Subject: Re: Job package: A lightweight multithreading tool, using promise/future mechanism
Posted by mirek on Sun, 10 Sep 2017 13:08:26 GMT
View Forum Message <> Reply to Message

Trying to figure out what is relative advantage of using Job vs CoWork vs C++11 future/promise....

For now, it looks like Job is not using worker threads, creating thread for each run. Which actually can have advantage if there are blocking operations (file I/O etc), but that is meant to be dealt with in CoWork by increasing the number of threads.

Other than that, your examples can be rewritten with CoWork easily. "WaitForJobs" is outright ugly, being single global function for all Jobs; I think CoWork::Finish has a clear upper hand here...

```
 {
  // Print a message to stdout.
  // Returns void.
  CoWork job;
  job & [=] { Cout() << "Hello world!\n"; };
  job.Finish();
 }

 {
  // Print all the divisors of random numbers.
  // Returns a String.
  CoWork jobs;
  Vector<String> results;
  for(int i = 0; i < 5; i++)
   jobs & [=, &results] { String h = GetDivisors2(); CoWork::FinLock(); results.At(i) = GetDivisors();
};
  jobs.Finish();
  for(auto r : results)
   Cout() << r << '\n';
 }
```