BitTorrent Client

Python’s AsyncIO

asyncio is a built-in Python module that **allows writing asynchronous code** using **coroutines, event loops, and tasks**. It is mainly used for **concurrent execution** in tasks like:

- **Handling multiple network requests** (e.g., making multiple API calls)
- **Reading/writing files asynchronously** (e.g., handling large files)
- **Building chat servers** (e.g., handling multiple clients in real-time)

Synchronous vs. Asynchronous Execution

Synchronous (Blocking) Code

```
import time

def task1():
    print("Task 1 starting")
    time.sleep(3)  # Simulates delay (blocking)
    print("Task 1 done")

def task2():
    print("Task 2 starting")
    time.sleep(2)  # Simulates delay (blocking)
    print("Task 2 done")

task1()
task2()
print("All tasks done")
```

Total time = 3s + 2s = 5 seconds

Asynchronous (Non-Blocking) Code

```
import asyncio

async def task1():
    print("Task 1 starting")
    await asyncio.sleep(3)  # Simulates async delay (non-blocking)
    print("Task 1 done")

async def task2():
    print("Task 2 starting")
    await asyncio.sleep(2)  # Simulates async delay (non-blocking)
    print("Task 2 done")

async def main():
    await asyncio.gather(task1(), task2())  # Runs both tasks in parallel

asyncio.run(main())  # Runs the event loop
```

Total time ≈ max(3s, 2s) = 3 seconds (faster than the sync version)

Since BitTorrent clients need to handle **multiple downloads/uploads concurrently**, asyncio is useful for:

- Downloading multiple pieces of a file **simultaneously**
- Communicating with multiple peers **at the same time**
- Handling **network requests asynchronously**

Coroutines

A **coroutine** is a special kind of function that can **pause** its execution (yield control) and **resume** later.

async def	Defines a coroutine (like def defines a function)
Coroutine	A paused task that can be resumed (async def returns this)
await	Pauses current coroutine until the awaited coroutine finishes
asyncio	Python’s built-in library to manage coroutines and async execution

---

Introduction to BitTorrent

BitTorrent is a peer-to-peer protocol, where *peers* join a *swarm* of other peers to exchange pieces of data between each other. Each peer is connected to multiple peers at the same time, and thus downloading or uploading to multiple peers at the same time.

Bencoding

```
>>> from pieces.bencoding import Decoder

# An integer value starts with an 'i' followed by a series of
# digits until terminated with a 'e'.
>>> Decoder(b'i123e').decode()
123

# A string value, starts by defining the number of characters
# contained in the string, followed by the actual string.
# Notice that the string returned is a binary string, not unicode.
>>> Decoder(b'12:Middle Earth').decode()
b'Middle Earth'

# A list starts with a 'l' followed by any number of objects, until
# terminated with an 'e'.
# As in Python, a list may contain any type of object.
>>> Decoder(b'l4:spam4:eggsi123ee').decode()
[b'spam', b'eggs', 123]
```

```
# A dict starts with a 'd' and is terminated with a 'e'. objects
# in between those characters must be pairs of string + object.
# The order is significant in a dict, thus OrderedDict (from
# Python 3.1) is used.
>>> Decoder(b'd3:cow3:moo4:spam4:eggse').decode()
OrderedDict([(b'cow', b'moo'), (b'spam', b'eggs')])
```

Difference Between Strings and Byte Strings

Strings (str)

- Text-based, human-readable
- Uses Unicode encoding

```
text = "hello"
print(type(text))  # Output: <class 'str'>
```

Byte Strings (bytes)

- Raw byte data
- Uses b"" notation

```
btext = b"hello"
print(type(btext))  # Output: <class 'bytes'>
```

Converting Between Strings and Bytes

- **String → Bytes:** encode()
- **Bytes → String:** decode()

```
text = "hello"
btext = text.encode()  # Convert str to bytes
print(btext)  # Output: b'hello'
```

```
decoded_text = btext.decode()  # Convert bytes to str
print(decoded_text)  # Output: 'hello'
```

Parsing .torrent file

There is a `.torrent` file that regulates how many pieces there is for a given file(s), how these should be exchanged between peers, as well as how the data integrity of these pieces can be confirmed by clients.

- The name of the file to download
- The size of the file to download
- The URL to the tracker to connect to

All these properties are stored in a binary format called *Bencoding*.

```
>>> with open('tests/data/ubuntu-16.04-desktop-amd64.iso.torrent', 'rb') as
f:
...     meta_info = f.read()
...     torrent = Decoder(meta_info).decode()
...
>>> torrent
OrderedDict([(b'announce', b'http://torrent.ubuntu.com:6969/announce'), (b'a
nnounce-list', [[b'http://torrent.ubuntu.com:6969/announce'], [b'http://ipv6.tor
rent.ubuntu.com:6969/announce']
]), (b'comment', b'Ubuntu CD releases.ubuntu.com'), (b'creation date', 146123
2732), (b'info', OrderedDict([(b'length', 1485881344), (b'name', b'ubuntu-16.0
4-desktop-amd64.iso'), (b'piece
length', 524288), (b'pieces', b'\x1at\xfc\x84\xc8\xfaV\xeb\x12\x1c\xc5\xa4\x1
c?\xf0\x96\x07P\x87\xb8\xb2\xa5G1\xc8L\x18\x81\x9bc\x81\xfc8*\x9d\xf4k\x
e6\xdb6\xa3\x0b\x8d\xbe\xe3L\xfd\xfd4\...')]))])
```

Connecting to Tracker

The `announce` property in the *meta-info* is the HTTP URL to the tracker to connect to using the following URL parameters:

Parameter	Description
info_hash	The SHA1 hash of the info dict found in the `.torrent`
peer_id	A unique ID generated for this client
uploaded	The total number of bytes uploaded
downloaded	The total number of bytes downloaded
left	The number of bytes left to download for this client
port	The TCP port this client listens on
compact	Whether or not the client accepts a compacted list of peers or not

Generating 20-byte peer ID

```
>>> import random
# -<2 character id><4 digit version number>-<random numbers>
>>> '-PC0001-' + ''.join([str(random.randint(0, 9)) for _ in range(12)])
'-PC0001-478269329936'
```

A tracker request can look like

```
http GET "http://torrent.ubuntu.com:6969/announce?info_hash=%90%28%9
F%D3M%FC%1C%F8%F3%16%A2h%AD%D85L%853DX&peer_id=-PC0001-
706887310628&uploaded=0&downloaded=0&left=699400192&port=6889&co
mpact=1"
HTTP/1.0 200 OK
Content-Length: 363
Content-Type: text/plain
Pragma: no-cache
```

```
d8:completei3651e10:incompletei385e8:intervali1800e5:peers300:£¬%ËÌyO
kÝ.ê@_<K+Ô\Ý Ámb^TnÈÕ^AËO*ÈÕ1*ÈÕ>¥³ÈÕBä)ðþ¸ÐÞ¦Ô/ãÈÕÈuÉæÈÕ
...
```

From the tracker response, there is two properties of interest:

- **interval** - The interval in seconds until the client should make a new announce call to the tracker.
- **peers** - The list of peers is a binary string with a length of multiple of 6 bytes. Where each peer consist of a 4 byte IP
  address and a 2 byte port number (since we are using the compact
  format).

Architecture

- Pieces - Original file to be shared is split into equal sized pieces, 256KB to 1 MB. SHA1 hashes of each piece is added to .torrent file under ‘pieces’ attribute. Piece is fetched by its SHA1. Downloaded file is verified using the same hash.
- Tracker - Only central entity for a torrent.
  - Tracks peers who are downloading and uploading the file.
  - Helps peer find other peer to download.

The Choke Algorithm

Free riders are peers that never upload, but always download, should be penalized. Downloading and uploading should be based on reciprocation. Choking is a temporary refusal to upload.

- **Choking** a peer means **refusing to upload** to them.

- **Unchoking** means allowing them to download from you.
- Algorithm:
    1. The peer looks at all other peers it's connected to.
    2. It sorts **interested peers** by their **upload rate to you**.
    3. It **unchokes the top 4** (default is 4) fastest uploaders.
    4. The rest remain **choked** (not allowed to download from you).

Unchoking algorithm:

- One **randomly chosen interested peer** is **optimistically unchoked**, even if it's not one of the top 4.
- This gives **new peers or slow ones** a chance to prove themselves.
- If they start uploading to you, they might enter the top 4 in the next round.

Piece Selection Algorithm

1. Rarest-First Piece First Algo.

Kademlia - DHT

#TODO

Resources

https://markuseliasson.se/article/introduction-to-asyncio/

https://markuseliasson.se/article/bittorrent-in-python/