**Lecture extra - Asyncronous calls and coroutines**

**Coroutines**

Coroutine = a code fragment that can get suspended and to be resumed later. A coroutine is a bit similar to a thread, but it explicitly transfer the control to another coroutine.

Simplest examples: Windows fibers. Basic API:

* CreateFiber()
* SwitchToFiber()

Example: [coroutine-fiber-sample.cpp](https://www.cs.ubbcluj.ro/~rlupsa/edu/pdp/progs/coroutine-fiber-sample.cpp)

Coroutine uses:

* Producer - consumer;
* Switching to something else when the current code has to wait for some external event, and continue after the external event arrives.

**Generators**

Python generator example: [coroutine-generator.py](https://www.cs.ubbcluj.ro/~rlupsa/edu/pdp/progs/coroutine-generator.py)

Windows fibers implementation: [coroutine-generator.cpp](https://www.cs.ubbcluj.ro/~rlupsa/edu/pdp/progs/coroutine-generator.cpp)

**Async-await mechanism**

* calling an *async* function creates a coroutine and starts it
* *await* switches to some other coroutine; completing its target *future* switches back to it
* *await* switching to the caller coroutine returns a *future*
* finishing the coroutine function completes the future

Simple demo: [async-await-short-demo.cs](https://www.cs.ubbcluj.ro/~rlupsa/edu/pdp/progs/async-await-short-demo.cs)

Downloading web pages examples

async Task Download(...) {

await Connect();

...

while(...) {

await Recv(...);

}

}

for(...) {

tasks[i] = Download(...)

}

Task.WaitAll(tasks);

**C++20 coroutines**

* Coroutines are stackless -> switching can occur only from top level function
* Coroutine = function containing co\_await, co\_yield or co\_return
* Return type = wrapper over coroutine handle (std::coroutine\_handle<promise\_type>)
* Return type defines various behaviors of the coroutine
* ...

**More about asynchronous functions**

Asynchronous function = function that starts some operations that will complete later.

Get the result: polling, blocking, or interrupt (via some callback).

(simple) callbacks - a bit hard to use

*Future* - allows polling or blocking. Advanced futures allow adding a callback, which is even harder to use.

Higher level handling: some developing on futures; or coroutines.

Composing asynchronous calls:

Task t1 = Func1();

Task t2 = t1.ContinueWith(Func2).Unwrap();