

Лабораторная работа №8

Задача: Используя структуры данных, разработанные для лабораторной работы №6 (контейнер 1-ого уровня и классы-фигуры) разработать алгоритм быстрой сортировки для класс-контейнера.

Необходимо разработать два вида алгоритма:

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

Для создания потоков использовать механизмы:

- future
- packaged task/async

Для обеспечения потокобезопасности структур использовать механизмы:

- mutex
- lock quard

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контейнер.
- Распечатывать содержимое контейнера.
- Удалять фигуры из контейнера.
- Проводить сортировку контейнера.

Фигуры: треугольник, квадрат, прямоугольник.

Контейнер: связный список.

1 Введение

Класс `std::future` представляет собой обертку, над каким-либо значением или объектом (даже значением), вычисление или получение которого происходит отложено. Точнее, `future` предоставляет доступ к некоторому разделяемому состоянию, которое состоит из 2-х частей: данные (здесь лежит значение) и флаг готовности.

Для получения значения из `future` предназначен метод `std::future::get`. При этом, поток вызвавший `get` блокируется до вычисления значения. В арсенале объектов, помогающих в реализации идиомы асинхронного программирования, в C++11, существует вполне логичная для такого рода программирования сущность, - задача. Ведь именно задача является базовым блоком асинхронного программирования. В C++ роль задачи выполняет объект класса `std::packaged_task`.

Асинхронные операции (созданные с помощью `std::async`, `std::packaged_task`, или `std::promise`) могут вернуть объект типа `std::future` создателю этой операции.

2 Код программы

`tlist.h`

```
#ifndef TLIST_H
#define TLIST_H
#include "tlist_iterator.h"
#include "tlist_item.h"
#include <memory>
#include <future>
#include <mutex>

template <class T>
class TListItem;

template <class T>
using TListItemPtr = std::shared_ptr<TListItem<T>>;

template <class T> class TList {
public:
    TList()
    {
        head = nullptr;
    }
};
```

```

}

void Push(T* item)
{
    TListItemPtr<T>other(new TListItem<T>(item));
    other->SetNext(head);
    head = other;
}

void Push(std::shared_ptr<T>item)
{
    TListItemPtr<T>other(new TListItem<T>(item));
    other->SetNext(head);
    head = other;
}

bool IsEmpty() const
{
    return head == nullptr;
}

size_t Size()
{
    size_t result = 0;
    for (auto i : *this)
        result++;
    return result;
}

TListItemPtr<T>begin()
{
    return TListItemPtr<T>(head);
}

TListItemPtr<T>end()
{
    return TListItemPtr<T>(nullptr);
}

void Sort()

```

```

{
if (Size() >1) {
std::shared_ptr<T>middle = Pop();
TList<T>left,right;
while (!IsEmpty()) {
std::shared_ptr<T>item = Pop();
if (!item->SquareLess(middle)) {
left.Push(item);
}
else {
right.Push(item);
}
}
left.Sort();
right.Sort();
while (!left.IsEmpty()) {
Push(left.PopLast());
}
Push(middle);
while (!right.IsEmpty()) {
Push(right.PopLast());
}
}
}

void SortParallel()
{
if (Size() >1) {
std::shared_ptr<T>middle = PopLast();
TList<T>left,right;
while (!IsEmpty()) {
std::shared_ptr<T>item = PopLast();
if (!item->SquareLess(middle)) {
left.Push(item);
}
else {
right.Push(item);
}
}
std::future<void>left_res = left.sort_in_background();
std::future<void>right_res = right.sort_in_background();
}

```

```
left_res.get();
```

```
while (!left.IsEmpty()) {  
    Push(left.PopLast());  
}  
Push(middle);  
right_res.get();  
while (!right.IsEmpty()) {  
    Push(right.PopLast());  
}  
}  
}
```

```
std::shared_ptr<T>Pop()  
{  
    std::shared_ptr<T>result;  
    if (head != nullptr) {  
        result = head->GetValue();  
        head = head->GetNext();  
    }  
    return result;  
}
```

```
std::shared_ptr<T>PopLast()  
{  
    std::shared_ptr<T>result;  
    if (head != nullptr) {  
        TListItemPtr <T>element = head;  
        TListItemPtr <T>prev = nullptr;  
        while (element->GetNext() != nullptr) {  
            prev = element;  
            element = element->GetNext();  
        }  
        if (prev != nullptr) {  
            prev->SetNext(nullptr);  
            result = element->GetValue();  
        }  
        else {
```

```

result = element->GetValue();
head = nullptr;
}
}
return result;
}

void Delete(std::shared_ptr<T>key)
{
bool found = false;
if (head != nullptr) {
TListItemPtr <T>element = head;
TListItemPtr <T>prev = nullptr;
while (element != nullptr) {
if (element->GetValue()->TypedEquals(key)) { //found :)
found = true;
break;
}
prev = element;
element = element->GetNext();
}
if (found) {
if (prev != nullptr) {
prev->SetNext(element->GetNext());
}
else {
head = element->GetNext();
}
}
}
}

template <class A>
friend std::ostream& operator<<(std::ostream& os,const TList<A>& list)
{
TListItemPtr<A>item = list.head;
if (list.IsEmpty())
os <<"List is empty\n";
while (item != nullptr) {
os <<*item;
item = item->GetNext();
}
}

```

```

}
return os;
}
virtual ~TList() {}
private:
std::future<void>sort_in_background()
{
std::packaged_task<void(void) >task(std::bind(std::mem_fn(&TList<T>::SortParallel),this),std::move(*this));
std::future<void>res(task.get_future());
std::thread thr(std::move(task));
thr.detach();
return res;
}
TListItemPtr<T>head;
};
#endif

```

3 Вывод программы:

```

-----
-----МЕНЮ-----
|1-Добавить треугольник      |
|2-Добавить прямоугольник   |
|3-Добавить квадрат         |
|4-Удалить фигуру           |
|5-Сортировка               |
|6-Параллельная сортировка  |
|7-Распечатать список       |
|8-Выход                    |
Выберете действие:
1
Введите значение a:3
Введите значение b:4
Введите значение c:5
Выберете действие:
2
Введите значение a:1
Введите значение b:2
Выберете действие:
3

```

Введите значение a:1

Выберете действие:

2

Введите значение a:1

Введите значение b:6

Выберете действие:

2

Введите значение a:1

Введите значение b:7

Выберете действие:

7

Прямоугольник со сторонами [1,7],Площадь = 7

Прямоугольник со сторонами [1,6],Площадь = 6

Квадрат со стороной [1],Площадь = 1

Прямоугольник со сторонами [1,2],Площадь = 2

Треугольник со сторонами [3,4,5],Площадь = 6

Выберете действие:

5

Квадрат со стороной [1],Площадь = 1

Прямоугольник со сторонами [1,2],Площадь = 2

Прямоугольник со сторонами [1,6],Площадь = 6

Треугольник со сторонами [3,4,5],Площадь = 6

Прямоугольник со сторонами [1,7],Площадь = 7

Выберете действие:

2

Введите значение a:3

Введите значение b:4

Выберете действие:

2

Введите значение a:1

Введите значение b:5

Выберете действие:

7

Прямоугольник со сторонами [1,5],Площадь = 5

Прямоугольник со сторонами [3,4],Площадь = 12

Квадрат со стороной [1],Площадь = 1

Прямоугольник со сторонами [1,2],Площадь = 2

Прямоугольник со сторонами [1,6],Площадь = 6

Треугольник со сторонами [3,4,5],Площадь = 6

Прямоугольник со сторонами [1,7],Площадь = 7

Выберете действие:

6

Квадрат со стороной [1],Площадь = 1

Прямоугольник со сторонами [1,2],Площадь = 2

Прямоугольник со сторонами [1,5],Площадь = 5

Треугольник со сторонами [3,4,5],Площадь = 6

Прямоугольник со сторонами [1,6],Площадь = 6

Прямоугольник со сторонами [1,7],Площадь = 7

Прямоугольник со сторонами [3,4],Площадь = 12

Выберете действие:

8

Ссылка на гитхаб: <https://github.com/alexma99/2-course/tree/master/oop>