Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №8 по курсу «Объектно-ориентированное программирование»

Студент: С. М. Бокоч
Преподаватель:
Группа: М8О-204Б
Дата:
Оценка:
Подпись:

Москва, 2017

Лабораторная работа №8

Задача:
Целью лабораторной работы является:

1. Знакомство с параллельным программированием в C++.

Используя структуры данных, разработанные для лабораторной работы No6 (контейнер первого уровня и классы-фигуры) разработать алгоритм быстрой сортировки для класса-контейнера. Необходимо разработать два вида алгоритма:

- Обычный, без параллельных вызовов.
- С использованием параллельных вызовов. В этом случае, каждый рекурсивный вызов сортировки должен создаваться в отдельном потоке.

Для создания потоков использовать механизмы:

1. future
2. packaged_task/async

Для обеспечения потоко-безопасности структур данных использовать:

1. mutex
2. lock_guard

Фигуры. Квадрат, трапеция, прямоугольник.
Контейнер первого уровня. Массив.
Контейнер первого уровня. Список.

1 Теория

Параллельное программирование – это техника программирования, которая использует преимущества многоядерных или многопроцесорных компьютеров и является подмножеством более широкого понятия многопоточности (multithreading). Это способ организации компьютерных вычислений, при котором программы разрабатываются как набор взаимодействующих вычислительных процессов, работающих параллельно (одновременно).

Термин охватывает совокупность вопросов параллелизма в программировании, а также создание эффективно действующих аппаратных реализаций. Теория параллельных вычислений составляет раздел прикладной теории алгоритмов. Существуют различные способы реализации параллельных вычислений. Например, каждый вычислительный процесс может быть реализован в виде процесса операционной системы, либо же вычислительные процессы могут представлять собой набор потоков выполнения внутри одного процесса ОС. Параллельные программы могут физически исполняться либо последовательно на единственном процессоре — перемежая по очереди шаги выполнения каждого вычислительного процесса, либо параллельно — выделяя каждому вычислительному процессу один или несколько процессоров (находящихся рядом или распределённых в компьютерную сеть).

2 Листинг

```
1  //TList.cpp
2  template <class T>
3  std::shared_ptr<TListItem<T>> TList<T>::Partition(std::shared_ptr<TListItem<T>> &head)
4  {
5      std::lock_guard<std::mutex> lock(mutex);
6      if (head->GetNext()->GetNext() = nullptr) {
7          if (head->GetNext()->GetFigure()->Square() > head->GetFigure()->Square()) {
8              return head->GetNext();
9          } else {
10             return head;
11         }
12     } else {
13         std::shared_ptr<TListItem<T>> i = head->GetNext();
14         std::shared_ptr<TListItem<T>> pivot = head;
15         std::shared_ptr<TListItem<T>> lastElSwapped = (pivot->GetNext()->GetFigure()->
               Square() >= pivot->GetFigure()->Square()) ? pivot->GetNext() : pivot;
16
17         while ((i != nullptr) && (i->GetNext() != nullptr)) {
18             if (i->GetNext()->GetFigure()->Square() >= pivot->GetFigure()->Square()) {
19                 if (i->GetNext() == lastElSwapped->GetNext()) {
20                     lastElSwapped = lastElSwapped->GetNext();
21                 } else {
22                     std::shared_ptr<TListItem<T>> tmp = lastElSwapped->GetNext();
23                     lastElSwapped->SetNext(i->GetNext());
24                     i->SetNext(i->GetNext()->GetNext());
25                     lastElSwapped = lastElSwapped->GetNext();
26                     lastElSwapped->SetNext(tmp);
27                 }
28             }
29             i = i->GetNext();
30         }
31         return lastElSwapped;
32     }
33 }
34
35 template <class T>
36 void TList<T>::Sort()
37 {
38     if (head == nullptr)
39         return;
40     std::shared_ptr<TListItem<T>> tmp = head->GetNext();
41     head->SetNext(PSort(tmp));
42 }
43
44 template <class T>
45 void TList<T>::ParSort()
46 {
```

```
47      if (head == nullptr)
48          return;
49      std::shared_ptr<TListItem<T>> tmp = head->GetNext();
50      head->SetNext(PParSort(tmp));
51  }
52
53  template <class T>
54  std::shared_ptr<TListItem<T>> TList<T>::PParSort(std::shared_ptr<TListItem<T>> &head)
55  {
56      if (head == nullptr || head->GetNext() == nullptr) {
57          return head;
58      }
59
60      std::shared_ptr<TListItem<T>> partitionedEl = Partition(head);
61      std::shared_ptr<TListItem<T>> leftPartition = partitionedEl->GetNext();
62      std::shared_ptr<TListItem<T>> rightPartition = head;
63
64      partitionedEl->SetNext(nullptr);
65
66      if (leftPartition == nullptr) {
67          leftPartition = head;
68          rightPartition = head->GetNext();
69          head->SetNext(nullptr);
70      }
71
72      std::packaged_task<std::shared_ptr<TListItem<T>>(std::shared_ptr<TListItem<T>>&)>
73          task1(std::bind(&TList<T>::PParSort, this, std::placeholders::_1));
74      std::packaged_task<std::shared_ptr<TListItem<T>>(std::shared_ptr<TListItem<T>>&)>
75          task2(std::bind(&TList<T>::PParSort, this, std::placeholders::_1));
76      auto rightPartitionHandle = task1.get_future();
77      auto leftPartitionHandle = task2.get_future();
78
79      std::thread(std::move(task1), std::ref(rightPartition)).join();
80      rightPartition = rightPartitionHandle.get();
81      std::thread(std::move(task2), std::ref(leftPartition)).join();
82      leftPartition = leftPartitionHandle.get();
83      std::shared_ptr<TListItem<T>> iter = leftPartition;
84      while (iter->GetNext() != nullptr) {
85          iter = iter->GetNext();
86      }
87
88      iter->SetNext(rightPartition);
89      return leftPartition;
90  }
```

3 Выводы

В данной лабораторной работе я получил алгоритмы работы с параллельным программированием в C++. Реализовал алгоритм быстрой сортировки и распараллелил его.