Subject: CoWork and shared memory
Posted by peek on Wed, 22 Feb 2012 16:53:56 GMT
View Forum Message <> Reply to Message

Hello all

Is it possible that processes running in different cores with CoWork can share the same memory through a mutex?

Peek

Subject: Re: CoWork and shared memory
Posted by mirek on Thu, 23 Feb 2012 19:38:13 GMT
View Forum Message <> Reply to Message

peek wrote on Wed, 22 February 2012 11:53Hello all

Is it possible that processes running in different cores with CoWork can share the same memory through a mutex?

Peek

Sure. Is there any reason why it shlould not?

Mirek

Subject: Re: CoWork and shared memory
Posted by peek on Thu, 23 Feb 2012 20:41:29 GMT
View Forum Message <> Reply to Message

Thank you Mirek

I will try with some INITBLOCK.

The reason of the question was that in the doc it says:

Quote:This class is indented as loop-parallelization tool. Whenever loop iterations are independent (they do not share any data)

Thank you again

Subject: Re: CoWork and shared memory
Posted by mirek on Fri, 24 Feb 2012 10:53:37 GMT

View Forum Message <> Reply to Message

peek wrote on Thu, 23 February 2012 15:41Thank you Mirek

I will try with some INITBLOCK.

The reason of the question was that in the doc it says:

Quote:This class is indented as loop-parallelization tool. Whenever loop iterations are independent (they do not share any data)

Thank you again

Ah, sorry, I guess it would need more clarification. The real deal is that 'not share any data' is best wrt performance.

Subject: Re: CoWork and shared memory
Posted by mirek on Fri, 24 Feb 2012 10:56:21 GMT
View Forum Message <> Reply to Message

Like this:

Whenever loop iterations are independent (they do not share any data between iterations), CoWork can be used to relatively easily spawn loop iterations over thread and thus over CPU cores. Note that previous statement does not preclude CoWork iterations to share data at all - sharing data using Mutex or similar serialization mechanisms still works.

Subject: Re: CoWork and shared memory
Posted by peek on Fri, 24 Feb 2012 12:04:18 GMT
View Forum Message <> Reply to Message

Great , I understand.

One question: This way it would be better to use CoWork instead of Thread because CoWork always tries to run in all available cores but Thread only runs in actual core?.

Subject: Re: CoWork and shared memory
Posted by mirek on Fri, 24 Feb 2012 12:52:29 GMT
View Forum Message <> Reply to Message

peek wrote on Fri, 24 February 2012 07:04Great , I understand.

One question: This way it would be better to use CoWork instead of Thread because CoWork always tries to run in all available cores but Thread only runs in actual core?.

Ah, the answer is a bit more complicated. CoWork actually uses thread pool - it creates a couple of threads (number of logical CPU cores + 2) the first time it is used and then uses all these threads to execute everything in parallel. So you are at least saved thread creation/destruction costs. Plus, it uses the very same pool for any nested looping (you can use another CoWork loop nested with within). Actually, it does not even have to be a loop, simply what you run with CoWork will possibly run in parallel and you can be sure that at CoWork destructor, all is finished. The order of running those tasks is of course unspecified.

Subject: Re: CoWork and shared memory
Posted by peek on Sat, 25 Feb 2012 19:05:18 GMT
View Forum Message <> Reply to Message

Hello Mirek

So this way have Cowork processes to end fast because they work in any way in the same thread that main program?

I mean, to do:
```
App::~App() {
 finish = true;
 ...

// and in Cowork function
if (finish)
 return;
```
is useless as destructor is not accessible until Cowork variable is destructed and that is is done when all Cowork calls have ended.

A question: If the Cowork jobs lasts time a solution could be to run Cowork in a separate thread?

Peek

Subject: Re: CoWork and shared memory
Posted by mirek on Sun, 26 Feb 2012 08:56:04 GMT
View Forum Message <> Reply to Message

peek wrote on Sat, 25 February 2012 14:05Hello Mirek

So this way have Cowork processes to end fast because they work in any way in the same thread that main program?

I mean, to do:
```
App::~App() {
 finish = true;
```

...

// and in Cowork function
if (finish)
 return;
is useless as destructor is not accessible until Cowork variable is destructed and that is is done when all Cowork calls have ended.

No, this still works - CoWork has no clue what the processing is and this one simply makes the processing "nop" and it ends quickly. The loop would still go through all iterations, thread will still preform them, but each iteration job will end immediately...

Quote:
A question: If the Cowork jobs lasts time a solution could be to run Cowork in a separate thread?

Well, not really, if you want a clean exit from the app...

The problematic thing is that for clean exit, you need cannot use any form on thread abortion - it has to exit through its normal exit point; otherwise it would result in resource leaks.

Mirek
