CS 340
Computer Systems

#25: Threading in Python

Apr. 25, 2023 · Wade Fagen-Ulmschneider

Programming Language Support for Threading

Most modern programming languages provide **language-level support for threading**:

25/async-await.py
```
 1  import asyncio
 2
 3  async def fibonacci(x, tid):
 4    # Base Cases:
 5    if x == 0: return 0
 6    if x == 1: return 1
 7
 8    print(f"{tid}: Calculating fibonacci({x})...")
 9    await asyncio.sleep(0.1)
10    fx_minus1 = await fibonacci(x - 1, tid)
11    fx_minus2 = await fibonacci(x - 2, tid)
12
13    return fx_minus1 + fx_minus2
```

The **async** keyword wraps the function (formally called a "coroutine") as an **Future** object.

- A **Future** object:

A **Future** has three states:

[1]: Unfulfilled:

[2]: Fulfilled:

[3]: Failed:

As a procedural programming language, the **await** keyword exists to synchronize your code based on the result of a **Future**:

25/async-await.py
```
10    fx_minus1 = await fibonacci(x - 1, tid)
11    fx_minus2 = await fibonacci(x - 2, tid)
```

You can "race" all multiple **Future** objects:

25/async-await.py
```
15  async def main():
16    r = await asyncio.gather(
17      fibonacci(15, "A"),
18      fibonacci(14, "B"),
19      fibonacci(13, "C"),
20    )
21
22    print(r)
```

Q: What output do we get?

Since every **async** function is just **Future**, you must **asyncio.run** your first one **async** function (often a function called **main**):

25/async-await.py
```
24  asyncio.run(main())
```

Otherwise: Python does nothing (but does provide a warning):

INCORRECT version of async-await.py:
```
24  main()
```
```
async-await.py:24: RuntimeWarning: coroutine 'main' was
never awaited
  main()
RuntimeWarning: Enable tracemalloc to get the object
allocation traceback
```

Multithreading in Python

Python is multi-threaded, but ________________________________:

25/countup.py

```python
import asyncio

ct = 0
THREAD_COUNT_AMOUNT = 10000000

async def countup():
  global ct
  for i in range(THREAD_COUNT_AMOUNT):
    ct += 1

async def main():
  await asyncio.gather(
    countup(),
    countup(),
    countup(),
  )

  print(ct)

asyncio.run(main())
```

Q: When we did this in C, what happened?

Q: What do we expect to happen in Python?

Q: What is going on that is different in Python than C?

Python is multi-threaded, but ________________________________:

25/countup.py

```python
 1  import asyncio
 2  import sys
 3
 4  ct = 0
 5  THREAD_COUNT_AMOUNT = 10000000
 6
 7  async def countup(tid):
 8    global ct
 9    for i in range(THREAD_COUNT_AMOUNT):
10      if i % 10000 == 0:
11        sys.stdout.write(tid)
12        sys.stdout.flush()
13
14      ct += 1
15      await asyncio.sleep(0)
16
17  async def main():
18    await asyncio.gather(
19      countup("A"),
20      countup("B"),
21      countup("C"),
22    )
23
24    print(ct)
25
26  asyncio.run(main())
```

Q: What is the difference between `countup` and `countup2`?

Q: What happens when we run this code with `:15` commented out?

....and if it's not commented out?

Q: What can we learn about how Python handles threading verses C?