multi threading c++11

Warsaw C++ User Group 13.05.2014

thread

default (1)

```
thread() noexcept;
```

initialization (2)

```
template <class Fn, class... Args>
explicit thread (Fn&& fn, Args&&... args);
```

copy [deleted] (3)

```
thread (const thread&) = delete;
```

move (4)

```
thread (thread&& x) noexcept;
```

thread

```cpp
void foo()
{
  // do stuff...
}

void bar(int x)
{
  // do stuff...
}
```

```cpp
int main()
{
  std::thread first (foo);     // spawn new thread that calls foo()
  std::thread second (bar,0);  // spawn new thread that calls bar(0)

  std::cout << "main, foo and bar now execute concurrently...\n";

  // synchronize threads:
  first.join();                // pauses until first finishes
  second.join();               // pauses until second finishes

  std::cout << "foo and bar completed.\n";

  return 0;
}
```

future, sync

```
int foo(int x) { return x + 1; }

std::future<int> fut = std::async(foo, 15);
…
std::cout<< fut.get();
```

```
vector<future<int>> v;
for (int i = 0 ; i < 10 ; i++)
    v.push_back(std::async([](int x) {
        sleep(x);
        return x + 1; })
```

```
int p = 0;
foreach(v.begin(), v.end(), [](future<int> f)
{
    p += f.get();
});
```

MapReduce

```
template <typename Iter, class MapFunction, class ReduceFunction>
void mapReduce(Iter first,
                        Iter last,
                        MapFunction mapFunction,
                        ReduceFunction reduceFunction,
                        size_t threadsCount);

/*Rozdziela przedział [first, last) na threadCount części, następnei
wykonuje na każdym z nich mapFunction a później scala za pomocą
reduceFunction */
```

```
{
    if (threadsCount > 1)
    {
        threadsCount--;
        int distance = std::distance(first, last) - 1;
        Iter middle = first;
        std::advance(middle, distance/2+1);
        auto secondPart = async(std::launch::async,
                                mapReduce<Iter,MapFunction, ReduceFunction>,
                                middle, last, mapFunction,
                                reduceFunction, threadsCount/2);
        threadsCount -= threadsCount/2;
        mapReduce(first, middle, mapFunction, reduceFunction, threadsCount);
        secondPart.wait();
        reduceFunction(first, middle, last);
    }
else
    {
        mapFunction(first, last);
    }
}
//wywołanie
typedef vector<int>::iterator it;
vector<int> s(...);
mapReduce(s.begin(), s.end(), sort<it>, inplace_merge<it>, 42)
```

Tutorial wielowątkowości c++11

Bartosz Milewski youtube

https://www.youtube.com/user/DrBartosz

playlista Concurrency

Dzięki za uwagę!