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
