Subject: Coroutines package for U++
Posted by Oblivion on Sat, 05 Nov 2022 11:19:16 GMT
View Forum Message <> Reply to Message

Hi,

I have added an experimental CoRoutines package to both UppHub and upp-components repo.
It provides a simple and basic interface for building coroutines.

Main github repo (for pull request etc.):

There are two types of CoRoutine in the package:

CoRoutine -> A regular coroutine that can return any value (or void)
CoGenerator -> A regular generator type function.

- The classes provided with the package expose "traditional" Upp-like interface (Do(), Get(), Pick() and Next() -methods, depending on their type.)

- Allows exception handling and propagation.

- Also an example code is included.

Example:

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

using namespace Upp;

CoRoutine<Vector<int>> CoMakeIota(int begin, int end, int step)
{
 Vector<int> v;
 for(int i = begin; i < end; i += step) {
  v.Add(i);
  RLOG(__func__);
  co_await CoSuspend();
 }
 co_return v;
}

CoGenerator<int> CoGenerateNumbers()
{
 int i = 0;
 for(;;) {
  RLOG(__func__);
  co_yield i++;
```

```
 }
}

CONSOLE_APP_MAIN
{
 StdLogSetup(LOG_COUT|LOG_FILE);

 auto co1 = CoMakeIota(0, 10, 2);
 auto co2 = CoGenerateNumbers();

 try {
  while(co1.Do())
   RLOG(__func__); // NOP;
  auto v = co1.Pick();
  RDUMP(v);

  for(int i = 0; i < 10; i = co2.Next())
   RDUMP(i);
 }
    catch(...) {
        RLOG("Unhandled exception");
    }

}
```

Suggestion, pull requests, reviews, etc. are welcome.

Best regards,
Oblivion

Subject: Re: Coroutines package for U++
Posted by Oblivion on Sun, 06 Nov 2022 19:20:14 GMT
View Forum Message <> Reply to Message

Hi,

Trivial iterator support is added for CoGenerator type coroutines.

Example:

```
CONSOLE_APP_MAIN
{
 StdLogSetup(LOG_COUT|LOG_FILE);
```

```
 auto generator = []() -> CoGenerator<int> {
  for(int i = 0;; i++)
   co_yield i;
 };

 for(auto i : generator()) {
  RLOG(i);
  if(i == 100)
   break;
 }

 for(auto i :[]() -> CoGenerator<int> { int i = 0; while(true) co_yield i++; }()) {
  RLOG(i);
  if(i == 100)
   break;
 }
}
```

There are rough edges, but they'll be smoothed in the following days...

Best regards,
Oblivion

Subject: Re: Coroutines package for U++
Posted by Klugier on Sun, 06 Nov 2022 19:44:57 GMT
View Forum Message <> Reply to Message

Hello Oblvion,

Why you decieded to create separate package in concuret to Core? Is there Co* family functions like CoWork, AsyncWork, CoPartition and CoDo not enough for your needs?

Klugier

Subject: Re: Coroutines package for U++
Posted by Oblivion on Sun, 06 Nov 2022 20:31:00 GMT
View Forum Message <> Reply to Message

Hello Klugier,

Quote:Hello Oblvion,

Why you decieded to create separate package in concuret to Core? Is there Co* family functions like CoWork, AsyncWork, CoPartition and CoDo not enough for your needs?

On the contary, I use them daily.

However, It seems that there is a misunderstading here.

C++20 coroutines are not "concurrency" functions. They are not threads. They are suspendable functions, meaning that you can return from the function in the middle of its operation and you can resume it later. This makes async programming without threads impressively easy!

Excerpt:

A coroutine is a function that can suspend execution to be resumed later. Coroutines are stackless: they suspend execution by returning to the caller and the data that is required to resume execution is stored separately from the stack. This allows for sequential code that executes asynchronously (e.g. to handle non-blocking I/O without explicit callbacks), and also supports algorithms on lazy-computed infinite sequences and other uses.

While it is one of the coolest features of C++20, the original design is, well, let's say it isn't very elegant... (no wonder even seasoned developers are having a hard time understanding how to use them - watch the cppcon's ever-increasing videos about them.)

So, I haven't created a "yet another" thread-based concurrency tool, in the traditional sense. I have implemented and made available something completely new to U++. (hence the upload.)

Best regards,
Oblivion

Subject: Re: Coroutines package for U++
Posted by Lance on Sun, 06 Nov 2022 23:14:49 GMT
View Forum Message <> Reply to Message

I did some casual search on coroutine but never figured out its intended use.

Quote:
A coroutine is a function that can suspend execution to be resumed later. Coroutines are stackless: they suspend execution by returning to the caller and the data that is required to resume execution is stored separately from the stack. This allows for sequential code that executes asynchronously (e.g. to handle non-blocking I/O without explicit callbacks), and also supports algorithms on lazy-computed infinite sequences and other uses.

This excerpt is very helpful.

Thank you, Oblivion! If you can add some small, non-trivial examples to your library, it will be great!

Regards,
Lance

Subject: Re: Coroutines package for U++
Posted by koldo on Mon, 07 Nov 2022 07:39:55 GMT
View Forum Message <> Reply to Message

Yes, this is interesting.

To understand this better, excuse me for asking you a "devil's advocate" question: why does a programmer need to use coroutines?

Subject: Re: Coroutines package for U++
Posted by peterh on Mon, 07 Nov 2022 17:38:18 GMT
View Forum Message <> Reply to Message

Hi,

Out of interest I tried the example from UppHub.

(I have used cooperative coroutines 40 years ago on DOS in Borland Pascal (and in Modula II but the latter only experimental).
We used a commercial library. The program was rather big and consisted out of a base routine and overlays which where swapped.
We used this for a program that had a complex user interface and that must communicate with a running machine, store the process data in realtime, visualize data and control machine parameters interactively, all at the same time, using a single computer)

So far I have read, C++20 coroutines are different, these are "stackless" coroutines, which implies some limitations. (cannot suspend or yield in a subroutine)
I do not fully understand it yet and the purpose of it.

I noticed, if I set a breakpoint inside the body of a coroutine, then it is not hit.
Edit: A call to DebugBreak() is hit.
If I set a breakpoint at the entry point of a coroutine, it is hit.
If I call a subroutine from a coroutine and set a breakpoint inside this subroutine, then it is hit.

Subject: Re: Coroutines package for U++

Posted by Oblivion on Mon, 07 Nov 2022 19:07:01 GMT
View Forum Message <> Reply to Message

Hello koldo,
Quote:why does a programmer need to use coroutines?

Simple: To write non-blocking APIs or programs without spaghetti code: Recall the initial version of our very own SSH package. It was designed to be fully NB. This was achieved using a blend of an event queue, storing callbacks in a bivector (in a predefined order) and tracking their state. It did work, but the underlying code was very complex. (Fun fact: The same mechanism (but a stripped down version) is still used in NetProxy package.)

Now, the coroutines eliminates BOTH. You don't need to track the state of the internals of the NB object externally or worry about a state machine. Basically it is done for you by the compiler.

You write a local code block, and mark some points (usually where the code WOULDBLOCK) to suspend the coroutine and the compiler simply suspends it and returns the control back to its caller. The suspended coroutine can be resumed later from where it is left off.

This simplifites writing NB/async apps without using threads. Not to mention that you you don't need to worry about on how to terminate it. (CoRoutines are scope bound objects.)

Hello Peter:

Quote:I do not fully understand it yet and the purpose of it.

Well I don't think this debate (stackfull/stackless, i mean) will ever end --theoretically, at least. In practice, however, stackless coroutines are selected for C++20 (as in AFAIK C# and Python).

They can't be nested directly, yes.Some see this as a disadvantage, some don't. (I'm on the latter camp, since it forces simpler coroutines. (as no nesting is possible.)

As for the GDB: Yes it still can't handle coroutines properly..

A side note: I will add more complex examples to the package along with docs...

Best regards,

Oblivion

Subject: Re: Coroutines package for U++
Posted by peterh on Mon, 07 Nov 2022 19:30:16 GMT
View Forum Message <> Reply to Message

I understand it this way, C++20 coroutines are meant to write fast state machines in a simpler way.
In earlier times in plain old C longjump() could be (ab)used for this, but more complicated.

I have implemented sort of a coroutine in assembler for a device driver for an embedded device which had to handle XON/XOFF handshake and timeout properly without deadlock and I implemented the interrupt routine as a coroutine which was called by UART interrupt. This simplified the job a lot.

I do yet not understand, where does a C++ coroutine store its local variables?
On yield the stack is destroyed, so it must store the local variables elsewhere because these are persistent. (Local variables are still alive at the next invocation)

Subject: Re: Coroutines package for U++
Posted by Oblivion on Mon, 07 Nov 2022 19:31:41 GMT
View Forum Message <> Reply to Message

Hello Peter,

Quote:
I understand it this way, C++20 coroutines are meant to write fast state machines in a simpler way.

Yes, that's what I tried to explain above. They are meant to write very fast state machines in a very simple way.

Quote:I do yet not understand, where does a C++ coroutine store its local variables?

Heap. (With some compiler optimizations, IIRC)

Best regards,
Oblivion

Subject: Re: Coroutines package for U++
Posted by peterh on Tue, 08 Nov 2022 08:09:09 GMT
View Forum Message <> Reply to Message

Thank you.
I tried to compile this with MSVC22, but when I add "/std:c++20" to C++ options, then Upp does not compile. I get a lot of error messages.
Is it possible?

Subject: Re: Coroutines package for U++
Posted by Oblivion on Wed, 09 Nov 2022 15:22:28 GMT
View Forum Message <> Reply to Message

Sorry, I haven't test it on MSC yet. (I usually work on CLANG/GCC and they compile U++ with C++20 spec.).

But I will asap.

Best regards,
Oblivion

Subject: Re: Coroutines package for U++
Posted by Oblivion on Wed, 09 Nov 2022 21:52:53 GMT
View Forum Message <> Reply to Message

Hello,

As per Lance's request, I am adding some "non-trivial" coroutines examples to the package.
To demonstrate its usage, I have started with adapting the uppsrc/reference/GuiWebDownload example, so that it can be compared:

```
#include <CtrlLib/CtrlLib.h>
#include <CoRoutines/CoRoutines.h>


// GuiWebDownloader example, adapted to coroutines.

using namespace Upp;


CoRoutine<void> HttpDownload(const String& url)
{
 Progress pi;
 String path = AppendFileName(Nvl(GetDownloadFolder(), GetHomeDirFile("downloads")),
GetFileName(url));
 HttpRequest http(url);
 http.Timeout(0);
 int loaded = 0;
 {
  FileOut out(path);
  http.WhenContent = [&out, &loaded](const void *ptr, int size) { out.Put(ptr, size); loaded += size;
};
  while(http.Do()) {
```

```
   if(http.GetContentLength() >= 0) {
    pi.SetText("Downloading " + GetFileName(url));
    pi.Set((int)loaded, (int)http.GetContentLength());
   }
   else {
    pi.Set(0, 0);
    pi.SetText(http.GetPhaseName());
   }
   if(pi.Canceled())
    http.Abort();
   else
    co_await CoSuspend();
  }
 }
 if(http.IsFailure()) {
  DeleteFile(path);
  Exclamation("Download has failed.&\1" +
   (http.IsError()
    ? http.GetErrorDesc()
     : AsString(http.GetStatusCode()) + ' ' + http.GetReasonPhrase()));
 }
}

GUI_APP_MAIN
{
 StdLogSetup(LOG_COUT|LOG_FILE);

 String url = "http://downloads.sourceforge.net/project/upp/upp/4179/upp-x11-src-4179.tar.gz";

 for(;;) {
  if(!EditText(url, "Download", "URL"))
   return;
  auto downloader = HttpDownload(url); // Multiple downloaders can be created/run at once.
  try
  {
   while(downloader.Do())
    Sleep(1); // Simulate work.
  }
  catch(const Exc& e)
  {

  }
  catch(...)
  {

  }
 }
}
```

Best regards,
Oblivion

Subject: Re: Coroutines package for U++
Posted by Lance on Wed, 09 Nov 2022 22:26:53 GMT
View Forum Message <> Reply to Message

Great, thank you, Oblivion!

I have been searching on material on C++20 new language features and library additions. coroutine has been puzzling me for a while. I will spend some time hopefully over the weekend to digest what you have kindly provided.

BR,
Lance

Subject: Re: Coroutines package for U++
Posted by Lance on Fri, 11 Nov 2022 23:33:54 GMT
View Forum Message <> Reply to Message

It will take more study for me to understand coroutine. It's still like magic to me.

Subject: Re: Coroutines package for U++
Posted by Oblivion on Sat, 12 Nov 2022 06:22:36 GMT
View Forum Message <> Reply to Message

Hello Lance,

One of the -now-old but- good explanations (this is more general presentation on concurrency and where coroutines stand): https://youtu.be/1Wy5sq3s2rg
The other newest one, which I find very useful: https://youtu.be/J7fYddslH0Q
Best regards,
Oblivion

Subject: Re: Coroutines package for U++
Posted by Lance on Sun, 13 Nov 2022 04:37:19 GMT
View Forum Message <> Reply to Message

Thank you, Oblivion!

Subject: Re: Coroutines package for U++
Posted by Lance on Mon, 14 Nov 2022 01:17:00 GMT
View Forum Message <> Reply to Message

This article seems to be quite interesting:
C++20 Coroutines and io_uring

Subject: Re: Coroutines package for U++
Posted by Oblivion on Mon, 14 Nov 2022 04:56:41 GMT
View Forum Message <> Reply to Message

Hello Lance,

Quote:This article seems to be quite interesting:
C++20 Coroutines and io_uring

Yes, a really nice article that sums up what coroutines/awaitable is useful for. Thank you for this article!

This is the reason why I made my package a "light-weight" implementation of this nice cpp20 feature, and this is the reason why I haven't (yet) implemented a CoAwaiter (using awaitables). IMHO, what the author achieves by using awaitables + threads, we already have it: Upp::CoDo & Upp::CoFor. And they work on also earlier versions of C++.

The main use of CoRoutines package is to simplify writing async/single threaded code and easily manageable generators.

Best regards,
Oblivion

Subject: Re: Coroutines package for U++
Posted by Lance on Mon, 14 Nov 2022 14:08:43 GMT
View Forum Message <> Reply to Message

It will take some time for me to fully understand the point the author is trying to make . There is also a reference in his article to a blog of one of the author(supposedly) of coroutine TS which seems to be a better point to start with.

Asymmetric Transfer

Subject: Re: Coroutines package for U++
Posted by zsolt on Mon, 12 Dec 2022 12:48:29 GMT

View Forum Message <> Reply to Message

I did not study coroutines yet, but I have a question, maybe you can answer it.
Is it possible some easy way to write a function / method to start rendering some big images or PDFs on other threads and awaiting them, to complete, not blocking the main (GUI) thread, that started it?
And canceling that rendering threads automatically if I leave that context by closing the dialog, that started it?

Subject: Re: Coroutines package for U++
Posted by Oblivion on Mon, 12 Dec 2022 16:42:51 GMT
View Forum Message <> Reply to Message

Hello zsolt

Quote:Is it possible some easy way to write a function / method to start rendering some big images or PDFs on other threads and awaiting them, to complete, not blocking the main (GUI) thread, that started it?

In my experience the real question would be: Will it worth it?

My answer would be: No. Because what you describe is already achievable by using Upp::CoWork or Upp::AsyncWork. You can offload a rendering operation using these tools without blocking the gui, check their status and wait for them to join in non-blocking way and cancel them when you need to (AsyncWork has a Cancel method). In fact, I have a small utility (app) called SshGet (an MT Sftp/Scp transfer queque), where I do someting very similar to what you describe (instead of rendering stuff, I am transfering them).

However, the direct answer to your questions is that it is partially possible. cpp/coroutines have a concept called awaitables. You can write an awaitable type (class or functor) and pass the coroutine handle among threads in a lock free way via awaitable. But 1) It is complicated, 2) It wont solve the real problem: termination of a thread. Plus, what you achive in the end will seem to be what CoWork et al. can already do.

This is the reason why I opted to implement a light-weight coroutines interface (only routine and generator types and not the awaitable type.) This makes things simple and clean.

Best regards,
Oblivion

Subject: Re: Coroutines package for U++
Posted by zsolt on Mon, 12 Dec 2022 20:38:12 GMT
View Forum Message <> Reply to Message

Thanks, Oblivion. I have started to get the idea behind these concepts, checking your code.

I was thinking about if it could be possible to implement something so simple as Javascript's async/await.
But I concluded the same as you.

Maybe it could be done by combining your coroutine with your AsyncWork and a timer infrastructure calling coroutines until their finished state, but too much work compared to using AsyncWork and showing a progress or using SetTimeCallback() to update the window periodically.