Subject: Re: bug in CoWork since C++11
Posted by mirek on Tue, 23 Aug 2016 06:34:45 GMT
View Forum Message <> Reply to Message

crydev wrote on Thu, 18 August 2016 20:58mirek wrote on Tue, 09 August 2016 11:43crydev wrote on Tue, 09 August 2016 11:36mirek wrote on Tue, 09 August 2016 11:15Well, the one possible explanation is that your CoWork is global (or static) variable. Is that so?

My CoWork instance is not a global variable. It is a variable as member of a class and it is not a pointer. It is a singleton class, though. I construct it with a private constructor and a GetInstance method.

```
class X
{
private:
    CoWork mThreadPool;
    X();
public:
    static X* GetInstance()
    {
        static X instance;
        return &instance;
    };
}
```

I see. It is static then.

I believe what happens here is that first, CoWork instance is constructed. Then you schedule the work, which creates global static thread pool. On app exit, pool is therefore destructed BEFORE destructor of CoWork, which would normally performed the remaining work (by calling Finish).

Can you try to call GetInstance()->Finish() at the end of APP_MAIN to prove this hypothesis? (if true, I will then start thinking if resolving this situation is worth the trouble or rather mentioning in docs...

Mirek

Hello Mirek,

I'm having trouble with understanding exactly what you mean, because my app doesn't end when the work is completed. I tried to prove it the way you provided but I am not convinced by the proof. However, I tried a construction where the CoWork instance is not static. I changed 'mThreadPool' to a 'CoWork*' and manually instantiated and destructed the 'mThreadpool' object. However, this does not change the situation for me: it still waits while todo > 0.

While trying to prove your hypothesis, I created a situation where todo > 0 and the work did not complete properly. Then, I exited the application, having it call Finish() and noticed that it passes

the todo > 0 check, but fails at the 'p.scheduled' check, resulting in an infinite wait.

```
void CoWork::Finish() {
 if(!pool) return;
 Pool& p = *pool;
 p.lock.Enter();
 while(todo) {
  LLOG("Finish: todo: " << todo << " (CoWork " << FormatIntHex(this) << ")");
  if(todo == 0) // Doesn't break because todo > 0.
   break;
  if(p.scheduled) // Doesn't pass this check because p.scheduled is NOT true.
   Pool::DoJob(); // Therefore doesn't execute the jobs;
  else {
   LLOG("WaitForFinish (CoWork " << FormatIntHex(this) << ")");
   waitforfinish.Wait(p.lock); // Infinite wait here!
  }
 }
 p.lock.Leave();
 LLOG("CoWork " << FormatIntHex(this) << " finished");
}
```

What can I do?

Thanks,

crydev

I believe you do not have latest trunk version, here is how Finish looks now:

```
void CoWork::Finish() {
 Pool& p = GetPool();
 p.lock.Enter();
 while(!jobs.IsEmpty(1)) {
  LLOG("Finish: todo: " << todo << " (CoWork " << FormatIntHex(this) << ")");
  p.DoJob(*jobs.GetNext(1));
 }
 while(todo) {
  LLOG("WaitForFinish (CoWork " << FormatIntHex(this) << ")");
  waitforfinish.Wait(p.lock);
 }
 p.lock.Leave();
 LLOG("CoWork " << FormatIntHex(this) << " finished");
}
```