**Title:**

**Python Threading and Concurrency – Task Report**

Internship @ CyArt

Submitted by: Vedant Patil

**Introduction:**

This report documents the Python concurrency task assigned during my internship at CyArt. The objective was to explore and implement multithreading techniques in Python, understand the limitations of the Global Interpreter Lock (GIL), and safely handle shared resources using synchronization primitives like **Lock**. The task involved measuring performance improvements and ensuring data safety across threads.

**Threading and the Global interpreter Lock (GIL)**

* **Python’s Threading Module:**

Python’s **threading** module allows concurrent execution of tasks, especially useful for I/O-bound operations. Each thread runs independently but shares memory with other threads.

* **GIL (Global Interpreter Lock):**

The GIL ensures only one thread executes Python bytecode at a time. This restricts Python threads from achieving parallelism in **CPU-bound** tasks but doesn't impact **I/O-bound** performance significantly.

Sources:

* https://docs.python.org/3/library/threading.html
* https://realpython.com/python-gil**/**
* **Locks and Synchronization:**

**threading.Lock** is used to synchronize access to shared resources. It prevents race conditions by allowing only one thread to access the critical section at a time.

**Code:**

lock = threading.Lock()

with lock:

# critical section

perform\_operation()

**Task 1 – Image Downloader:**

**Objective:**

Download 20 images using:

* A single-threaded script
* A multithreaded script with 20 threads

**Single-threaded Version (image\_downloader\_single.py)**

import requests

import time

from pathlib import Path

IMG\_COUNT = 20

SAVE\_DIR = Path("downloads\_single")

SAVE\_DIR.mkdir(exist\_ok=True)

*#Function to download images.*

def download\_image(index: int) -> None:

    url = "https://picsum.photos/200"   *#using this site to download images.*

    response = requests.get(url)

    if response.status\_code == 200:

        with open(SAVE\_DIR / f"image\_{index}.jpg", "wb") as f:

            f.write(response.content)

    else:

        print(f"Failed to download image {index}")

def main():

    start = time.perf\_counter()

    for i in range(IMG\_COUNT):

        download\_image(i)

    end = time.perf\_counter()

    print(f"Single-threaded download took {end - start:.2f} seconds")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

This above version performs image downloads sequentially.

**Multithreaded Version (image\_downloader\_threaded.py)**

import threading

import requests

import time

from pathlib import Path

IMG\_COUNT = 20

SAVE\_DIR = Path("downloads\_threaded")

SAVE\_DIR.mkdir(exist\_ok=True)

*#Function to download images.*

def download\_image(index: int) -> None:

    url = "https://picsum.photos/200"

    response = requests.get(url)

    if response.status\_code == 200:

        with open(SAVE\_DIR / f"image\_{index}.jpg", "wb") as f:

            f.write(response.content)

    else:

        print(f"Thread {index}: failed to download image")

def main():

    threads = []

    start = time.perf\_counter()

    for i in range(IMG\_COUNT):

        t = threading.Thread(target=download\_image, args=(i,))

        threads.append(t)

        t.start()

    for t in threads:

        t.join()

    end = time.perf\_counter()

    print(f"Threaded download took {end - start:.2f} seconds")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

This version spawns 20 threads to download concurrently.