(PSort(tmp));
73  }
74
75  template <class T>
76  void TList<T>::ParSort()
77  {
78      if (head == nullptr)
79          return;
80      std::shared_ptr<TListItem<T>> tmp = head->GetNext();
81      head->SetNext(PParSort(tmp));
82  }
83
84  template <class T>
85  std::shared_ptr<TListItem<T>> TList<T>::PParSort(std::shared_ptr<TListItem<T>> &head)
86  {
87      if (head == nullptr || head->GetNext() == nullptr) {
88          return head;
89      }
90
91      std::shared_ptr<TListItem<T>> partitionedEl = Partition(head);
92      std::shared_ptr<TListItem<T>> leftPartition = partitionedEl->GetNext();
93      std::shared_ptr<TListItem<T>> rightPartition = head;
94
95      partitionedEl->SetNext(nullptr);
96
97      if (leftPartition == nullptr) {
98          leftPartition = head;
99          rightPartition = head->GetNext();
100         head->SetNext(nullptr);
101     }
102
103     std::packaged_task<std::shared_ptr<TListItem<T>>(std::shared_ptr<TListItem<T>>&)>
104         task1(std::bind(&TList<T>::PParSort, this, std::placeholders::_1));
105     std::packaged_task<std::shared_ptr<TListItem<T>>(std::shared_ptr<TListItem<T>>&)>
106         task2(std::bind(&TList<T>::PParSort, this, std::placeholders::_1));
107     auto rightPartitionHandle = task1.get_future();
108     auto leftPartitionHandle = task2.get_future();
109
110     std::thread(std::move(task1), std::ref(rightPartition)).join();
```

```
111     rightPartition = rightPartitionHandle.get();
112     std::thread(std::move(task2), std::ref(leftPartition)).join();
113     leftPartition = leftPartitionHandle.get();
114     std::shared_ptr<TListItem<T>> iter = leftPartition;
115     while (iter->GetNext() != nullptr) {
116         iter = iter->GetNext();
117     }
118 
119     iter->SetNext(rightPartition);
120     return leftPartition;
121 }
```

3 Консоль

```
denis@ubuntu:~/Desktop/OOP/oop_lab8$ ./run
TAllocationBlock: Memory init
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Sort list by qsort
0) Exit
1
Enter side A: 2
Enter side B: 3
Enter side C: 4
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Sort list by qsort
0) Exit
2
Enter side: 3
Choose an operation:
1) Add triangle
2) Add foursquare
```

```
3) Add octagon
4) Delete figure from list
5) Print list
6) Sort list by qsort
0) Exit
3
Enter side: 5
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Sort list by qsort
0) Exit
2
Enter side: 1
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Sort list by qsort
0) Exit
5
Side A = 2,side B = 3,side C = 4,type: Triangle

Side = 3,type: Foursquare

Side = 5,type: Octagon

Side = 1,type: Foursquare


Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
```

```
6) Sort list by qsort
0) Exit
6
1) using regular sort
2) to parallel
1
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Sort list by qsort
0) Exit
5
Side = 1,type: Foursquare

Side A = 2,side B = 3,side C = 4,type: Triangle

Side = 3,type: Foursquare

Side = 5,type: Octagon


Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Sort list by qsort
0) Exit
1
Enter side A: 2
Enter side B: 3
Enter side C: 5
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
```

```
5) Print list
6) Sort list by qsort
0) Exit
5
Side = 1,type: Foursquare

Side A = 2,side B = 3,side C = 4,type: Triangle

Side = 3,type: Foursquare

Side = 5,type: Octagon

Side A = 2,side B = 3,side C = 5,type: Triangle


Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Sort list by qsort
0) Exit
2
Enter side: 2
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Sort list by qsort
0) Exit
5
Side = 1,type: Foursquare

Side A = 2,side B = 3,side C = 4,type: Triangle

Side = 3,type: Foursquare

Side = 5,type: Octagon
```

```
Side A = 2,side B = 3,side C = 5,type: Triangle

Side = 2,type: Foursquare


Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Sort list by qsort
0) Exit
6
1) using regular sort
2) to parallel
2
Choose an operation:
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Sort list by qsort
0) Exit
5
Side A = 2,side B = 3,side C = 5,type: Triangle

Side = 1,type: Foursquare

Side A = 2,side B = 3,side C = 4,type: Triangle

Side = 2,type: Foursquare

Side = 3,type: Foursquare

Side = 5,type: Octagon


Choose an operation:
```

```
1) Add triangle
2) Add foursquare
3) Add octagon
4) Delete figure from list
5) Print list
6) Sort list by qsort
0) Exit
```

4 Выводы

В данной лабораторной работе необходимо было осуществить быструю сортировку двумя способами: используя технологии параллельного программирования: используя потоки, и традиционным способом. Технологии параллельного программирования широко исползуются во многих сферах в наши дни.