53. Нови стандартизирани конструкции в C++11. Асинхронни задачи, нишки и локални за тях данни.

Асинхронни задачи:малки порции на оригинален алгоритъм свързани само с данните, които те произвеждат или употребяват.

Нишки: звена за изпълнение на администрира от Runtime Environment. Те се отнасят до задачи в смисъл, че задачите се изготвят на ниво нишки.

**1Асинхронни задачи**

**int a, b, c;**

**int calculateA() { return a+a\*b; }**

**int calculateB() { return a\*(a+a\*(a+1)); }**

**int calculateC() { return b\*(b+1)-b; }**

int main(int argc, char \*argv[])

{ getUserData(); // initializes a and b

c = calculateA() \* (calculateB() + calculateC());

showResult();

}

Основната функция изсиква информация от потребителя и след това я предава на трите функции: calculateA, calculateB and calculateC.Резултатите след това се преобразуват ,за да могат да се представят на потребителя.

Тъй като тези функции са независими, ние можем да ги изпълняваме паралелно с помощта на асинхронена функция:

int main(int argc, char \*argv[])

{ getUserData();

future<int> f1 = async(calculateB), f2 = async(calculateC);

c = (calculateA() + f1.get()) \* f2.get();

showResult();

}

В даден момент ще имеме нужда от резултатите от тези паралелно изпълнени функции.Извикването на гет методите на всеки future блокира изпълнението , докато стойността е достъпна.Най-голямто закъснение на този подход е 3 секунди срещу 9 от страна на секвенциалната версия.

**2.Нишки:**

Въпреки, че е по-сложен програмен моделнишките предлагат по-добри методи за синхронизация и координация, което им позволява да доведат до изпълнение на друга нишка и да чакат за определен период от време или докато друга нишка е завършена преди да продължи.

auto multiple\_finder = [](int n) {

for (int i = 0; i < 100000; i++)

if (i%n==0)

cout << i << " is a multiple of " << n << endl; };

int main(int argc, char \*argv[]) {

thread th(multiple\_finder, 23456);

multiple\_finder(34567);

th.join();

}

В основната функция стартираме тази функция в две нишки с различни параметри.

Ние може да изпълним примера за асинхронни задачи от предишния раздел нишки. За тази цел трябва да се въведе понятието обещание(promise). Обещанието може да се разбира като мивка, чрез които резултатът ще бъде премахнат, когато е налична. Къде този резултат ще излезе веднъж отпаднал? Всяко обещание има асоциирано бъдеще.

**и локални за тях данни:**

В C + + може да се определят глобални променливи, чийто обхват е обвързан с цялото приложение, включително нишки. Но по отношение на нишките, сега има начин да се определят тези глобални променливи, които са такива, че всяка нишка поддържа свое копие. Тази концепция е известна като thread local storage и е декларирана, както следва:

**thread\_local int subtotal = 0;**

Ако декларацията се извършва в обхвата на функцият, видимостта на променливата ще се стесни до тази функция, но всяка нишка ще поддържа свое собствено статичен копие.

Въпреки че thread\_local не е налична в Visual C + + 11, то тя може да се симулира с нестандартно разширение на Microsoft:

#define thread\_local \_\_declspec(thread)