Subject: Simple Thread
Posted by lovmy on Mon, 05 Oct 2020 14:16:10 GMT
View Forum Message <> Reply to Message

Hi,

i try to create a simple thread in my application, this is a part of the code:

```
Processus::Processus() {}

Processus::~Processus() {}

void Processus::Start( String parametres )
{
 Thread().Run(THISBACK1(Boucle,parametres));
 while(Thread().GetCount());
}

void Processus::Boucle( String parametres )
{
 for (int i = 0; i < 10; ++i)
 {
  std::cout << i << ",";
 }
}
```

I have error:

F:\upp/uppsrc/Core/Callback.h (110): error: 'CLASSNAME' has not been declared

Can you help me ?

Subject: Re: Simple Thread
Posted by Klugier on Mon, 05 Oct 2020 16:12:08 GMT
View Forum Message <> Reply to Message

Hello,

It seems that you have missed "typedef Processus CLASSNAME;" (C++03) "using CLASSNAME =Processus" (c++11) declaration in your class. However, in c++11 world you don't need this just replace with lambda:

```
void Processus::Start( String parametres )
{
	Thread thr;
 thr.Run([=] { Boucle(parametres); }); // <- In previous implementation you delete thread model
```

```
immediately...
 while(thr.GetCount()); // <- Not very nice synchronization mechanism :)
}

void Processus::Boucle( String parametres )
{
 for (int i = 0; i < 10; ++i)
 {
  Cout() << i << ","; // <- Cout() better in U++ world!
 }
}
```

My working test code below:

```
#include <Core/Core.h>

using namespace Upp;

class Processus {
public:
 void Start( String parametres )
 {
  Thread thr;
  thr.Run([=] { Boucle(parametres); }); // <- In previous implementation you delete thread model
immediately...
  while(thr.GetCount()); // <- Not very nice synchronization mechanism :)
 }

 void Boucle( String parametres )
 {
  for (int i = 0; i < 10; ++i)
  {
   Cout() << i << ",";
  }
 }

};

CONSOLE_APP_MAIN
{
 Processus().Start("asda");
}
```

Klugier

Subject: Re: Simple Thread
Posted by Oblivion on Mon, 05 Oct 2020 17:04:23 GMT
View Forum Message <> Reply to Message

Klugier has put it nicely.

However, there is an even simpler way if you need async behaviour, using AsyncWork worker threads.

```
CONSOLE_APP_MAIN
{
 auto Boucle = [](String parametres) -> void
 {
  for (int i = 0; i < parametres.GetLength(); ++i)
  {
   Cout() << String(parametres[i], 1) << ",";
  }
 };

 Async(Boucle, "Hello world").Get();
}
```

Note that AsyncWork is the Upp interpretation and implementation of future/promise pattern and there are other ways to utilize it.

Best regards,
Oblivion

Subject: Re: Simple Thread
Posted by lovmy on Tue, 06 Oct 2020 06:15:50 GMT
View Forum Message <> Reply to Message

Hello !

Thank you for your help, i need to learn more on lambda expression...

Just a question, what's the différence between thread and CoWork ?

Have a nice day !

Subject: Re: Simple Thread
Posted by mirek on Tue, 06 Oct 2020 07:33:22 GMT

lovmy wrote on Tue, 06 October 2020 08:15Hello !

Thank you for your help, i need to learn more on lambda expression...

Just a question, what's the différence between thread and CoWork ?

Have a nice day !

CoWork is basically an interface to thread pool. That is a pool of waiting threads that get used on demand - this saves you the costs of creating and exiting threads.

CoWork allows you to schedule any number of jobs to be processed, possibly in parallel and then wait for all of them to be done.

That said, for most of work, CoWork is now superseded by CoDo, which basically starts the same job, usually represented by lambda, on as many CPU cores as possible.

Long story short:

If you are after improving performance by parallelizing work, check CoDo first (most iterations are best done in CoDo).

If CoDo is not suitable, check CoWork (e.g. quick-sort cannot be implemented with CoDo nicely, but can be implemented with CoWork).

For some background tasks, consider Async.

If you are after something else entirely, like long running background tasks (e.g. a thread that backups data in your application periodically), use raw Thread.

Eh, that was not so short after all :)

Mirek