Asynchroniczne wywołanie

dr inż. Ireneusz Szcześniak

jesień 2017

1 Wprowadzenie

Asynchroniczne wywołanie realizujemy funkcją `std::async`. Można wywoływać funkcje, funktory lub wyrażenia lambda.

Funkcja `std::async` zwraca obiekt klasy `std::future<T>`, gdzie `T` jest typem wartości zwracanej przez funkcję. Wartość zwracaną przez funkcję wywołaną asynchronicznie otrzymujemy wywołując funkcję `get` na rzecz otrzymanego obiektu klasy `std::future<T>`.

Obiekty klasy `std::future<T>` są tylko do przenoszenia, nie można ich kopiować.

Programy kompilujemy z GCC z flagą `-pthread`.

```
#include <iostream>
#include <future>

using namespace std;

void
foo()
{
  std::cout << "function" << std::endl;
}

struct A
{
  char operator () (char c)
  {
    std::cout << "functor:␣" << c << std::endl;
    return c + 1;
  }
};

int main()
{
  std::async(foo);

  std::cout << "Done␣#1!" << std::endl;

  A a;
  auto f = std::async(a, 'a');
  auto v = f.get();
  cout << "Got␣" << v << endl;

  std::cout << "Done␣#2!" << std::endl;

  return 0;
}
```

2 Polityka wywołania

Domyślną polityką jest `std::launch::async` std::launch::deferred|. Jeżeli chcemy zmienić politykę wywołania, to możemy użyć przeciążenia funkcji `std::async`, gdzie jako pierszy argument przekazujemy politykę wywołania.

```
#include <chrono>
#include <future>
#include <iostream>
#include <vector>

using namespace std;

int
foo(int i)
{
  // 1s to literał
  this_thread::sleep_for(1s);
  return i * i;
}

int
main()
{
  vector<future<int>> vf;

  for(int i = 0; i < 10; ++i)
    vf.push_back(async(launch::async, foo, i));
    // vf.push_back(async(launch::deferred, foo, i));

  // Iterujemy z użyciem referencji, a nie wartości (kopii obiektu z
  // kontenera), bo std::future jest typem tylko do przenoszenia.
  for(auto &f: vf)
    cout << f.get() << endl;
}
```

3 Przenoszenie wyniku

Jeżeli funkcja zwraca swój wynik przez wartość, to wywołanie asynchroniczne będzie wynik przenosiło, a nie kopiowało.

```
#include <future>

using namespace std;

struct A
{
  A() = default;

  A(const A &a) = delete;
  A& operator = (const A &a) = delete;

  A(A &&a) = default;
  A& operator = (A &&a) = default;
};

A print()
```

```
{
  return A();
}

int main()
{
  const A &a = std::async(print).get();

  return 0;
}
```

4 Przechwytywanie wyjątku

Jeżeli funkcja rzuca wyjątek, to ten sam wyjątek będzie rzucony wtedy, kiedy będzie wywołana funkcja `future<T>::get`.

```
#include <iostream>
#include <future>

using namespace std;

void
foo()
{
  throw true;
}

int
main()
{
  auto f = async(foo);

  try
    {
      f.get();
    } catch (bool)
    {
      cout << "Gotcha!\n";
    }
}
```