Subject: CoWork buggy!?
Posted by Werner on Fri, 21 Mar 2008 12:39:09 GMT
View Forum Message <> Reply to Message

I might be terribly wrong, but as far as I can see, CoWork doesn't work correctly - at least if there are less jobs than cpu cores.

Look at this:

```
/* analysis of a CoWork instance doing only 1 job */

/* create a CoWork instance named "coWork" */
// initializes "todo" with "0" (unnecessary)
CoWork coWork;
// jobs.GetCount:  0
// threads.GetCount: 0
// todo:    0
// waitforfinish:  0
// waitforjob:   0
// waiting_threads:  0

/* assign a job to "coWork" */
// "coWork.Do" calls "CoWork::pool" which creates a static Pool referenced by "p"
//   this calls "CoWork::Pool::Pool" which assigns cpu( core)s + 2 Threads ...
//  ... - following "max" - to the Threads in "p"
//    the OS calls "max" times "CoWork::Pool::ThreadRun", ...
//    ... which sets "waiting_threads" to "max"
//    now each "CoWork::Pool::ThreadRun" is waiting for a job
// "coWork.Do" adds the job to "jobs" in "p"
// "coWork.Do" increments "todo"
// "coWork.Do" decrements "waiting_threads"
// "coWork.Do" increments "waitforjob"
coWork.Do(fn_to_run_in_a_Thread);
// jobs.GetCount:  1
// threads.GetCount: max
// todo:    1
// waitforfinish:  0
// waitforjob:   1
// waiting_threads:  max - 1

/* now the OS activates one of the Threads */
// "waitforjob" is now "signaled", so "waitforjob" is set to "unsignaled" and ...
// "CoWork::Pool::DoJob" is called
//   CoWork::Pool::ThreadRun decrements "waitforjob" and ...
//   calls "CoWork::Pool::DoJob"
//     "CoWork::Pool::DoJob" removes the job from "jobs" and ...
//     ... runs the job's function, then decrements "todo"
//     as "todo" is now "0", "waitforfinish" is incremented
```

```
//   CoWork::Pool::ThreadRun increments "waiting_threads" and ...
//  ... waits for a new job
CoWork::Pool::ThreadRun(threadNumber);
// jobs.GetCount:  0
// threads.GetCount: max
// todo:    0
// waitforfinish:  1
// waitforjob:   0
// waiting_threads:  max

// as "todo" is now "0", "coWork.Finish" which might be called directly ...
// ... or by "coWork.~CoWork" does nothing
// that means:
// although CoWork should be terminated, all Threads are still running ...
// ... and, as "p" is static, even "CoWork::Pool::~Pool" which kills the running Threads ...
// ... via "CoWork::Pool::DoJob" and "CoWork::Pool::ThreadRun" is called only ...
// ... when the application itself is terminated
```

Maybe I don't understand how to handle multithtreading under Ultimate++. If so, a little documentation might help ...

Werner

Subject: Re: CoWork buggy!?
Posted by mirek on Sun, 23 Mar 2008 07:18:44 GMT
View Forum Message <> Reply to Message

Werner wrote on Fri, 21 March 2008 08:39I might be terribly wrong, but as far as I can see, CoWork doesn't work correctly - at least if there are less jobs than cpu cores.

```
// as "todo" is now "0", "coWork.Finish" which might be called directly ...
// ... or by "coWork.~CoWork" does nothing
// that means:
// although CoWork should be terminated, all Threads are still running ...
// ... and, as "p" is static, even "CoWork::Pool::~Pool" which kills the running Threads ...
// ... via "CoWork::Pool::DoJob" and "CoWork::Pool::ThreadRun" is called only ...
// ... when the application itself is terminated
[/code]
```

Well, not sure if I fully undestand the complaint, but if the issue is that Finish does not terminate threads, it is exactly what we wanted. Once created, threads are terminated only at app exit.

The reason is obvious - creating / terminating threads is somewhat expensive. So we have a pool of worker threads ready when next CoWork hits. And they are shared across all CoWorks as well, even nested.

Mirek

Subject: Re: CoWork buggy!?
Posted by Werner on Sun, 23 Mar 2008 10:38:38 GMT
View Forum Message <> Reply to Message

luzr wrote on Sun, 23 March 2008 08:18Well, not sure if I fully undestand the complaint, but if the issue is that Finish does not terminate threads, it is exactly what we wanted. Once created, threads are terminated only at app exit.

The reason is obvious - creating / terminating threads is somewhat expensive. So we have a pool of worker threads ready when next CoWork hits. And they are shared across all CoWorks as well, even nested.

Mirek
Ok, maybe I misunderstood the idea of "CoWork" . If so, sorry for that . After all there isn't any documentation and the code is written in a style which - well, let's say - I'm not familiar with .

Anyway, this raises bags of questions of which the following 2 are my favorites:

1.

Under which conditions should "CoWork::Finish" be called?

2. (independent from the answer to question #1)

What is "CoWork::waitforfinish" for?

If "CoWork::Finish" is called there are 2 possibilities:

a) no waiting job, i. e. "todo == 0":

Nothing is done. "CoWork::waitforfinish" remains unchanged.

b) waiting job, i. e. "todo > 0", e. g. "todo == 1":

That means "Pool:jobs.GetCount == 1" too, because both "todo" and "jobs" are incremented by "CoWork::Do" and decremented by "CoWork::Pool::DoJob". That again means "CoWork::Pool::DoJob" increments "CoWork::waitforfinish".

The conclusion is: "CoWork::waitforfinish" is never decremented and might raise without limit .

Werner

Werner wrote on Sun, 23 March 2008 06:38luzr wrote on Sun, 23 March 2008 08:18Well, not sure if I fully undestand the complaint, but if the issue is that Finish does not terminate threads, it is exactly what we wanted. Once created, threads are terminated only at app exit.

The reason is obvious - creating / terminating threads is somewhat expensive. So we have a pool of worker threads ready when next CoWork hits. And they are shared across all CoWorks as well, even nested.

Mirek
Ok, maybe I misunderstood the idea of "CoWork" . If so, sorry for that . After all there isn't any documentation and the code is written in a style which - well, let's say - I'm not familiar with .

Anyway, this raises bags of questions of which the following 2 are my favorites:

1.

Under which conditions should "CoWork::Finish" be called?

Finish waits until all the work required by calling "Do" is finished.

Quote:
What is "CoWork::waitforfinish" for?

It is used for synchronization. If there are any unfinished jobs, Finish has to wait until they are finished.

See:

```
bool CoWork::Pool::DoJob()
{
 Pool& p = pool();
 if(p.jobs.Top().work == NULL) {
  LLOG("Quit thread");
  return true;
 }
 MJob job = p.jobs.Pop();
 p.lock.Leave();
 job.cb();
 p.lock.Enter();
 if(--job.work->todo == 0) {
  LLOG("Releasing waitforfinish of (CoWork " << FormatIntHex(job.work) << ")");
  job.work->waitforfinish.Release();
```

```
 }
 LLOG("Finished, remaining todo " << job.work->todo << " (CoWork " << FormatIntHex(job.work)
<< ")");
 return false;
}
```

In this method, thread picks a new job to do. Each job has associated pointer to CoWork, where todo is number of jobs that yet has to be finished (you order a job to be finished by calling "Do"). If this goes to zero, all the work is done and waitforfinish semaphore can be released, so that Finish will get past waitforfinish.Wait.

a) no waiting job, i. e. "todo == 0":

Nothing is done. "CoWork::waitforfinish" remains unchanged.

It is not waiting job, it is *unfinished* job! (Includes waiting jobs and jobs that are currently being processed).

The conclusion is: "CoWork::waitforfinish" is never decremented and might raise without limit 8o .

Well, it is true that the last "Release" can have no matching Wait, but that is hardly a problem, as in that situation (todo == 0), everything is finised and we do not need to wait...

Mirek

Subject: Re: CoWork buggy!?
Posted by Werner on Sun, 23 Mar 2008 16:56:31 GMT
View Forum Message <> Reply to Message

luzr wrote on Sun, 23 March 2008 14:11Finish waits until all the work required by calling "Do" is finished.

...

... "CoWork::waitforfinish" ... is used for synchronization. If there are any unfinished jobs, Finish has to wait until they are finished.

...

... todo ... is not waiting job, it is *unfinished* job! (Includes waiting jobs and jobs that are currently being processed).

Thank you very much. This clarification was extremely helpful . Indeed I misunderstood CoWorks's design .

Do I get it right now when I assume that usage of this module just requires to

1.
create a CoWork instance, e. g.:
CoWork coWork;

2a.
assign a job, e. g.:
coWork.Do(a_job_in_form_of_a_callback);

or

2b.
assign a couple of jobs, e. g.:
coWork & job_0 & job_1 & job_2;

and *basically nothing more*? And that CoWork::Finish is *only* needed for synchronization purposes?

(I dumped all the other questions as "insignificant").

Werner

Subject: Re: CoWork buggy!?
Posted by mirek on Sun, 23 Mar 2008 17:28:16 GMT
View Forum Message <> Reply to Message

Werner wrote on Sun, 23 March 2008 12:56luzr wrote on Sun, 23 March 2008 14:11Finish waits until all the work required by calling "Do" is finished.

...

... "CoWork::waitforfinish" ... is used for synchronization. If there are any unfinished jobs, Finish has to wait until they are finished.

...

... todo ... is not waiting job, it is *unfinished* job! (Includes waiting jobs and jobs that are currently being processed).

Thank you very much. This clarification was extremely helpful . Indeed I misunderstood CoWorks's design .

Do I get it right now when I assume that usage of this module just requires to

1.
create a CoWork instance, e. g.:
CoWork coWork;

2a.
assign a job, e. g.:
coWork.Do(a_job_in_form_of_a_callback);

or

2b.
assign a couple of jobs, e. g.:
coWork & job_0 & job_1 & job_2;

and *basically nothing more*? And that CoWork::Finish is *only* needed for synchronization purposes?

(I dumped all the other questions as "insignificant").

Werner

Yes, but the primary usage is loop paralelization.

```
{
   CoWork co;
   for(int i = 0; i < n; i++)
      co & callback1(processI, i);
}
```

For example, imagine large Image brightness adjustment.... (and Finish in destructor enforces that the brightness for the full Image is adjusted after the block....)

Mirek

Subject: Re: CoWork buggy!?
Posted by Mindtraveller on Mon, 24 Mar 2008 09:48:56 GMT
View Forum Message <> Reply to Message

As far as I understand, CoWork detects number of processors present and tries to execute threads on different cores. Is that so? Can I have CoWork statistics how many theads are executed on each processor?

Subject: Re: CoWork buggy!?
Posted by mirek on Mon, 24 Mar 2008 10:22:51 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Mon, 24 March 2008 05:48As far as I understand, CoWork detects number of processors present and tries to execute threads on different cores. Is that so?

Yep, that is the main goal.

Quote:
Can I have CoWork statistics how many theads are executed on each processor?

No But the thread-pool is set to number-of-logical-cpus + 2.

"2" is arbitrarily choosen number. In reality, "jobs" can do I/O, so you want a reserve for blocking operations. OTOH, too much more threads would cause too much overhead - it seems that >5 slows things down.

If you want some deeper insight into CoWork operations, you can activate LLOG and LDUMP in Core/CoWork.cpp.

BTW: In debug mode, it is very common to see less than 100% CPU utilization even when using CoWork. This is caused by fact that quite frequently called memory allocator is in debug locked (to do diagnostics agenda).

Mirek

Subject: Re: CoWork buggy!?
Posted by Werner on Mon, 24 Mar 2008 10:54:11 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Mon, 24 March 2008 10:48As far as I understand, CoWork detects number of processors present and tries to execute threads on different cores. Is that so?

I had quite a nice time analyzing "CoWork" (and multithreading under Ultimate++ in general) and sometimes I was painfully wrong , but eventually - withs Mirek's help, of course - I was able to document the whole stuff . So just look at this doc cutting (also published in the hope that Mirek will correct possibly still lurking errors):

Quote:Cf. ...upp/uppsrc/Core/CoWork.cpp

Pool();

Starts number_of_cpu(_core)s_+_2 Threads which will wait for jobs and do them as soon as they

are available.

That is: all these Threads are delivered to the OS to be called by the OS when appropriate.

That means: as soon as Pool is constructed, the machine-dependant maximum number of Threads is active (periodically called by the OS) and waiting for jobs to do.

What, in the end, is periodically called by the OS, are number_of_cpu(_core)s_+_2 ThreadRun-methods (identifiable by the index passed while running "Pool()").

```
CoWork::Pool::Pool()
{
 for(int i = 0; i < CPU_Cores() + 2; i++)   // cpu( core)s + 2
  threads.Add().Run(callback1(&ThreadRun, i)); // add Thread to Pool & start Thread
}
```

Werner

Subject: Re: CoWork buggy!?
Posted by Mindtraveller on Tue, 25 Mar 2008 00:23:09 GMT
View Forum Message <> Reply to Message

Frankly speaking, I don`t see any errors in this code.
Documentation is kind of painful question for me, but Mirek told that new release after 2008 will be focused on that, so for now I`m silent. But I want to say again and again that good manual is critical for U++ successful survival in the nearest and far future - of course, not as author project but as widely used worldwide framework.

Subject: Re: CoWork buggy!?
Posted by Werner on Tue, 25 Mar 2008 10:32:32 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Tue, 25 March 2008 01:23Frankly speaking, I don`t see any errors in this code.
Documentation is kind of painful question for me, but Mirek told that new release after 2008 will be focused on that, so for now I`m silent. But I want to say again and again that good manual is critical for U++ successful survival in the nearest and far future - of course, not as author project but as widely used worldwide framework.

Sorry, it seems that you misunderstood me. This is certainly my fault because I expressed myself quite imprecisely .

The "CoWork::Pool::Pool" code is obviously error-free.

What I wanted to say is:

I analyzed and documented the entire Ultimate++ multithreading code, "CoWork" included (albeit the MSC-related code only). In my previous posting I published a scrap of this documentation (and I'm ready to continue this if somebody really wants).

When I said "published in the hope that Mirek will correct possibly still lurking errors" I meant MY DOCUMENTATION - ***NOT*** Mirek's code.

Sorry again. I sincerely hope that Mirek himself didn't understand it the wrong way. If so, APOLOGIES !

Werner
